High-Level Algorithmic Complexity Analysis for the Implementation of a Motion-JPEG2000 Encoder

Massimo Ravasi[1], Marco Mattavelli[1], Paul Schumacher[2], Robert Turney[2]

[1] Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland
{Massimo.Ravasi, Marco.Mattavelli}@epfl.ch
[2] Xilinx Research Labs
Longmont, CO, USA
{Paul.Schumacher, Robert.Turney}@xilinx.com

Abstract. The increasing complexity of processing algorithms has lead to the need of more and more intensive specification and validation by means of software implementations. As the complexity grows, the intuitive understanding of the specific processing needs becomes harder. Hence, the architectural implementation choices or the choices between different possible software/hardware partitioning become extremely difficult tasks. Automatic tools for complexity analysis at high abstraction level are nowadays a fundamental need. This paper describes a new automatic tool for high-level algorithmic complexity analysis, the Software Instrumentation Tool (SIT), and presents the results concerning the complexity analysis and design space exploration for the implementation of a JPEG2000 encoder using a hardware/software co-design methodology on a Xilinx Virtex-II™ platform FPGA. The analysis and design process for the implementation of a video surveillance application example is described.

1 Introduction

The evolution of digital silicon technology enables the implementation of signal processing algorithms that have reached extremely high levels of complexity. This fact, among others, has two relevant consequences for the system designer. The first is that processing algorithms cannot be specified in ways other than developing a reference software description. The second important consequence is that the understanding of the algorithms and the evaluation of their complexity have to be derived from such software description. As consequence of the greatly increased complexity, the generic intuitive understanding of the underlying processing becomes a less and less reliable design approach. Considering that, in many cases, the complexity of the processing is also heavily input-data dependent, the system designer faces a very difficult task when beginning the design of a system architecture aiming at efficiently implementing the processing at hand.

This difficulty is evident when considering for instance the case of hardware/software co-design for System-on-Chip integration. A typical design flow for this implementation case is shown in Fig 1. All the relevant information must be

extracted from the software description; indeed, the analysis of the complexity of single functions does not give any information without the knowledge of the interconnection, occurrence and actual use of all functions composing the algorithm. Some other traditional styles of design, such as complexity analysis based on "*pencil and paper*" or *worst-case* applied to some portions of the algorithm, not only become more and more impractical for the required effort, but can also results in very inaccurate results for not taking into account the correct dependency of the complexity on the input data.

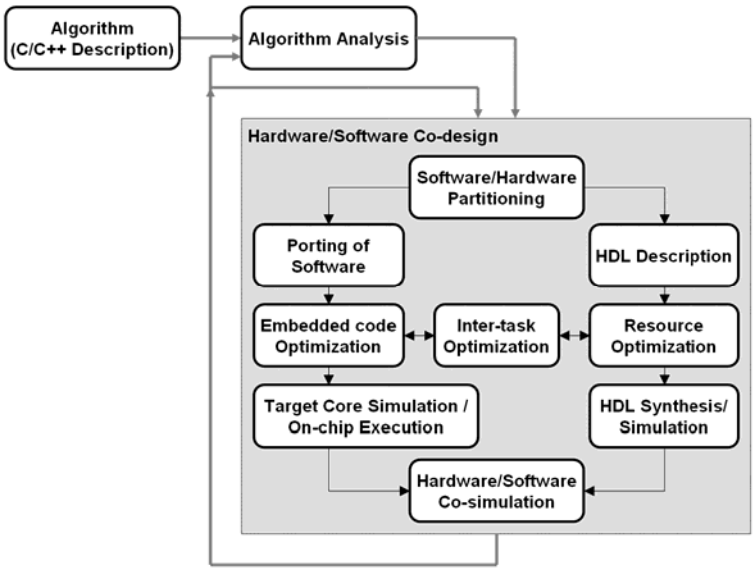

Fig 1. Typical simplified design flow of a hardware/software embedded system

It can be noticed that for hardware/software co-design [1], [2], a large variety of tools is available at all levels. Conversely no suitable automatic tools are available to assist the fundamental task partitioning stage or to gather detailed and reliable information on the computational complexity of the algorithm for optimizing the implementation, starting from the generic software description.

