2011 8th International Multi-Conference on Systems, Signals & Devices

FAST PROTOTYPING H.264 DEBLOCKING FILTER USING ESL TOOLS

Taheni Damak[1], Imen Werda[1], Nouri Masmoudi[1], Sébastien Bilavarn [2]
[1] University of Sfax, National School of Engineering Sfax, Tunisia,
Laboratoire d'Électronique et des Technologies de l'Information - LETI
[2] LEAT Université de Nice-Sophia Antipolis Nice, France
Laboratoire d'Electronique, Antennes et Télécommunications - - CNRS UMR6071
Nouri.Masmoudi@enis.rnu.tn, Sebastien.BILAVARN@unice.fr;

Abstract—**This paper presents a design methodology for hardware/software (HW/SW) architecture design using ESL tools (Electronic System Level). From C++ descriptions, our design flow is able to generate hardware blocks running with a software part and all necessary codes to prototype the HW/SW system on Xilinx FPGAs. Therefore we use assistance of high level synthesis tools (Catapult C Synthesis), logic synthesis and Xilinx tools. As application, we developed an optimized Deblocking filter C code, designed to be used as a part of a complete H.264 video coding system [1]. Based on this code, we explored many configurations of Catapult Synthesis to analyze different area/time tradeoffs. Results show execution speedups of 95,5% compared to pure software execution etc.**

Keywords- H.264, Deblocking filter, ESL, Catapult C.

1. INTRODUCTION

Recently, the intensifying demand of multimedia services has led to the development of embedded systems supporting ever-increasing functionality and flexibility [2]. Because of this evolution, multimedia systems become more and more complex with shorter time to market constraints. The H.264 [1] video coding standard is a representative example of this trend, especially if we consider embedded implementations. To speedup performances and to overcome the complexity of H.264 codec, several works have been proposed considering both software optimization [3] and hardware acceleration [4]. Comparison between the two approaches is difficult because of the differences between hardware and software design characteristics. Software implementation does not allow the exploration of parallel implementation structure that hardware design can expose. However, when the complexity of the system grows, hand coding at Register-Transfer Level (RTL) hardware design becomes both time consuming and error prone. Therefore, Electronic System Level (ESL) brings a solution to decrease the design time of dedicated hardware and keep the high abstraction level of software design. The automation introduced also enables a wider exploration of the design space. It has been reported to provide around five times speed-up of design time as compared to manual RTL methodologies [5]. Many other commercial and academic high-level synthesis tools can be used to generate an RTL code from a C++ description: AccelFPGA (AccelChip), SystemC Compiler (Synopsys) for commercials, SPARK, Cathedral, Catapult-C (Mentor Graphics), etc.

In this paper, we propose a specific design methodology to develop software and hardware execution achieving complete and efficient reconfigurable implementations from pure C specifications. We first developed a software decoder based on H.264 video coding standard, detailed in [6]. Then we analyzed the individual functions of the decoding process in terms of hardware acceleration potential. Profiling results reveal that the most expensive task is the Deblocking filter with an average cost of 24% in the overall decoding process. Due to its highly computationally intensive nature, the Deblocking filter seriously impacts the real-time performance of the target video application. For these reasons, we chose the Deblocking filter to generate a HW/SW optimized solution using our proposed flow. Catapult C is used to generate and explore a multitude of solutions.

The remaining of the paper is organized as follows. In Section 2, the proposed design methodology is explained. Then in Section 3, an overview of the H264 Deblocking filter is presented. Section 4 exposes and discusses the implementation results obtained on a Virtex-5 FPGA with a Power PC processor and dedicated filter accelerator in different parallelism configurations. A comparison with other H264 implementation is also provided. Finally, a conclusion and perspectives are drawn in Section 5.

2. DESIGN METHODOLOGY

Our proposed method to develop a mixed architecture is based on a combination of different tools: Catapult C synthesis, Xilinx EDK and Mentor Graphics Precision RTL (Figure 1).The H.264 Deblocking filter written in C++ is the input of Catapult C. It is used to automatically develop the corresponding VHDL code by means of HLS. Platforms constraints are also defined in Catapult to take into account the FPGA resources. To test the resulting hardware block, we used a PowerPC processor that communicates with Deblocking Filter through a PLB bus. Xilinx EDK environments were used to generate and implement both software and hardware components and create an interface between them. XPS is one of Xilinx tools. It compiles and builds software projects on a PowerPC processor and creates a user logic template. The later, generates the communication material needed between the processor and the accelerator. ISE tool from Xilinx is used to implement the overall platform on the FPGA. The remaining of this section is split into three sub-sections. Contributions of each used tools are detailed in each one.

