Code improvements towards implementing HEVC decoder

Israr, Adeel; Nazir, Sajid; Kaleem, Muhammad

Published in:
2018 3rd International Conference on Emerging Trends in Engineering, Sciences and Technology (ICEEST)

DOI:
10.1109/ICEEST.2018.8643307

Publication date:
2019

Document Version
Author accepted manuscript

Code Improvements Towards Implementing HEVC Decoder

Adeel Israr
Department of Electrical and Computer Engineering, COMSATS University
Islamabad, Pakistan
adeel.israr@comsats.edu.pk

Sajid Nazir
School of Computing, Engineering and Built Environment
Glasgow Caledonian University,
Glasgow, UK
sajid.nazir@gcu.ac.uk

Muhammad Kaleem
Department of Electrical and Computer Engineering, COMSATS University
Islamabad, Pakistan
mkaleem@comsats.edu.pk

***Abstract*—HEVC is the latest video coding standard achieving better compression as compared to previous standards but at a higher complexity. Many applications require fast and real-time decoding of compressed video for which Field Programmable Gate Array (FPGA) is one of the efficient options. In this paper we provide the code improvement and implementation of the computationally intensive HEVC decoder. We describe our investigation for the implementation of HEVC decoder on Visual C++ 2008 platform and present improvements to the code of HEVC decoder required prior to its conversion to a hardware description language. The results for decoding time are also provided.**

Keywords—video decoding, compression, HEVC decoder, complexity, Software debugging

I. Introduction

The advancements in Internet speed, processing and social media applications are driving the video traffic over the Internet. This requires efficient compression algorithms.

The latest video coding standard is High Efficiency Video Coding (HEVC) which has 50% compression efficiency compared to earlier coding standards but at a cost of higher complexity [1]. A comparison analysis of H.264, VP9 and HEVC is provided in [2] using Peak Signal to Noise Ratio (PSNR), encoding time, and Structural Similarity (SSIM) concluding that objective quality of HEVC is far better than other encoders. The intra coding performance (lossy and lossless) for HEVC, H.264, JPEG 2000 and JPEG LS is described in [3] concluding that HEVC provides performance gains over the other coding standards. A CUDA based parallel H.264 implementation of 4k UHD live streaming over wireless was demonstrated in [4]. A unified software decoder is described in [5] for the different HEVC extensions, such as Scalable, 3D and range.

HEVC enables video transmission over mobile networks [6]. A subjective evaluation of HEVC and H.264 in mobile environments is provided in [7]. Software innovations are in scope but hardware decoders are the key in the long term for mass-market reach of HEVC [6]. HD video is common but HEVC is a key enabler for Ultra HD [6]. A review of the building blocks and architecture of HEVC is provided in [8][9]. Software only solution has high power consumption and therefore hardware based implementation needs to be relied on for low-power devices and applications. Hardware decoders can be built using Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Array (FPGA) but FPGA provides a reconfigurable platform, with corresponding cost savings [1][10]. A hardware based implementation of HEVC is described in [11] that could address the power and real-time constraints.

We provide preliminary investigation of compiling the standard HEVC implementation available from [12] on Visual C++ 2008. This platform was selected due to its simplicity, ease of use, and relatively faster application compilations although other Windows and Linux based platforms are available. A number of problems were faced during the process of implementation of HEVC tool. For HEVC decoder code, it was observed that the constructor and destructor code for many classes were incomplete, and therefore it was needed to be reconstructed. Similar problems were found and corrected in various other functions of many classes.

The focus of this paper is to describe the various fixes for HEVC decoder using VC++ 2008 platform and provide our preliminary research work towards implementation of HEVC decoder in hardware.

This paper is organized as follows. Section II describes the related work. The system model is described in Section III. Section IV details the improvements in the HEVC decoder implemented on Visual C++ 2008 platform. Section V describes the results and the conclusion is provided in Section VI.