All these considerations, although relevant for most of signal processing implementation problems, become fundamental for video, still image, audio and multimedia coding, where the latest generation of compression standards (i.e. MPEG-4 [3] and JPEG-2000 [4]) reaches a very high level of complexity that is also extremely sensitive to the encoder optimization choices and strongly data-dependent.

This paper presents a new approach to complex system design by means of a tool-assisted high-level algorithmic complexity analysis based only on the pure software description of an algorithm. This analysis is carried out by means of the *Software Instrumentation Tool* (SIT) [5], an automatic tool allowing the extraction of relevant information about the complexity of the algorithm under study. The dependencies on

the underlying architecture and the compilation process used to verify and analyze the algorithm are not taken into account, because only the software description of the algorithm is relevant for the analysis and not how it is compiled and run on an architecture chosen for verification purposes only. Furthermore, the analysis is performed in real working conditions on real input data, to take into account the input-data dependency of the performance and the complexity of signal processing algorithms.

Section 2 gives a brief overview of current state of the art methods in complexity analysis and of their drawbacks for complex systems design. Section 3 presents the new design approach by means of the *Software Instrumentation Tool*. Section 4 shows how the new design approach was applied to the implementation of a motion-JPEG2000 encoder for a video surveillance application, using a hardware/software co-design methodology applied on a Xilinx Virtex-II™ [6] platform FPGA. Section 5 presents the main details about the implementation example.

2 Complexity Analysis and System Design

An in-depth understanding of the algorithm complexity is a fundamental issue in any system design process. Questions such as *how many operations*? *of which type*? *on which type of data*? *how many memory accesses*? *on which memory architecture*? *which processing functions are necessary to correctly perform the algorithm*? are fundamental for the design of efficient processing architectures that aim to match the processing requirements. Having this information in advance and as a reliable support to the hardware/software task partitioning and task optimization can reduce or even eliminate the need of the costly and time-consuming redesign iterations shown in Fig 1. The same type of analysis is also useful for other system optimization tasks such as data-transfer and power consumption minimization that require several methodological steps starting from a generic algorithm specification [7].

2.1 State-of-the-Art Approaches to Complexity Analysis

Depending on the specific goals of the desired complexity analysis to be performed, very different approaches and tools can be chosen [8]. All these approaches are perfectly suited for their specific applications, but they present serious drawbacks when applied to the design of complex systems. The most common approaches can be classified into the following categories:

– *Profilers*, modifying the program to make it produce run-time data [8]. Profilers can basically provide two types of results: number of calls of a given section of a program and execution time of that section. The results provided by profilers strictly depend on the architecture on which the code is executed and on the compilation optimizations. These results cannot therefore be easily used for complexity analysis concerning the implementation of the same algorithm onto another architecture and they could even yield misleading complexity evaluations.

– *Static methods*. For these methods state of the art solutions rely on annotation at high-level programming language so as to determine lower and upper bounds of resource consumption [9]. The main drawback of these techniques is that the real processing complexity of many multimedia algorithms heavily depends on the input data, while static analysis depends only on the algorithm. These approaches are better suited for real-time control applications for which strict *worst-case* analysis is required. Moreover restricted programming styles such as absence of dynamic data structures, recursion and bounded loops are required.
– *Hardware Description Languages* and *Hardware/Software Co-Design tools*, allowing describing (at different abstraction levels), synthesizing and simulating hardware or heterogeneous hardware/software systems [1], [2]. Through synthesis and simulation, these approaches allow gathering very reliable results about the implementation complexity and performance of the described algorithm. However, the analysis can only be performed at the end of the design cycle, *after* all architectural choices have already been taken. If it is realized that the a priori architectural choices are not appropriate for the desired performance constraints, a costly redesign of the system is necessary.

3 The *Software Instrumentation Tool* (*SIT*)

It is assumed that a software implementation of an algorithm is available and that it can be run in realistic input data conditions. The goal is to measure the complexity of the algorithm, whose performance can be data-dominated. In other words, the interest is not only about the measure of complexity of the algorithm itself, but also about its dependencies on specific input-data. Moreover, the software implementation is a high-level description of the algorithm whose complexity has to be measured independently of the underlying architecture on which the software is run for verification purposes. This approach is fully in line with methodological approaches, aiming at optimizing data-transfer and memory bandwidths at a high-level description of the algorithm [7].

