FPGA- and Java-based Rapid Prototyping of a Real-time H.264/AVC Decoder

Alexander Petrovsky, Marek Parfieniuk
Department of Real-Time Systems, Bialystok Technical University
Bialystok, Poland
Email: marekpk@wi.pb.edu.pl, palex@bsuir.by

Andrew Stankevich, Alexey Petrovsky
NTLab: New Technologies Laboratory
Minsk, Belarus
Email: www.ntlab-soc.com, petrovsky@bsuir.by

***Abstract*—The paper reports on an attempt to implement a real-time hardware H.264 video decoder. The initial results of the project are presented, especially a customized RISC core and some digital modules, both of which have been implemented in Xilinx FPGA. The former has to serve as a host processor that supervises the latter, which speed up the essential decoding subtasks. The system is designed and tested based on a software decoder and diagnostic tool, which are implemented in Java using the object-oriented paradigm. Based on our experiences, we recommend the combination of FPGA and the Java platform as a good basis for rapid prototyping of advanced DSP algorithms.**

***Index Terms*—H.264 decoder, Plasma RISC, Xilinx FPGA, Java.**

I. Introduction

H.264/AVC (Advanced Video Coding: MPEG-4 Part 10) is the state-of-art standard video codec, which is recommended by both ITU-T and ISO/IEC [4], [6], [8], [10]. Designed to be flexible and network-friendly, it is expected to dominate the market of multimedia devices and services in the near future. The most notable areas of H.264 use are the Digital Video Broadcasting (DVB), 3GPP mobile communication, and Blu-ray Disc.

Adaptability to various applications and better bandwidth usage, which are advantages of H.264 over its predecessors, especially over MPEG-2 Video (H.262), have been achieved at the price of higher complexity and greater memory requirements. Although the main principles of hybrid video coding are still used, the subalgorithms: transform, prediction, motion compensation, as well as entropy coding have been revised and given new options, which improve effectiveness but decrease efficiency. Significant modifications of both bitstream format and decoding process limit reusing of existing software and hardware, so that new infrastructure needs to be created.

Since 2003, when the first version of the standard had been published, many H.264-related products have been issued, like encoding engines, decoder chips, software players, and bitstream analyzers. However, there are still reasons for developing new solutions, because the existing ones often do not follow standard's nuances or recent extensions, or have deficiencies related to speed or power consumption.

As working in this field is interesting from both engineering and commercial points of view, the authors have undertaken the task of developing a real-time hardware H.264 decoder. Its novel architecture has to be not only computationally efficient but also flexible from the design point of view. Namely, modularization and reconfigurability should guarantee that: i) future extensions of the standard can easily be incorporated into the system without entirely redesigning it, ii) speed can be traded off for resource consumption, iii) the digital circuit can be customized and optimized in order to satisfy application requirements without wasting resources. An additional decision was to widely use free-of-charge and open-source development tools in order to keep investments small and to be able to customize the toolset in accordance with needs.

Because so far the team mainly specialized in speech processing, see e.g. [7] or [9], in order to gain more konwledge, they have split work between two directions. One is to develop from scratch a reliable object-oriented model of the decoder and implement it in software. The second one is to use the resulting code, after testing, to design hardware modules, which can be functionally verified based on data generated using the software.

In order to achieve high productivity, the Java platform has been used in object-oriented development. In addition to preventing errors that are common in C programming, it was expected to facilitate building a consistent development toolset adjusted to our needs.

The paper presents the initial results of the project, which are related to both software and hardware, and justifies main design decision the team made. After characterizing H.264 briefly, we describe both Java code organization and the corresponding hardware architecture of the decoder. Then the Plasma-NTLab processor is presented, which has been developed in order to supervise the prototype platform, especially hardware decoding. Finally, FPGA designs of some modules that speed up the decoding algorithm are shown. In particular, the transform unit is compared with a known solution, in order to show that our methodology brings benefits.

