An Efficient NEON-based Quarter-pel Interpolation Method for HEVC

Hao Lv[*†], Ronggang Wang[*1], Jie Wan[*], Huizhu Jia[†], Xiaodong Xie[†] and Wen Gao[†]
[*]School of Computer & Information Engineering, Peking University Shenzhen Graduate School, Shenzhen, China
E-mail: hlv@pku.edu.cn, rgwang@pkusz.edu.cn, wanjie@sz.pku.edu.cn
[†]National Engineering Laboratory of Video Technology, Peking University, Beijing, China
E-mail: hzjia@pku.edu.cn, donxie@pku.edu.cn, wgao@pku.edu.cn

***Abstract*—SIMD (Single Instruction Multiple Data) instructions have been widely used for digital signal processing and multimedia applications, especially video codec. This paper proposes the quarter-pel interpolation acceleration method of the HEVC (High Efficiency Video Coding), which is implemented with ARM SIMD instructions. Data level parallelism is utilized to use the SIMD capability of NEON effectively. Experiment results show that the implementation of the proposed method is approximately five times faster than that of the HEVC reference software for the HEVC quarter-pel interpolation operation.**

Key words: SIMD, NEON, HEVC, quarter-pel interpolation

I. INTRODUCTION

With the irresistible trend of high-definition video, the next generation video codecs will be expected to achieve at least 4k×2k Quad Full High Definition (QFHD) resolution for ultra-high definition. And now the next generation video coding standard HEVC, developed by the Joint Collaborative Team on Video Coding (JCT-VC), has received increased attention. A number of new algorithmic tools are proposed in HEVC, covering many aspects of video compression technology. One of the techniques to enhance the coding efficiency is the quarter-pel motion estimation and compensation, and the adopted interpolation filter is DCTIF (DCT-based interpolation filter) [1~3].

HEVC still adopts block-based hybrid video coding framework. The biggest block in HEVC is called LCU (largest coding unit), whose size ranges from 16×16 to 64×64, and LCU can be further divided into CU (coding unit), PU (prediction unit) and TU (transform unit). The concept of a macroblock as the basic processing unit in standardized video coding is called CTB (coding tree block), and the nested quadtree structure indicates how the blocks are further subdivided for the purpose of prediction and residual coding [4]. The basic unit for compression, termed CU, is a 2N×2N square block, and one CU can be recursively split into four smaller CUs until the predefined minimum size is reached. Each CU contains one or multiple PUs. According to the technique AMP (Asymmetric Motion Partitioning) [5], the square block CU can be split in one rectangle block of 3/4 of the square width or height, and another rectangle block of 1/4 of the square width or height. Hence the PU sizes can be 2N×2N, 2N×N, N×2N, N×N by symmetrically partitioning, and 2NxnU, 2NxnD, nLx2N, nRx2N by asymmetrically partitioning. So there are a large number of block sizes of PUs, and consequently the quarter-pel interpolation in HEVC becomes considerably complicated.

Nowadays with the development of multimedia application for mobile devices such as smart phone and tablet PC, video codec have become the indispensable part. In fact the data processing operations of video codec are mostly done at pixel level and similar operations are performed on each pixel in the entire block, so it can be efficiently performed for a group of pixels (e.g. 4, 8 or 16) in the packed form by using SIMD because SIMD instructions can process multiple packed data in parallel with a single operation. Most microprocessors available today support SIMD instructions to accelerate their application programs.

In this paper we focus on the design and implementation of the luma quarter-pel interpolation module of the HEVC video codec. To take advantage of the NEON SIMD architecture, the algorithms should be designed well to make SIMD instructions execute efficiently. The rest of the paper is organized in six sections. In Section II, we will describe the NEON SIMD architecture. In Section III, a brief introduction of the quarter-pel interpolation filter and interpolation process in HEVC will be given. The Section IV will present the acceleration method on quarter-pel interpolation in detail. The Section V will provide the acceleration results. At last, we give a brief conclusion in Section VI.