II. Related Work

The Rate-Distortion optimization is used to select initial set of motion vectors in Advanced Motion Vector Prediction (AMVP) for HEVC standard [13]. As there are many different sizes of motion blocks therefore it is a critical to optimize the RD process. A scheme is proposed for the HEVC encoder that reduces the encoder complexity by 1.7% but is accompanied by a PSNR loss of 0.10 dB and increases in bit rate by 2.4%. FPGA architecture is proposed for AMVP on Xilinix Virtex-7 FPGA which processed 8K (7680 x 4320) resolution at 60 fps with 1% of FPGA resource utilization [13].

FPGA implementation on Xilinix Zynq using Verilog that exploited critical path delays, single-cycle reference pixel processing, sparsity of transformed coefficient matrix, and a latency aware cache architecture is described in [1] achieving 4K 30 frames/seconds HEVC decoder. The authors claim their work to be the first implementation of a real-time HEVC decoder on a commercial FPGA [1].

An energy consumption reduction of 34.66% compared to the original HEVC intra prediction is reported in [14]. One DSP block in FPGA implemented one intra angular

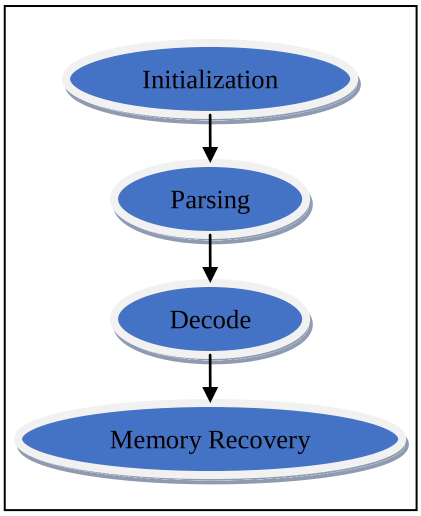

Fig. 1. The flowchart of HEVC decoder.

prediction. The implementation could process HD video at 55 frames/seconds [14].

HEVC Sub-pixel interpolation operation is computationally intensive [15]. Sub-pixel HEVC interpolation was implemented on a Xilinix Vivado High-level Synthesis (HLS) which could process 45 quad full HD (3840x2160) frames/seconds demonstrating use of HLS tools [15].

III. System Model

We downloaded the HEVC software from [12], version 16.9 using Slicksvn. The HEVC tool consists of two main classes namely, TAppDecCfg and TAppDecTop. TAppDecCfg manages the names of input file, number of the layers to be decoded, and the names of the output files. The methods in this class parse the command line string input to the decoder to extract the information to populate these attribute variables. TAppDecTop, on the other hand, is a child class of TAppDecCfg, and actually performs the process of decoding. It takes the compressed file and passes it through various steps to generate various layers of decompressed video.

The main decompression starts with declaration of an instance of TAppDecTop class. It is followed by parsing of the input arguments for the decoder, this is done by calling the method function inherited from the TAppDecCfg class. Then the decode function that actually decompresses the input to the various layers of video as requested in the input arguments. After the completion of the decompression process, the destroy function cleans and recover all the memory allocated to various attribute variables of classes of the HEVC tool.

IV. Member Variable Initializations in Constructors

The warnings and error settings can be defined in Microsoft Visual C++. For our work we used Visual C++ 2008 platform.

A total of 63 classes make HEVC decoder. Some are large while othere are significantly smaller. Out of these, we made slight changes to the eleven larger classes to improve the coding structure as shown in Annex A.