II. H.264 Codec

The general scheme of the H.264 codec is shown in Fig. 1. Like the older standards, it is a hybrid algorithm that uses both transform coding and motion-compensated predictive coding to remove spatial as well as temporal redundancy of video

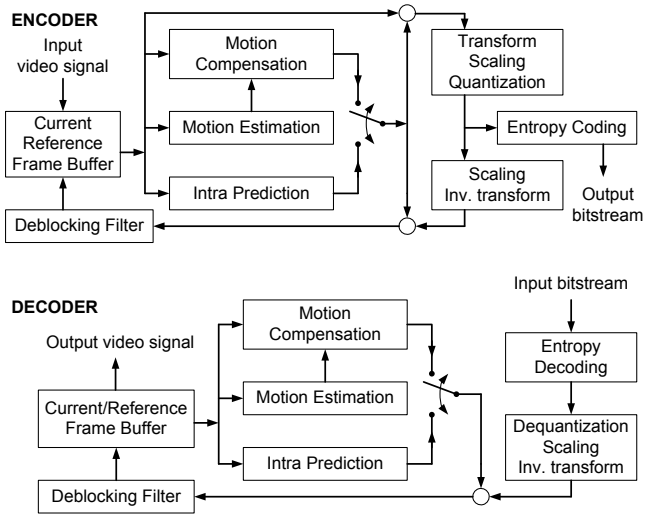

Fig. 1. The general scheme of the H.264 codec.

signals. Better flexibility and compression efficiency have been achieved by only improving subalgorithms.

Fine-grained macroblock partitioning into smaller units and quarter-pixel accuracy allow motion estimation/compensation to be more effective. Estimation accuracy is further improved by employing an in-loop deblocking filter that removes the blocking artifact before using a frame for prediction. Reduction of temporal redundancy is improved by allowing multiple (up to 16) reference frames to be used and by making bidirectional prediction possible, in which future frames can be referenced in addition to past ones. Removing spatial dependencies among pixels by a decorrelating transform can be supported with multi-mode intra prediction of a block using adjacent fragments of the same frame. Moreover, transform size can be switched between 4×4 and 8×8 in order to best fit macroblock contents. Finally, more effective methods of entropy coding have been employed: Context-based Adaptive Binary Arithmetic Coding (CABAC) and Context-Adaptive Variable Length Coding (CAVLC). Specific needs of studio and wireless applications have been satisfied by incorporating Fidelity Range Extensions (FREXt) and Scalable Video Coding (SVC), respectively, into the standard.

Thus, the encoder, which is equipped with a lot of switches, allows the output bitstream to be customized in order to best fit a particular usage of the codec. This has been rationalized by defining several H.264 profiles, which correspond to various trade-offs among quality, bitrate, and computational requirements.

Owing to these advanced techniques and high flexibility, H.264 offers even two times better compression efficiency than MPEG-2 Video and is much better suited to contemporary applications. However, decoding can require even four times more operations, even though in the new standard, the Discrete Cosine Transform (DCT) has been replaced with efficient multiplierless approximations.

Because of its complexity and little connection to previous standards, implementing H.264 is not trivial, especially if a small and energy efficient device is expected to operate in real time. Thus, there is a great interest in novel solutions in this field.

III. Software prototype

Our review of the existing support for implementing H.264 has shown that a lot of information is accessible but they are distributed among papers, reports, web pages, etc. Moreover, they usually are not conveyed readily, so that there are no reliable implementation patterns and ready-to-use high-quality source code. Especially, the standard documents and the reference software [1] are very unclear and very difficult to understand even for experienced developers. Thus we decided to design from-the-scratch a software H.264 decoder that does not necessarily work with real-time performance but forms a clear, well-documented foundation for developing and than testing hardware modules.

Such objectives has motivated us to employ the Java platform instead of the C or C++ languages, which are commonly used for implementing DSP algorithms. The former is undoubtedly slower but greatly increases productivity and code quality. Its strict type-control prevents many errors that can easily be made when using C, some other bugs become easy to detect, and finally, the language helps programmers with object-oriented design. Moreover, there is no need for combining different open-source tools/libraries or for relaying on platform-dependent commercial products. The standard Java packages provide all that is necessary for creating advanced GUI- and network-based applications, which work on both Windows and Linux. A rich set of OS-independent development tools can be downloaded as a single bundle, including the sophisticated RAD development environment, NetBeans, and JavaDoc, a simple means for generating well-organized documentation from code comments.

