$Y'UV$-to-$R'G'B'$ Color Space Conversion on FPGA-augmented TriMedia-32 Processor

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

Abstract— **This paper investigates $Y'UV$-to-$R'G'B'$ color space conversion on FPGA-augmented TriMedia-32 processor. First, we outline the extension of TriMedia-32 architecture consisting of FPGA-based Reconfigurable Functional Units (RFU) and associated generic instructions. Then we analyse a YUV-RGB (RFU–specific) instruction which can process four pixels per call, and propose a scheme to implement the YUV-RGB operation on RFU. When mapped on an ACEX EP1K100 FPGA, the proposed YUV-RGB exhibits a latency of 10 and a recovery of 2 TriMedia-32@200 MHz cycles, and occupies 34% of the device. By configuring the YUV-RGB facility on the RFUs at application load-time, YUV-to-RGB color space conversion can be computed on FPGA-augmented TriMedia-32 with a speed-up of $3.3\times$ over the standard TriMedia-32.**

Keywords— **Reconfigurable computing; color space conversion; 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 [1], [2], [3]. 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/CPU64+FPGA hybrid [4], which proved promising results with respect to several applications: Inverse Discrete Cosine Transform [5], Entropy Decoding [6], and $Y'CbCr$-to-$R'G'B'$ Converter [7].

In this paper, we investigate the potential impact on performance yielded by augmenting a TriMedia-32 processor with a reconfigurable core. TriMedia-32 is a 32-bit 5 issue slot VLIW processor with a media-oriented instruction set [8], [9]. We first describe the skeleton of an extension of the TriMedia-32 architecture, which consists of FPGA-based Reconfigurable Functional Units (RFU) and associated generic instructions. With such extension, the user is given the freedom to define and use any computing facility subject to the FPGA size and TriMedia-32 organization. Then we address the $Y'UV$-to-$R'G'B'$ color space conversion and demonstrate that significant speed-up can be achieved on FPGA-augmented TriMedia-32 over standard TriMedia-32 for such task.

Since the $Y'UV$-to-$R'G'B'$ conversion exhibits large data and instruction-level parallelisms, it can be implemented on TriMedia-32 with high efficiency. Obtaining improvements for a task having a computational pattern which TriMedia-32 has been optimised for, is indeed challenging. The main idea in achieving speed-up is to configure a pipelined YUV-RGB converter on FPGA and to unroll the software loop calling the YUV-RGB to reduce the penalty associated to firing-up and flushing the pipeline. In particular, we provide configurable-hardware support for a YUV-RGB operation which can process four pixels per call. When mapped on an ACEX EP1K100 FPGA, the computing unit performing the YUV-RGB operation has a latency of 10 and recovery of 2 TriMedia@200 MHz cycles, and occupies 34% of the device.

The experimental results indicate that by configuring the YUV-RGB unit on FPGA at application load-time, $Y'UV$-to-$R'G'B'$conversion can be computed on FPGA-augmented TriMedia-32 $3.2\times$ faster over the standard TriMedia-32. Given the fact that TriMedia-32 is a 5 issue-slot 32-bit media VLIW processor [8], such an improvement within the target media processing domain indicates that TriMedia-32 + FPGA hybrid is a promising approach.

Summarizing, the paper contributions are:

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

The paper is organized as follows. We present several issues related to $Y'UV$-to-$R'G'B'$color space conversion in Section II. Section III outlines the architectural extension of TriMedia-32. The FPGA-based implementation of a $Y'UV$-to-$R'G'B'$converter which processes four pixels per call and its associated instructions are discussed in Section IV. The experimental framework and the results are presented in Section V. Section VI concludes the paper.

II. Background

In composite NTSC and PAL video [10], the color difference signals required to convey color information are combined into a *chroma* signal by quadrature modulation using a color subcarrier, f_{SC}. The luminance signal, *luma*, and *chroma* are then summed into a composite signal for recording or transmission. Summing combines brightness and color into one signal, at the expense of introducing a certain degree of mutual interference.

The earliest digital video equipment processed video signals in composite forms. The composite analog video signal is sampled with a frequency equal to a multiple of four of the color subcarrier, $4f_{SC}$ [11]. Since *luma* and *chroma* are subject to cross-contamination, image manipulation cannot be performed in the composite domain. Thus, decoding and reencoding, and $Y'UV$-to-$R'G'B'$ color space conversion in particular are mandatory.