There are two main classes in the HEVC decoder, TAppDecCfg and its drived class TappDecTop. It was

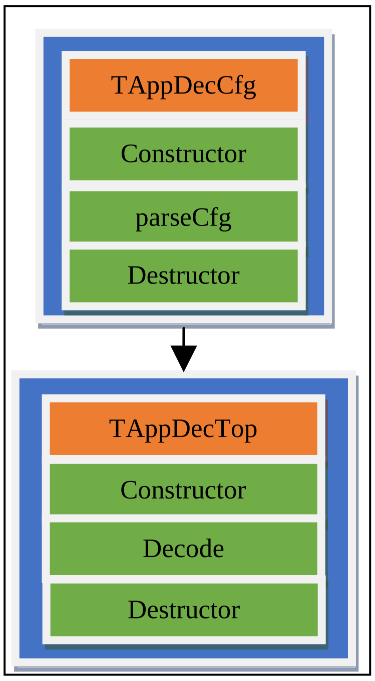

Fig. 2. The main function for the two main classes in the HEVC software.

found that a number of member variables of these classes were not properly initialized, and the constructor for TAppDecCfg was entirely missing. A constructor function of the class was written and the variables were properly initialized. In addition, a destructor function was written to recover the memory allocated for these variables. Similarly, the drived class TAppDecTop had an empty constructor function in the orignal code that resulted in the crash of this class during execution because of the unitialization of its member variables.

Fig. 1 shows the flow chart of HEVC decoder. First an object variable of the class TAppDecTop is decleared and is initialized thereby allocating memory for all the member variables of that class and the ones inherited from TAppDecCfg class. This is followed by the process of parsing the command line string which instructs the HEVC decoder, the name of the compressed file, the number of layers to be decoded and the names of the files where the decoded layers will be stored. This is achieved by calling parseCfg function of the class which is actually inherited from TAppDecCfg class as depicted in Fig. 2. After the completion of parsing, the actual decoding process is started. The decode function is an actual function of TAppDecTop class which is not inherited from TAppDecCfg class. Once the process of decoding is completed, the memory allocated for various member variables (inherited and otherwise) is recovered. The orignal code had no such provision and it used to crash. Therefore, memory recovery functions were added in the original HEVC code to make it compatible for VC++2008 platform.

The decode function is part of the TAppDecTop class and it uses a number of objects of various classes during the process of decoding. There were a number of problems in many of these classes that resulted in the crash of decode function. These problems were fixed to make a working version of HEVC decoder for said platform.

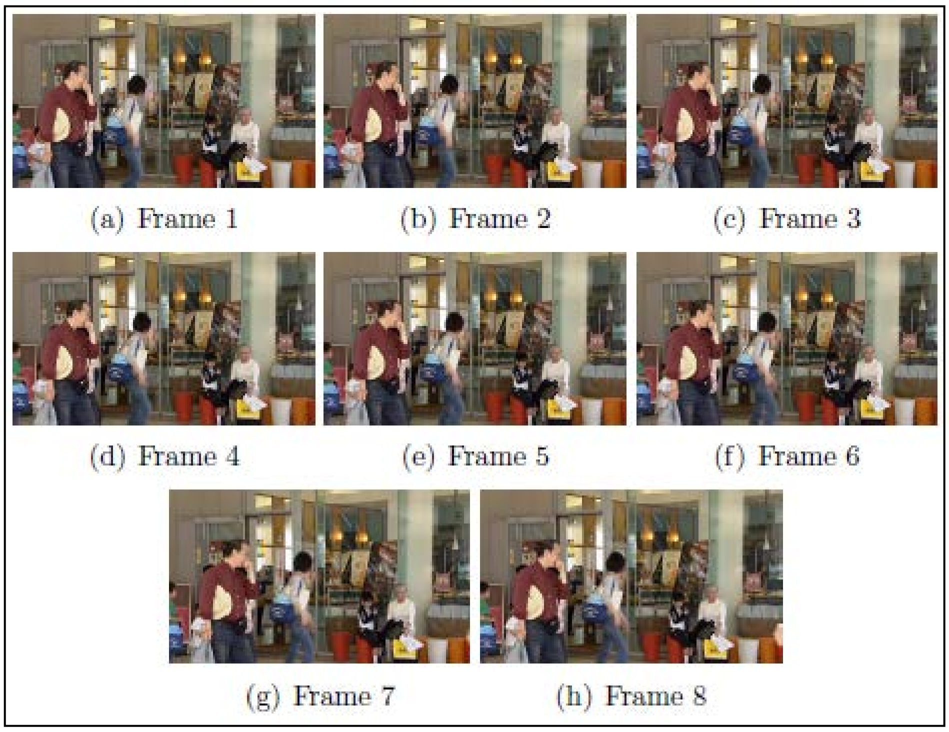

Fig. 3. Frames of the Base Layer.