A class diagram of our object-oriented model of the H.264 decoder is shown in Fig. 2. It comprises about 50 classes, which represent data and subprocesses related to decoding. They have been designed in such a way that it is easy to identify objects and methods with hardware modules, registers, or state changes.

For most of classes, it is possible to strictly determine the number of instances. Knowing the latter allows objects to be preallocated as static fields and to exist continuously during program execution. This significantly reduces computational load related to memory management and garbage collection. It seems that using this technique is crucial for developing a Java-based H.264 decoder that works in real time.

Another conclusion, which does not directly result from the standard document, is that most of operations can be performed without explicit integer multiplications. The latter can widely be replaced with binary shifts, possibly supplemented by additions. As to data types, 16 bits (including sign) seem sufficient to store variables related to decoding, but in some cases, auxiliary results need 32 bits.

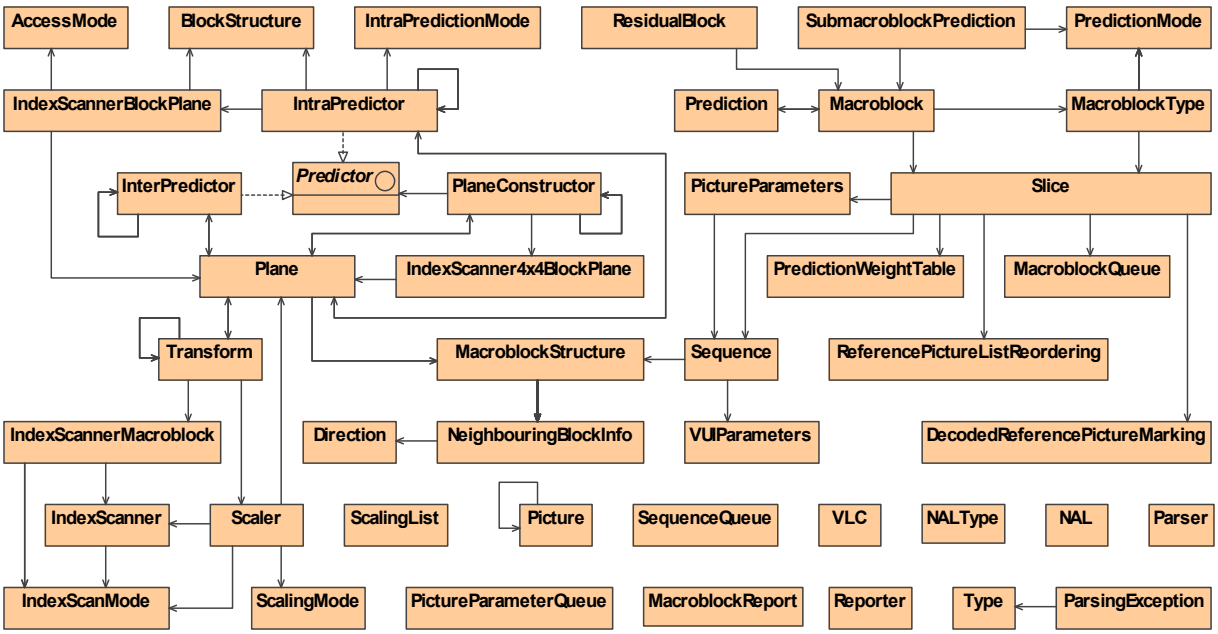

Fig. 2. UML class diagram of the Java-based H.264 decoder.

Internal variables of decoding pipelines do not occupy much memory. Quantization tables and sample buffers for transform and prediction purposes take up the most space, yet it seems possible to incorporate them into a chip. The main problem is in storing reference frames for inter-prediction, which requires large out-of-chip memory. Some of known decoders require encoders to limit the number of reference frames depending on video resolution and accessible storage space, and we will probably employ this approach in our chip.

