TEXAS INSTRUMENTS

White Paper
SPRAAX8–August 2008

Translational Motion Stabilization for Embedded Video Applications

Fitzgerald J Archibald

In this paper, translation motion stabilization of video in embedded applications like digital still camera (DSC), digital camcorder, and security camera is discussed. In embedded systems, computational load and power consumption are critical parameters along with the quality of stabilization. The method and apparatus described in this paper use boundary signal computation (BSC), a single instruction multiple data (SIMD) core, enhanced DMA, and the ARM9EJ processor to accelerate the computations and save power.

The algorithm starts with a projection vector (boundary signal) computation of each block in a frame. A block is a logical partitioning of a frame. The projection vectors are used for estimating the block motion vector of each block within the frame. The block motion vector estimation uses sum of absolute differences (SAD) computation and its derivatives to find the best matching position of boundary signals from successive frames. The best match position results in block motion vectors which are passed through histogram stages of frame or global motion estimation.

The histogram module consists of raw, windowed, accumulated and weighted histogram sub-modules. All of these sub-modules along with spurious data detection logic are used in the estimation of global motion. The global motion is passed though a biquad filter array with delay and clip compensation to estimate jitter. The jitter is compensated for by generating a cropped window within the frame. The position of the window is chosen to negate the global motion. The cropping can be done either using DMA or using the line offset feature in on screen display (OSD).

Index Terms: image stabilizer, motion jitter estimation and compensation, translational motion estimation and compensation, video stabilizer

Contents

List of Figures

List of Tables

1 Introduction

The video sequences recorded by a digital still camera (DSC) or digital camcorder typically have unwanted frame motion due to unintentional shaking of the camera. The unwanted motion can be translational (vertical and/or horizontal) or rotational. The video sequences may have desired frame motion as in panning shots. The purpose of video stabilization is to remove unwanted motion from the video sequences being recorded or captured. Video stabilization can also be performed during playback of captured video. The vertical axis tends to have more jitter generally due to gravity affecting a camera held by hand. The horizontal motion is prominent when the video is shot from moving vehicles. This paper provides a video stabilization algorithm for compensating or eliminating translational motion by estimating frame motion from block motion vectors. Videos shot while walking tend to have rotational motion. The rotation motion needs quite a bit of processing on motion compensation and hence is not discussed.

Section 2 provides prior art so that the reader can understand the problem and existing solutions. Section 2 dwells in depth into the three major computational stages: the block motion estimation (Section 3.1), frame motion estimation (Section 3.2), unwanted motion estimation (Section 3.3), and motion compensation by cropping (Section 3.4). The embedded system design for video stabilization is illustrated in Section 4. Section 5 and Section 6 provide brief algorithm evaluation results and a summary of the algorithm.

2 Prior Art

Integral projection matching based translation video stabilization algorithm using a frame level boundary signal (or projection vector) and the unrestricted motion compensation described in [1] have their limitations on quality of stabilization and ability to compensate motion. If stabilization is performed by cropping, the stabilized frames are of smaller size. The cropped image can be resized to original scale if required at the cost of video quality. Most often, the capture sensors produce more pixels than that is used for the video stream and hence cropping is not a problem. If a single projection vector is used, object movement can get interpreted as motion jitter. However, projection vector approach is useful with hardware acceleration and use of multiple vector spaces for eliminating local motion vector resulting from object motion.

Rotational jitter along with translation motion jitter removal by use of binary motion estimation in selected area of the frames, local motion outlier removal for global motion estimation and use of 1st order damping filter for ARM based platform is illustrated in [2]. This paper is useful in understanding the complexity and loading of video stabilization on an ARM based system-on-a-chip (SoC).

Matsushita et al. [3] present a method for enhancing the quality of stabilization by means of using motion inpainting and de-blurring. Due to limitations on available clock cycles for video stabilization in SoC intended for low end digital still cameras, the proposed motion compensation scheme is not practical. This method can be used if accelerators are available for performing stabilization.

In [4], a method incorporating circular block matching is used for translational and large rotational motion stabilization for handheld digital cameras. The paper does not provide computational complexity or hardware requirements for implementation of circular block matching. The probability based approach in global motion estimation should be computationally less intensive. Rotation motion compensation has additional problems in compensation further requiring specialized hardware acceleration. A method for estimating translational, rotational motion jitter in the presence of depth changes using a 2.5-D inter-frame motion model is presented in [10]. This method uses an inertial motion filtering for estimating jitter component of the global motion vector. A method for affine motion estimation based on local and global projection is presented in [11]. Affine motion compensation requires hardware acceleration to meet real-time constraints required by cameras.

