International Journal of Applied Research 2015; 1(9): 490-497

ISSN Print: 2394-7500
ISSN Online: 2394-5869
Impact Factor: 5.2
IJAR 2015; 1(9): 490-497
www.allresearchjournal.com
Received: 24-06-2015
Accepted: 25-07-2015

Mani Laxman Aiyar
Dept. of ECE, Alliance University, Bangalore, Karnataka, 562106, India.

K Ramesha
Dept. of ECE, Dr. Ambedkar Institute of Technology, Bangalore, Karnataka, 560056, India

Correspondence
Mani Laxman Aiyar
Dept. of ECE, Alliance University, Bangalore, Karnataka, 562106, India.

A High-Precision Sub-Pixel Motion Estimator-Interpolator for Real-Time Ultra High Definition QFHD Video Coding in HEVC/ MPEG-H(2)/H.265, for Next Generation Multimedia Wireless applications

Mani Laxman Aiyar, K Ramesha

Abstract
Despite increased complexity sub-pixel motion estimation and compensation significantly outperforms integer motion estimation and compensation in MPEG4/H.264 Video Codec's, since moving objects do not necessarily move by integer pixel locations between successive video frames. Typically, fractional pixel accuracy is obtained by means of bilinear interpolation producing a spatially blurred predicted signal. The motion estimation and compensation is improved in this paper by means of the filtering effect using a very effective spatial digital low pass FIR filter, which allows the motion vectors to be determined with higher levels of precision and accuracy than existing algorithmic implementations. The fractional pixel accuracy was achieved using a total of 112 8-tap digital FIR filter for one-eighth pixel precision, which includes half and quarter pixel accuracy. The design has been implemented on a 28nm TSMC process, with a speed of 1.101 GHz and it has achieved 2262 GOPS at this speed, outputting data at the rate of 1.8 Tera bits per second, for one-eighth pixel accuracy. Computational complexity, Memory & I/O Bandwidth has been reduced by inputting the Mean Square Error Map of the pixels to the Fractional Pixel Estimator and then searching in the sub-pixel grid.

Keywords: MPEG4, H.264, HEVC, HDTV, DVB, FIR.

1. Introduction
The H.264 is the state-of-the-art codec and covers a wide range of applications with excellent results such as video-conferencing, video streaming, and video transmission over fixed and wireless networks with different transport protocols [1].
In the early motion-compensation algorithms like H.261, only full-pixel motion vectors were used i.e. the components of a motion vector were restricted to an integer number. Because the real motion between two successive images of a sequence is in general not exactly described by steps of one pixel, this restricted resolution leads to errors in the motion-compensated prediction and thus to an increased prediction error [2].
Despite the increased complexity, sub-pixel motion estimation and compensation can significantly outperform integer motion estimation and compensation. This is because, a moving object will not necessarily move by an integer number of pixels, between successive video frames. Searching sub-pixel locations as well as integer locations is likely to find a good match in a large number of cases [3].
Accurate motion estimation is essential to effective motion compensated video signal processing and sub-pixel resolutions are required for high quality applications [6]. It has been reported that motion vectors with fractional accuracy may be more efficient in encoding, motion compensated blocks [5].
The majority of the motion-compensated predictors that are reported in the literature use motion-compensation with "integer-pixel accuracy". Typically fractional pixel accuracy is achieved by means of bilinear interpolation, which produces spatially blurred predicted signal [3]. By using a spatial low pass filter, the predictor can improve the motion compensation. Improvement gained in this way is referred to as the filtering effect [4].
The prediction capability of motion compensation algorithm in MPEG4-10/H.264 is further improved by allowing motion vectors to be determined with higher levels of spatial accuracy than in existing standards. One-eighth pixel accuracy is currently the highest precision

accuracy that can be achieved in motion compensation, in MPEG4-10/H.264, in contrast with prior standards based primarily on half-pixel accuracy. Quarter-pixel accuracy is available only in the newest versions of MPEG4-2 visual. One-eighth pixel accuracy is adopted as a feature for increased coding efficiency at high bit rates and high video resolutions in HDTV [7].

The main objective of this work was to achieve motion-compensation with fractional-pixel accuracy, up to one-eighth pixel precision. Here pixel precision is achieved by what is referred to as the filtering effect, by designing an appropriate filter to overcome the spatial blurring. Further to this the filtering is done in three stages. The first stage is at half-pixel precision, the second stage being at quarter-pixel level and the third stage at the one-eighth pixel precision level done by bilinear interpolation. Computational complexity, Memory and I/O Bandwidth is reduced by inputting the mean square error map, of the integer pixels from the Integer pixel locations to the fractional pixel estimator, and then further searching in the sub-pixel grid.