Even though the software decoder is currently developed only in order to help engineers with implementing the H.264 decoder in hardware, it can become a stand-alone project. Our results suggests that for low resolution videos, real-time performance can be achieved on current PCs even without rewriting the code in the C language.

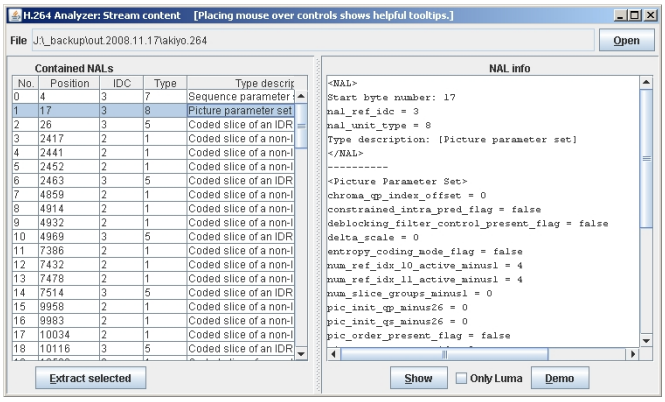

Fig. 3. Stream analysis window of the diagnostic tool.

IV. Diagnostic tools

The software decoder is a foundation of our platform-independent diagnostic tool. Being written in Java, the program works in any operating system equipped in the JVM, especially on Linux. It allows for interactively testing the decoder against errors and for preparing data for verification of digital modules. This is possible via two main functionalities, which are GUI-controlled using the windows of Figs. 3 and 4. Firstly, H.264 bitstreams can be analyzed and restructured, in order to identify and extract input data that cause the decoder to fail. Secondly, correctness of decoding of a single frame can be examined both visually and by following the dataflow step by step.

The latter is based on a quite advanced reporting mechanism, which collects data in a synthetic form, so that they can be both displayed on screen and exported to verification tools. The mechanism has been designed in a way that allows it to be easily incorporated into the decoder, without refactoring and messing up the essential code.

Similar commercial tools are accessible, e.g. H264Visa [2], but they are quite expensive, work only on Windows, and it is impossible to customize them as desired. Especially, they offer only limited access to the internals of the decoding pipeline, and the data inspected via GUI cannot be efficiently translated to a form suitable for verification.

On the other hand, our reporting and verification tools can be extended as necessary. Especially, filters are to be developed that allow interesting information to be quickly extracted. Another functionality under development is automatic detection and extraction of erroneously decoded frames in a long stream. Nevertheless, in most cases, interactive testing the program supports is sufficient.

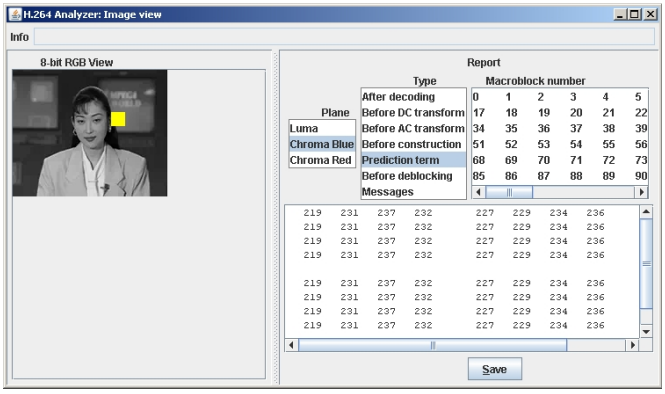

Fig. 4. Frame analysis window of the diagnostic tool.