Kalman filter based jitter motion estimation is presented in [5]. This approach is practical as it considers constraints on the amount of rotational and translational motion that can be compensated in addition to interdependencies of the constraints. Frame Position Smoothing (FPS) based jitter estimation is presented by Ertuk in [9]. Viterbi dynamic programming algorithm based motional jitter estimation method is presented in [6]. The computational load is the least on jitter estimation irrespective of the use of this algorithm or any of the other methods like Kalman filter, damping filter or IIR filters. In [8], Litvin et al. present a method for estimating jitter using a Kalman filter and mosaicking for compensation. Mosaicking requires extra computational cycles.

A method for implementing motion detector using 11,000 gates is explained in [7]. This method uses band extract representative points (BERP) for motion estimation and object movement recognition for improving stabilization quality.

3 Algorithm Design

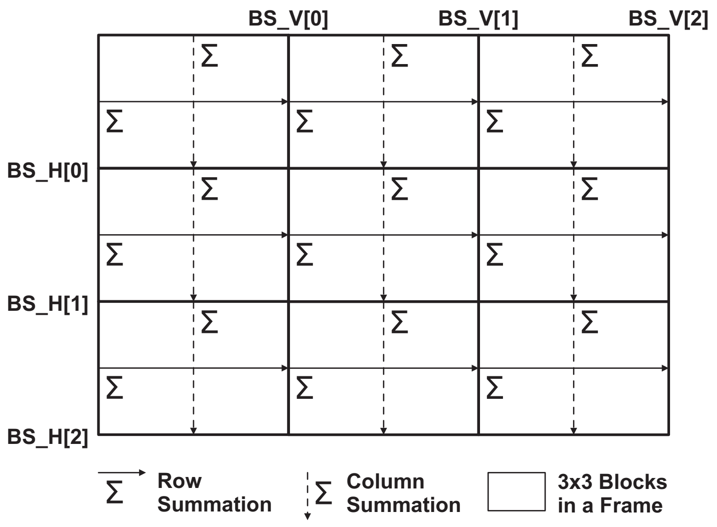
The video stabilization algorithm consists of:

1. Block motion estimation (BME)
2. Frame motion estimation (FME)
3. Unwanted motion estimation (UME)
4. Frame motion compensation (FMC)

The computations need to be performed for horizontal and vertical axes. The boundary signal computation (BSC) and sum of absolute differences (SAD) computations used in BME are computationally intensive. The computational complexity of FMC depends on the technique used for compensation.

The video frame is divided vertically and horizontally to produce blocks. Only luma (Y) samples are used for BSC and SAD computations used in block motion vector (BMV) and frame motion vector (FMV) estimation. The reasons for using Y are because of the availability of Y samples for all pixels in a frame, and the texture information contained in the samples. Figure 1 illustrates the 3×3 division of a frame resulting in nine blocks. This division is carried out for getting block motion vectors, rather than a frame vector. If the frame has a moving object, only a few of the BMV will be incorrect. Thus, this division helps in finding FMV even in the presence of moving objects in the video frames being stabilized. The number of blocks in a frame can equal as much as the number of Y pixels. The smaller the block dimension, the higher the computational load for BMV and FMV computation. The division of 3×3 is more optimal in the sense there is a center block with a focus on the object of interest and eight boundary blocks. The division of 5×5 is good for larger frame sizes since there are nine inside blocks and 16 boundary blocks.

Figure 1. Blocks in a Frame



The smaller the block dimension, the higher the computational load for BMV and FMV computation. The division of 3×3 is more optimal in the sense there is a center block with focus on object of interest and eight boundary blocks. The division of 5×5 is good for larger frame sizes since there are nine inside blocks and 16 boundary blocks.

3.1 Block Motion Estimation (BME)

The computational steps in BME are BSC, SAD and derivatives computation and Motion Estimation (ME). Luma (Y) samples in a video frame are used for Boundary Signal (BS) calculation. The BS is used for computing SAD vector. The SAD and derivative vectors are used for estimating Motion Vector (MV) for each of the blocks in the video frame.