To make the presentation self-consistent, we will address some issues related to $Y'UV$-to-$R'G'B'$ color space conversion.

A. 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 $Y'UV$.

A.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, a nonlinear transfer function is applied to RGB intensities to give *Gamma–Corrected Red*, *Green*, and *Blue* ($R'G'B'$). The gamma-corrected red, green, blue are defined on a scale from 0 to 1.0, chosen such that shades of gray are produced when $E'_R = E'_G = E'_B$, where $E'_{\cdot}$ denotes an analog gamma–corrected signal.

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.

A.2 $Y'UV$ color space

Since the human visual system has poor color acuity, $R'G'B'$ is transformed into luminance-related quantity called *luma* (Y'), and two color difference components, U, V, as specified by the ITU-R Recommendation BT.470 [10]. Then, color detail can be lowpass filtered without the viewer noticing. It is worth mentioning that $Y'UV$ space is unique to NTSC, PAL: it is not used in component video, and is not used in HDTV. However, an $Y'UV$-to-$R'G'B'$ color space conversion is useful when the analog composite signal is sampled and discretized in order to be processed digitally.

As described in [11], the digital composite signal is represented on an 8-bit unsigned integer having a range of 64 to 211 for PAL and 60 to 200 for NTSC. Therefore, in order to preserve accuracy, it is realistic to assume that filtering and quadrature demodulation are carried out on 16-bit integers. As a consequence, each of Y', U, V will be represented on an 16-bit signed integer.

Several considerations regarding the sampling rate of the *chroma* signal with respect to *luma* are worth to be provided. Since the entire composite digital waveform is sampled with a single frequency ($4f_{SC}$), the color components U and V will have the same resolution of the *luma* Y' after band-pass filtering and quadrature demodulation. That is, an upsampling procedure which is required for component digital video decoding, e.g., [12], does not have to be carried out in front of the proper $Y'UV$-to-$R'G'B'$conversion.

A.3 $Y'UV$-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 $Y'UV$-to-$R'G'B'$ conversion [11]:

$$\begin{cases} R' = Y' & + 1.140\,V \\ G' = Y' - 0.394\,U - 0.581\,V \\ B' = Y' + 2.028\,U \end{cases} \quad (1)$$

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.

Before we will present the FPGA–based implementation of the YUV-RGB computing facility and its associated user-defined instruction, we will outline the architectural extension of the TriMedia-32 processor.

III. Architectural extension for TriMedia

TriMedia-32 is a processor which features a rich instruction set optimized for media processing. Specifically, TriMedia-32 is a 5 issue-slot 32-bit VLIW core, launching a long instruction every clock cycle [8]. 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 [13]. 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 [5], [6] for a TriMedia/CPU64 processor, TriMedia-32 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 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 [4].

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-32. 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*.

For the subsequent experiment, two instances of the TriMedia-32+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 $Y'UV$-to-$R'G'B'$computing unit, and its associated instruction are presented.

IV. YUV-RGB 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 YUV-RGB operation which reads the $Y'UV$ triplet and returns the $R'G'B'$ triplet. 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 on 16-bit signed integers, and gamma-corrected red, green, and blue are represented on 8-bit unsigned integers, at most four pixels can be processed in parallel. Indeed, the **YUV-RGB** is a 4-way SIMD operation which transforms six 16-bit signed integer vectors $(Y'_1, U_1, V_1, Y'_2, U_2, V_2)$ into three 8-bit unsigned integer vectors (R', G', B'):

$$\textbf{YUV-RGB} \quad \boldsymbol{Y'_1, U_1, V_1, Y'_2, U_2, V_2} \longrightarrow \boldsymbol{R', G', B'}$$

where $Y'_1, U_1, V_1, Y'_2, U_2, V_2, R', G', B'$ are all 32-bit registers. This translates to a number of $6 \times 2 \times 16 + 3 \times 4 \times 8 = 288$ 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 YUV-RGB operation has to be emulated by sequences of 1- and/or 2-slot operations. Therefore, we define two 2-slot YUV-RGB instructions: **YUV-RGB_R**, which performs the proper conversion and returns only the *red* information, and **YUV-RGB_GB**, which returns the *green* and *blue* information:

$$\begin{aligned} \textbf{YUV-RGB\_R} \quad & \boldsymbol{Y'_1, U_1, V_1, Y'_2, U_2, V_2} \longrightarrow \boldsymbol{R'} \\ \textbf{YUV-RGB\_GB} \quad & \longrightarrow \boldsymbol{G', B'} \end{aligned}$$

We have to emphasize that this approach is carried out only for experimental purpose. Fortunately, our choice is conservative, since it is easier to schedule a single 3-slot instruction than multiple 1- and/or 2-slot instructions.