The implementation of the Software Instrumentation Tool (SIT) [5] is based on the concept of the instrumentation of all the operations that take place during the execution of the software program. Instrumenting code by overloading C++ operators has been already proposed in literature, but it has always been considered an approach presenting severe practical and functional limitations [8]. Major drawbacks were considered the applicability only to C++ program, the impossibility to instrument pointers and other data types such as structures and unions, resulting into not accurate analysis of data-transfer oriented operations and to an extensive manual rewriting of the original code. All known functional and practical limitations of the operator overloading approach have been overcome with SIT. The current version of SIT is able to instrument a C program by translating it into a corresponding C++ program by means of an automatic tool: both programs have the same behavior but, by substituting C simple types with C++ classes and by substituting all C operators with C++ overloaded operators, standard C operations can be intercepted during the execution and counted. The great advantage of this approach is that no manual code rewriting is

necessary. Moreover, SIT allows associating an appropriate and customizable memory model to the algorithm, in order to complete the complexity analysis with data-transfer analysis.

The results gathered with SIT are presented on a per-context basis, which can be chosen to correspond to the function call tree or to be extended to the single compound statements for a more detailed analysis. The results of the computational complexity analysis are in terms of executed operations within a context node and are collected on the two axis *operations* and *data-types*. The *operations* axis is an extension of the C operator set (+, +=, etc.) as well as the *data-types* axis is an extension of the C data type set (`int`, `float`, `struct`, etc.). The results of the data-transfer analysis depend on the simulated memory model, which may include the simulation of cache hierarchies.

In order to validate the SIT methodology, a real-world design example was used. The chosen design was a JPEG2000 encoding system for video surveillance applications.

4 Hardware/Software Co-Design of a Motion-JPEG2000 Encoder

JPEG2000 standard [4] includes a specification for the encoding and storage of motion sequences [10]. Whereas well-known video standards such as MPEG-2 and MPEG-4 [3] use inter-frame dependencies and motion compensation, motion-JPEG2000 involves encoding each frame independently.

Whereas the standard specifies the bitstream to ensure interoperability between encoder and decoder systems, it leaves the actual implementation open. This section presents the implementation of a JPEG2000 encoder system capable of handling video data rates, created using a hardware/software co-design methodology on a platform field programmable gate array (FPGA). A cohesive and programmable hardware/software co-design is created.

The targeted video surveillance application involves a low-grade video coding system coding. The frame size is 640 × 480 × 24 bits and the rate is 15 frames/sec. High compression of the video data is expected as quality is not of high importance.

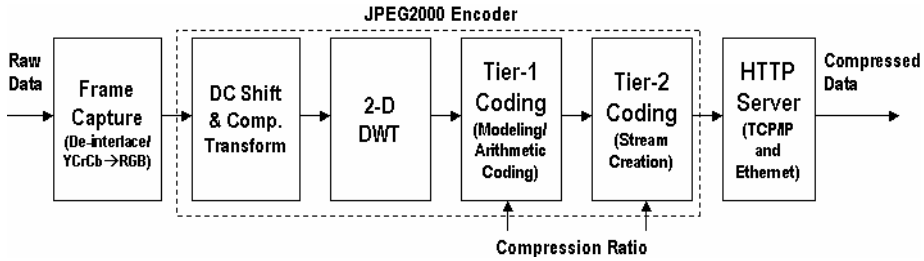

Fig 2. Block diagram of motion-JPEG2000 system

Fig 2 shows the block diagram of the designed JPEG2000 encoder system. After the frames are captured and the data is DC shifted, the user can decide to perform a component transformation on the three components for each pixel as specified by the standard [4]. A 2-D DWT is then performed on each tile within each frame. It was decided to have each frame be a single tile in order to eliminate the tiling effects. That is, the DWT is performed on the entire frame, and the number of decomposition levels performed is decided by the user (for the designed system, up to five levels is supported). The 5/3 DWT kernel was chosen for the implementation. The sub-bands of the DWT results are then divided into code-blocks and the *Tier-1* coder operates on each code-block independently. The code-block size for this system was chosen to be 64x64. *Tier-2* coding then involves adding the appropriate headers, compiling the compressed data into packets, and delivering the data as a complete codestream.