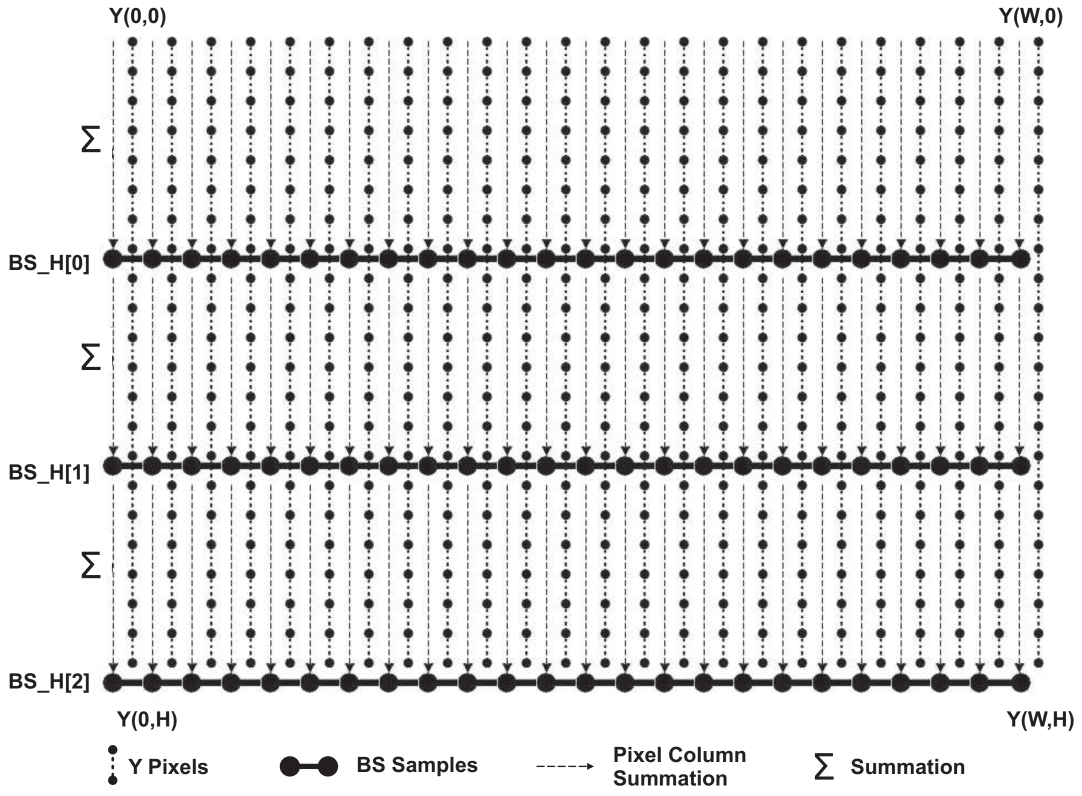
3.1.1 Boundary Signal Computation (BSC)

The boundary signal (BS) is the 1-D vector representing the texture information contained in the 2-D video frame. Horizontal BS (BS_H) is computed by summation of all pixels along a column, as given by (1) for the entire frame (1×1). Thus, the length of the boundary signal will equal the width of the video frame (W).

$$BS_H(h) = \sum_{\upsilon=0}^{\upsilon=H} Y(h,\upsilon),\ where\ h = 0, 1, 2, ..., W \tag{1}$$

Figure 2 shows BS_H computation in the case of 3×3 division. In 3×3 block division of the video frame, the summation is performed for each block. The frame is divided into three equal divisions horizontally. The summation is performed for each horizontal division. This results in three BS_H signals of a length equal to W. Each BS_H can be divided into three equal pieces to produce one BS_H signal per block.

