2016 11th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

FPGA Implementations of HEVC Sub-Pixel Interpolation Using High-Level Synthesis

Firas Abdul Ghani, Ercan Kalali, Ilker Hamzaoglu
Faculty of Engineering and Natural Sciences, Sabanci University
34956 Tuzla, Istanbul, Turkey
{firas, ercankalali, hamzaoglu}@sabanciuniv.edu

Abstract—**Sub-pixel interpolation is one of the most computationally intensive parts of High Efficiency Video Coding (HEVC) video encoder and decoder. High-level synthesis (HLS) tools are started to be successfully used for FPGA implementations of digital signal processing algorithms. Therefore, in this paper, the first FPGA implementation of HEVC sub-pixel (half-pixel and quarter-pixel) interpolation algorithm using a HLS tool in the literature is proposed. The proposed HEVC sub-pixel interpolation hardware is implemented on Xilinx FPGAs using Xilinx Vivado HLS tool. It, in the worst case, can process 45 quad full HD (3840x2160) video frames per second. Using HLS tool significantly reduced the FPGA development time. Therefore, HLS tools can be used for FPGA implementation of HEVC video encoder.**

Keywords—HEVC, Sub-Pixel Interpolation, HLS, FPGA.

I. INTRODUCTION

Joint collaborative team on video coding (JCT-VC) recently developed a new video compression standard called High Efficiency Video Coding (HEVC) [1]-[3]. HEVC provides 50% better coding efficiency than H.264 video compression standard. Sub-pixel (half-pixel and quarter-pixel) interpolation is one of the most computationally intensive parts of HEVC video encoder and decoder. On average, one fourth of the HEVC encoder complexity and 50% of the HEVC decoder complexity are caused by sub-pixel interpolation [4].

In H.264, a 6-tap FIR filter is used for half-pixel interpolation and linear interpolation is used for quarter-pixel interpolation [5]. In HEVC, three different 8-tap FIR filters are used for both half-pixel and quarter-pixel interpolations [6]. In H.264, 4x4 and 16x16 block sizes are used. In HEVC, prediction unit (PU) sizes can be from 4x4 to 64x64. Therefore, HEVC sub-pixel interpolation is more complex than H.264 sub-pixel interpolation.

In this paper, the first high-level synthesis (HLS) implementation of HEVC sub-pixel interpolation algorithm in the literature is proposed. The proposed HEVC sub-pixel interpolation hardware is implemented on Xilinx FPGAs using Xilinx Vivado HLS tool. HLS tools accept their inputs in different formats [7]. Xilinx Vivado HLS tool takes C or C++ codes as input, and generates Verilog or VHDL codes. The C codes given as input to Xilinx Vivado HLS tool are developed based on the HEVC sub-pixel interpolation software implementation in the HEVC reference software video encoder (HM) version 15 [8]. Three HEVC sub-pixel interpolation HLS implementations are done. In the first one (MM), in the C codes, multiplications with constants are implemented using multiplication operations. In the second one (MAS), multiplications with constants are implemented using addition and shift operations. In the last one (MMCM), addition and shift operations are implemented using Hcub multiplierless constant multiplication algorithm [9].

Verilog RTL codes generated by Xilinx Vivado HLS tool for the three HEVC sub-pixel interpolation HLS implementations are verified to work in a Xilinx Virtex 6 FPGA. The implementation results show that the proposed HEVC sub-pixel interpolation FPGA implementation, in the worst case, can process 45 quad full HD (3840x2160) video frames per second with acceptable hardware area. Using HLS tool significantly reduced the FPGA development time. Therefore, HLS tools can be used for FPGA implementation of HEVC video encoder.

A few HLS implementations for HEVC video compression standard are proposed in the literature [10], [11]. A few HLS implementations for H.264 video compression standard are proposed in the literature [12]. In Section III, the HEVC sub-pixel interpolation HLS implementation proposed in this paper is compared with the handwritten HEVC sub-pixel interpolation hardware implementations proposed in the literature [13]-[16].

II. HEVC SUB-PIXEL INTERPOLATION ALGORITHM

Integer pixels ($A_{x,y}$), half pixels and quarter pixels in a PU are shown in Fig. 1. Type A, type B and type C 8-tap FIR filters used for sub-pixel interpolation are shown in (1), (2), and (3), respectively. The shift1 value is determined based on bit depth of the pixel. The half pixels a, b, c are interpolated from nearest integer pixels in horizontal direction using type A, type B and type C filters, respectively. The half-pixels d, h, n are interpolated from nearest integer pixels in vertical direction using type A, type B and type C filters, respectively. The quarter pixels e, f, g are interpolated from the nearest a, b, c half pixels respectively in vertical direction using type A filter. The quarter pixels i, j, k are interpolated similarly using type B filter. The quarter pixels p, q, r are interpolated similarly using type C filter.