4.1 Complexity Analysis

For initial system definition, the JPEG2000 encoder system was defined in ANSI C. This gives an excellent starting point for the eventual co-design, provides a reference and test bed for verification, and allows numerous modes and parameters available from the standard to be tested. For an initial complexity assessment, this software was analyzed with SIT. This software JPEG2000 encoder implements only lossless coding, while the targeted video surveillance application is based on lossy coding, aiming to an average compression ratio of 20:1. For these reasons, the results for the *Tier-1* and *Tier-2* blocks were respectively scaled by 1/3 and 1/6.5 [11], while the performance of *DC Shift*, *Component Transformation* and *2-D DWT* is unaffected by the coding type.

The results of the complexity analysis with SIT, concerning the encoding of one frame, are summarized in Table 1, which shows how the computational complexity and data-transfers are distributed over the main processing blocks of Fig 2. With *SITview*, the graphical visualization tool of SIT, the results for the computational complexity (operation counts) were mapped onto the instruction set of the targeted Xilinx MicroBlaze™ 32-bit RISC soft processor by means of a set of weights representing the latencies of the operations [12]. Therefore the results in the "*Operations*" columns of Table 1 are an estimate of the clock cycles required by the targeted core to perform the processing of each block, without taking into account the data-transfers.

Table 1. Computational complexity and data-transfers of the main blocks of the encoder

	Operations		Data-Transfers			
	Tot	Tot [%]	R	R [%]	W	W [%]
DC Shift & Comp. Transf.	1.72e7	8 %	3.69e6	7 %	3.69e6	23 %
2-D DWT	8.33e7	37 %	1.95e7	37 %	9.77e6	62 %
Tier-1	1.22e8	55 %	2.96e7	56 %	2.12e6	13 %
Tier-2	2.64e5	< 1 %	5.49e5	1 %	1.78e5	1 %
All Blocks	*2.23e8*		*5.33e7*		*1.58e7*	

As with the results of the computational complexity analysis, the results of the data-transfer analysis were mapped onto the MicroBlaze instruction set, taking into account the latencies of the *load* and *store* instructions of the core [12]. By summing the results of this mapping with the results of the computational complexity analysis and multiplying the obtained total by the desired frame rate of 15 frames per second, an estimate of the required processing power was obtained in term of clock frequencies, as shown in Table 2. This estimate represents the clock frequencies at which the MicroBlaze core should run in order to perform the processing of each block of Fig 2.

Table 2. Estimation of the minimum clock rate of the MicroBlaze core required to perform the processing of each block at the desired frame rate of 15 frames per second

DC Shift & Comp. Transf.	*2-D DWT*	*Tier-1*	*Tier-2*
479 MHz	2126 MHz	2786 MHz	26 MHz

Considering that the maximum clock frequency for the MicroBlaze core is approximately 125 MHz for a Xilinx Virtex-II™ FPGA, the results in Table 2 clearly show that only the *Tier-2* coder can be implemented in software, while for all the other blocks a hardware accelerator is necessary.

4.2 Hardware Software Co-Design

The target platform for implementing the system is the Xilinx MicroBlaze Multimedia Demonstration Board designed around a Xilinx Virtex-II XC2V2000 FPGA, which is the heart of the user-defined video processing engine. The FPGA is supported with five independent banks of 512K × 36-bit ZBT RAM with byte write capability. These memories may be used as microprocessor code/data storage or as video frame buffers. The microprocessor supported is a soft 32-bit RISC processor (MicroBlaze™) which can utilize IBM Power PC™ peripheral busses and IP. MicroBlaze is a pre-synthesized soft core implemented in the FPGA fabric and can be included in the FPGA design source as a black box. The MicroBlaze design supports full 32-bit operands, 32-bit data paths and 32-bit registers to provide high performance.

According to the results of the complexity analysis, the three blocks targeted for hardware acceleration were the *DC shift & Component Transform*, the *2-D DWT* and the *Tier-1* coder. The MicroBlaze core was targeted for the software implementation of the *Tier-2 coder*, as well as for mastering the whole processing by handling the interrupt requests from the hardware blocks and scheduling the different tasks. Furthermore, the data-transfer results of SIT Memory Simulation (Table 1) clearly show that *DC Shift & Component Transform*, *2-D DWT* and *Tier-1* are the most I/O intensive tasks, accounting for about all the read and write operations. Mapping only the I/O operations onto the MicroBlaze instruction set, similarly to what previously done for obtaining the global results of Table 2, yields an estimate of the impact of the I/O

operations only on the clock frequencies required by the MicroBlaze to sustain the desired frame rate, as shown in Table 3.