978-1-4577-0411-6/11/$26.00 ©2011 IEEE

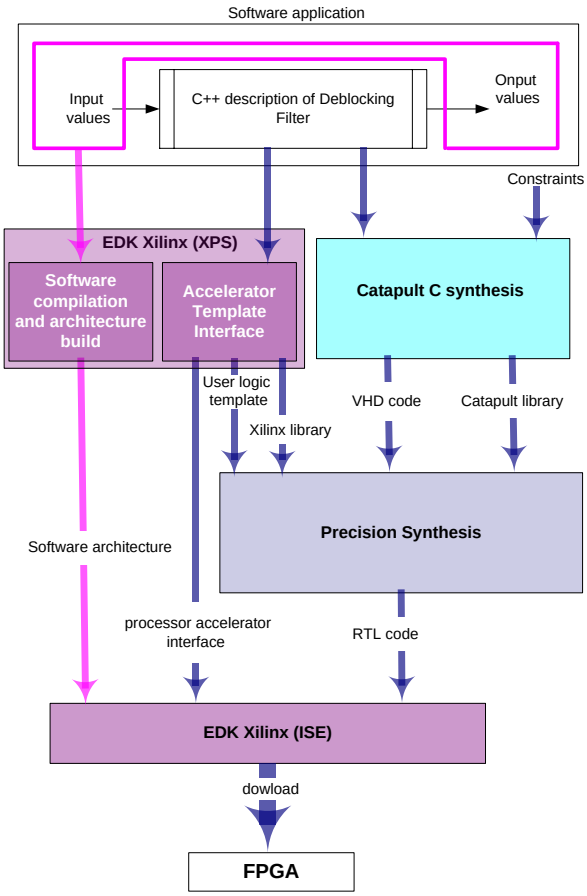

Figure 1. Design flow to develop software/hardware architecture.

2.1. Catapult:

Catapult C Synthesis [7] is the central tool in the proposed flow. It is a high-level synthesis tool developed by Mentor Graphics. Its role is mainly to generate optimized RTL from algorithmic specifications written in ANSI-standard C/C++. But it is more then a simple translator from a high level software language to hardware description language, it provides mechanisms to control and to optimize the generated hardware solution. Catapult C Synthesis is dedicated to ASIC and FPGA hardware designers who need to deliver optimal implementations with high time-to-market requirements.

The Catapult design starts with a high level specification that is used to define the algorithm and verify the functionality. The input description in C++ should be adapted to Catapult rules, e.g. dynamic memory allocation is not supported since it has no equivalence in a hardware description. Therefore all pointers should be replaced by a static allocation (arrays). In addition, some C/C++ design styles can be used to expose more parallelism for the RTL architecture [8]. For hardware description, a manual design should start with a Micro-architecture definition, and then a long time is depleted to develop an optimized RTL description, to verify and optimize it. Conversely, in Catapult C, designers have only to define implementation constraints (at the Scheduling step) in terms of latency and clock cycle time selection. In this step, the designer also has the possibility to fix input and output memory types to be used.

In our case, input and output arrays are implemented in single port RAM memory, while other I/O scalar parameters are mapped into registers. As result Catapult C generates automatically RTL VHDL code, hierarchical schematic, speed estimation and the design area. Catapult C output is not directly supported by Xilinx generation tools, thus another tool is used for logic synthesis of Catapult solutions (Mentor Precision).

Catapult C allows the designer to easily explore different area and speed trade-offs through loop unrolling and/or pipelining. By adding annotations to the source code, the designer can specify which of the identified loops needs to be unrolled and how many times it will be unrolled. Unroll loop in C++ description language means sequentially execution iterations, that correspond in VHDL to a repetitive hardware block that allows parallel execution implementation. In consequence, loop unrolling increases the execution time and affects FPGA surface. Therefore, several pipelining/unrolling configurations are investigated to reach the most efficient VLSI architecture. Comparative results of Catapult C configuration are given in results section.

2.2. Xilinx EDK:

The target device for this study is a Virtex 5 platform including a PowerPC hard core. The Xilinx EDK environment [9] handles both the software and hardware development flow. In particular, it automatically generates a VHDL template to interface a dedicated hardware block with the PLB system bus. The software project is compiled using XPS. An ELF file is generated and executed on the Power PC processor in standalone mode. The accelerator is connected to the PowerPC [10] through the Processor Local Bus (PLB v4.6) 32-bits system bus [11]. The PLB bus can be clocked until 133 MHz. In our case, we set it at 100MHz because it is the maximum bus frequency supported when the PowerPC is clocked at 400MHz. As a consequence the accelerator clock speed has been set to 100MHz. In the Deblocking filter, pixels are presented as bytes of data. Consequently pixel by pixel transfer uses only 25% of the PLB bandwidth since it is 32bit wide. Minimizing transfer time is mandatory in order to grant important accelerator speedups. So we have to take advantage of 32-bit data transfers. Four pixels are then packed into 32-bit data to decrease transfers times by four.