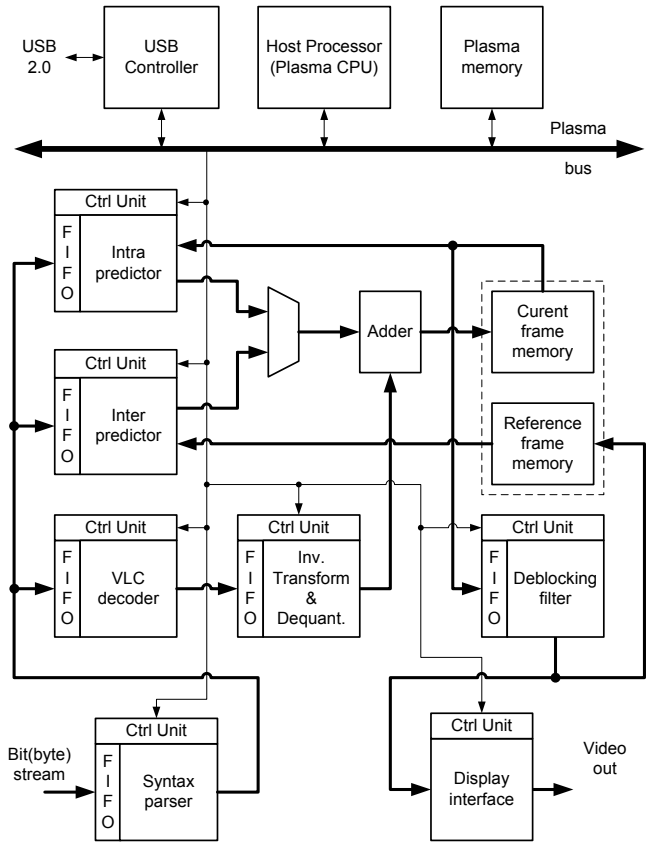

Fig. 5. Decoder architecture.

V. Decoder architecture

The software allowed us to design the hardware architecture of Fig. 5. One its part is a host CPU with a USB controller and general-purpose memory. The second one is a H.264 decoding pipeline with dedicated memory for video frames. Except both memories, all circuits have been implemented on a single FPGA chip. After reaching a mature state, the design is expected to be translated to the ASIC technology.

The Virtex-4 ML 401 Evaluation Platform was used in our experiments. It is powered by the Xilinx XC4VLX25 FPGA device and supported by industry-standard peripherals, interfaces, and connectors like DB15 VGA and USB. The main clock source is a 100 MHz oscillator. The memory resources comprise 64 MB DDR SDRAM, 1 MB ZBT SRAM, 32 MB Compact Flash, 8 MB Flash, 4 kb IIC EEPROM, and 32 Mb Platform Flash. They are connected to the FPGA via 32-bit data buses. The main DDR SDRAM runs up to 266 MHz data rate. Such a configuration is expected to be able to decode videos of resolutions from 320×240 to 720×576 at the rate of 30 frames per second. Both Baseline and Main H.264 profiles have to be supported.

The decoder has been made programatically reconfigurable. Most of its modules have microprogrammable control units. The host processor is responsible for loading up-to-date microprograms before starting a decoding job.

In our architecture, decoding modules are cascaded so that they form a pipeline, in addition to being connected to the system bus. The latter is used only to initialize and roughly control block states. Data related to decoding are passed from module to module via dedicated FIFO-buffered connections between them, which are also responsible for interblock synchronization. This is expected to greatly improve concurrency. Firstly, it helps with avoiding bottlenecks caused by sharing one bus by many devices. Secondly, a connection can be made wide enough to pass an entire 4×4 or 8×8 block of samples at once, which allows them to be processed in parallel. This is especially the case of transform and prediction.

The current prototype has only one pipeline which is switched between luma and chroma processing. A future approach we consider is to have three separate pipelines, which allows decoding to be totally parallelized, but requires a lot of FPGA resources.

VI. Plasma-NTLab Processor

The host processor we use is a customized version of the Plasma CPU [3]. The latter is a simple RISC processor that is accessible as a VHDL project (about 4000 lines of source code + documentation), and thus can be modified and used as a part of advanced SOPC solutions. Moreover, it is free for commercial use, even though its features are sufficient to a wide range of applications, especially those DSP-related.

Fig. 6 shows the block diagram of the processor. The 32-bit address bus allows the core to handle large memory. Excessive accesses to the latter can be avoided by wise use of 32 32-bit general-purpose registers. Two additional special-purpose registers for storing the results of both integer multiplication and division allow the ALU to fully support fixed-point arithmetic. At the VHDL level, it is possible to select between big- and little-endian byte-ordering. A number of peripherals are also accessible: UART, Interrupt Controller, Interrupt Timer, SRAM Controller, Flash Controller, DDR SDRAM Controller, and Ethernet MAC.

