Extending RISC-V ISA for Accelerating the H.265/HEVC Deblocking Filter

M. Alizadeh, and M. Sharifkhani
Department of Electrical Engineering,
Sharif University of the Technology, Iran
E-mail: alizadehms, msharifk@sharif.edu

***Abstract*—In this paper, we present an RISC-V based Application Specific Instruction Set Processor (ASIP) for accelerating the HEVC deblocking filter. The proposed ASIP extends the RISC-V instruction set to include new instructions specifically targeted to expedite the HEVC deblocking filter. These instructions are designed by profiling the implementation of the OpenHEVC filter. Our proposed solution improves the performance of the deblocking filter compared to the standard implementation by 11%. Furthermore, our approach can also be applied to accelerate other parts of the decoder by taking advantage of the programmability and flexibility of ASIPs.**

***Index Terms*—HEVC; Declocking filter; ASIP; RISC-V.**

I. Introduction

The past 20 years have seen fast growth in the usage of video streaming and video broadcasting. Over this period, video codecs improved significantly regarding compression and performance. High Efficiency Video Coding (HEVC), also known as H.265, is the most recent video codec released by the Joint Collaborative Team on Video Coding (JCT-VC), a joint activity of ITU-T Video Coding Experts Group (VCEG) and ISO/IEC Moving Picture Experts Group (MPEG) in 2013. HEVC can reduce consuming bitrate up to 50% compared to previous standards at the cost of increasing complexity. The HEVC standard has different units for decoding/encoding of a video sequence, such as ”Intra Prediction”, ”Inter Prediction”, ”Entropy Coding” and ”In-loop Filter”. It specifies two loop filters, a deblocking filter and a sample adaptive offset filter (SAO). The HEVC deblocking filter is designed for attenuating artifacts at the boundary of blocks. The HEVC decoding is a complex process and the deblocking filter can consume up to 15% of the codecs total decoding time [1].

Due to increase the speed of processing, a lot of research has been done on optimizing the video codec processing with different methods [2], [3]. The offloading complex functions from the processor into specialized accelerators is common. These accelerators are specialized for one application and optimized for increasing the performance and decreasing power. These accelerators can be implemented as application specific instruction-set processors (ASIPs) or as fixed-function hardware. An ASIP is superior in terms of flexibility and programmability. The design of ASIPs comes with the need to define the best-suited processor architecture and instruction set, to develop both the hardware implementation and the associated software development kit (SDK).

In this work, we proposed an ASIP solution for accelerating and improving the efficiency of the HEVC deblocking filter by designing and developing new instructions based on the 32-bit RISC-V instruction set.

RISC-V [4] is a new general purpose instruction set architecture (ISA) with wide support and adoption developed at the University of California, Berkeley. In contrast to most ISAs, RISC-V is open source and free which provides a small but comprehensive ISA and supports both 32 bit and 64-bit address space. Its toolchain consists of a GCC and a LLVM cross-compiler, a software ISA simulator, verificator and hardware descriptions. This makes it a good baseline for investigating of the ASIP approach with user-level ISA extensions and specialized variants.

Very little work, ASIP targeting HEVC was found in the literature. However, there are several ASIPs for the previous standard such as H.264/AVC and H.263 code. Ilkka Hautala *et. al.* [5] implemented a loop filter based on their TTA structure with some software modifications on the loop filter of HEVC. They claim that their solution can process 30 fps full HD. Kim *et al.* [6] design an ASIP for decoding and encoding of H.264 and reduced the number of clock cycles by 20%. Yanjun *et al.* [7] explored the ASIP approach for decoding H.263 standard by adding VLIW instructions. Gracieli Posser *et. al.* [8] designed a MIPS based ASIP for H.264 and decreased the running time of ”Invert Transform” by ten times.

The rest of the paper is organized as follows. Section 2 presents the typical design methodology of an ASIP. Section 3 describes the HEVC deblocking loop filter algorithm, and Section 4 proposed two methods for accelerating of the deblocking filter on RISC-V. The experimental results and evaluation of instructions are discussed in Section 5.