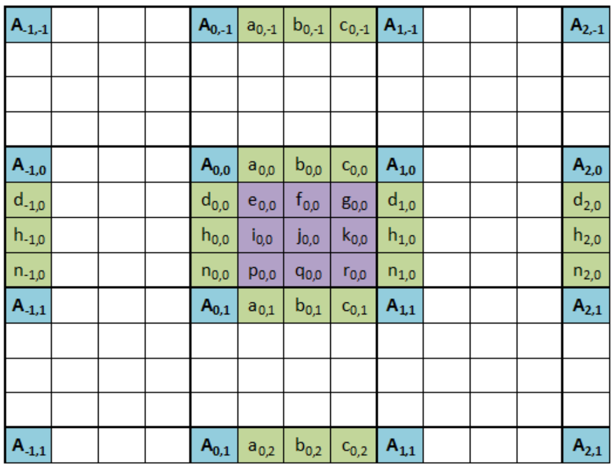

Fig. 1. Integer, Half and Quarter Pixels

$$a_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 10 * A_{-1,0} + 58 * A_{0,0} + 17 * A_{1,0} - 5 * A_{2,0} + A_{3,0}) \gg shift1 \quad (1)$$

$$b_{0,0} = (-A_{-3,0} + 4 * A_{-2,0} - 11 * A_{-1,0} + 40 * A_{0,0} + 40 * A_{1,0} - 11 * A_{2,0} + 4 * A_{3,0} - A_{4,0}) \gg shift1 \quad (2)$$

$$c_{0,0} = (-A_{-2,0} - 5 * A_{-1,0} + 17 * A_{0,0} + 58 * A_{1,0} - 10 * A_{2,0} + 4 * A_{3,0} - A_{4,0}) \gg shift1 \quad (3)$$

III. HEVC Sub-Pixel Interpolation HLS Implementations

The proposed HLS implementation of HEVC sub-pixel interpolation is shown in Fig. 2. The proposed HLS implementation is synthesized to Verilog RTL using Xilinx Vivado HLS tool. The C codes given as input to Xilinx Vivado HLS tool are developed based on the HEVC sub-pixel interpolation software implementation in the HEVC reference software video encoder (HM) version 15 [8].

In the proposed HLS implementation, half pixels and quarter pixels for an 8x8 PU are interpolated using 15x15 integer pixels. Half pixels and quarter pixels for larger PU sizes can be interpolated by interpolating the half pixels and quarter pixels for each 8x8 part of a PU separately. In the C codes, 15 integer pixels are taken as input in each clock cycle. 8 a, 8 b and 8 c half-pixels are interpolated in parallel in each clock cycle. 15x8 a, 15x8 b, and 15x8 c half pixels are interpolated in 15 clock cycles, and they are stored into registers for quarter pixel interpolation. In the same 15 clock cycles, 15x8 integer pixels are also stored into registers for interpolating d, h, n half pixels. Then, 8x8 d, 8x8 h, 8x8 n half pixels are interpolated using 15x8 integer pixels. Finally, all quarter pixels (e, f, g, i, j, k, p, q, r) are interpolated using 15x8 a, 15x8 b, and 15x8 c half pixels.

Three HEVC sub-pixel interpolation HLS implementations are done. In the first one (MM), in the C codes, multiplications with constants are implemented using multiplication operations. In the second one (MAS), multiplications with constants are implemented using addition and shift operations. In the last one (MMCM), addition and shift operations are implemented using Hcub multiplierless constant multiplication algorithm [9].

Verilog RTL codes generated by Xilinx Vivado HLS tool for these three HLS implementations are verified with RTL simulations. RTL simulation results matched the results of HEVC sub-pixel interpolation software implementation in the HEVC reference software video encoder (HM) version 15 [8]. The Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX550T FF1760 FPGA with speed grade 2 using Xilinx ISE 14.7. The FPGA implementations are verified with post place and route simulations.

We used several optimizations offered by Xilinx Vivado HLS tool to increase the performance and decrease the area of the proposed HLS implementations [17]. We tried to use loop unrolling directive. However, loop unrolling directive did not work correctly for the proposed HLS implementations. In [18], it is mentioned that loop unrolling may cause memory access problems in HLS designs, and current generation of HLS tools may ignore these problems. As shown in Table I, the performance of the HLS implementation, which implements multiplications with constants using multiplication operations, without loop unrolling is very low. Therefore, we performed manual loop unrolling in the proposed HLS implementations to increase their performances.