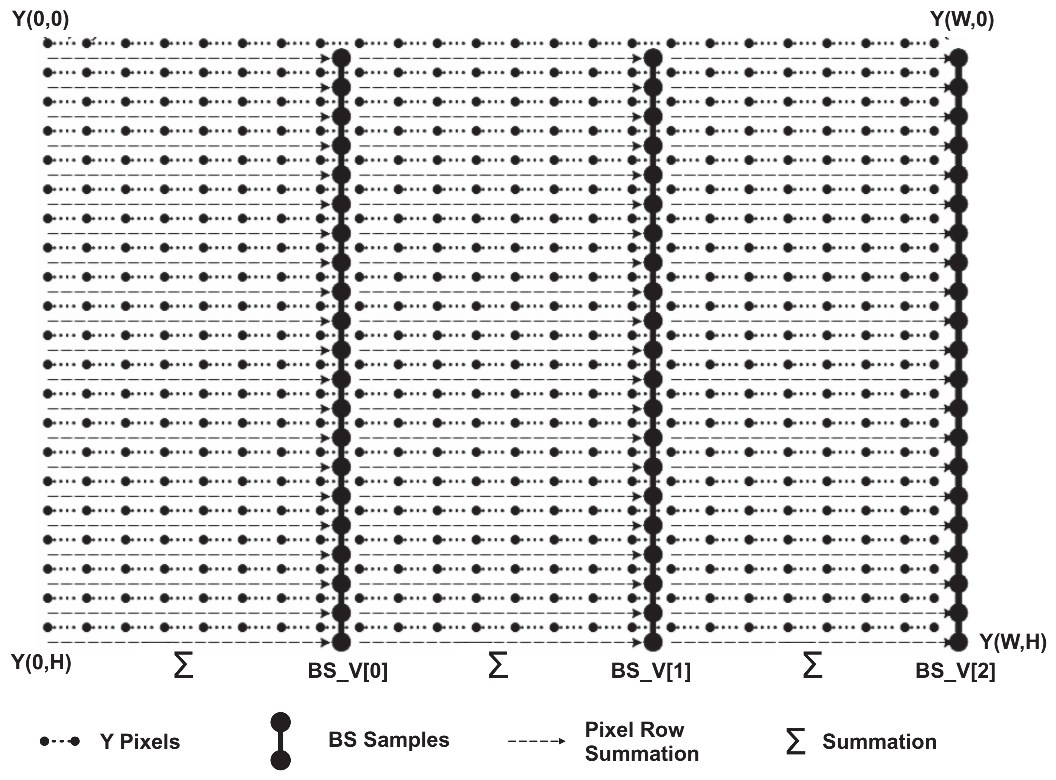
Figure 2. Horizontal BS Computation



Vertical BS (BS_V) is computed in a manner similar to BS_H. The summation of pixels along a row produces a BS_V of a length equal to the height of the video frame (H), as given by (2). The image is divided vertically into three equal pieces. The row summation on each division produces three BS_V, as shown in Figure 3. Each BS_V is divided into three equal pieces to produce one BS_V signal per block. Thus, there are nine BS_V and nine BS_H vectors.

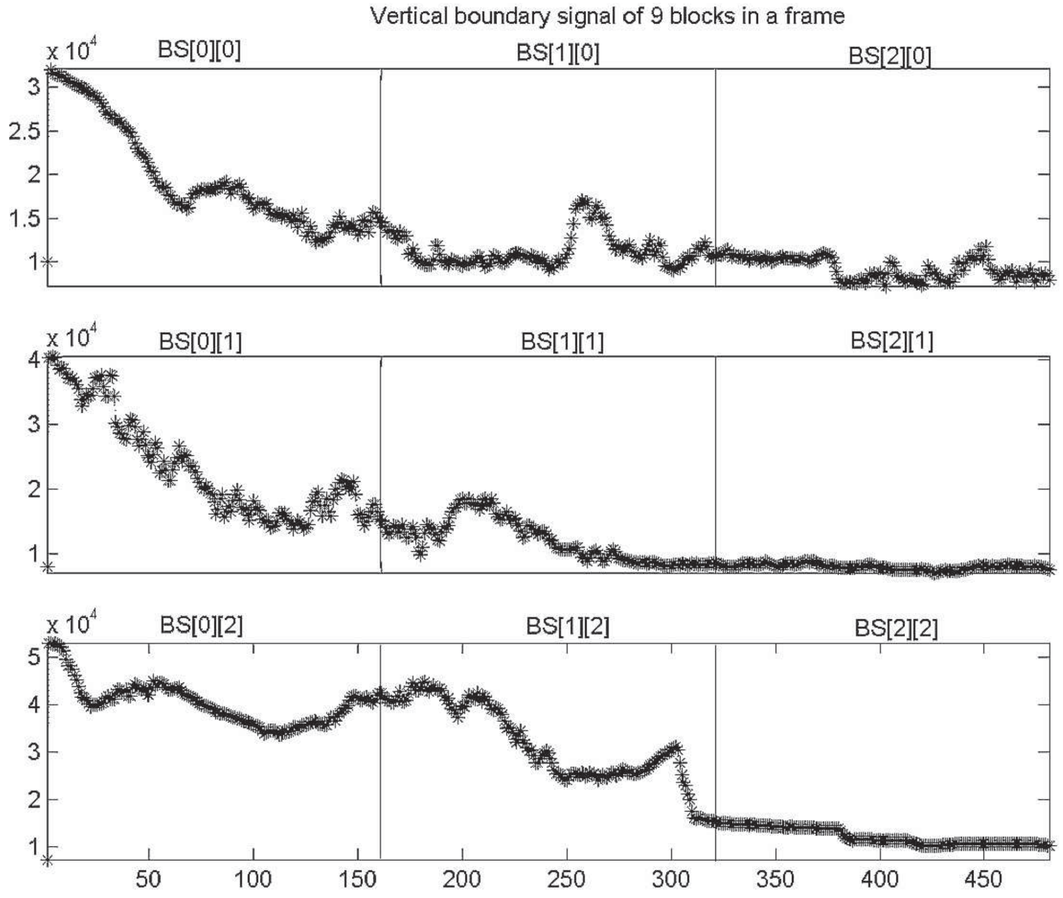
$$BSv(\upsilon) = \sum_{h=0}^{h=W} Y\,(h,\upsilon),\ where\ v = 0,\ 1,\ 2,\ldots,\ H \tag{2}$$

Figure 3. Vertical BS Computation



A practical example of boundary signal is provided in Figure 4.

Figure 4. Waveform of Boundary Signal



3.1.2 Sum of Absolute Differences (SAD)

The SAD computation on a 1-D BS results in a 1-D SAD vector. The SAD computation requires the reference (BS_{ref}) and current BS. The previous frame BS is the reference BS for the current frame. The current BS is slid on BS_{ref} in one sample step. For each sliding position, summation is carried out on all absolute differences between reference and current BS samples, as given by (3).

$$SAD(n) = \sum_{n=-BMV\,max}^{n=+BMV\,max} \sum_{m=0}^{m=L_{BS}-2xBMV\,max} |BSref\,(m + n + BMV\,max) - BS(m + BMV\,max)| \tag{3}$$

Sliding is carried out from negative maximum BMV ($-BMV_{max}$) to positive maximum ($+BMV_{max}$) position with respect to BS of length LBS. This means the BS is shorter than BS_{ref} by $2\times BMV_{max}$, as shown in Figure 5 and Figure 6.