II. INSTRUCTION SYSTEM OF NEON

NEON technology is introduced in the ARMv7 architecture and optionally available only for the ARMv7-A and ARMv7-R architectures [6], designed to provide flexible and powerful acceleration for the low power mobile multimedia applications. NEON technology provides 128-bit wide vector operations, and allows SIMD computations to be performed on packed byte, word, doubleword and quadword integers. It has thirty-two 64-bit doubleword registers (D0-D31) and sixteen 128-bit quadword registers (Q1-Q15) which are composed of two consecutive doubleword registers. Registers

[1] The corresponding author.

are considered as vectors of elements of the same data type. Data types can be signed integer, unsigned integer, or integer of unspecified type with 8-bit, 16-bit, 32-bit, 64-bit wide, and in addition floating-point number and polynomial over {0,1} are also allowed. NEON supports multiple instructions such as addition, multiplication, rounding, shifting and saturation, which are essential to video codec implementation. Fig. 1 shows a typical SIMD computation process, and instructions perform the same operation in all lanes.

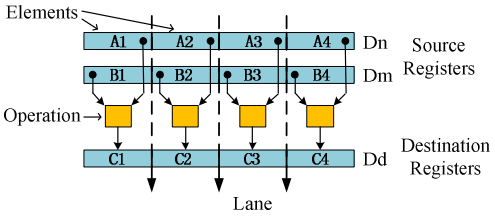

Fig. 1 NEON instructions processing.

III. QUARTER-PEL INTERPOLATION IN HEVC

In inter prediction, interpolation filter is applied to generate fractional-pel values. Current motion vector accuracy for luma component is quarter-pel, and 15 fractional-pel pixels will be interpolated as showed in Fig. 2. The DCTIF adopted in HEVC is a 2D separable interpolation filter. For fractional positions a, b and c, horizontal 1D filter is used. For fractional positions d, h and n, vertical 1D filter is used. For remaining positions, the interpolation process is separable, first horizontal 1D filter is applied for extended block and then vertical 1D filter is used.

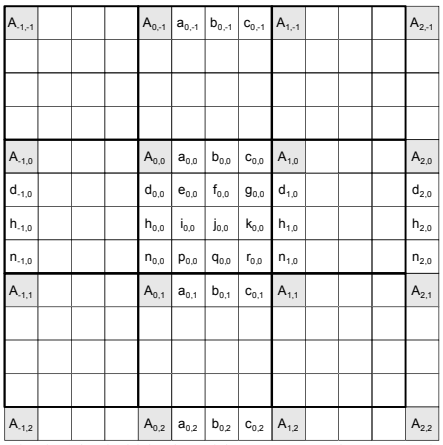

Fig. 2 Integer pixels (shaded blocks with upper-case letters) and fractional-pel pixels (un-shaded blocks with lower-case letters) for quarter-pel luma interpolation. [7]

For example, The fractional-pel pixels $a_{0,0}$, $b_{0,0}$ and $c_{0,0}$ shall be derived by applying the 8-tap filter in horizontal direction to the adjacent integer pixels as described by (1a)~(1c). The fractional-pel pixels $d_{0,0}$, $h_{0,0}$ and $n_{0,0}$ shall be derived by applying the 8-tap filter in vertical direction.

$$a_{0,0} = \left(-A_{-3,0} + 4\times A_{-2,0} - 10\times A_{-1,0} + 58\times A_{0,0} + 17\times A_{1,0} - 5\times A_{2,0} + A_{3,0}\right) >> shift1 \quad (1a)$$

$$b_{0,0} = \left(-A_{-3,0} + 4\times A_{-2,0} - 11\times A_{-1,0} + 40\times A_{0,0} + 40\times A_{1,0} - 11\times A_{2,0} + 4\times A_{3,0} - A_{4,0}\right) >> shift1 \quad (1b)$$

$$c_{0,0} = \left(A_{-2,0} - 5\times A_{-1,0} + 17\times A_{0,0} + 58\times A_{1,0} - 10\times A_{2,0} + 4\times A_{3,0} - A_{4,0}\right) >> shift1 \quad (1c)$$

The fractional-pel pixels $e_{0,0}$, $i_{0,0}$, $p_{0,0}$, $f_{0,0}$, $j_{0,0}$, $q_{0,0}$, $g_{0,0}$, $k_{0,0}$ and $r_{0,0}$ shall be derived by applying the 8-tap filter to the fractional-pel pixels $a_{0,i}$, $b_{0,i}$ and $c_{0,i}$ in vertical direction where $i = -3$, ..., 4. Equation (2a)~(2c) show the interpolation formulas of the fractional-pel pixels $e_{0,0}$, $i_{0,0}$ and $p_{0,0}$.