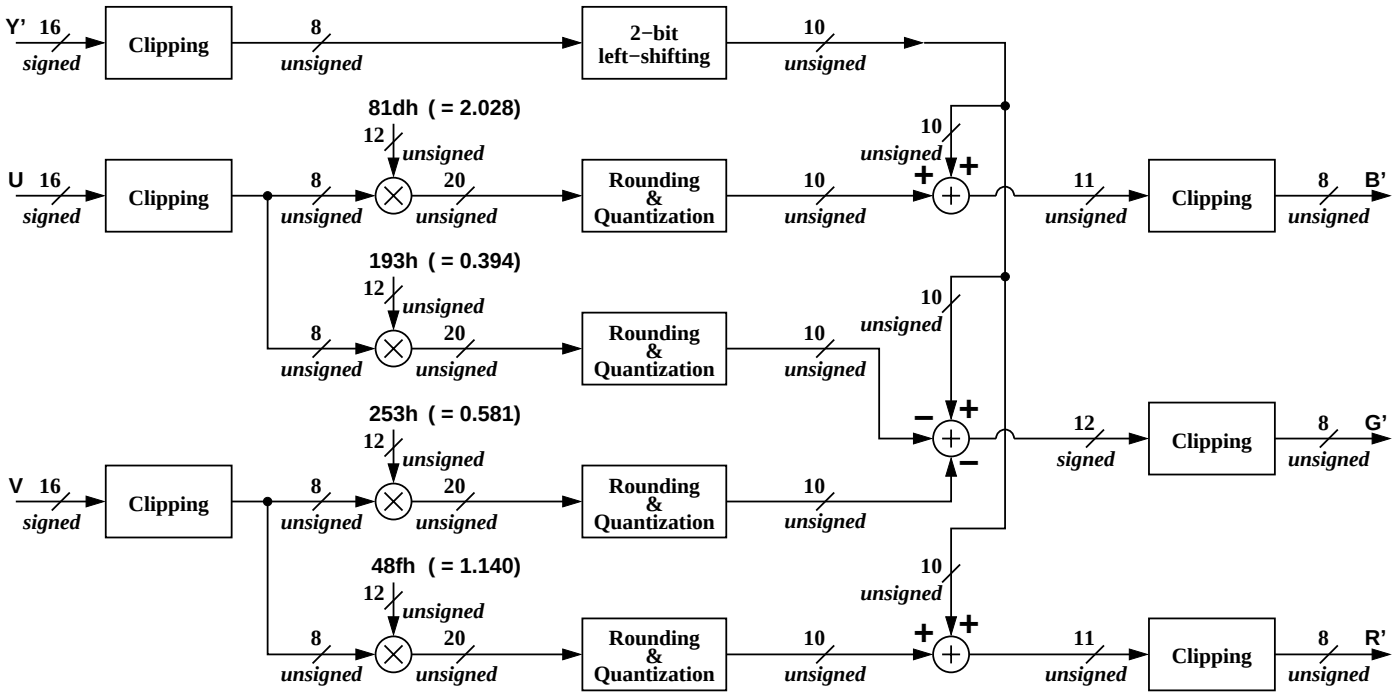

Fig. 1. The YUV-RGB implementation on FPGA

The FPGA–based YUV-RGB implementing the Equation set 1, is presented in Figure 1. 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 YUV-RGB 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 YUV-RGB having the latency of 10 and recovery of 4.

V. Experimental results

For the first extended TriMedia instance, a YUV-RGB computing unit having the latency of 10 and recovery of 2 cycles is configured on the RFU@100MHz, while a YUV-RGB 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 $Y'UV$-to-$R'G'B'$ conversion for an image, calls to YUV-RGB are issued within a software loop. The scheduled code when the RFU@100MHz is considered is presented in Figure 2. First, LOAD operations fetch the pixels in $Y'UV$ format from memory. Then, pairs of YUV-RGB_R + YUV-RGB_GB operations are launched to perform color space conversion, four pixels per call. Finally, STORE operations send the results to a display FIFO.