Figure 5. Horizontal SAD Computation

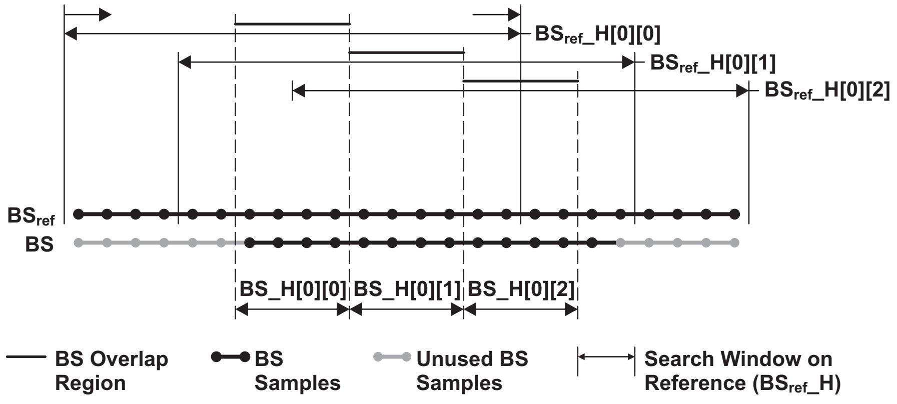
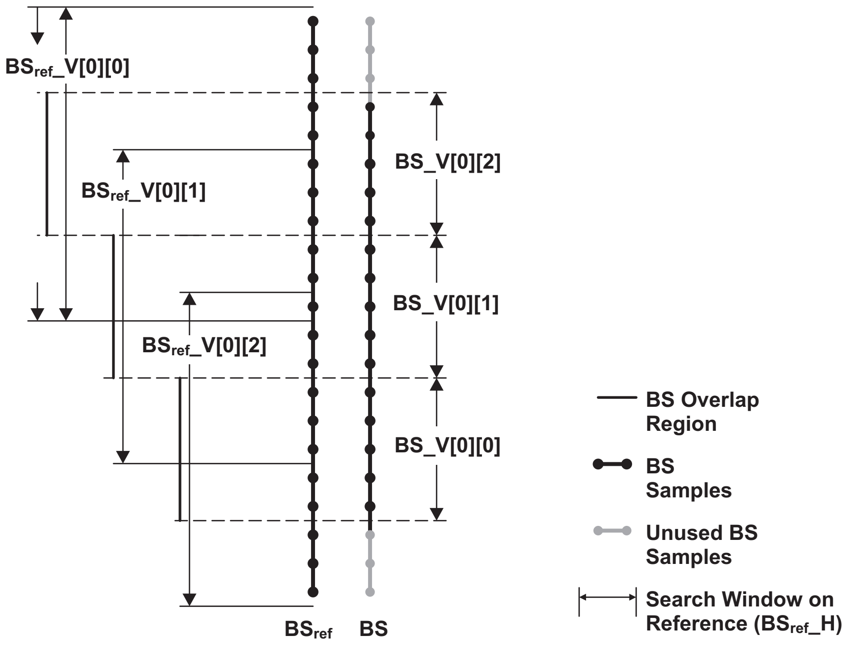


Figure 6. Vertical SAD Computation



The SAD vector is computed for both the horizontal and vertical BS of each block in the video frame. Thus, there are nine horizontal SAD vectors (SADh) and nine vertical SAD vectors (SADv). In other words, one SAD vector per BS vector is computed. The three horizontal and three vertical SAD vector computations illustrated in Figure 5 and Figure 6, respectively, need a $2 \times BMV_{max}$ search margin on the edge of the boundary signals BS for sliding. The inner vector can use the outer boundary signal samples for sliding.

The SAD values will require 22-bit memory locations for 640×480 resolution with 3×3 blocks. The result can be shifted right to fit in 16-bit memory without any significant loss in BME capability. The amount of shift (rsh) is dynamically computed based on the past frame maximum SAD value as given in (4), (5) and (6). The SAD values are right shifted by rsh and saturated to 216-1 before storing in 16-bit memory.

$$estSAD_{max} = |SAD|_{max} \tag{4}$$

$$rsh = rsh + 1, \text{ if } (estSAD_{max} \geq 2^{16} - 1) \tag{5}$$

$$rsh = rsh - 1, \text{ if } (estSAD_{max} < 2^{15} - 1) \tag{6}$$

If automatic exposure or gain control is applied on input YUV frames, the boundary signals (reference and current) will vary in amplitude by a constant bias for correlated data. This variation will cause SAD to detect the reference and current frame as uncorrelated data due to the non-linear nature of the building block in SAD, namely absolute computation. However, equal or proportional gain change (gain or exposure time) as applied to the current frame can be applied to the reference BS to circumvent this problem. Note that the uncorrelated data problem that is due to the variation in luma magnitude still causes a problem in lighting changes caused by shadows or obstructions in the light path for the same scene content.