$$e_{0,0} = \left(-a_{0,-3} + 4\times a_{0,-2} - 10\times a_{0,-1} + 58\times a_{0,0} + 17\times a_{0,1} - 5\times a_{0,2} + a_{0,3}\right) >> shift2 \quad (2a)$$

$$i_{0,0} = \left(-a_{0,-3} + 4\times a_{0,-2} - 11\times a_{0,-1} + 40\times a_{0,0} + 40\times a_{0,1} - 11\times a_{0,2} + 4\times a_{0,3} - a_{0,4}\right) >> shift2 \quad (2b)$$

$$p_{0,0} = \left(a_{0,-2} - 5\times a_{0,-1} + 17\times a_{0,0} + 58\times a_{0,1} - 10\times a_{0,2} + 4\times a_{0,3} - a_{0,4}\right) >> shift2 \quad (2c)$$

IV. NEON-BASED QUARTER-PEL INTERPOLATION METHOD

In this section, an acceleration method for the quarter-pel interpolation in HEVC is proposed for NEON SIMD architecture, by minimizing the number of memory access and arithmetic operations such as multiplication.

Because of the quadtree structure and AMP technique in HEVC, the sizes of PU can be (4, 8), (4, 16), (8, 4), (8, 8), (8, 16), (8, 32), (12, 16), (16, 4), (16, 8), (16, 12), (16, 16), (16, 32), (16, 64), (24, 32), (32, 8), (32, 16), (32, 24), (32, 32), (32, 64), (48, 64), (64, 16), (64, 32), (64, 48) and (64, 64). As NEON supports instructions that can operate on 128-bit data at most, it can process a PU in parallel whose width is up to 8. So NEON can process the PU whose width is 4 or 8, and when the width is greater than 8, such as 12, 16, 24, 32, 48 and 64, we can match up the PU through the combination of the PUs corresponding to width=4 and width=8. For example, the PU (12, 16) can be a combination of a PU (4, 16) and a PU (8, 16), and special attention is required to the read and write address offsets. Now we take the PU with width=8 as an example to illustrate the NEON-based quarter-pel interpolation method as follows. The method for a PU with width=4 is similar to a PU with width=8.

Fig. 3 shows the proposed data arrangement methods for interpolations in vertical and horizontal directions with NEON instructions. For vertical interpolation, different integer pixels of each row are referenced. In order to carry out interpolations using NEON instructions, the pixel data should be loaded in registers. For 8×8 PUs, 8 integer pixels in the same row can be loaded in a Q register and 8 rows can be loaded in 8 different Q registers, as showed in Fig. 3(a). For horizontal interpolation, the 8 referenced integer pixels are in the same row. We need to arrange the pixels so that the SIMD instructions can be executed efficiently. NEON has a VEXT [6] instruction, through which we can arrange the pixels in one row. Fig. 3(b) shows how to load 16 integer pixels in two Q registers and use the VEXT instruction to arrange them into 8 different Q registers, namely Q0, Q2, Q3, Q4 Q5, Q6, Q7 and Q1.

Now all the multiplication, addition, rounding and saturation instructions can be done in parallel using SIMD operations. Many NEON data processing instructions are available in normal, long, wide, narrow and saturating variants. In HEVC reference software, both the source address and the destination address are 16-bit pointers, but the pixel is 8-bit for Y, U and V, so for rounding and saturation operations we need to use VRSHR, VQMOVN and VMOVL instructions in NEON. The first R in VRSHR instruction means rounding, and the Q in VQMOVN instruction means saturation. With these three instructions, the results are rounded and saturated to the range of 0~255.

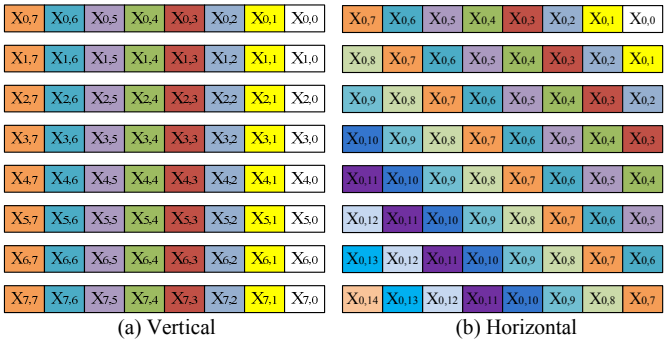

Fig. 3 Data arrangements for vertical and horizontal interpolations.