II. ASIP Design Methodology

Figure 1 shows the design methodology of a typical ASIP. First, the application is profiled at different levels: instructions level, function level, and algorithmic level. Then based on these analyses, application constraints, instruction generation, and architectural design are conducted to produce the ASIPs hardware description and its corresponding software SDK [9].

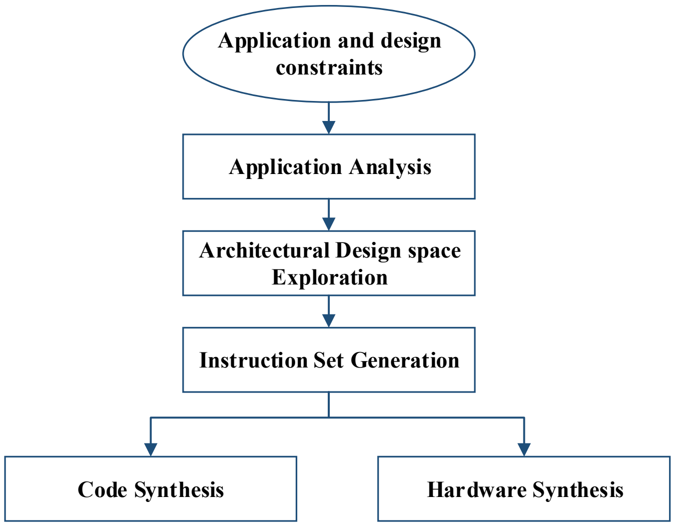

Application and design constraints
Application Analysis
Architectural Design space Exploration
Instruction Set Generation
Code Synthesis
Hardware Synthesis

Fig. 1. Flow diagram of designing an ASIP [4].

In this work, we have used RISC-V ASIP tool, which is a tool chain targeted for designing and developing ASIPs. It can automatically generate RISC-V based ASIP model along with its complete SDK.

III. HEVC Deblocking Filter

This section explains the deblocking filter as described in the HEVC Video standard. The deblocking filter can be divided into two parts, Boundary Strength (BS) and filter operation. The filter is applied to the vertical and horizontal boundaries of the 8×8 blocks of Prediction Unit (PU) and/or Transform Unit (TU) boundaries. Depending on the coding modes and conditions of the current block, such as prediction mode, motion vector (MV) and so on, a BS value is assigned to each boundary. These values can be 0, 1 or 2.

The filtering operation can be divided into two processes. The first process is decision making which determines, the filter should apply or not and the second process is How a boundary is filtered, strong or weak. For the first process, a decision is made based on the condition of (1). For this process, the following data are needed BS, Quantization Parameter (QP) for determining and to values. Two adjacent 4×4 blocks, block P and block Q pixels, to be filtered, as shown in fig.2 [10]. If (1) was true the filter will be applied.

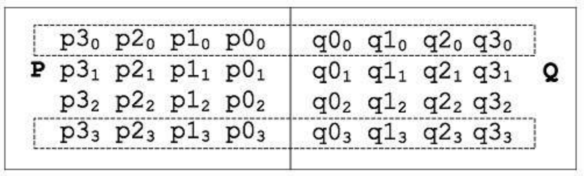
$$|p2,0-2p1,0+p0,0|+|p2,3-2p1,3+p0,3|+ \\ |q2,0-2q1,0+q0,0|+|q2,3-2q1,3+q0,3|>\beta, \quad (1)$$

$$|p2,i-2p1,i+p0,i|+|q2,i-2q1,i+q0,i|<\beta/8, \quad (2)$$

$$|p3,i-p0,i|+|q0,i-q3,i|<\beta/8, \quad (3)$$

$$|p0,i-q0,i|<2.5tC. \quad (4)$$

For the second process, based on the two adjacent 4x4 block pixels, two parameters d and dE are calculated, then to, β, d and dE decide how a boundary is filtered, if (2),(3) and (4) are true, strong filtering will be applied and based on (5),(6),(7) three pixels will be modified. In the other hand, weak filter will be applied to one or two pixels based on (8). The modifications will be done for q pixels as well.