According to Figure 2, 16 pixels can be processed with the latency of 29 cycles. In order to keep the pipeline full, a new YUV-RGB_R instruction has to be issued every two cycles (or, every four cycles in the RFU@50MHz–based instance). Due to the limited memory bandwidth (only two Load/Store operations can be issued per cycle) this is not possible, and $Y'UV$-to-$R'G'B'$conversion can be performed with a throughput of only 16/20 = 0.8 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 YUV-RGB pipeline has to be taken into consideration. As a rule of thumb, the throughput drops to $N/(\mathit{latency}/16 + (N-1)/\mathit{ideal\ throughput})$, where N is the number of times which the loop is unrolled. For example, the ideal throughput drops to 0.72 pixels/cycle for $4\times$ loop unrolling, and to 0.55 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.

The testing database for both pure-software and FPGA-based color space converters consists of a stream of images, for which the Y', U, V components are stored in separate tables in the main memory. The data is organized as 16-bit signed integer vectors as resulted from filtering and quadrature demodulation. The $Y'UV$-to-$R'G'B'$ conversion is done in an SIMD fashion, by sequentially processing four triplets of Y', U, V values at a time.

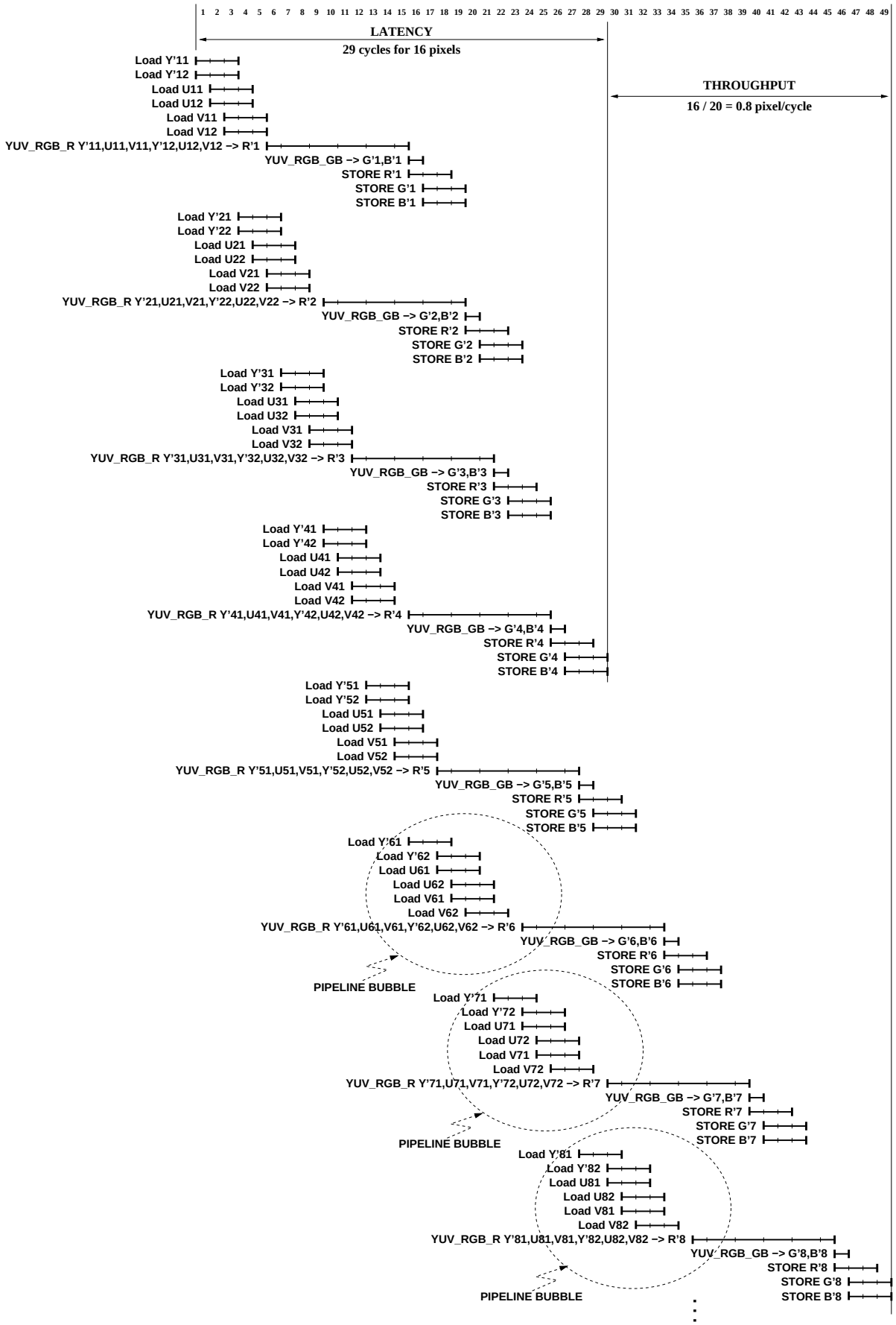

Fig. 2. The scheduling result for a YUV-RGB computing unit having the latency of 10 and recovery of 2

The linear mapping defined by Equation set 1 is carried out, and the result is sent to a display FIFO.

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 [14]. The reference color space converter is implemented as a loop, where each iteration processes 16 pixels. Since 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-32 cycle accurate simulator, we determined that an iteration which processes 16 pixels can be scheduled into 72 cycles, which translates into 0.22 pixels/cycle. It is also worth mentioning that 4.6 of 5 issue slots are filled in with operations in the pure-software implementation. This result is indeed a challenging reference for the TriMedia-32+FPGA hybrid.

Therefore, our experiment includes two approaches: *pure software* and *FPGA-based*. As mentioned, 0.22 pixels/cycle are decoded in the pure software approach, while 0.8 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 $Y'UV$-to-$R'G'B'$ performance evaluation has been carried out considering two FPGA-augmented TriMedia-32 instances: TriMedia-32 + 1 RFU@100 MHz and TriMedia-32 + 2 RFUs@50 MHz. A program has been written in C, and further compiled and scheduled with TriMedia-32 development tools. To overcome the penalty associated to firing-up and flushing the pipeline, the loop calling the YUV-RGB instruction has been manually unrolled different numbers of times. The experimental results reveal speed-ups on extended Trimedia-32 over standard TriMedia-32 as follows:

- 0.55 / 0.22 $\approx 2.5\times$ for no unrolling;
- 0.65 / 0.22 $\approx 3.0\times$ for $4\times$ unrolling;
- 0.72 / 0.22 $\approx 3.3\times$ for $4\times$ unrolling;
- 0.80 / 0.22 $\approx 3.6\times$ for infinite unrolling.

Given the fact that TriMedia-32 is a 5 issue-slot VLIW processor with 32-bit datapaths and a very rich multimedia instruction set [8], such an improvement within the target media processing domain indicates that the TriMedia-32 + FPGA hybrid is a promising approach with respect to color space conversion.

VI. Conclusions and future work

We have described an $Y'UV - to - R'G'B'$ converter on FPGA-augmented TriMedia. For such a task, the performance improvement over a simple TriMedia is approx. 250% 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] Rahul Razdan and Michael D. Smith, "A High Performance Microarchitecture with Hardware-Programmable Functional Units," in *Proceedings of the 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 *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, April 1996, pp. 126–135.

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

[4] 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 *Proceedings of the IEEE International Conference on Computer Design*, Austin, Texas, September 2001, pp. 425–430.

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

[6] Mihai Sima, Sorin D. Cotofana, Jos T.J. van Eijndhoven, Stamatis Vassiliadis, and Kees A. Vissers, "MPEG-compliant Entropy Decoding on FPGA-augmented TriMedia/CPU64," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, April 2002.

[7] Mihai Sima, Stamatis Vassiliadis, Sorin Cotofana, and Jos T.J. van Eijndhoven, "Color Space Conversion for MPEG decoding on FPGA-augmented TriMedia," in *Design, Automation, and Test in Europe Conference*, Messe Munich, Germany, March 2003, submitted.

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

[9] TriMedia Product Group, *TM-1000 Data Book*, Philips Electronics North America Corporation, Sunnyvale, California, 1998.

[10] International Telecommunication Unit, "Conventional Television Systems," ITU-R Recommendation BT.470-6, November 1998.

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

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

[13] 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.

[14] Mihai Sima, "Color Space Conversion on TriMedia/CPU64," Private Communication, August 2002.