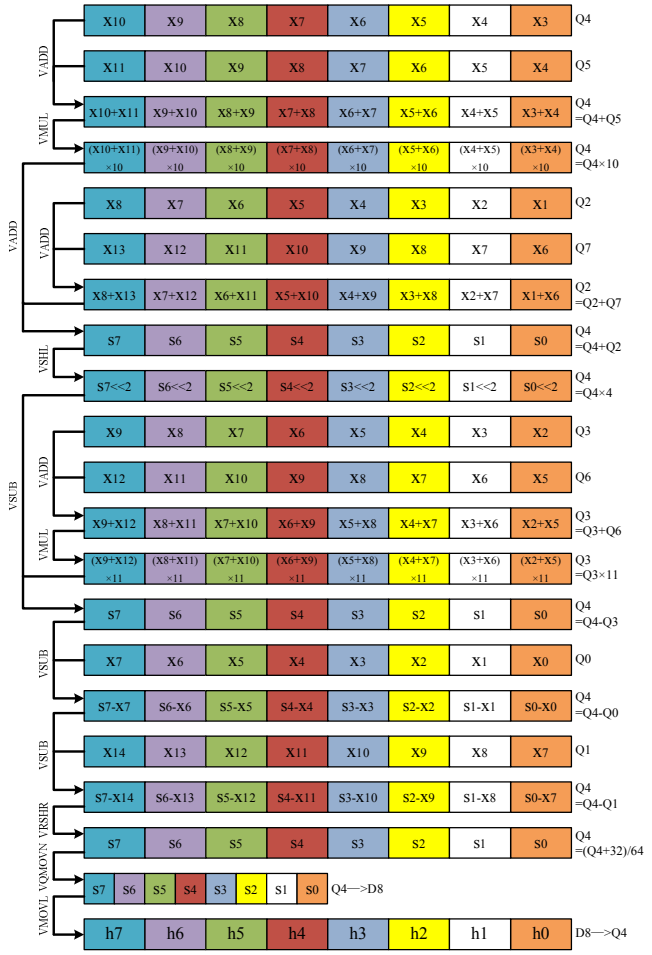

Fig. 4 A method to interpolate half-pel pixels by using the 8 tap DCTIF following the (3a) or (3b).

The formulas to compute half-pel interpolations are proposed by using the symmetry of the 8-tap DCTIF coefficients, resulting in significant reduction of the multiplications. Fig. 4 shows the 8-tap filtering to compute the (3a) or (3b) for eight half-pel values. For the quarter-pel pixels a, c and d, n, the computing processes are similar to the half-pel pixels b and h, respectively. So the fractional-pel pixels a, b, c, d, h and n can all be obtained.

$$\begin{aligned} b_{0,0} &= \left(-A_{-3,0}+4\times A_{-2,0}-11\times A_{-1,0}+40\times A_{0,0}+40\times A_{1,0}-11\times A_{2,0}+4\times A_{3,0}-A_{4,0}+32\right)>>6 \\ &= \left(40\times\left(A_{0,0}+A_{1,0}\right)-11\times\left(A_{-1,0}+A_{2,0}\right)+4\left(A_{-2,0}+A_{3,0}\right)-\left(A_{-3,0}+A_{4,0}\right)+32\right)>>6 \\ &= \left(\left(10\times\left(A_{0,0}+A_{1,0}\right)+A_{-2,0}+A_{3,0}\right)\times 4-11\times\left(A_{-1,0}+A_{2,0}\right)-A_{-3,0}-A_{4,0}+32\right)>>6 \end{aligned} \quad (3a)$$

$$\begin{aligned} h_{0,0} &= \left(-A_{0,-3}+4\times A_{0,-2}-11\times A_{0,-1}+40\times A_{0,0}+40\times A_{0,1}-11\times A_{0,2}+4\times A_{0,3}-A_{0,4}+32\right)>>6 \\ &= \left(40\times\left(A_{0,0}+A_{0,1}\right)-11\times\left(A_{0,-1}+A_{0,2}\right)+4\left(A_{0,-2}+A_{0,3}\right)-\left(A_{0,-3}+A_{0,4}\right)+32\right)>>6 \\ &= \left(\left(10\times\left(A_{0,0}+A_{0,1}\right)+A_{0,-2}+A_{0,3}\right)\times 4-11\times\left(A_{0,-1}+A_{0,2}\right)-A_{0,-3}-A_{0,4}+32\right)>>6 \end{aligned} \quad (3b)$$

