POLITECNICO DI TORINO
Repository ISTITUZIONALE

High Level Synthesis based FPGA Implementation of H.264/AVC Sub-Pixel Luma Interpolation Filters

(Article begins on next page)

14 July 2024

High Level Synthesis based FPGA Implementation of H.264/AVC Sub-Pixel Luma Interpolation Filters

Waqar Ahmad, Javed Iqbal, Maurizio Martina, Guido Masera
Department of Electronics and Telecommunication
Politecnico Di Torino, Turin, Italy
waqar.ahmad@polito.it, javed_iqbal@polito.it, maurizio.martina@polito.it, guido.masera@polito.it

Abstract—In High Efficiency Video Coding (HEVC) and H.264/AVC video coding standards, Interpolation filtering used for sub-pixel interpolation is one of the most computational intensive parts of the standards. Video processing systems are becoming more complex thus decreasing the productivity of the hardware designers and the software programmers, producing design productivity gap. To fill this productivity gap, hardware and software fields are bridged through High Level Synthesis (HLS), thus improving the productivity of the hardware designers. In this paper, we present a HLS based FPGA Implementation of sub-pixel Luma Interpolation of H.264/AVC. Xilinx Vivado HLS tools are used for the FPGA implementation of interpolation filtering on Xilinx xc7z020clg481-1 device. Our design can achieve the frame processing speed of 41 QFHD, i.e. 3840x2160@41fps. The development time is significantly decreased by the HLS tools.

Keywords–H.265; H.264/AVC; HLS; Interpolation; Filters; FPGA;

I. INTRODUCTION

Significant storage is needed for uncompressed digital videos. Digital video is handled by high compression and efficient video coding standards, such as H.264/AVC and H.265/HEVC. The temporal redundancy present in the video signal is exploited by the process of Motion Compensated Prediction(MCP). MCP, reduces the amount of data to be sent to the decoder [1]. We can get rid of large amount of video data by temporal motion prediction. Current block/object location is compared with the previous frame to measure if there exist the same block/object. Hence, reducing the amount of data required to transmit to the video decoder. In MCP, to process the current frame, the similar data/object of the current frame and the previous frame are measured first by the video encoder. For this purpose the frame is divided into blocks of pixels. MCP sends the motion vector as side information to tell the decoder about the similarity between the current frame and the previous frame for prediction. Prediction error is also sent along with the motion vector, for new frame reconstruction. The objects in the consecutive video frames may differ by fractional position i.e. these displacements are continuous. These objects are independent of the sampling grid of the digital video sequence. Fractional motion vector accuracy make the video encoder efficient and reduce the prediction

error [2]. Interpolation filter are used for fractional value motion vector. The design of the interpolation is carried out by keeping in view the important factors such as visual quality, coding efficiency and implementation complexity [3]. H.264/AVC and HEVC video coding standards, support half and quarter pixel accuracy. Interpolation filtering used for sub-pixel interpolation is one of the most computational intensive parts of H.264/AVC and HEVC. Computational complexity of the interpolation filters is about 20% and 25% of total time in 3D-HEVC encoder and decoder, respectively, as reported in our previous work [4]. In industry and academia, HLS is being studied for many years and there exist many operational projects [5]. In maximum devices the H.264/AVC is still being used, as it is the marketdominant video coding standard. In new efficient devices the H.264/AVC is being replaced by state-of-the-art video standard i.e. High Efficiency Video Coding (HEVC). This standard migration will occur steadily because HEVC is only supported by the latest models and H.264/AVC is already present in the most devices. Interest for efficient implementation of H.264/AVC is still desirable because both coding standards will coincide in the market sideways a number of years. In this article, HLS based FPGA implementation of H.264/AVC sub-pixel luma interpolation is presented. Xilinx Vivado HLS tools are used for FPGA implementation of H.264/AVC luma sub-pixel interpolation. HLS has some specific benefits over the conventional RTL based VLSI design. One key benefit is its power to render micro-architectures with specific area vs. performance trade-offs for the same behavioral description by surroundings different synthesis choice [6]. To the best of our knowledge, in the literature, there is no prior work on HLS based FPGA implementation of H.264/AVC sub-pixel interpolation. In [7], HLS based FPGA implementation of HEVC sub-pixel luma interpolation is described. A comparison between the HLS based FPGA implementation of sub-pixel luma interpolation of H.264/AVC and HEVC is also carried out in this article.