Allocation (ALC) directive is used to specify the maximum number of resources that can be used in hardware. It forces the HLS tool to perform resource sharing. It therefore decreases the hardware area. In the proposed HLS implementations, ALC is used for subtraction, addition, multiplication, and shifting operations. Pipeline (PIPE) directive performs pipelining to increase the performance. It is used in the proposed HLS implementations.

Resource (RES) directive is used to specify which resource will be used to implement a variable such as an array, arithmetic operation or function argument. In the proposed HLS implementations, it is used to store input integer pixels into BRAMs. Array map (AMAP) directive is used to map multiple small arrays into a single large array. The large array can be targeted to a single large memory (RAM or FIFO) resource. It is also used to control how (horizontal or vertical) data is stored in BRAMs. In the proposed HLS implementations, it is used to control how data is stored in BRAMs so that the number of BRAMs used in the hardware is reduced as much as possible.

Array partition (APAR) directive partitions the large arrays into multiple smaller arrays or individual registers for parallel data accesses. In the proposed HLS implementations, it is used to partition the arrays that store a, b, and c half pixels to increase quarter pixel interpolation performance. Xilinx Vivado HLS tool provides a specific library for designing bit-accurate (BIT) models in C codes. In the proposed HLS implementations, bit accurate model is used to decrease adder bit widths and therefore hardware area.

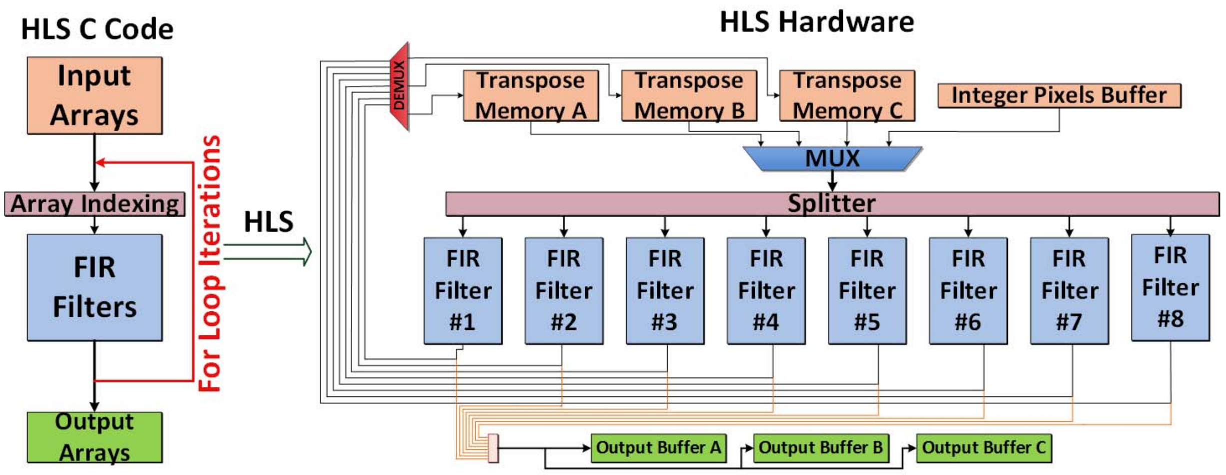

Fig. 2. HEVC Sub-Pixel Interpolation HLS Implementation

The FPGA implementation results for the first HLS implementation (MM) are given in Table II. In this HLS implementation, in the C codes, multiplications with constants are implemented using multiplication operations. These multiplication operations are mapped to DSP48 blocks in RTL synthesis. This decreased the number of LUTs and DFFs used in the hardware. Allocation (ALC), pipeline (PIPE), resource (RES) and array map (AMAP) directives are used in this HLS implementation. In the table, M shows the number of multipliers used in the ALC directive.

The FPGA implementation results for the second HLS implementation (MAS) are given in Table III. In this HLS implementation, multiplications with constants are implemented using addition and shift operations. This HLS implementation does not use any DSP48 blocks, but it uses more LUTs and DFFs than MM. It also has higher performance than MM. Pipeline (PIPE), resource (RES) and array map (AMAP) directives are used in this HLS implementation.