This work involves the design, modelling and simulation of the sub-pixel motion estimation in Matlab, C & Verilog HDL and implementation of the same on a TSMC 28nm process, using Cadence & Synopsys Design tools. The output of the integer pixel motion estimation unit, are the integer motion vectors and the mean square error map. The MSE map consists of 9 x 9 grid around the optimal integer motion vector. This is inputted to the sub-pixel precision estimation unit, for interpolating the sub-pixel values.

The rest of the paper is organized as follows. Section-2 describes the design and implementation of the Motion estimator-interpolator and Section-3 describes the results achieved and analysis of the same.

2. Design of Algorithm and Its Implementation

Motion Estimation and Compensation

Motion Estimation is a process in video compression, by which the current frame is predicted from a previous frame(s) or future frame(s) or both. The Motion Estimator generates motion Vectors to indicate where the objects have moved from, as shown in Figure 1.

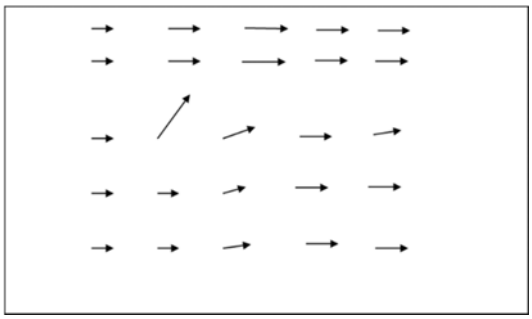

Fig 1: Motion Vectors

This motion Estimation process is done only at the Encoder. The Motion Compensator subtracts the predicted frame from the original frame, to produce the residual frame as shown in Figure 2.

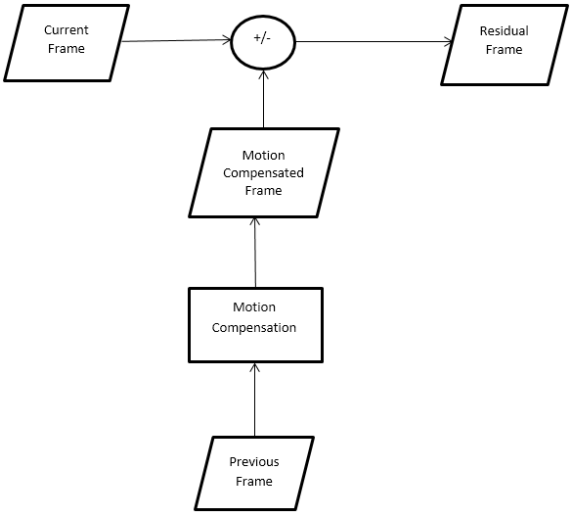

Fig 2: The Motion Compensator

Only the residual frame is encoded and transmitted to the Receiver. The Video Decoder reconstructs the original frame from the motion vectors based on an identical prediction scheme at the receiver. The best compression performance is achieved when the size of the residual frame transmitted through the channel is minimized.

The Block Matching Process

The initial motion estimate is achieved by means of a block matching process. For each block of luminance samples in the current frame, the motion estimation algorithm searches a neighboring area in the reference frame for a match. The best match is one that minimizes the energy of the difference between the two blocks. The complete set of mean square error values for each position in the frame forms the MSE map for the frame, as shown in Figure3 and Figure 4.

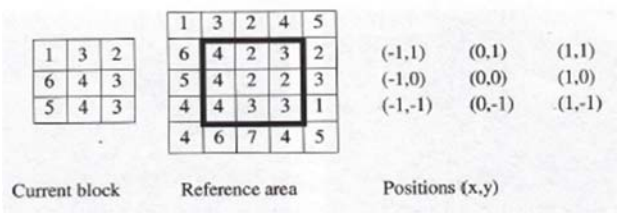

Fig 3: Block Matching Process and MSE positions

The MSE provides a measure of the energy remaining in the difference block for a N xN sample block the MSE is given by:

$MSE = 1/N^2 \sum\sum (C_{ij} - R_{ij})^2$; $\sum$ from 0 to N-1,

Where C_{ij} is a sample in the current block and Rij is a sample in the reference block.

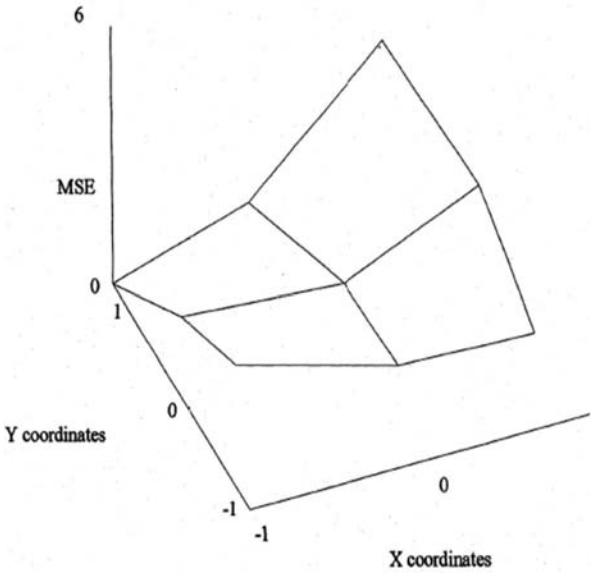

Fig 4: Mean Square Error Map

Mean Absolute Error

Mean absolute error provides a reasonable good approximation of residual energy and is easier to calculate then the MSE, since it requires magnitude calculation instead of square calculation for each pair of samples is given by:

$MAE = 1/N^2 \sum\sum |(C_{ij} - R_{ij})|$; summation from 0 to N-1

The comparison may be further simplified by neglecting the $1/N^2$ term giving the sum of absolute errors (SAE) or the sum of absolute differences (SAD), which gives a reasonable approximation and is used as a matching criterion for block-based motion compensation.

$SAD = \sum\sum |(C_{ij} - R_{ij})|$; summation from 0 to N-1

Sub-Pixel Motion Estimation

The best match for the block matching process is not necessarily a region offset from the current block by an integer number of pixels. For many blocks, a better match is found by searching a region interpolated to sub-pixel accuracy. This will produce a smaller Displaced Frame Difference (DFD), which reduces the energy in the residual frame, which leads to increased compression efficiency.

Estimating sub-pixel values using Bilinear Interpolation in 2D

The interpolating filter is a digital low-pass FIR filter, used to filter the image frequencies generated by increasing the sampling rate. The Interpolation process is shown in Figure 5.

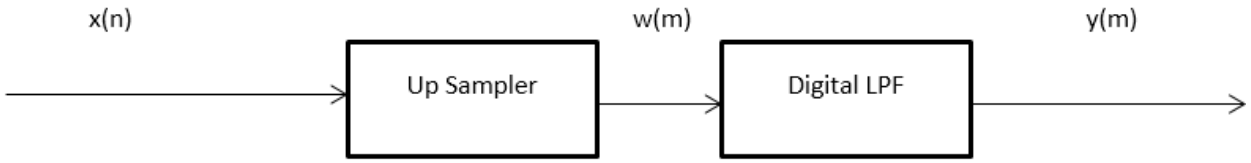

Fig 5: Interpolation Process

The computational overhead of sub-pixel interpolation is reduced by interpolating between the SAE values calculated at integer position, rather than carrying out search at sub-pixel positions [3]. SAE is a continuous function in the region of the minimum. Figure 3 shows the approach of bilinear interpolation in 1D. The integer SAE calculations are carried out to produce the points shown in black. By fitting a curve to the integer SAE results, estimated SAE results are found for the neighboring half-pixel positions as shown in Figure 6.

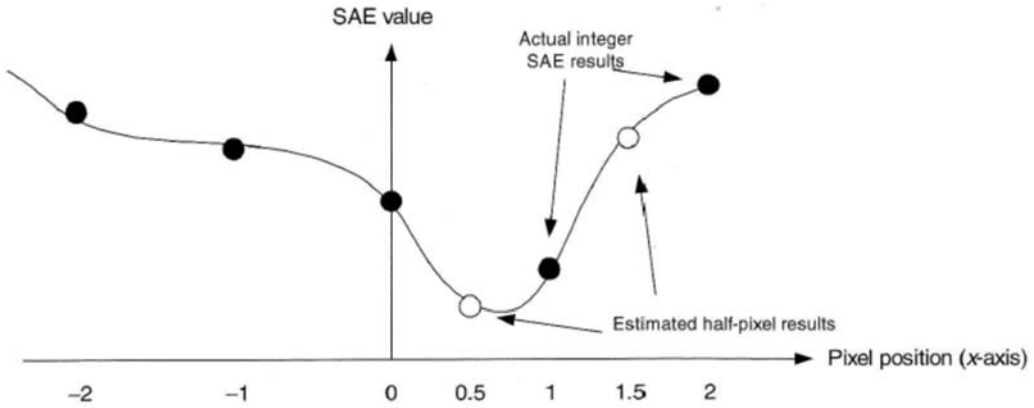

Fig 6: Estimating half-pixel positions from and Curve Fitting

The complexity of half-pixel estimation is reduced using this method. The accuracy of the motion-compensation depends on, how accurately the curve fitting approximates the actual half-pixel results. The 1D interpolation process is extended

by doing bilinear interpolation in 2D.