This article is organized as follows. Section II describes interpolation filtering process of H.264/AVC video coding standard. Section III presents the HLS based FPGA implementation of the sub-pixel interpolation. Section IV gives

$A_{0,0}$	$a_{0,0}$	$b_{0,0}$	$c_{0,0}$	$A_{1,0}$
$d_{0,0}$	$e_{0,0}$	$f_{0,0}$	$g_{0,0}$	$d_{1,0}$
$h_{0,0}$	$i_{0,0}$	$j_{0,0}$	$k_{0,0}$	$h_{1,0}$
$n_{0,0}$	$p_{0,0}$	$q_{0,0}$	$r_{0,0}$	$n_{1,0}$
$A_{0,1}$	$a_{0,1}$	$b_{0,1}$	$c_{0,1}$	$A_{1,1}$

Figure 1. Pixel positions for Integer, Luma half and Luma quarter pixels.

the comparison between the HLS based sub-pixel luma interpolation of HEVC and H.264/AVC. Finally, Section V gives the conclusion summary and future work.

II. H.264/AVC SUB-PIXEL INTERPOLATION

For 4:2:0 color format video in H.264/AVC, luma samples supports the quarter-pel accuracy and chroma samples supports one-eight pel accuracy of the motion vectors [8]. Motion vector may points to integer and/or fractional samples position. In the latter case, the fractional pixel are generated by interpolation. A one-dimensional 6-tap FIR filter is used for prediction signals at the half-sample value, in vertical and horizontal directions. Average of the sample values at full and half-pixel are used for the quarter sample values generation of the prediction signal.

The luma sub-pixel interpolation process in H.264/AVC is shown in Fig. 1. The half pixel values $b_{0,0}$ and $h_{0,0}$ are obtained by applying the 6-tap filter in the horizontal and vertical directions, respectively, as follows:

$$b_{0,0} = (A_{-2,0} - 5*A_{-1,0} + 20*A_{0,0} + 20*A_{1,0} \\ -5*A_{2,0} + A_{3,0} + 16) >> 5 \quad (1)$$

$$h_{0,0} = (A_{0,-2} - 5*A_{0,-1} + 20*A_{0,0} + 20*A_{0,1} \\ -5*A_{0,2} + A_{0,3} + 16) >> 5 \quad (2)$$

where $A_{n,0}, A_{0,n}$ with values of $n = -2, -1, 0, 1, 2, 3$, are integer pixels in horizontal and vertical directions, respectively. Intermediate half-pel samples $b'n$ or $h'n$ are used for the calculation of half pixel value $j_{0,0}$, by applying the 6-tap filter in the vertical or horizontal directions, as follows:

$$b'_n = b'_{n,-2} - 5*b'_{n,-1} + 20*b'_{n,0} + 20*b'_{n,1} \\ -5*b'_{n,2} + b'_{n,3} \quad (3)$$

$$j_{0,0} = (b'_n + 512) >> 10 \quad (4)$$

where $n = -2, -1, 0, 1, 2, 3$ and $b' = b << 5 - 16$, i.e. we can use the values of b. We can obtain the values of $j_{0,0}$ alternatively, as given by (5) and (6).
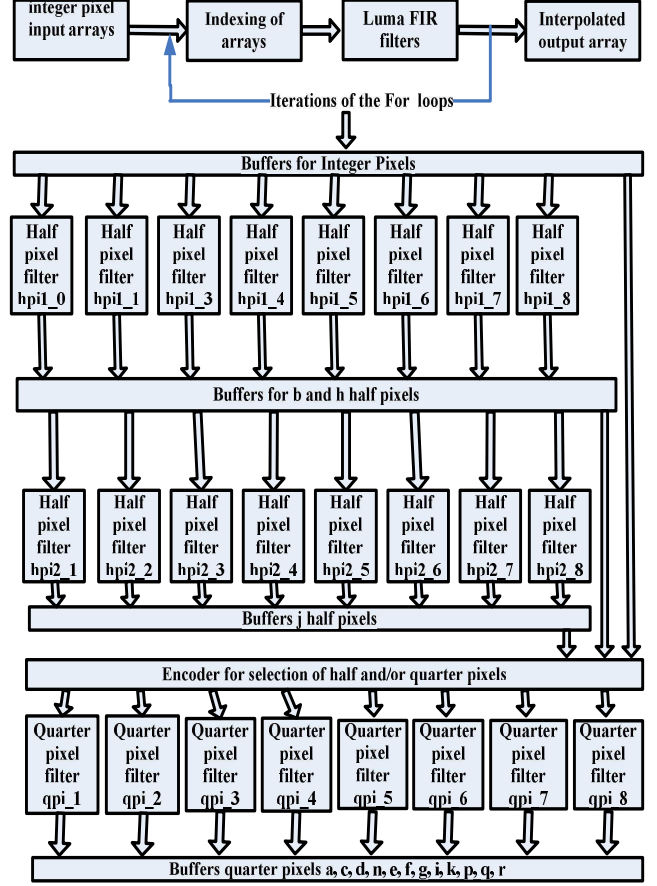