For the fractional-pel pixels e, f, g, i, j, k, p, q and r, two interpolation operations are needed. In the first horizontal filtering, the results are stored as intermediate values, and in the second vertical filtering the intermediate values are applied an 8-tap filter to obtain the fraction-pel pixels. The intermediate values are 16-bit, so the data processing unit should be 32-bit. Because of the maximum register Q is 128-bit, one operation can implement four fractional interpolations in parallel, and hence two operations are needed to process all the loading data. Fig. 5 shows the first operation, illustrating the computation process of the second vertical interpolation. The intermediate values calculated by the first horizontal interpolation are stored in Q8~Q15. The first operation processes the data in registers D16, D18, D20, D22, D24, D26, D28 and D30 as showed in Fig. 5, and the results are stored in register D0; the second operation processes the data in registers D17, D19, D21, D23, D25, D27, D29 and D31 with the similar process, and the results are stored in register D1. In the process from Q0 to D0, operation VQRSHRUN is used. VQRSHRUN takes each element in a quadword vector of integers, right shifts them by an immediate value, and places the results in a doubleword vector. The Q means saturation, the first R means rounding, the U means transforming signed data to unsigned data, and the N means placing quadword vector operands in a doubleword vector.

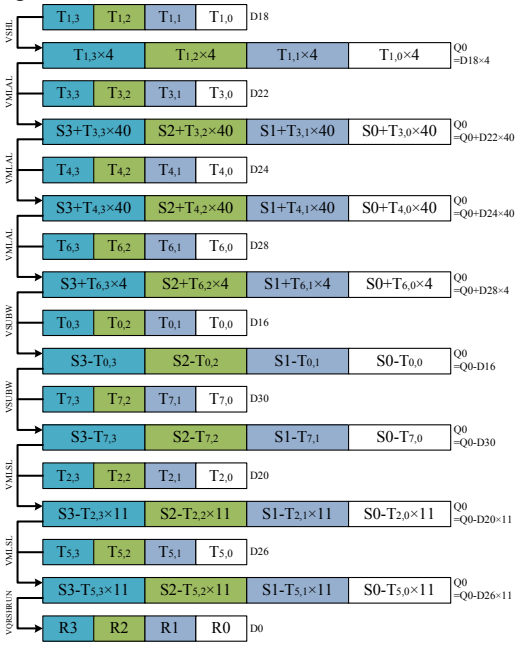

Fig. 5 A method of the second vertical interpolation based on the intermediate values following the (2b).

After two operations, Q0 which consists of D0 and D1 stores eight values, and they also need to be rounded and saturated to the range of 0~255 by using VQMOVN and VMOVL instructions as showed in Fig. 4. The computing processes of fractional-pel pixels e, f, g, j, k, p, q and r are similar to i. So all the 15 fractional-pel pixels can be obtained.

V. ACCELERATION RESULTS

We transplant the HEVC reference software HM5.2 [8] to the mobile phone operating system Android. The Android NDK allows Android application developers to embed native machine code compiled from C, C++ and ARM NEON assembly source files into their application packages. The Android VM allows the application's source code to call methods implemented in native code through the JNI (Java Native Interface). We use the LG P920 mobile phone as the platform to test our acceleration methods. This phone has a Cortex-A9 MPCore processor with NEON running at 1GHz. Eclipse Indigo is used to compile the Java codes and Android NDK r6b is used to compile the C++ codes in HEVC reference software and ARM NEON assembly codes of the accelerated modules. We conduct experiments in HEVC decoder, and the input bitstreams are obtained by encoding the video sequences with HM5.2 encoder under the common test conditions defined in JCTVC-F900 [9], and the configuration, low delay P high efficiency (LPHE) is used.