Since the maximum clock frequency of the MicroBlaze core is about 125 MHz, the results in Table 3 clearly show that the modules for the *DC Shift & Component Transform*, *2-D DWT* and *Tier-1* tasks, targeted for hardware implementation, must access their respective input and output data independently of the MicroBlaze core. It is therefore necessary to provide hardware-controlled memory access so that each hardware block has direct access to the off-chip memory and the processor is not involved in these transactions. For this reason, it was decided to dedicate to the aforementioned hardware modules three of the five independent banks of ZBT SRAM. A Multi-Memory/Multi-Port ZBT Interface was created in hardware (Fig 3, Fig 4), in charge of round-robin interfacing, frame after frame, the three hardware accelerators to the three dedicated ZBT SRAM banks; thanks to the round-robin interfacing scheme and to the fact that the ZBT SRAMs can be accessed independently, the three hardware accelerators work independently and concurrently on their respective data, without charging the MicroBlaze core of any extra data-transfer load.

Table 3. Impact of I/O operations on the estimated performance of the MicroBlaze core

	DC Shift & Comp. Transf.	*2-D DWT*	*Tier-1*	*Tier-2*
Read	111 MHz	584 MHz	888 MHz	16 MHz
Write	111 MHz	293 MHz	64 MHz	5 MHz
Total	221 MHz	877 MHz	952 MHz	22 MHz

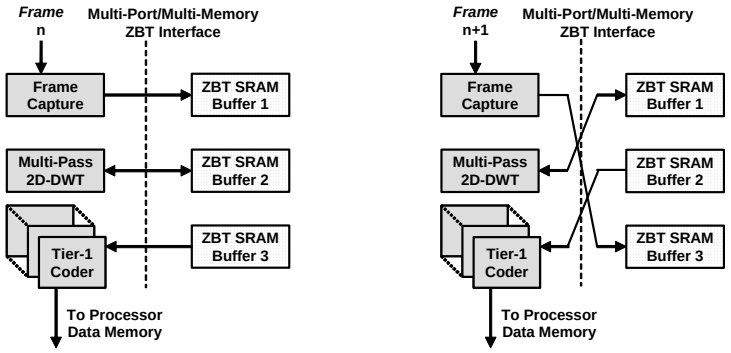

Fig 3. Round-Robin connections between the hardware accelerators and the dedicated ZBT RAMs for two successive frames, as managed by the Multi-Port/Multi-Memory ZBT Interface

The *DC Shift & Component Transform* tasks only rely on a single pixel value for their computations. For this reason it was decided to integrate them into the frame capture streaming data path, resulting in a straightforward implementation and in reducing the overall bandwidth of the corresponding hardware module, since the memory accesses for the temporary intermediate results are eliminated.

The second block that was targeted for hardware acceleration was the *2-D DWT* function. Since the DWT is being performed on the entire frame as a single tile, it becomes unreasonable to store an entire tile on chip. The DWT was implemented using a line-based design so that it can accept a stream of input data, while the output results are written over the already-read values off-chip; line buffers can easily be implemented using the many on-chip 18Kbit BlockRAMs that Virtex-II provides, thus reducing the bandwidth toward the external ZBT SRAM by avoiding storing the temporary data between horizontal and vertical filtering.

The last block that was hardware accelerated was the *Tier-1* coder. This coding involves bit/context modeling and arithmetic coding. A hardware core was designed to accept a single code-block containing up to 4096 words (as specified by the standard), perform the modeling and arithmetic coding on that code-block, and store the compressed byte stream. Three Tier-1 coders were implemented to operate in parallel and guarantee the required processing power for the desired frame rate. As with the other hardware modules, the bandwidth from the ZBT SRAM buffer was optimized by re-scheduling the operations on temporary data in order to reduce of the corresponding I/O accesses.