We defined constructors and destructor functions for some classes which were missing and also provided the member function initializations. The constructors and member variables initializations were provided for the following classes:

- InputNALUnit
- TDecTop
- NALUnit
- TComPic
- TComPicYuv

V. RESULTS

HEVC compressed file used to check the functionality of the improved decoder has the following details. The file used for decompression is BQMall.bin. It consists of 165472 bytes. The process of decompression generates two YUV420 video files representing base and first layers. Both of the layers generated by HEVC tool consist of 8 frames. Fig. 3 depicts the scaled down version of all the frames of the base layer. The base layer consists of 1,198,080 bytes in total. Every frame of this layer consists of 416x240 pixels, the luminance (Y) component consists of 99840 bytes, and the two chrominance components each consists of 24960 bytes. On the other hand, the first layer video consists of 4,792,320 bytes. A single frame of this layer consists of 832x480 pixels, the luminance (Y) component consists of 399360 bytes, and the two chrominance components each consists of 99840 bytes. It can be clearly observed that the base layer is more than 7 times the size of the compressed file and the first layer is 28 times of that size.

Table 1: Attributes of the decoded layers

	Base Layer	First Layer
Total Size (all frames, Bytes)	1,198,080	4,792,320
Frame Size (Pixels)	416×240	832×480
Luminance (Y) Size (Frame)	99840	399360
Chrominance (U) Size (Frame)	24960	99840
Chrominance (V) Size (Frame)	24960	99840

To estimate the performance of the decompression software we measured the computing time on an Intel Pentium 4 Dual Core 3.00 GHz machine with 2 GB memory. The calculated execution time for the decompression, display and storage of frames as files was 13.5 seconds.

VI. CONCLUSION

This paper describes the importance of the emerging HEVC standard as a preferred video coding standard for high resolution videos. The complexity issues motivating the need of a hardware implementation are also discussed. We provide our initial investigations of the HEVC decoder on Visual C++ 2008 platform and provide many significant improvements to the HEVC decoder code.

In our future work we will be implementing HEVC decoder code in a hardware description language and thereby significantly improving the performance of the decoder.

References

[1] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. De Silva, and A. Pasqual, "4K Real-Time HEVC Decoder on an FPGA," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 236-249, January 2016.

[2] L. Mengzhe, J. Xiuhua, and L. Xiaohua, "Analysis of H.265/HEVC, H.264 and VP9 coding efficiency based on video content complexity," *IEEE International Conference on Computer and Communications (ICCC)*, Chengdu, 2015, pp. 420-424. doi: 10.1109/CompComm.2015.7387608.

[3] Q. Cai, L. Song, G. Li, and N. Ling, "Lossy and lossless intra coding performance evaluation: HEVC, H.264/AVC, JPEG 2000 and JPEG LS," *Proceedings of The 2012 Asia Pacific Signal and Information Processing Association Annual Summit and Conference*, Hollywood, CA, 2012, pp. 1-9.

[4] A. O. Adeyemi-Ejeye and S. Walker, "4kUHD H264 Wireless Live Video Streaming Using CUDA," *Journal of Electrical and Computer Engineering*, vol. 2014, Article ID 183716, 12 pages, 2014. https://doi.org/10.1155/2014/183716.

[5] B. Martin, W. Hamidouche, J. Le Feuvre, and M. Raulet, "Unified real time software decoder for HEVC extensions," *IEEE International Conference on Image Processing (ICIP)*, Paris, 2014, pp. 2186-2188. doi: 10.1109/ICIP.2014.7025442.

[6] HEVC: Cutting Through The Hype, A realistic evaluation of what HEVC means for your business: today, tomorrow and in 2020, Avni Rambhia, Industry Manager, Digital Media.