The FPGA implementation results for the last HLS implementation (MMCM) are given in Table IV. In this HLS implementation, addition and shift operations are implemented using Hcub multiplierless constant multiplication (MCM) algorithm [9]. Common sub-expressions are calculated in different equations and same integer pixel is multiplied with different constant coefficients in different equations. Therefore, in this HLS implementation, common sub-expressions in different FIR filter equations are calculated once, and the result is used in all the equations. This HLS implementation also uses Hcub MCM algorithm in order to reduce number and size of the adders, and to minimize the adder tree depth [9]. Hcub algorithm tries to minimize number of adders, their bit size and adder tree depth in a multiplier block, which multiplies a single input with multiple constants. This HLS implementation has the best performance with acceptable hardware area. Allocation (ALC), pipeline (PIPE), array partition (APAR) directives and bit-accurate (BIT) model are used in this HLS implementation. In the table, A and S show the number of adders and subtractors used in the ALC directive, respectively.

The best HEVC sub-pixel interpolation HLS implementation proposed in this paper (MMCM with ALC(A500_S500)_APAR_PIPE_BIT) is compared with the handwritten HEVC sub-pixel interpolation hardware implementations proposed in the literature [13]-[16]. The comparison results are shown in Table V.

The proposed MMCM HLS implementation is similar to the handwritten HEVC sub-pixel interpolation hardware implementation proposed in [13]. In [13], common sub-expressions in different FIR filter equations are calculated once, and the result is used in all the equations. Also, addition and shift operations are implemented using Hcub MCM algorithm. In [13], the handwritten Verilog RTL codes are synthesized and mapped to a Xilinx XC6VLX130T FF1156 FPGA with speed grade 3. In this paper, the handwritten Verilog RTL codes proposed in [13] are synthesized and mapped to a Xilinx XC6VLX550T FF1760 FPGA with speed grade 2 for fair comparison with the proposed MMCM HLS implementation. The proposed MMCM HLS implementation has higher performance than the handwritten HEVC sub-pixel interpolation hardware implementation proposed in [13] at the expense of larger area.

Since the handwritten HEVC sub-pixel interpolation hardware implementation proposed in [14] is designed only for motion compensation (MC), it has higher performance and lower area than the proposed MMCM HLS implementation. The handwritten HEVC sub-pixel interpolation hardware implementation proposed in [15] has lower performance and therefore lower area than the proposed MMCM HLS implementation. In addition, it requires higher clock frequency to achieve real time performance. The handwritten HEVC sub-pixel interpolation hardware implementation proposed in [16] has both lower performance and larger area than the proposed MMCM HLS implementation.

TABLE I. HLS IMPLEMENTATION WITHOUT MANUAL LOOP UNROLLING WITH MULTIPLIERS RESULTS

Optimizations	Slice	LUT	DFF	BRAM	DSP48	Freq. (MHz)	Clock Cycles (8x8 PU)	Fps
NOOPT	885	2565	1411	1	15	250	1921	1 3840x2160

TABLE II. HLS IMPLEMENTATION WITH MULTIPLIERS RESULTS

Optimizations	Slice	LUT	DFF	BRAM	DSP48	Freq. (MHz)	Clock Cycles (8x8 PU)	Fps
NOOPT	4623	14110	7526	0	113	200	156	10 3840x2160
ALC(M128)	4769	14133	6226	0	135	168	148	9 3840x2160
PIPE	4938	14086	8736	0	113	201	56	28 3840x2160
RES(BRAM)	4723	13883	7395	4	113	201	156	10 3840x2160
ALC(M128)_RES(BRAM)_PIPE	5197	14366	8000	4	147	167	56	23 3840x2160
ALC(M128)_AMAP(4)_RES(BRAM)_PIPE	4299	12401	7964	2	147	167	56	23 3840x2160
ALC(M20)_AMAP(4)_RES(BRAM)_PIPE	4299	13100	8037	2	59	168	56	23 3840x2160

TABLE III. HLS IMPLEMENTATION WITH ADDERS AND SHIFTERS RESULTS

Optimizations	Slice	LUT	DFF	BRAM	DSP48	Freq. (MHz)	Clock Cycles (8x8 PU)	Fps
NOOPT	4809	15629	9095	0	0	202	133	12 3840x2160
AMAP(4)_RES(BRAM)_PIPE	4891	15716	9436	2	0	200	55	28 3840x2160

TABLE IV. HLS IMPLEMENTATION WITH MCM RESULTS

Optimizations	Slice	LUT	DFF	BRAM	DSP48	Freq. (MHz)	Clock Cycles (8x8 PU)	Fps
NOOPT	4850	15632	6673	2	0	201	195	8 3840x2160
ALC(A500_S500)_APAR_PIPE	5288	14619	10118	0	0	168	29	45 3840x2160
ALC(A500_S500)_APAR_PIPE_BIT	4426	14225	9984	0	0	168	29	45 3840x2160