5 System Implementation

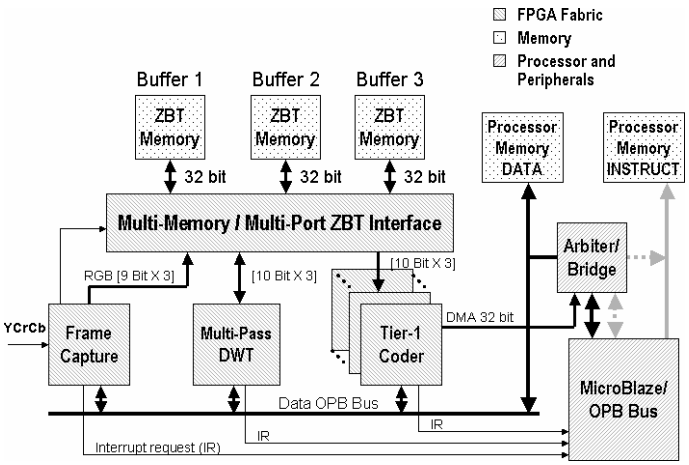

Fig 4. Block diagram of the motion-JPEG2000 system on multimedia board

Fig 4 shows the motion-JPEG2000 encoder system as implemented on the multimedia board using a Xilinx MicroBlaze soft processor. Note that two of the five off-chip ZBT SRAMs are utilized by the processor for data and instructions, while the other three are used for storage of intermediate frame data. First, the frame capture block grabs the YCrCb data from the NTSC camera, performs de-interlacing and conversion to RGB, and also performs the DC shift and component transformation for the

JPEG2000 encoder. This data is stored in one of the ZBT buffers. Secondly, the *2-D DWT* block performs the required transformation on all three components for a user-specified number of decomposition levels. Lastly, the DWT coefficients are read and processed by three *Tier-1* coders in parallel, with each coder operating on a 64x64 code-block. The output byte streams of each of these code-blocks is transferred to the processor data memory, where the compiling of the codestream (i.e., *Tier-2* coding) is performed by the MicroBlaze processor. The three tasks: frame capture, DWT, and coding are all given a frame time to complete their job. This gives an overall latency of the system of three frames.

6 Conclusions

This paper presented a new approach to hardware/software co-design based on high-level algorithmic complexity analysis and showed how it was applied to a real design case of a motion-JPEG2000 encoder for a video surveillance application. The target architecture was based on an FPGA with an embedded RISC core, providing an excellent platform for a hardware/software co-design. A software representation of a JPEG2000 encoder was analyzed by means of an automatic tool, the Software Instrumentation Tool, in order to extract relevant information about the algorithmic complexity of the encoder, such as the number of operations and data-transfers. The results of the complexity analysis were applied to the design of the encoder, allowing a fast evaluation of the system implementation requirements as well as of the hardware/software partitioning constraints.

References

1. CoWare Corporation: CoWare N2C Design System. Available on the Internet at www.CoWare.com
2. Synopsys Corporation: Designing Complex Digital Communications for Systems on a Chip. Available on the Interned at www.synopsys.com/products/dsp/digital_br.html
3. Ebrahimi, T. et al.: Dynamic Coding of Visual Information, Technical Description. JTC1/SC2/WG11/M0320, MPEG-4, International Organization for Standardization ISO/IEC (Oct. 1995)
4. Boliek, M., Christopoulos, C., Majani, E. (Editors): JPEG 2000 Part 1 Final Publication Draft. ISO/IEC JTC1/SC29/WG1 N2678 (July 2002)
5. Ravasi, M., Mattavelli, M.: High-Level Algorithmic Complexity Evaluation for System Design. Journal of Systems Architecture, Vol. 48/13-15. Elsevier Science B.V., (May 2003) 403-427
6. Xilinx: Virtex-II Platform FPGA Handbook v1.4. (November 2002)
7. Nachtergaele, L., Moolenaar, D., Vanhoof, B., Catthoor, F., De Man, H.: System-Level Power Optimization of Video Codecs on Embedded Cores: A Systematic Approach. Journal of VLSI Signal Processing, 18 (1008) 89-109
8. Kuhn, P.: Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Kluwer Academic Publisher (1999) 61-81

9. Li , Y. S., Malik, S.: Performance Analysis of Embedded Software Using Implicit Path Enumeration. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16 (December 1997) 1477-1487
10. Fukuhara, T., Singer, D.: Motion JPEG2000 Version 2, MJP2 Derived from ISO Media File Format. ISO/IEC JTC1/SC29/WG1 N2718 (October 2002)
11. Taubman, D.: Software Architectures for JPEG2000. International Conference on Digital Signal Processing, vol. 1 (2002) 197-200
12. Xilinx: MicroBlaze Software Reference Guide, v2.2 (April 2002)