3.1.3 Derivatives of SAD

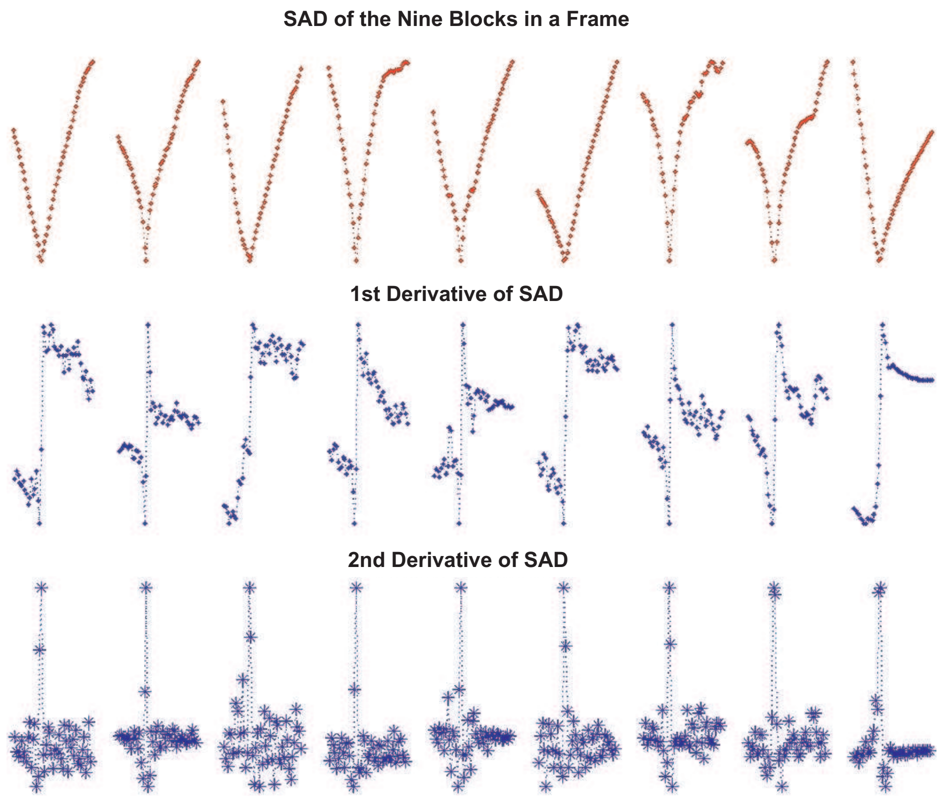
The first and second derivatives of SAD are computed as in (7) and (8).

$$\frac{d}{dn} SAD = SAD(n) - SAD(n-1),$$
$$\text{where } n = 1,2,3,\ldots,2^{*}\,BMVmax \tag{7}$$

$$\frac{d^2}{dn^2} SAD = \frac{d}{dn} SAD(n) - \frac{d}{dn} SAD(n-1),$$
$$\text{where } n = 1,2,3,\ldots,2^{*}\,BMVmax - 1 \tag{8}$$

The derivatives are computed for both horizontal and vertical SAD vectors. There is a first and second derivative vector for each vertical and horizontal SAD vector in a block. The example waveforms are shown in Figure 7.

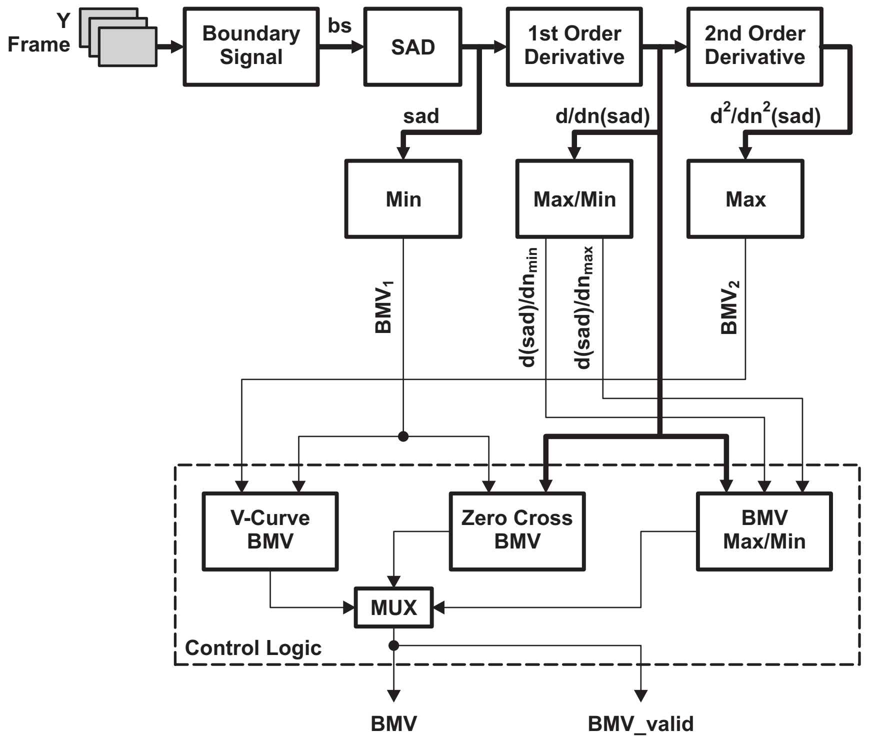
Figure 7. Waveforms of SAD and Derivatives