TABLE V. HEVC SUB-PIXEL INTERPOLATION HARDWARE COMPARISON

	[13]	[14]	[15]	[16]	Proposed (MMCM)
Tech.	Xilinx Virtex 6	90 nm	150 nm	90 nm	Xilinx Virtex 6
Gate/Slice Count	1597	32.5 K	30.2 K	224 K	4426
Freq. (MHz)	200	171	312	333	168
Fps	30 QFHD	60 QFHD	30 QFHD	30 FHD	45 QFHD
Design	ME + MC	MC	ME + MC	ME + MC	ME + MC

ACKNOWLEDGMENT

This research was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) under the contract 115E290.

REFERENCES

[1] High Efficiency Video Coding, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013.

[2] E. Kalali, Y. Adibelli, I. Hamzaoglu, “A High Performance and Low Energy Intra Prediction Hardware for High Efficiency Video Coding”, International Conference on Field Programmable Logic and Applications, Aug. 2012.

[3] E. Kalali, E. Ozcan, O. M. Yalcinkaya, I. Hamzaoglu, “A low energy HEVC inverse transform hardware,” IEEE Transactions on Consumer Electronics, vol. 60, no. 4, pp. 754-761, Nov. 2014.

[4] J. Vanne, M. Viitanen, T.D. Hämäläinen, A. Hallapuro, “Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs”, IEEE Trans. on Circuits and Systems for Video Technology, vol. 22, no. 12, pp.1885-1898, Dec. 2012.

[5] S. Yalcin, I. Hamzaoglu, “A High Performance Hardware Architecture for Half-Pixel Accurate H.264 Motion Estimation”, 14th IFIP Int. Conference on VLSI-SoC, Oct. 2006.

[6] E. Kalali, Y. Adibelli, I. Hamzaoglu, “A Reconfigurable HEVC Sub-Pixel Interpolation Hardware”, IEEE Int. Conference on Consumer Electronics - Berlin, Sept. 2013.

[7] W. Meeus, K. V. Beeck, T. Goedeme, J. Meel, D. Stroobandt, “An overview of today's high-level synthesis tools,” Springer Design Automation for Embedded Systems, vol. 16, no. 3, pp. 31-51, Sept. 2012.

[8] K. McCann, B. Bross, W.J. Han, I.K. Kim, K. Sugimoto, G. J. Sullivan, “High Efficiency Video Coding (HEVC) Test Model (HM) 15 Encoder Description”, JCTVC-Q1002, June 2014.

[9] Y. Voronenko, M. Püschel, "Multiplierless Constant Multiple Multiplication", ACM Trans. on Algorithms, vol. 3, no. 2, May 2007.

[10] E. Kalali, I. Hamzaoglu, “FPGA Implementations of HEVC Inverse DCT Using High-Level Synthesis,” Conf. on Design and Architectures for Signal and Image Processing (DASIP), pp. 1-6, Sept. 2015.

[11] P. Sjovall, J. Virtanen, J. Vanne, T. D. Hamalainen,“High-Level Synthesis Design Flow for HEVC Intra Encoder on SoC FPGA,” Euromicro Conf. on Digital System Design, pp. 49-56, Aug. 2015.

[12] S. Kim, H. Kim, T. Chung, J-G. Kim, “Design of H.264 video encoder with C to RTL design tool,” Int. SoC Design Conference, pp. 171-174, Nov. 2012.

[13] E. Kalali, I. Hamzaoglu, “A low energy HEVC sub-pixel interpolation hardware,” IEEE Int. Conference on Image Processing, pp. 1218-1222, Oct. 2014.

[14] Z. Guo, D. Zhou, S. Goto, “An Optimized MC Interpolation Architecture for HEVC”, IEEE Int. Conf. on Acoustics, Speech and Signal Processing, March 2012.

[15] C. M. Diniz, M. Shafique, S. Bampi, J. Henkel, “High-Throughput Interpolation Hardware Architecture with Coarse-Grained Reconfigurable Datapaths for HEVC”, IEEE Int. Conference on Image Processing, Sept. 2013.

[16] G. Pastuszak, M. Trochimiuk, “Architecture Design and Efficiency Evaluation for the High-Throughput Interpolation in the HEVC Encoder”, Euromicro Conference on Digital System Design, Sept. 2013.

[17] UG902, “Vivado Design Suite User Guide: High-Level Synthesis,” May 2014.

[18] P. Coussy, A. Morawiec, High-Level Synthesis from Algorithm to Digital Circuit, Springer, 2008.