$$\begin{array}{cccc|cccc} p3_0 & p2_0 & p1_0 & p0_0 & q0_0 & q1_0 & q2_0 & q3_0 \\ p3_1 & p2_1 & p1_1 & p0_1 & q0_1 & q1_1 & q2_1 & q3_1 \\ p3_2 & p2_2 & p1_2 & p0_2 & q0_2 & q1_2 & q2_2 & q3_2 \\ p3_3 & p2_3 & p1_3 & p0_3 & q0_3 & q1_3 & q2_3 & q3_3 \end{array}$$

Fig. 2. Four-pixel long vertical block boundary formed by the adjacent blocks P and Q [10].

TABLE I
Function Level Profiling of The Deblocking Filter.

ID	Symbol	Sample	Percentage
1	Main	270002	43.6
2	Hevc_loop,_filter	240920	38.9
3	Clip	72556	13.7
4	Abs	35120	5.7

$$\begin{aligned} Strong\,new\,p0 =& clip(p0-2tc, p0+2tc, 1/8 \\ & (p2+2p1+2p0+2q0+q1+4)), \quad (5) \\ Strong\,new\,p1 =& clip(p1-2tc, p1+2tc, 1/4 \\ & (p2+p1+p0+q0+2)), \quad (6) \\ Strong\,new\,p2 =& clip(p2-2tc, p2+2tc, 1/8 \\ & (2p3+3p2+p1+p0+q0+4)), \quad (7) \\ Weak\ new\ p0 =& clip(p0+clip(-1/2tc, 1/2tc, 1/2 \\ & (1/2(p2+p0+1)-p1+d))). \quad (8) \end{aligned}$$

IV. Profiling and Analysis

New instructions are added to the instruction set based on profiling the deblocking filter. Generally, profiling can be done at three level:

- Instruction level: In this level, the most frequent sequences of 2 to 5 are investigated and based on this analysis new instructions will be designed.
- Function level: Here the number of function calls is explored and frequently called chosen to be implemented as new instructions.
- Algorithm level: New instructions are designed by investigating the algorithm.

Profiling is done by executing the deblocking filter on an RISC-V processor model and provides instruction-level and function level profiling. Two methods are used for accelerating the deblocking filter, scalar instruction, and complex instruction:

A. *Scalar Instructions*

In the first method, new scaler instructions based on profiling are added. These instructions are simple and implemented without any extra registers. For adding this instruction, profiling is done on functional and algorithmic levels.

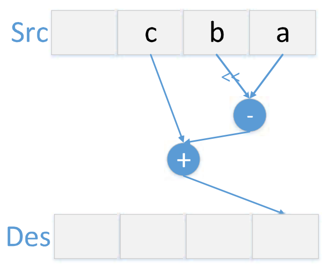


Fig. 3. Implementation of HAS.

1) Function level: Table I shows the number of function calls obtained by running the foreman sequence video on our model with their address in the memory and number of their samples. Based on these numbers, two functions are selected for being implemented as new instructions.

- Absolute (abs), is consumed 5.7% on average and is defined as (9) This was implemented as a new instruction with assembly description: *ABS Des, Src*

$$abs(src) = |src| \tag{9}$$

- Clip, which consumes 13% on average and defined as (10), is implemented as a new instruction with assembly description: *CLIP Des,Src,MinMax*

$$\text{CLIP(Src,MinMax)} = \begin{cases} MinMax & \text{if } Src > MinMax \\ -MinMax & \text{if } Src < MinMax \\ Src & \text{otherwise} \end{cases} \tag{10}$$

2) Algorithmic level: At this level, the algorithm of the deblocking filter is analyzed with the goal of replacing certain frequently repeated terms with new instructions. One of these terms is:

- *a-2b+c*, Defined as *HAS Dst, Src* which is frequently repeated in the decision-making process of the deblocking filter. The function of this instruction is shown in fig.3.