Figure 2. HLS implementation of H.264/AVC Luma Sub-pixel.

$$h'_n = A_{-2,0} - 5*A_{-1,0} + 20*A_{0,0} + 20*A_{1,1} \\ -5*A_{2,0} + A_{3,0} \quad (5)$$

$$j_{0,0} = (h'_{n,-2} - 5*h'_{n,-1} + 20*h'_{n,0} + 20*h'_{n,1} \\ -5*h'_{n,2} + h'_{n,3} + 512) >> 10 \quad (6)$$

Nearest, half pixel and/or integer pixel averaging is used for the calculation of the quarter-pixel sample. The samples used in the averaging could be both half-pel and a combination of the half-pel and integer-pel samples.

As an example, the following equations show the method to calculate quarter-pixel samples for some of the quarter-pixel positions i.e. $a_{0,0}, f_{0,0}$ and $e_{0,0}$ out of $a_{0,0}, c_{0,0}, d_{0,0}, n_{0,0}, f_{0,0}, i_{0,0}, k_{0,0}, q_{0,0}, e_{0,0}, g_{0,0}, p_{0,0}$ and $r_{0,0}$:

$$a_{0,0} = (A_{0,0} + b_{0,0} + 1) >> 1 \quad (7)$$

$$f_{0,0} = (b_{0,0} + j_{0,0} + 1) >> 1 \quad (8)$$

$$e_{0,0} = (b_{0,0} + h_{0,0} + 1) >> 1 \quad (9)$$

TABLE I
RESOURCES REQUIRED FOR HLS IMPLEMENTATION OF H.264/AVC LUMA SUB-PIXEL INTERPOLATION USING MULTIPLIERS FOR MULTPLICATION.

Opt.	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	706	1188	430	128	1489	0.5
LOOP UNROLL	0	0	3011	5084	1670	110	577	1.5
LOOP UNROLL + ARRAY PARTITION	0	0	3451	8653	2655	112	473	2
PIPELINE + ARRAY PARTITION	0	0	10224	27995	8817	102	19	41

TABLE II
RESOURCES REQUIRED FOR HLS IMPLEMENTATION OF H.264/AVC LUMA SUB-PIXEL INTERPOLATION USING ADD AND SHIFT OPEATIONS FOR MULTPLICATION.

Opt.	BRAM18K	DSP48E	FF	LUT	SLICE	Freq. (MHz)	Clock Cycles	Fps
NO OPTIMIZATION	0	0	302	413	110	129	1432	1
LOOP UNROLL	0	0	2304	3033	422	212	577	3
LOOP UNROLL + ARRAY PARTITION	0	0	2843	4056	748	210	449	3
PIPELINE + ARRAY PARTITION	0	0	11001	12774	2606	102	19	41