The instruction set is compatible with the MIPS I architecture, e.g. with the MIPS R2000 CPU. From another point of view, it is equivalent to the user-mode subset of the MIPS32

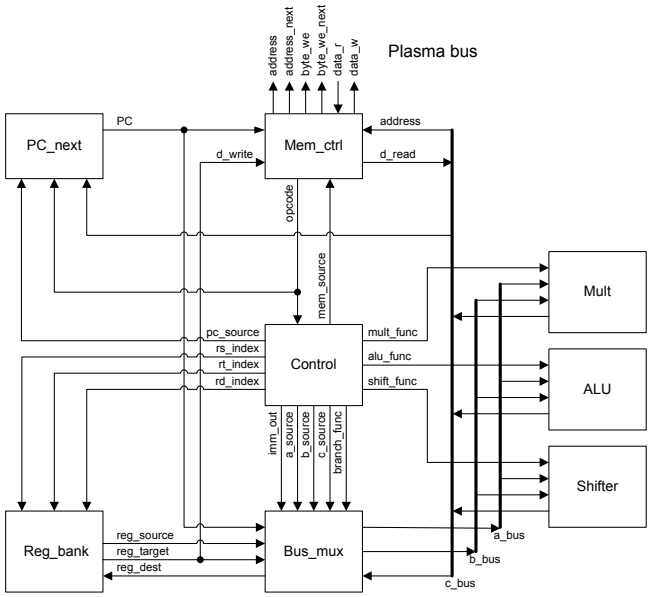

Fig. 6. The block diagram of the Plasma microprocessor.

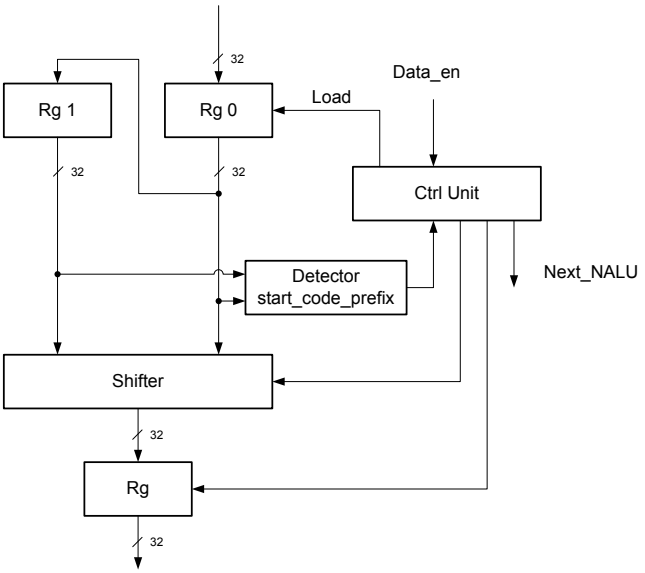

Fig. 7. NALU detector.