B. Complex Instruction

The second method is implementing rather complex instructions. These complex instructions handle a larger part of the algorithm. However, the hardware cost is also increased. In our work, we have implemented an instruction-set extension that can calculate p3,..,p0 and q3,..,q0 in one instruction. pi and qi require 8 bytes, furthermore, the parameter to requires 1 byte. This makes it clear that these data do not fit in the provided 32-bit register file. For this reason, we have added a 128 bit wide register file with two registers (one for source and one for destination). As next step, we plan to parallelize the filtering of two rows of pixels in one step.

As a new register file is introduced, it is also necessary to transfer data between the memory and the register file. Thus a LOAD and a STORE instruction were implemented. These instructions load a register from memory or store a register to memory.

The deblocking filter instruction performs the actual filtering. It read p3,..,p0 and q3,..,q0 from the 128 bit register argument. The tc parameter is read from a 32 bit register.Then the filtered p2, p1, p0, q0, q1, q2 values are calculated. Finally, the updated values are written to the 128 bit destination register. Because this instruction handles the deblocking filter calculation, the whole filter can be done in three instruction, load, filter, and store.

TABLE II
RESULTS OF ADDING NEW INSTRUCTION METHOD 1.

ID	Instruction	Speed Up %
1	Original	0
2	ABS	2.1
3	CLIP	4.1
4	HAS	1.0
5	Total	6.9

V. EXPERIMENTAL RESULTS

For implementing the instructions, a 32 bit, 5 stage pipelines RISC-V model with 32×32 bit registers and 8 MB of memory is used. The RISC-V ISA defines four kind of instructions I, R, U, S which are shown in fig.4.

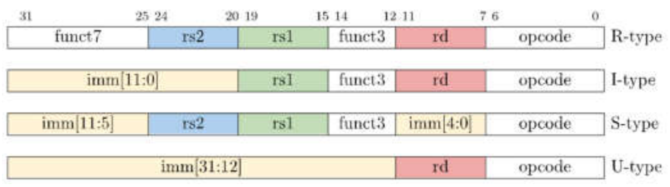


Fig. 4. Instruction type of RISC-V model [4].

In the first method, scaler instructions, ABS and HAS are implemented as R-type instructions with opcodes $(80b3h)$,$(81b3h)$ and the CLIP instruction is implemented as an I-type with opcode $(5400h)$. In the second method, the filter instruction is defined as an I-type, with opcode $(5041h)$ and the related load and store instruction are defined as I-type with opcodes $(5402h)$, $(5403h)$ respectively. The new 128 register file is called "LPRF" which is defined as 2×128 bit register file.

For testing and evaluating our ASIP model, we used the Foreman sequence video. It was encoded via the HEVC standard implementation HM $V.15$ [11] using the main profile, IPPPP G.O.P, and QP 32, as input to the deblocking filter running on our model implementation. The results are verified by comparing the obtained output frames with the OpenHEVC output frames.

The HEVC deblocking filter is modified using newly added instruction independently to measure their impacts on the performance. Table II shows speed up, the percentage of decreased number of total instruction obtained by executing test sequences using the newly added instructions in the first method. The results are depicted per instruction along with the total improvement, which is about 7%. The most effective instruction is CLIP with about 4 percent. The results of running the second method and the number of calls are shown in Table III. In this method, the speed up is about 11.2% on average, which is 8% better the first method. However, with the

TABLE III
RESULTS OF ADDING NEW INSTRUCTION METHOD 2.

Instruction		Number of Instruction	Speed Up %
Original		69857562	0
Method 2	Load	18808	-
	Filter	18808	-
	Store	18808	-
	Total	61726266	11.6

extra cost of adding a 2×128 register file and more complex hardware.