2.3. Precision synthesis:

At this stage, the architecture is generated by Xilinx EDK and the VHDL code is generated by Catapult C. The major problem is how to combine code of different tools until the final bitstream to be downloaded onto FPGA. Mentor Graphic Precision synthesis was used to synthesize Catapult RTL code with its libraries within the modified template EDK code and its libraries. The resulting code is used by Xilinx tools to generate the bitstream (ISE) and download it onto FPGA (EDK).

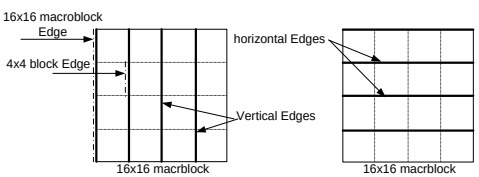

Figure 2. Horizontal and vertical edges in a macroblock.

3. DEBLOCKING FILTER ALGORITHM

Adaptive Deblocking filter is a new feature of H.264/AVC video coding standard used in the reconstruction loop of the encoder and the decoder. This filter greatly contributes to provide a better visual quality for the user and to enhance the rate-distortion performance of H.264/AVC-based video coding systems. As shown in Figure 2, the Deblocking filter is applied to each edge of the 4x4 blocks of pixels decoded inside and between 16x16 macroblocks (MB). It is conditionally applied to smooth horizontal and vertical block edges: vertical edges are first processed from left to right, and then horizontal edges from top to bottom. The proposed Deblocking filter process consists of two phases that occur at a MB level: boundary strength (BS) computation and filter core modules. The first function generates 32 BS values for 32 block edges within a MB: 16 horizontal edges and 16 vertical edges. This module calculates the BS value for each left or top edge of a 4x4 block within a given MB to evaluate the necessity of filtering and choose a filter type: No filter, Standard or Strong filter. The filter core module acquires an input MB from the reconstructed buffer and processes it depending on the filter type. Each of Standard and Strong filter is characterized by mathematic equations [1] that are applied directly to pixels. Since the Deblocking filter process is repetitive, a for-loop syntax structure is used to look into a MB direction edge (vertical/horizontal), 4 edges in each direction and 16 pixels in each edge. More details are given in the diagram of Figure 3: a first loop L1 is applied using the *HOR_VER_counter* parameter, the first iteration is for the vertical edge, the second iteration is for the horizontal edge. L2 is the next loop over *luma_edge_counter* to look over four MB edges. For strong and standard filter process four loops are used to read pixels and to apply corresponding equations: L3 and L4 for strong luma and chroma pixels, L5 and L6 for standard luma and chroma pixels.

4. IMPLEMENTATION RESULTS AND COMPARISON

The proposed Deblocking filter is based on the H.264 reference code from ITU [1]. It was first tested on a Pentium platform to valid its functionality, and then executed on the PowerPC processor on a Virtex-5 FX FPGA (ML507) in standalone mode. Using Catapult, the C code of the Deblocking filter was translated to VHDL RTL, wrapped with the PLB us interface template and tested on a Power PC with the corresponding generated accelerator. We used the PLB bus to connect the Deblocking filter accelerator to the processor. The proposed architecture was tested in a configuration of 400MHz for the PowerPC and 100MHz for the system bus and accelerator. Table 1 gives different accelerator parallelism configurations and the corresponding results in terms of execution time, execution speedup over full PowerPC execution, accelerator area and system area. In each proposed solution, we make use of different loop unrolling settings to check their impact and highlight an optimal solution. The first solution is the basic Catapult configuration without unrolling any loop (worst performance/lowest area). Then loops are progressively unrolled as shown in Table 1.