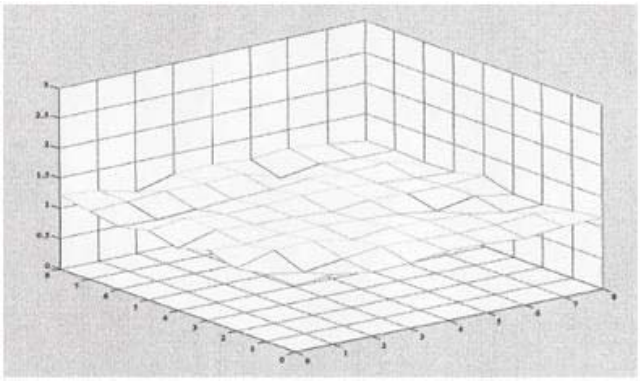

Fig 7: The 3-D SAE Map

2D interpolation takes a series of (x,y,z) points and generates estimated values of z at new (x,y) points. This interpolation process is used since the function that originally generated the (x,y,z) is unknown. Interpolation is related to but very distinct from fitting a function to a series of points. Here, the interpolated function goes through all the original points, while a fitted function does not. Data is available for a rectangular grid of points and bilinear interpolation is performed here for points off the grid. Here this process was modeled in Matlab as shown in Figure 7, and then was designed and implemented using Verilog HDL and synthesized on a 28nm TSMC process.

Bilinear interpolation by designing Digital FIR LPF's

In this paper, bilinear interpolation at sub-pixel levels has been achieved by designing 112 8-tap FIR digital filters using Matlab. The FIR filter is represented by the following filter difference equation:

$Y(n) = \sum h(k)\, x(n-k)$; k varying from 0 to L, L = 7 in this case;

The frequency response of this filter is determined from the equation: $H(e^{j\theta}) = \sum h(k)\, e^{-jk\theta}$; k varying from 0 to L, L = 7 in this case;

The z-transform of the above equation can be represented as: $H(z) = \sum h(k)\, z^{-k}$; varying from 0 to L, L = 7 in this case;

Simulation of the Design

The coefficients of the 8-tap FIR filters for the half-, quarter- & one-eighth pixel position was designed in Matlab. The filter responses including magnitude and the phase plots for the quarter-pixel, half-pixel and one-eighth pixel are shown in Figure 8.

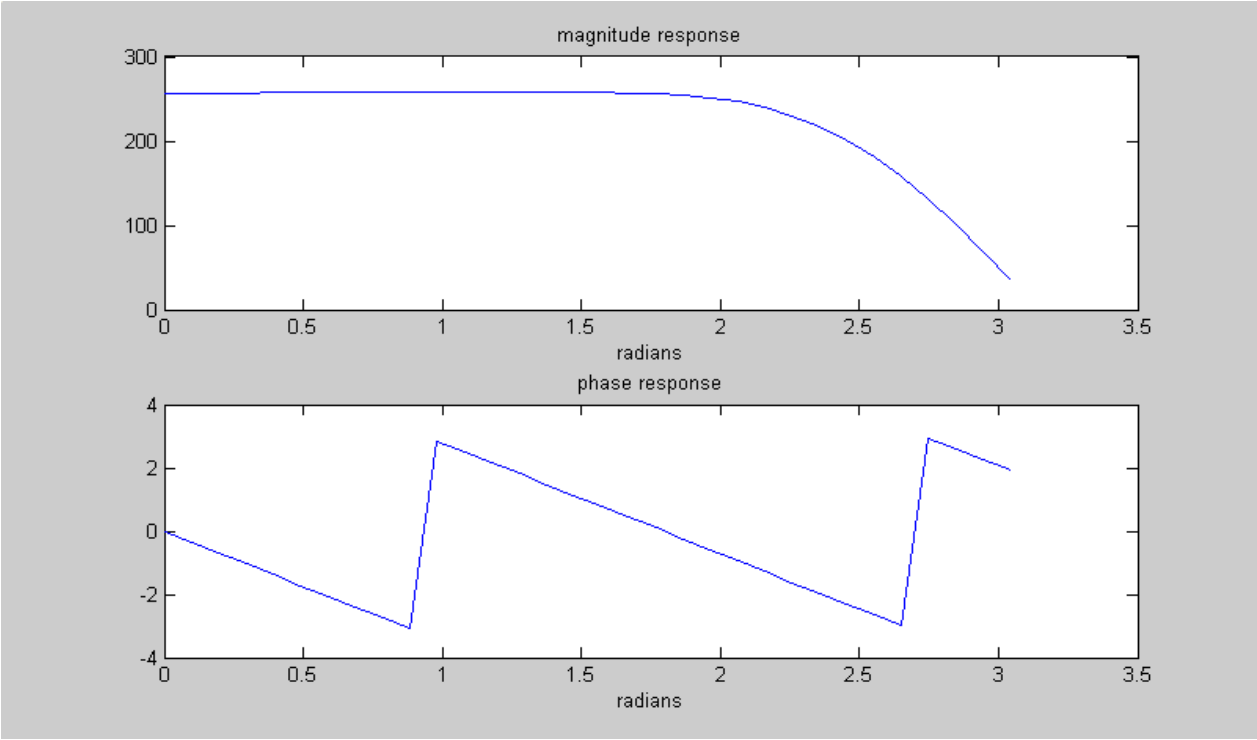

Fig 8: FIR Filter Response

Further the designswere implemented in hardware by using Verilog HDL and simulated in Synopsys VCS simulator.