III. HLS BASED FPGA IMPLEMENTATION

In Fig. 2, the proposed HLS implementation of H.264/AVC sub-pixel interpolation is shown. In our proposed design, 13x13 integer pixels are used for the half and quarter pixel interpolation of the 8x8 PU. For the larger PU sizes, half and quarter pixel can be interpolated using each 8x8 PU part of the larger block i.e. dividing the larger block in PU sizes of 8x8. 13 integer pixesl are given as input to the first half pixel interpolator array $hpi1$ in each clock cycle. 8 half pixels $b_{0,0}$ are interpolated in parallel in each clock cycle, so in total it will interpolate 13x8 half pixels in 13 clock cycles. These half pixels are stored into registers for interpolating the half pixesl $j_{0,0}$ or quarter pixels $a_{0,0}$ and $c_{0,0}$. During the interpolation of $b_{0,0}$ half pixels interpolation, 13x13 integer pixels are stored for the half pixel interpolation of the $h_{0,0}$. Then the $h_{0,0}$ half pixels are interpolated using these stored 13x13 integer pixels using $hpi1$, meanwhile, in parallel the $j_{0,0}$ half pixel are interpolated using $hpi2$ from the already available intermediate $b_{0,0}$ half pixels. The half pixels $h_{0,0}$ and $j_{0,0}$ are also stored in the registers for the quarter pixel interpolation. Finally all the $a_{0,0}, c_{0,0}, d_{0,0}, n_{0,0}, f_{0,0}, i_{0,0}, k_{0,0}, \quad q_{0,0}, e_{0,0}, g_{0,0}, p_{0,0}$ and $r_{0,0}$ quarter pixels are generated using the already computed registered half pixels $b_{0,0}, h_{0,0}, j_{0,0}$ and the 13x13 integer pixels.

Vivado HLS tools are used for the FPGA implementation of the design. The HLS implementation is synthesized to verilog RTL. Xilinx Vivado HLS tools take $C, C++$ or $SystemC$ codes as input. In our case the C code is applied as input to the vivado HLS tool. The C code is written according to the H.264/AVC reference software video encoder. Vivado HLS provides various optimization techniques called as optimization directives or pragmas. Many variants of the HLS implementation of H.264/AVC luma sub-pixel interpolation are possible depending on the area vs performance trade-off requirements. Design Space Exploration (DSE) of the H.264/AVC luma sub-pixel interpolation is carried out using these optimization directives.

A. Discussion on Results

Vivado HLS kepts the loops as rolled by default. Loops are considered and operated as single sequence of operations defined within the body of the loop. All operations of the loops defined in the body of the loops are synthesized as hardware. So, all iterations of the loops use the same common hardware. Loop UNROLL directive available in the Vivado HLS, unrolls the loops partially or fully, depending on the appication requirements. If the application is performance critical, then the loop UNROLL directive can be used to unroll the loops for better optimized hardware in terms of performance by parallel processing, but it will increase the area e.g. if the loops are fully unrolled then the multiple copies of the same hardware will be synthesized. The other directive which we used in our design is PIPELINE. Pipeline directive can be applied to function or loop, it is basically the pipelining. The new inputs can be processed after every N clock cycles. Here the N is the Initiation Interval (II), the number of clock cycles after which the new inputs will be processed by the design.

When the pipeline directive is applied, it automatically unrolls all the loops within the scope of the pipeline region i.e. you do not need to apply loop UNROLL directive separately if the pipeline directive is already applied to the scope containing loop. For the parallel processing the data requirement must be satisfied. In our design the arrays are used as input to the HLS tools. Arrays are by default mapped to block RAMs in the vivado HLS i.e. you can only read or write or both read write at the same time if the block RAM is dual port. So, ARRAY PARTITION directive is used to partition the arrays into individual registers. It makes the data available for the parallel processing.

Two different implementations of the H.264/AVC luma sub-pixel interpolation is carried out using two different techniques for constant multiplication i.e. multiplication using multipliers, multiplication using add and shift operations.

Table I and II enlist the optimization directives used and the corresponding hardware resources required for HLS implementation using the multpliers as constant multplication and multiplication by shift and add operations. Mainly three directives are used for the efficient implementation of the H.264/AVC Luma interpolation designs. As shown in Table I and II , when there is NO OPTIMIZATION directive applied, the latency is much higher i.e. to process 8x8 PU it takes higher clock cycles as compared to the optimized ones. For the optimized design we use the combination of optimization directives such as LOOP UNROLL + ARRAY PARTITION and PIPELINE + ARRAY PARTITION. In both designs the application of optimizations shows significant area vs performance trade-off. In case of constant multplication using add and shift operations, we have better optimized design in terms of area and performance.