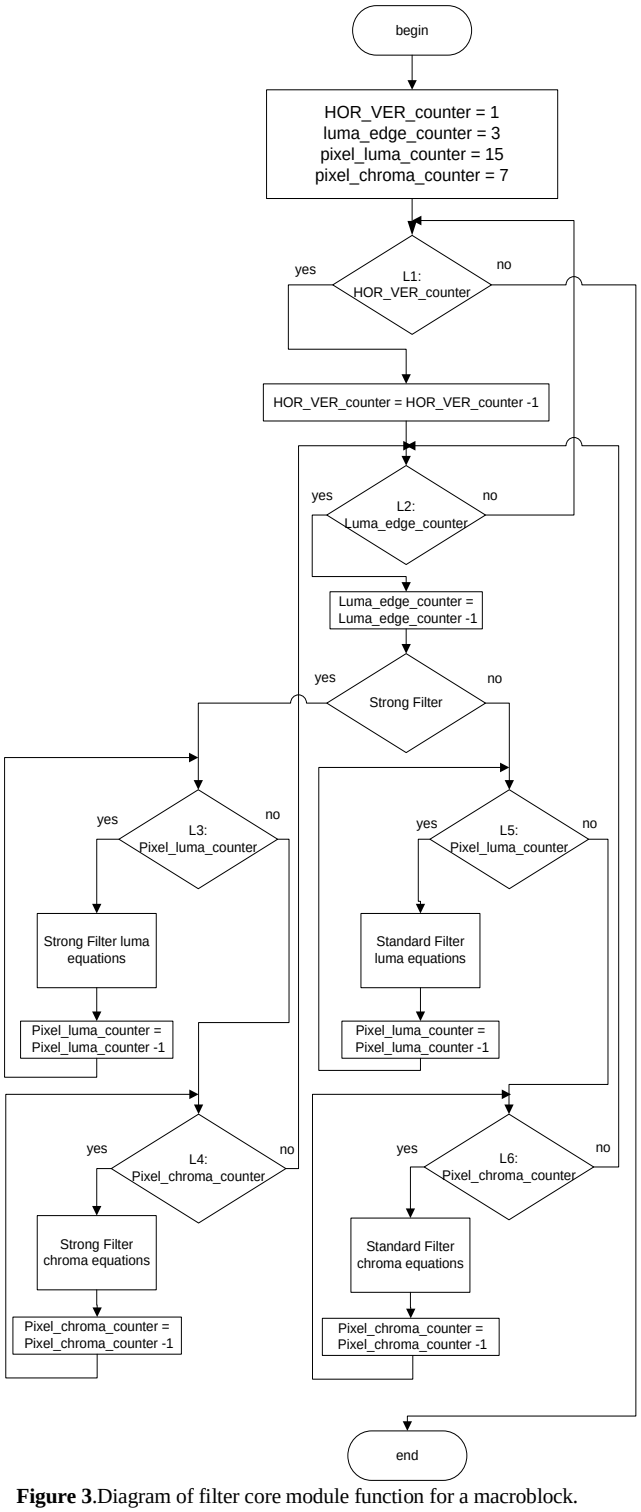

Figure 3.Diagram of filter core module function for a macroblock.

Since loops in our Deblocking filter are classified in three scales per MB (direction, edge and pixels scale) for both luminance and chrominance processing, we have started unrolling the pixels scale loop (L3, L4, L5 et L6).We proceeded as follow; solution 2 unrolls pixels scale loop for both chrominance loops (L4 and L6). Solution 3 keeps preceding loops unrolling and add luminance standard filter loop (L5). Solution 4 unrolls all luminance and chrominance fields of the pixel scale loop (L3, L4, L5 and L6). Solution 5 unrolls the edge loop (L2) with pixel scale loops.

Table 1 . Catapult loop unrolling configuration

	execution time(cycles)	execution speedup	Accelerator area(Slices)	system area(Slices)
Solution 1	475	88,26 %	5167	6223
Solution 2	402	90.07 %	7235	7276
Solution 3	382	90,5 %	9070	9919
Solution 4	324	91.99 %	10253	11024
Solution 5	260	93.57 %	11972	12952
Solution 6	180	95.55 %	13440	14323

Finally all loops are unrolled in solution 6 to process all MB Deblocking filter loops in parallel. In the results, we observe sensitive execution time decrease and area increase when we unroll the loops. For our application, area is not considered as a bottleneck since the FPGA is large enough to implement the accelerator. The highest area is reached by solution 6 which presents 40% of FPGA slice LUTs. Therefore we select solution 6 that provides the best execution time.

In order to compare with state-of-the-art architectures, we have summarized existing implementations of H.264 Deblocking filters in Table 2. State-of-the-art architectures are classified by Deblocking filter MB throughput per second: our proposed work doesn't achieve the highest throughput because of difference in frequency, technology and decoding filter algorithm. In fact, the proposed accelerator and [12], [13], [14] operate at the same frequency (100 MHz). An efficient comparison is justified here: our throughput is better in term of the number of cycles per MB processing. *H. L and all* [15] solution achieves a better throughput compared with our solution. They use a hand coded design with a well considered architecture exploiting the maximum of parallelism in the Deblocking filter algorithm. Consequently, results are effective and much optimized compared to our automated approach, at the expense of a very long development time. Furthermore, FPGA technology is not flexible enough to generate different architectural solutions. HLS allows larger possibilities of amending and improving solutions implemented on the platform. It also allows system portability that is more difficult with other tools.