Video Encoder System Architecture

As shown in Figure 9 the motion estimation accelerator unit forms a major part of the video encoder system architecture. This unit is separate from that of the system CPU, memory & I/O. The current and reference blocks are stored in the current frame buffer RAM and search window buffer RAM respectively. These form the inputs to the motion estimation unit as shown if figure. The motion estimation unit accesses this data through the global address & data bus.

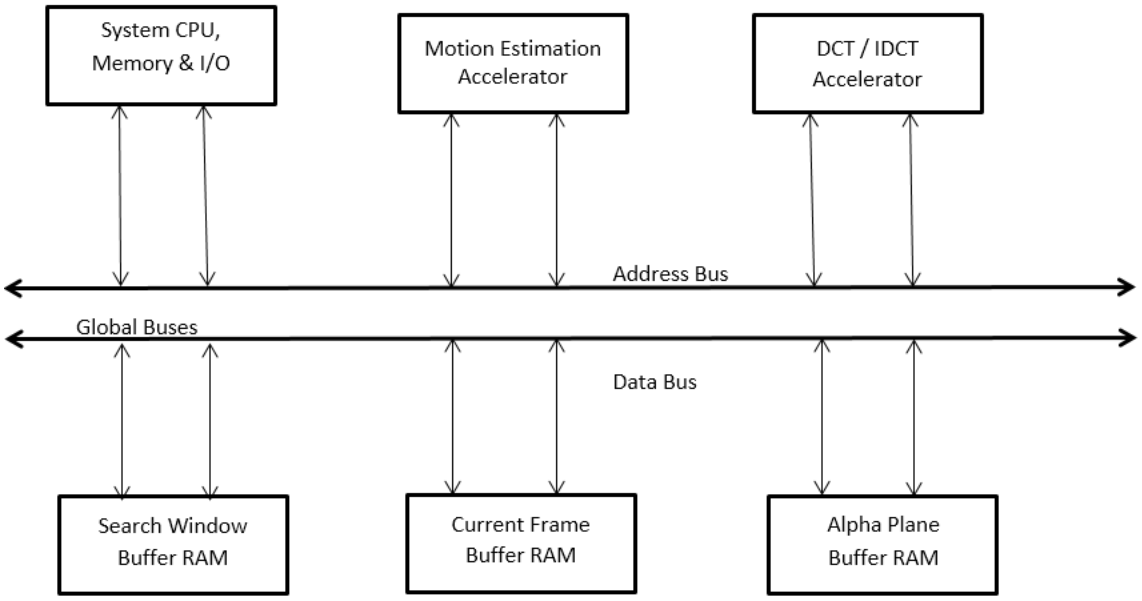

Fig 9: Video Encoder System Architecture

Integer Motion Estimator

The inputs to the motion estimation unit are the current 8 x 8 block and reference 16 x 16 block. The integer unit finds a match for the current 8 x 8 block, in the reference 16 x 16 search window. For the full search motion estimation, the Integer Unit calculates the mean square error (MSE) map for a 9 x 9 grid, as shown in figure. The location of the minimum MSE in the 9 x 9 grid forms the optimal integer motion vector. The motion vectors and the MSE error map are the outputs from the integer unit and are stored in memory.

Sub-Pixel Estimator

The output of the integer pixel motion unit the inputs to the sub-pixel motion estimation unit, being designed. The input to the current design is the optimal motion vector and the mean square error map with the optimal integer position at the center (4,4). The Mean Square Error map consists of a 9x9 grid around the optimal integer motion vector. This design also can be used as a sub-pixel interpolation unit in order to interpolate the sub-pixel values for higher resolution. In that case, the actual pixel values are loaded as an input to the design. This is achieved by multiplexing the inputs as shown in figure.

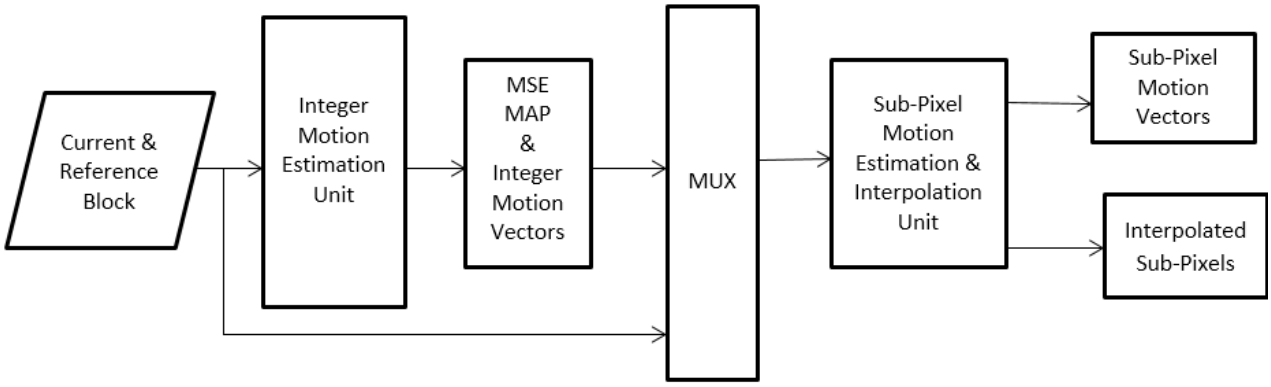

Fig 10: The Sub-Pixel Estimator

The design of the sub-pixel estimator has been done in three modular stages as follows:

1. The Half-Pixel design, which interpolates the half-pixels
2. The Quarter-Pixel design, which interpolates the half- and quarter- pixels.
3. The One-Eighth-Pixel which interpolates the half, quarter, and one-eighth pixels.

The Half-Pixel Design

The heart of the design consists of several multipliers, adders, comparators and the clipping function. The whole design was created using a Verilog HDL model.

Inputs

The inputs to the design are the mean square error map from the integer-pixel motion estimation unit. These inputs come as a 9x9 array. The total number of inputs are eight one. Each of these inputs has a value between 0 and 255 which are represented in 8 bits. The motion vector is at the center of the array at location (4,4). The array starts from (0,0) and ends at (8,8). The data is stored row-wise. The first nine inputs are from (0,0) to (0,8). Then the next nine values come in from (1,0) to (1,8) and so on up to (8,8).

Outputs

The output is the optimal motion vector location and value. The value of the output is in 8 bit form.

Also, there are eight other half-pixel outputs, in addition to the optimal integer-pixel. All these outputs have 8 bit values. These outputs are the actual half-pixel interpolation values. The half-pixel values are calculated from the input 9x9 array. Figure shows these half-pixel values which were calculated in the grid array (33x33).

The Quarter-Pixel Design

The heart of the design consists of several multipliers, adders, comparators and the clipping function. The whole design was created using a Verilog HDL model.

Inputs

The inputs to the design are the mean square error map, from the integer-pixel motion estimation unit. These inputs come as 9x9 array. The total number of inputs is eighty one. Each of these inputs have a value between 0 and 255 which are represented in 8 bits. The motion vector is at the center of the array at location (4,4). The array starts from (0,0) and ends at (8,8). The data is stored row-wise. The first nine inputs are from (0,0) to (8,8). Then the next nine values come in from (1,0) to (1,8) and so on up to (8,8).

Outputs

The output is the optimal motion vector location and value. The value of the output is in 8 bit form. Also, there are 48 other quarter-pixel outputs in addition to the optimal integer-pixel. All these outputs have 8 bit values. These outputs are the actual quarter-pixel interpolation values.

Multipliers and Adders

These multipliers and adders have been modeled in behavioral modeling, using Verilog hardware description language. The actual architectural implementation of these multipliers and adders has been done during synthesis. The synthesis was done using Synopsys Design Compiler, using the Design Ware foundation components, for architectural implementation of the multipliers and adders. The quarter-pixel values were calculated from the input 9x9 array. Figure 11 shows the quarter-pixel values which were calculated in the grid array.

A	d	b-h	d	b-h	d	b-h	d	A
d	e	d	f-h	d	f-h	d	e	d
b-v	d	c-q	d	c-q	d	c-q	d	b-v
d	f-v	d	g	d	g	d	f-v	d
b-v	d	c-q	d	c-m	d	c-q	d	b-v
d	f-v	d	g	d	g	d	f-v	d
b-v	d	c-q	d	c-q	d	c-q	d	b-v
d	e	d	f-h	d	f-h	d	e	d
A	d	b-h	d	b-h	d	b-h	d	A

Fig 11: 9 x 9 Quarter Pixel Array

The Comparators

The final 49 quarter-pixel output values are compared for the minima. The input to the comparator logic is an array of 17x17 quarter-pixels around the optimal integer-pixel, as shown in Figure 12. The comparator logic is implemented as a 50 input comparator, with an additional input for the reference value.

	3.000	3.125	3.250	3.375	3.500	3.625	3.750	3.875	4.000	4.125	4.250	4.375	4.500	4.625	4.750	4.875	5.000
3.000	33	d	b-h	d	b-h	d	b-h	d	34	d	b-h	d	b-h	d	b-h	d	35
3.125	d	e	d	f-h	d	f-h	d	e	d	e	d	f-h	d	f-h	d	e	d
3.250	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
3.375	d	f-v	d	g	d	g	d	f-v	d	f-v	d	g	d	g	d	f-v	d
3.500	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
3.625	d	f-v	d	g	d	g	d	f-v	d	f-v	d	g	d	g	d	f-v	d
3.750	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
3.875	d	e	d	f-h	d	f-h	d	e	d	e	d	f-h	d	f-h	d	e	d
4.000	33	d	b-h	d	b-h	d	b-h	d	34	d	b-h	d	b-h	d	b-h	d	35
4.125	d	e	d	f-h	d	f-h	d	e	d	e	d	f-h	d	f-h	d	e	d
4.250	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
4.375	d	f-v	d	g	d	g	d	f-v	d	f-v	d	g	d	g	d	f-v	d
4.500	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
4.625	d	f-v	d	g	d	g	d	f-v	d	f-v	d	g	d	g	d	f-v	d
4.750	b-v	d	c-q	d	c-q	d	c-q	d	b-v	d	c-q	d	c-q	d	c-q	d	b-v
4.875	d	e	d	f-h	d	f-h	d	e	d	e	d	f-h	d	f-h	d	e	d
5.000	33	d	b-h	d	b-h	d	b-h	d	34	d	b-h	d	b-h	d	b-h	d	35