TABLE III
H.264/AVC Luma Sub-pixel HLS vs Manual RTL
Implementations.

	[9]	[10]	Proposed
Tech.	SMIC 130 nm	130 nm	Xilinx Virtex 7
Slice/Gate Count	75 K	67 K	2606
Freq. (MHz)	340	200	102
Fps	30 QFHD	2160p@30fps	41 QFHD
Design	ME	ME	ME + MC

TABLE IV
H.264/AVC vs HEVC Luma Sub-pixel HLS implementation.

	Proposed	[7]
Tech.	Xilinx Virtex 7	Xilinx Virtex 6
Slice/Gate Count	2606	4426
Freq. (MHz)	102	168
Fps	41 QFHD	45 QFHD
Design	ME + MC	ME + MC

IV. COMPARISON OF RESULTS

A. Comparison with Manual RTL implementation

Table III gives the comparison between HLS and manaul RTL implementations of H.264/AVC luma sub-pixel interpolation. It is evident that the HLS implementation is more efficient in terms of performance. Even though the other two implementations are VLSI based, we expect the same performance for the FPGA implementations of the corresponding implementations.

B. Comparison with HLS implementation of HEVC

Table IV gives the comparison between HLS implementations of H.264/AVC and HEVC luma sub-pixel interpolation. The proposed implementation takes less area as compared to the HLS implementation of HEVC because the HEVC uses larger interpolation filters and hence larger area. The throughput of the HEVC luma interpolation is also higher because the quarter pixel interpolation is independent of the half pixel interpolation e.g. $a_{0,0}, b_{0,0}, d_{0,0}$ and $h_{0,0}$.

V. CONCLUSION

HLS tools provide wider scope for Design Space Exploration (DSE) of complex systems through the use of optimization directives/pragmas. HLS based implementation of complex systems like HEVC and H.264/AVC video coding standards algorithms can be implemented as optimized as manual RTL implementation. The design time and time to market could be reduced significantly by the use of HLS tools. HLS tools give the flexibilty to easily implement the optimization techniques for efficient and optimtimized designs by the application of optimization directives or pragmas.

REFERENCES

[1] K. Uger, A. Alshin, E. Alshina, F. Bossen, W.J. Han, J.H. Park, and J. Lainema. Motion Compensated Prediction and Interpolation Filter Design in H.265/HEVC. *IEEE Journal of Selected Topics in Signal Processing*, 7:6, December 2013.

[2] B. Girod. Motion-Compensating Prediction with Fractional-Pel Accuracy. *IEEE Transaction on Communications*, 41:604–612, April 1993.

[3] T. Wedi. Motion Compensation in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:577–586, July 2003.

[4] W. Ahmad, M. Martina, and G. Masera. Complexity and Implementation Analysis of Synthesized View Distortion Estimation Architecture in 3D High Efficiency Video Coding. In *International Conference on 3D Imaging (IC3D)*. Liege, Belgium, 2015.

[5] Y. Chen, S. T. Gurumani, Y. Liang, G. Li, D. Guo, K. Rupnow, and D. Chen. FCUDA-NoC: A Scalable and Efficient Network-on-Chip Implementation for the CUDA-to-FPGA Flow. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24, June 2016.

[6] B. C. Schafer. Enabling High-Level Synthesis Resource Sharing Design Space Exploration in FPGAs through Automatic Internal Bitwidth Adjustments. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11, October 2015.

[7] F. A. Ghani, E. Kalili, and I. Hamzaoglu. FPGA Implementation of HEVC Sub-Pixel Interpolation Using High Level Synthesis. In *International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, volume April. Liege, Belgium, 2016.

[8] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13, July 2003.

[9] J. Liu, X. Chen, Y. Fan, and X. Zeng. A Full-mode FME VLSI Architecture Based on 8x8/4x4 Adaptive Hadamard Transform For QFHD H.264/AVC Encoder. In *IEEE/IFIP 19th International Conference on VLSI and System-on-Chip*, 2011.

[10] G. Pastuszak and M. Jakubowski. Optimization of the Adaptive Computationally-Scalable Motion Estimation and Compensation for the Hardware H.264/AVC Encoder. *Journal of Signal Processing Systems*, 82:391–402, March 2016.