For comparison, the newly added instructions are implemented by using just simple 32 bit adders and comparators, the hardware cost of the first method and the second method are estimated. The area cost is calculated based on the number of required Programmable Logic Array (PLA). As in [12] the number of PLAs used for an adder is 1056 and for a comparator is 33. Also, the hardware cost of additional register file introduced in Method 2 is calculated with the factor of 1 PLA for each 1 bit.

As shown in Table IV method 1 with scaler instruction has 7% lower number of instructions with the area cost of 4323 and the second method with complex instructions, reduced the number of instructions by 11.6% with the cost of 8 times more additional hardware compare to the running of deblocking filter without any optimization.

TABLE IV
HARDWARE AND AREA COST OF THE FIRST AND THE SECOND METHOD.

Instruction		Hardware Cost	Area Cost (#PLA)
Method 1	ABS	1 Comp+ 1 Adder	1089
	HAS	2 Adder	2112
	CLIP	2 Comp + 1 Adder	1122
	Total	3 Comp + 4 Adder	4323
Method 2	Filter	3 Comp + 35 Adder	37059
	Load/Store	256 Bit Reg	256
	Total	Comp+ 12 Adder + 256 Reg	37315

The proposed ASIP can do loop filter for an HD bitstream 30 fps if the CPU is clocked 900 MHz with complex instruction. Also, it is compatible for extending to other parts of HEVC and applications, however, TTA architecture is more complex for adding a new application.

VI. CONCLUSION

In this work, an ASIP for accelerating the HEVC deblocking filter is designed. By profiling the filter at the algorithmic and function level, new instructions were added to the RISC-V standard ISA. We have obtained an improvement of about 7% in the first method and 11.6% in the second method with costs of more additional hardware unit usage. This approach can be applied to other parts of the standard to accelerate the full HEVC video decoding.

REFERENCES

[1] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, 2012.

[2] K. Chen, Y. Duan, L. Yan, J. Sun, and Z. Guo, "Efficient SIMD optimization of HEVC encoder over X86 processors," in *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, Dec 2012, pp. 1–4.

[3] M. S. Alizadeh and M. Sharifkhani, "Low complexity downsizer h.264 decoder," in *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*, Oct 2014, pp. 604–607.

[4] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The RISC-V Instruction Set Manual. Volume 1: User-Level ISA, Version 2.0," DTIC Document, Tech. Rep., 2014.

[5] I. Hautala, J. Boutellier, and J. Hannuksela, "Programmable lowpower implementation of the HEVC Adaptive Loop Filter." in *ICASSP*, 2013, pp. 2664–2668.

[6] S. D. Kim and M. H. Sunwoo, "ASIP approach for implementation of H. 264/AVC," *Journal of Signal Processing Systems*, vol. 50, no. 1, pp. 53–67, 2008.

[7] Y. Zhang, H. He, Z. Shen, and Y. Sun, "ASIP approach for multimedia applications based on a scalable VLIW DSP architecture," *Tsinghua Science & Technology*, vol. 14, no. 1, pp. 126–132, 2009.

[8] G. Posser, G. Corr, R. Reis, L. Carro, S. Bampi *et al.*, "A MIPS-based ASIP to accelerate the inverse Hadamard tranform for H. 264/AVC video coding," in *2010 First IEEE Latin American Symposium on Circuits and Systems (LASCAS)*. IEEE, 2010, pp. 109–112.

[9] M. K. Jain, M. Balakrishnan, and A. Kumar, "ASIP design methodologies: survey and issues," in *VLSI Design, 2001. Fourteenth International Conference on*. IEEE, 2001, pp. 76–81.

[10] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van der Auwera, "HEVC deblocking filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1746–1754, 2012.

[11] Hm. [Online]. Available: https://hevc.hhi.fraunhofer.de/

[12] J. Jacob, P. S. Sivakumar, and V. D. Agrawal, "Adder and comparator synthesis with exclusive-OR transform of inputs," in *VLSI Design, 1997. Proceedings., Tenth International Conference on*. IEEE, 1997, pp. 514–515.