3.1.4 Block Motion Vector (BMV) Computation

The BMV computation relies on the nature of SAD curves for correlated data. In the case of highly correlated data, the SAD vector will have a minimum at the best match position. On either side of this minimum, the correlation will gradually decrease resulting in the increasing magnitude of SAD values. If the relative motion between adjacent frames is higher than the search range (-BMV_{max} to + BMV_{max}), then the SAD vector may simply have a negative or positive slope, depending on whether the motion is to right or to the left. This is because the best match position lies outside the search range. If the reference and current frame data are uncorrelated, then SAD may have many minima (false minima). In some cases of uncorrelated data, the SAD vector may even show a positive or negative slope line similar to the high motion case. The building blocks and data/control flow of the BME stage is given in Figure 8.

Figure 8. BME Block Diagram



Step 1:

If the distance between the positions on which SAD reached minima (minSADpos) and the second derivative reached maxima (maxSAD2pos) is within a threshold ($2 \times BMV_{max}/12$), the BMV for the block is estimated as the position of the second derivative maxima(maxSAD2pos).

Step 2:

If BMV cannot be determined using step 1, a window of region ($2 \times BMV_{max}/5$) is established in the first derivative of SAD around the position on which SAD reached minima. All the positions on which the first derivative of SAD transitions from negative to positive within this window are recorded. If the number of such transitions is unity and the distance between the positions of this zero crossing (xingPos) and the SAD minima are within a threshold ($2 \times BMV_{max}/12$), the BMV for the block is estimated as the position of the zero crossing.

Step 3:

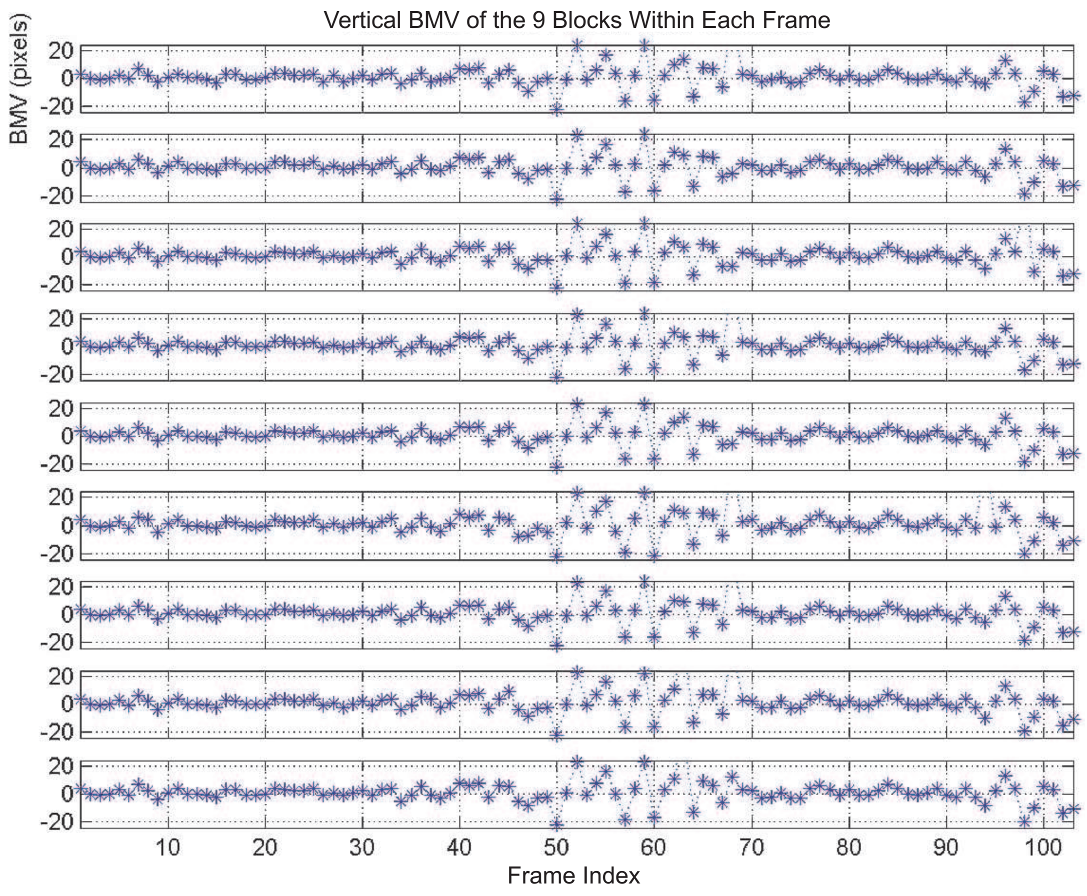
If step 1 and 2 do not yield a BMV, then BMV may have reached maxima. The maxima and minima values in the first derivative of SAD are added. If the sum is positive, a negative threshold of a value equivalent to 10% of the maxima is established. Otherwise, a positive threshold of 10% of the minima is established. If all values in the first derivative of the SAD vector are below the threshold, BMV is on positive maxima (BMV_{max}). If all the values in the first derivative of SAD are above the threshold, BMV is on negative maxima ($-BMV_{max}$).