5. CONCLUSION AND PERSPECTIVES

This paper presents the design flow and implementation of H.264 Deblocking filter on a Virtex 5 FPGA. Catapult C Synthesis provides reduced development time and allows designers to enhance performances applications from a high abstraction level when controlling implementation steps.

Since the Deblocking Filter interfaces with a processor, future works will address the full implementation of a decoder on Power PC processor and the global acceleration impact of the dedicated hardware filter.

The development flow proposed in this work will be applied to other time consuming modules of the H.264 decoder in order to accelerate the global video throughput. We expect significant design time reductions to develop those solutions from the use of the underlying design methodology.

REFERENCES

[1] ITU-T Recommendation H.264 & ISO/IEC 14496-10, "Advanced Video Coding for Generic Audiovisual Services", Version 4, 2005.

[2] F. Sun, and et al, "Application-specific heterogeneous multiprocessor synthesis using extensible processors," IEEE Trans. CAD, Vol. 25, Issue 9, pp. 1589-1602, 2006.

[3] Z. Wei, K. L. Tang, and K. N. Ngan, "Implementation of H.264 on Mobile Device", IEEE Transactions on Consumer Electronics, vol. 53, no. 3, pp. 1109-1116, Aug. 2007.

[4] M. Parlak and I. Hamzaoglu, "An efficient hardware architecture for H.264 adaptive deblocking filter algorithm," in Proc. Conf. Adaptive Hardware Syst., 2006, pp. 381–385.

[5] Feng Wang Yuan Xie Andres Takach "Variation-Aware Resource Sharing and Binding in Behavioral Synthesis", 978-1-4244-2749-9/09/$25.00 ©2009 IEEE.

[6] T.Damak, I.Werda, M.Ben Ayad, N.Masmoudi**,** "An Efficient Zero Length Prefix Algorithm for H.264 CAVLC Decoder on TMS320C64"**,** IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2010, Tunisia.

[7] M. G. Datasheet, "Catapult Synthesis," Mentor Graphics, Tech. Rep.,2008, http://www.mentor.com/products/esl/high_level_synthesis/catapult_synthesis/index.cfm.

[8] Catapult-C Manual and C/C++ style guide, Montor Graphics, 2004.

[9] EDK Reference Guide "EDK Concepts, Tools, andTechniques",A Hands-On Guide to Effective Embedded System Design XTP013 EDK10.1

[10] PowerPC Processor Reference Guide "Embedded Development Kit ".EDK 6.1 September 2, 2003.

[11] PLB Reference Guide Processor Local Bus (PLB) v4.6 (v1.00a) , DS531 August 9, 2007.

[12] Bin Sheng' Wen Goo', Di Wu'" An implemented architecture of deblocking filter for H.264/AVC", 2004 International Conference on Image Pfocessing (ICIP)

[13] Shen-Yu Shih; Cheng-Ru Chang; Youn-Long Lin, "A near optimal deblocking filter for H.264 advanced video coding,", Asia and South Pacific Conference on Design Automation, Jan. 2006.

[14] Tsu-M ing Liu; Wen-Ping Lee; Chen-Yi Lee, "An In/Post-Loop Deblocking Filter With Hybrid Filtering Schedule," IEEE Transactions on Circuits and Systems for Video Technology, July 2007.

[15] H. Loukil , A. Ben Atitallah, N. Masmoudi, "Hardware architecture of H.264/AVC deblocking filter algorithm », 6th International Multi-Conference Systems, Signals and Devices, IEEE 2009.

Table 2 . Implementation comparison.

	[4]	[12]	[13]	[14]	Proposed	[15]
Technology	0.15 µm	0.25 µm	0.18 µm	0.18 µm	65 nm	NA
RAM type	Dual-port	Dual-port	Dual-port	Dual-port	Single-port	NA
RAM size	18 x2	32x64 32x96	32x32, 32x1.5x1280x720	2x32x (1280x720+12)	912x8	1792
Processing cycles/MB	6144	446	214 or 246	243	180	105
Max frequency	72MHz	100 MHz	100 MHz	100 Mhz	100MHz	150 MHz
Throughput (KMB/s)	11.7	224	406 or 467	411	555,5	1428