TABLE I
COMPARISON OF EXECUTION TIME AND PERFORMANCE OF QUARTER-PEL INTERPOLATION

Pel	Time (sec)		Speedup
	HM5.2	HM5.2 with NEON	
A	0.032857	0.005864	5.6032
a	0.036119	0.006569	5.4984
b	0.027891	0.004697	5.9380
c	0.029931	0.005624	5.3220
d	0.042760	0.006743	6.3414
e	0.075879	0.011865	6.3952
f	0.065607	0.009907	6.6223
g	0.083075	0.013346	6.2247
h	0.040399	0.005727	7.0541
i	0.065752	0.010942	6.0091
j	0.087957	0.014525	6.0556
k	0.069814	0.011425	6.1106
n	0.043587	0.006527	6.6780
p	0.075879	0.012237	6.2008
q	0.073659	0.010863	6.7807
r	0.087407	0.014003	6.2420

Table I and II show the comparison of actual execution time of quarter-pel interpolation module of the HM5.2 decoder and the proposed algorithm using the NEON. Among so many kinds of PUs, we choose an 8×8 block to test the performance of proposed algorithm, and the video sequence used in our experiment is *RaceHorses* which consists of 300 frames. The results in Table I show that each interpolation function with NEON is about six times faster than that in HEVC reference software. In addition, we test all the 8×height PUs, and find that it makes little difference when the height of PUs is 8 or greater than 8, which can be testified by the architecture of assembly codes of each interpolation function. 4×height PUs also are tested, and results show more than four times of speed improvement.

When decoding the bitstreams of the four test sequences in D class, we implement all the interpolation functions of all the kinds of PUs by NEON instructions, and Table II shows that our proposed method is about five times faster than the HEVC reference software for the quarter-pel interpolation.

TABLE II
COMPARISON OF EXECUTION TIME AND PERFORMANCE OF QUARTER-PEL INTERPOLATION IN DIFFERENT SEQUENCES

Sequence	Time (sec)		Speedup
	HM5.2	HM5.2 with NEON	
BasketballPass	3.689823	0.741390	4.9769
BQSquare	6.875181	1.396061	4.9247
BlowingBubbles	5.130787	1.000368	5.1289
RaceHorses	3.222310	0.612770	5.2586

VI. CONCLUSIONS

An optimized method for the quarter-pel interpolation in HEVC by using SIMD instructions is implemented on ARM processor. According to the experiment results, the proposed implementation of the quarter-pel interpolation is about five times faster than that of the HEVC reference software HM5.2. With the promotion of the next generation video coding standard HEVC and the increasing number of mobile multimedia applications, it is possible that the proposed SIMD based quarter-pel interpolation will be integral.

ACKNOWLEDGMENT

This work was partially supported by the National Science & Technology Pillar Program 2011BAH08B03, the Chinese National Natural Science Foundation under contract No. 61171139 and No. 61035001, National Basic Research Program of China under contract No. 2009CB320907 and No. 2009CB320906, and Shenzhen Basic Research Program of JC201104210117A and JC201105170732A.

REFERENCES

[1] Ken McCann, Woo-Jin Han and Il-Koo Kim, "Samsung's Response to the Call for Proposals on Video Compression Technology", JCTVC-A124, Dresden, Germany, Apr. 2010.

[2] Elena Alshina, Jianle Chen, Alexander Alshin, Nikolay Shlyakhov, Woo-Jin Han, "CE3: Experimental results of DCTIF by Samsung", JCTVC-D344, Daegue, Korea, Jan. 2011.

[3] Elena Alshina, Alexander Alshin, JeongHoon Park, "CE3: 7 taps interpolation filters for quarter pel position MC from Samsung and Motorola Mobility", JCTVC-G778, Geneva, CH, Nov. 2011.

[4] Detlev Marpe, Heiko Schwarz, Sebastian Bosse, etc. , Highly efficient video compression using quadtree structures and improved techniques for motion representation and entropy coding, 28th Picture Coding Symposium, PCS2010, Nagoya, Japan, Dec. 2010.

[5] Il-Koo Kim, Woo-Jin Han, JeongHoon Park, Xiaozhen Zheng, "CE2: Test results of asymmetric motion partition (AMP)", JCTVC-F379, Torino, IT, Jul. 2011.

[6] ARM Corp., RealView® Compilation Tools Version 4.0 Assembler Guide, DUI 0204J, Dec. 2010.

[7] Benjamin Bross, Woo-Jin Han, Jens-Rainer Ohm, Gary J. Sullivan, Thomas Wiegand, "High efficiency video coding (HEVC) text specification draft 7", JCTVC-I1003, Geneva, CH, Apr. 2012.

[8] https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/

[9] Frank Bossen, "Common test conditions and software reference configurations", JCTVC-F900, Torino, Italy, Jul. 2011.