Fig 12: The 17 x 17 one-eighth-Pixel Array

The One Eighth-Pixel Design

The heart of the design consists of several multipliers, adders, comparators and the clipping function. The whole design was created using a Verilog HDL model.

Inputs

The input to the design is the mean square error map, from the integer-pixel motion estimation unit. These inputs come as 9x9 array. The total number of inputs is eighty one.

Each of these inputs have a value between 0 and 255 which are represented in 8 bits. The motion vector is at the center of the array at location (4,4). The array starts from (0,0) and ends at (8,8). The data is stored row-wise. The first nine inputs are from (0,0) to (8,8). Then the next nine values come in from (1,0) to (1,8) and so on up to (8,8).

Outputs

The output is the optimal motion vector location and value. The value of the output is in 8 bit form. Also, there are 225 other one eighth-pixel outputs in addition to the optimal integer-pixel. All these outputs have 8 bit values. These outputs are the actual one eighth-pixel interpolation values.

The one eighth-pixel values were calculated from the 9x9 array.

3. Results and Discussion

Comparison of the results with other implementations

Reference [8] describes a cascadable 200GOPS motion estimation chip for HDTV Applications. Reference [9] describes a parallel processor for motion estimation. [10] and [11] describes a VLSI Implementation of mean-corrected block-matching motion estimation of Partial Quad trees. A Verilog standard cell based synthesis approach was used for this design contrary to that of the full custom design methodology used by [8] which offers the advantage of faster migration to newer VLSI technology libraries. The main differences of the presented work to that of the previous implementations are shown in Table 1 below.

Table 1: Comparison of Results

Comparison Criteria	UcdL Belgium	RWTC Aachen Germany	Tech. University Munich	Our Design Half-Pixel	Our Design Quarter-Pixel	Our Design 1/8th Pixel
Search Strategy	Prog.	Full-Search	Full-Search	Full-Search	Full-Search	Full-Search
# of Pixel Element	12	32 x 32	32 x 32	3 x 3	9 x 9	15 x 15
Clock Frequency	50 MHZ	200 MHZ	100 MHZ	1120 MHZ	1010 MHZ	1010 MHZ
Design Style	Full Custom	Full Custom	VHDL Synth	Verilog	Verilog	Verilog
Process	1 um	0.5 um, 2LM	0.35um Compass	28 nm TSMC	28 nm TSMC	28 nm TSMC
Processing Power	N.A.	200 GOPS	100 GOPS	430 GOPS	1616 GOPS	2262 GOPS
Speed(MHZ)		200	100	1120	1010	1010
On-Chip Memory	512 Bytes	No	12,288 Bytes	802 Bytes	802 Bytes	802 Bytes
MV Search Area	Prog.	+/- 15	[-16,+15], [-32, +31]	+/- 15	+/- 15	+/- 15

The main objective of this project is to achieve motion-compensation with fractional-pixel accuracy, up to one eighth-pixel accuracy. In this report, fractional pixel accuracy is achieved by what is referred to as the filtering effect by using the FIR filter coefficients as described in the H.264 standard. In this work, filtering is done at two stages. The first filtering is done at the half-pixel stage and the second filtering is done at the quarter-pixel stage. The one eighth-pixels are extracted by means of bilinear interpolation. Computational Complexity and Memory & I/O bandwidth is reduced by inputting the mean square error map, of the pixels to the Fractional Pixel estimator and then searching in the sub-pixel grid.

The sub-pixel motion estimation unit was designed in Verilog HDL and synthesized using Synopsys and Cadence tools. The output of the integer pixel motion unit is an input to the sub-pixel motion estimation unit being modelled. The input to the current design is the optimal motion vector and the mean square error map with the optimal integer position at the center (4,4). The Mean Square Error map consists of a 9x9 grid around the optimal integer motion vector. This design can also be used as a sub-pixel interpolation unit in order to interpolate the sub-pixel values for higher resolution. The computational complexity and Memory & I/O bandwidth is reduced by inputting the mean square error map of the pixels to the Fractional Pixel estimator, and then searching in the sub-pixel grid.