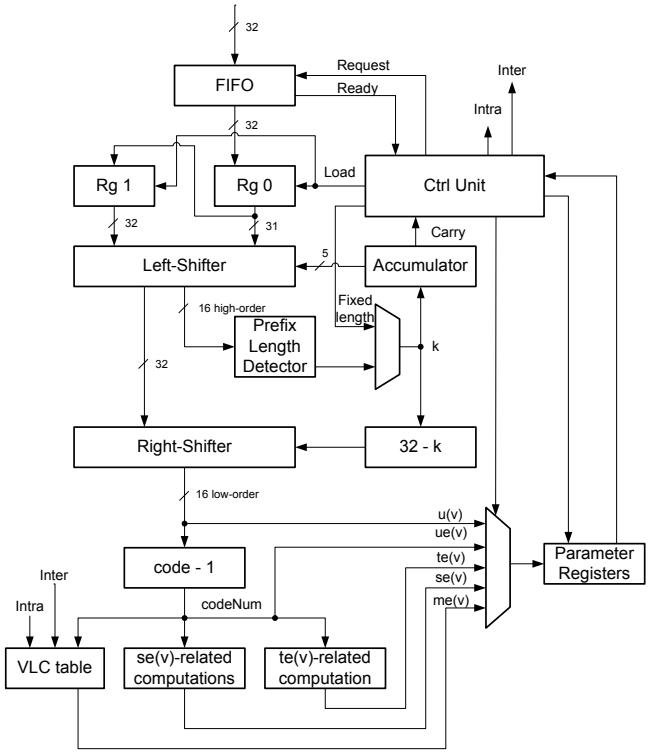

Fig. 8. Parser unit.

instructions, except nonaligned data access and exceptions. Two-stage command pipeline can be extended to three stages.

The Plasma core is not very resource consuming. If Xilinx Spartan-3 XC3S1000 is the target platform, it takes up 1604 slices (20% of chip area) and can operate at the maximum clock of 32–33 MHz. For Xilinx Virtex-4 XC4VLX25, is takes up 1588 slices (14% chip area) whereas the maximum clock is 64–67 MHz.

The original Plasma has been customized in order to match the memory organization and I/O interfaces the development board provides. Especially, a USB controller has been added, which allows communications between the system and a PC workstation. The testing environment runs on the latter, which allows output of the prototype decoder to be verified after sending a video stream to it. This required the system to be extended in such a way that the CPU can control and monitor stages of the decoding pipeline. The resulting microprocessor architecture has been called the Plasma-NTLab CPU.

VII. FPGA DESIGN OF MODULES

Based on some parts of the software decoder, for which code had been frozen after thorough tests, digital circuits have been developed in FPGA. These are the NALU (Network Abstraction Layer Unit) detector, bitstream parser, including the VLC (Variable Length Coding) decoder, and residual transform unit. Their schemes are shown in Figs. 7, 8, and 9, respectively.

The first module is responsible for determining NALU boundaries in a bitstream, and to extract the contents units carry. The contents form a higher-level bitstream, which is analyzed by the parameter parser in order to decode syntax elements. The related operations require only iterating binary shifts and comparisons, which seems simple, but takes many cycles of a general-purpose CPU.

In order to relieve the latter, only a fraction of the FPGA chip area needs to be sacrificed. Namely, the NALU detector of Fig. 7 use only 70 of 21504 Slice Flip-Flops, and 174 of 21504 LUTs that XC4VLX25 contains.

A much more complex part of the decoder is the transform unit of Fig. 9, which computes an approximation of 2-dimensional DCT. It needs quite large memory buffers for

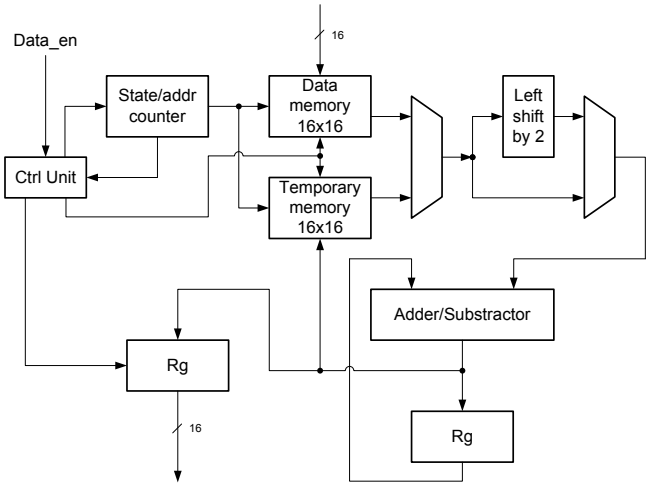

Fig. 9. Transform unit.

storing two 16×16 arrays of integers: one of input data and one of auxiliary/output values.

The 4×4 version of our transform unit can be compared with that of [5], where detailed data of a H.264 implementation in FPGA are given. Two variants of the circuit are considered therein, which we have also realized in our architecture. In the first, calculations on vectors are performed element by element, in order to conserve chip area. In the second, an entire inner product of 4-element vectors is computed at once, which requires replicated arithmetic blocks to work in parallel. The synthesis results of Tables I and II show that our preliminary designs are comparable to those by others, or even slightly better in terms of chip area utilization.