[7] R. Garcia and H. Kalva, "Subjective evaluation of HEVC and AVC/H.264 in mobile environments," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 1, pp. 116-123, February 2014. doi: 10.1109/TCE.2014.6780933.

[8] J. Ohm and G. J. Sullivan, "High Efficiency Video Coding: The next frontier in video compression," *IEEE Signal Processing Magazine*, Jan 2013.

[9] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard", *IEEE Trans. Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[10] R. Green and A. Behman, "The benefits of FPGA,", broadcastengineeringworld.com, May 2012.

[11] D. Engelhardt, J. Moller, J. Hahlbeck, and B. Stabernack, "FPGA implementation of a full HD real-time HEVC main profile decoder," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 3, pp. 476-484, Aug. 2014. doi: 10.1109/TCE.2014.6937333.

[12] High Efficiency Video Coding (HEVC): https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/trunk

[13] A. M. Abdelsalam, A. Shalaby, and M. S. Sayed, "Towards an FPGA-Based HEVC Encoder: A Low-Complexity Rate Distortion Scheme for AMVP," *Circuits Syst Signal Process* (2017) 36:4207–4226.

[14] H. Azgin, A. C. Mert, E. Kalali, and I. Hamzaoglu, "An efficient FPGA implementation of HEVC intra prediction," *IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2018, pp. 1-5. doi: 10.1109/ICCE.2018.8326332.

[15] F. A. Ghani, E. Kalali, and I. Hamzaoglu, "FPGA implementations of HEVC sub-pixel interpolation using high-level synthesis," *International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Istanbul, 2016, pp. 1-4. doi: 10.1109/DTIS.2016.7483883.

Annex ‘A’

Changes to the HEVC Decoder

Classes	**Uninitialized Member Variables**
InputNALUnit	m_Bitstream
InputByteStream	m_Input, m_FutureBytes, m_NumFutureBytes

AnnexBStats	m_numLeadingZero8BitsBytes
	m_numZeroByteBytes;
	m_numStartCodePrefixBytes;
	m_numBytesInNALUnit;
	m_numTrailingZero8BitsBytes;

TAppDecTop	m_acTDecTop, m_apcTDecTop, m_acTVideoIOYuvReconFile, m_aiPOCLastDisplay
TAppDecCfg	m_pchBitstreamFile m_iSkipFrame m_outputBitDepthY m_outputBitDepthC m_iMaxTemporalLayer m_decodedPictureHashSEIEnabled m_colourRemapSEIEnabled m_tgtLayerId m_pchBLReconFile

TDecTop	m_pocRandomAccess
	m_cListPic
	m_parameterSetManagerDecoder; // m_apcSlicePilot;
	m_SEIs;
	m_cGopDecoder
	m_ m_cGopDecoder
	m_cLoopFilter
	m_bFirstSliceInPicture
	m_layerId
	m_pocDecrementedInDPBFlag
	m_pBLReconFile

NALUnit	temporalId layerId
TComReferencePictureSet	m_numberOfPictures m_numberOfNegativePictures m_numberOfPositivePictures m_numberOfLongtermPictures m_deltaPOC m_POC m_used m_interRPSPrediction; m_deltaRIdxMinus1; m_deltaRPS; m_numRefIdc; m_refIdc m_bCheckLTMSB

TComYuv	m_apiBufY;
	m_apiBufU;
	m_apiBufV;

TComPic	m_uiTLayer m_bUsedByCurr m_bIsLongTerm m_apcPicSym m_apcPicYuv m_pcPicYuvPred m_pcPicYuvResi m_bReconstructed;

	m_bNeededForOutput;
	m_uiCurrSliceIdx
	m_bCheckLTMSB;

TComPicYuv	m_apiPicBufY
	m_apiPicBufU;
	m_apiPicBufV;
	m_piPicOrgY
	m_piPicOrgU;
	m_piPicOrgV;