Table 2: Comparison of Mathematical Operations

S.No.	Design	1/2-Pixel	1/4-Pixel	1/8 Stage 1	1/8 Stage 2	1/8 Stage 3	1/8 Total
1	# of Comparisons	20	100				552
2	# of Equations	24	100	112	48	128	288
3	# of Multiplier /Equation	8	8	8	1	0	9
4	# of Adder / Equation	7	7	7	2	1	10
5	# of Divisions / Equation	1	1	1	1	1	3
6	# of Operations	16	16	16	4	2	22
7	Total # of Multiplications	192	800	896	48	0	944
8	Total # of Additions	168	700	784	96	128	288
9	Total # of Divisons	24	100	112	48	128	288
10	Total # of Operations	384	1600	1792	192	256	2240
11	(GOPS)	430.08	1616.00	1809.92	193.92	258.56	2262.40
12	Speed(GHZ)	1.12	1.01	1.01	1.01	1.01	1.01

The half-, quarter- one eight- pixel motion estimator was designed, simulated, synthesized and verified at the gate level using the 28 nm TSMC process. The fractional pixel accuracy was achieved using a 2 stage spatial 8-Tap Digital FIR filter for the one eighth-pixel estimator of MPEG4/H.264. A total of 112 8-Tap FIR filters were used at the one eighth-pixel level in comparison to 188 8-Tap FIR filters and 24 8-Tap filters at the quarter- and half-pixel levels, respectively, as shown in Table 2.

The half-pixel design which is a 160,000 gate design, is working at a speed of 1.12 GHz and is achieved a maximum of 430 Giga Operations Per Second (GOPS), at this speed.

The quarter-pixel design which is a 320,000 gate design is working at a speed of 1.01 GHz and it achieved a maximum of 1616 Giga Operations Per Second (GOPS) at this speed. The one eighth-pixel design which is a 320,000 gate design is working at a speed of 1.01 GHz and it achieved a maximum of 2262 Giga Operations Per Second (GOPS) at this speed. This design outputs data at the rate of 1.8 Tera bits per second, at 1.01 GHz, as shown in Table 2.

References

1. Jou SY, Chang SJ, TS Chang. Fast Motion Estimation Algorithm and Design for Real Time QFHD High-Efficiency Video Coding, IEEE Transactions on Circuits and Systems for Video Technology, 2015.
2. He G, Zhou D, Li Y, Chen Z, Zhang T, Goto S. High-Throughput Power Efficient VLSI Architecture for Fractional Motion Estimation for Ultra-HD HEVC Video Coding, IEEE Transactions on VLSI Systems, 2015.
3. Sinangil ME, Sze V, Zhou M, Chandrakasan AP. Cost and Coding Efficient Motion Estimation Design Considerations for HEVC Standard, IEEE Journal of selected topics in signal processing. Dec, 2013, 7(6).
4. Zhou D, Zhou H, He G, Goto S. A 1.59 Gpixels/s Motion Estimation Processor with -211-to-211 Search Range for UHDTV Video Encoder, Symposium of VLSI Circuits and Digital Technology Papers, 2013.
5. Tsai SF, Li CT, Chen HH, Tsung PK, Chen KY, Chen LG. A 1062 Mpixels/s 8192x4320p High Efficiency Video Coding (h.265) encoder chip, Symposium of VLSI Circuits and Digital Technology Papers, 2013.
6. Dominguez DJO *et al.* The H.264 Video Coding Standard, IEEE Potentials, 2014.
7. Pereira F, Ebrahimi T. The MPEG-4 Hand Book", Prentice Hall PTR, New Jersey, 2002.
8. Richardson IEG. Video CODEC Design", John Wiley & Sons, England, 2002.
9. Girod B. Motion Compensating prediction with fractional-pixel accuracy, IEE Transactions on Communications 1993; 41(4):604-612,
10. Yang K *et al.*, A family of VLSI designs for the motion compensation block-matching algorithm, IEEE Transactions on circuits and systems 1989; 36(10):1317-1325.
11. Lix X, Gonzales C. Alocally quadratic model of the motion estimation error criterion function and its application to sub-pixel interpolation, IECE Transactions on circuits and systems for video technology, 1996, 6(1).
12. H.264 standard UB Video Inc, 2014. www.ubvideo.com.
13. Berns JP, Noll TG. A cascadable 200 GOPS motion estimation chip for HDTV applications, IEEE custom integrated circuits conference, San Diego, 1996, 335-358.
14. Hanssens E, Legat JD. A Parallel processor for Motion Estimation, SPIE, 1996, 1006-1016.
15. Kuhn P. Algorithms complexity analysis and VLSI architectures for MPEG-4 Motion Estimation" Kluwer Academic Publishers, 1999.
16. Kuhn P *et al.*, VLSI Implementation of Mean-corrected Block-Matching Motion Estimation of Partial Quad trees, VLBV 97, Workshop for very low Bitrate Video Coding, Sweden, 1997.