This a proof that our software prototype well accomplishes its task. It allows hardware engineers to quickly understand what is expected and to construct digital circuits of good quality, i.e. high performance is achieved at low resource utilization.

The inter- and intra-prediction units are under development. Finishing them will allow for assembling a first version of the decoder

TABLE I
EVALUATION OF SINGLE-STAGE RESIDUAL TRANSFORM

Parameter	Authors' realization	[5]
FPGA chip	XC4VLX25-12	XC2VP7
Wordlength	16	16
Slice Flip-Flop	37	65
LUT	111	*
Slice	93	103
Max. clock [MHz]	200	150

VIII. CONCLUSION

Using Java routines as a basis for FPGA development has turned out to be a good methodology for implementing such an advanced DSP algorithm as the H.264 decoder. Clear and well-documented code has allowed hardware specialists both to design a flexible modularized architecture of the real-time system and to rapidly prototype digital circuits that speed up decoding subtasks. The advantage of high productivity is accompanied by good quality of FPGA designs, which can compete with the solutions by others, in terms of throughput and resource utilization. Additionally, without the necessity of looking for other tools, the Java platform has allowed us to develop in parallel a multi-platform GUI-based diagnostic application for test and verification purposes. On the other hand, the simple Plasma RISC core, which is publicly available as a VHDL source code, has served as a foundation for developing a customized host processor for controlling the hardware decoding pipeline. We estimate that our project has reached its half-way point. Further results are expected soon, which will be the subjects of future papers.

TABLE II
EVALUATION OF PARALLEL RESIDUAL TRANSFORM

Parameter	Authors' realization	[5]
FPGA chip	XC4VLX25-12	XC2VP7
Wordlength	16	16
Slice Flip-Flop	-	257
LUT	1008	*
Slice	512	644
Max. critical delay [ns]	9.7	9.3

ACKNOWLEDGMENT

This work was supported by Bialystok Technical University under the grant W/WI/8/08.

REFERENCES

[1] "The H.264/AVC reference software (JM)." [Online]. Available: http://iphome.hhi.de/suehring/tml/
[2] "H264Visa." [Online]. Available: http://www.h264visa.com
[3] "Plasma CPU." [Online]. Available: http://www.opencores.org/projects/mips
[4] ITU-T and ISO/IEC, *ITU-T Rec. H.264 Advanced video coding for generic audiovisual services / ISO/IEC 14496-10 MPEG-4 AVC*. Geneva: ITU, 2003. [Online]. Available: http://www.itu.int/rec/T-REC-H.264
[5] R. Kordasiewicz and S. Shirani, "On hardware implementations of DCT and quantization blocks for H.264/AVC," *J. VLSI Signal Process.*, vol. 47, pp. 189–199, 2007.
[6] S.-k. Kwon, A. Tamhankar, and K. R. Rao, "Overview of H.264/MPEG-4 part 10," *J. Vis. Commun. Image R.*, vol. 17, pp. 186–216, 2006.
[7] M. Livshitz, M. Parfieniuk, and A. Petrovsky, "Wideband CELP coder with multiband excitation and multilevel vector quantization based on reconfigurable codebook," *Digital Signal Process. (OOO "KBWP", Moscow, Russia)*, no. 2, pp. 20–35, 2005, in Russian.
[8] D. Marpe, T. Wiegand, and G. J. Sullivan, "The H.264/MPEG4 Advanced Video Coding standard and its applications," *IEEE Commun. Mag.*, pp. 134–143, Aug. 2006.
[9] A. A. Petrovsky, A. Borowicz, M. Parfieniuk, and A. Petrovsky, "Warped Discrete Fourier Transform in perceptual speech and audio processing," in *Proc. X Symp. "New Trends in Audio and Video"*, Wroclaw, Poland, 16–18 Sep. 2004, pp. 143–152.
[10] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*. Wiley, 2003.