Step 4:

If steps 1, 2 and 3 failed to estimate BMV, BMV is marked as invalid. If the maximum and minimum value of SAD is the same or within a threshold (either statically defined or dynamically estimated) then BMV is considered invalid. In this case, the minimum SAD most likely is not an index of motion.

The steps 1 to 4 are performed for each block in a frame for both the vertical and horizontal direction. The result from this stage is BMV and BMV validity in vertical and horizontal directions for each block within a frame. The BMV and its validity are passed on to the application via the interface to facilitate algorithms like image stabilization. An example of BMV waveforms is shown in Figure 9.

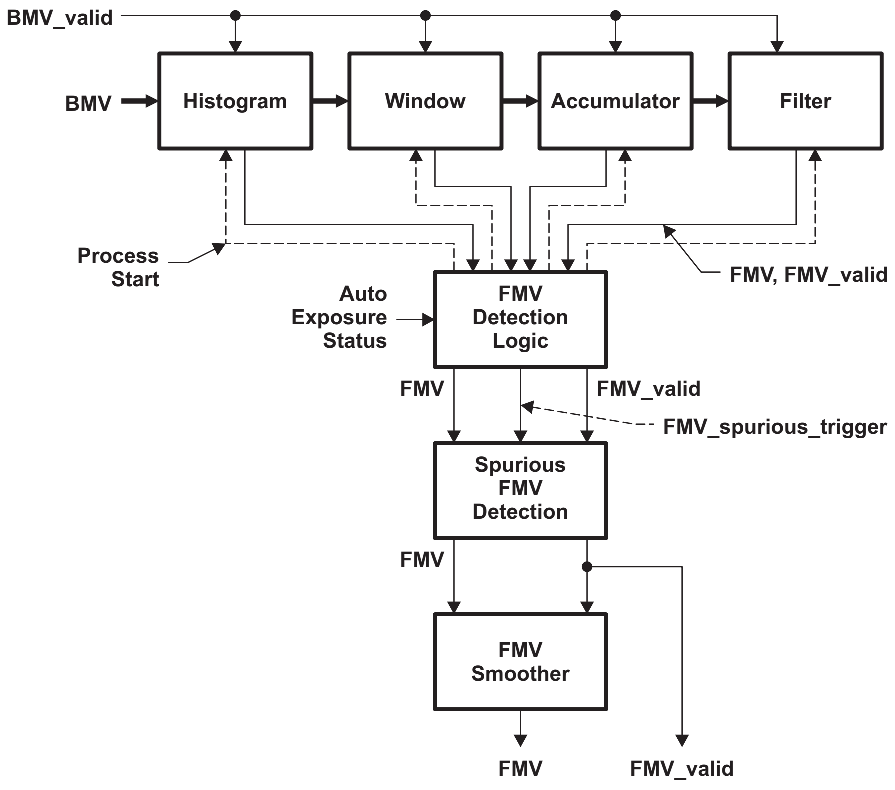
Figure 9. Example Waveforms of BMV



3.2 Frame Motion Estimation (FME)

The FME uses the BMV of each block in a frame to estimate the global or frame motion vector. The estimation step consists of raw histogram, windowed histogram, accumulated histogram, filtered histogram, FMV computation, spurious FMV detection, and FMV smoothing as shown in Figure 10. The histograms provide information to the FMV computer for estimating the frame or global motion vector. In addition, the parameters from the histogram are used by a spurious FMV detector for isolating potential jitter introduced by the algorithm. The main causes for spurious values are due to scene change (uncorrelated data), Luma variation (shadows and auto exposure algorithms), and object motion across frames. The spurious FMV detector validates whether the estimated FMV is erroneous. The FMV smoother minimizes the unwanted motion caused from valid to invalid transition and vice versa, when invalid FMV are detected.

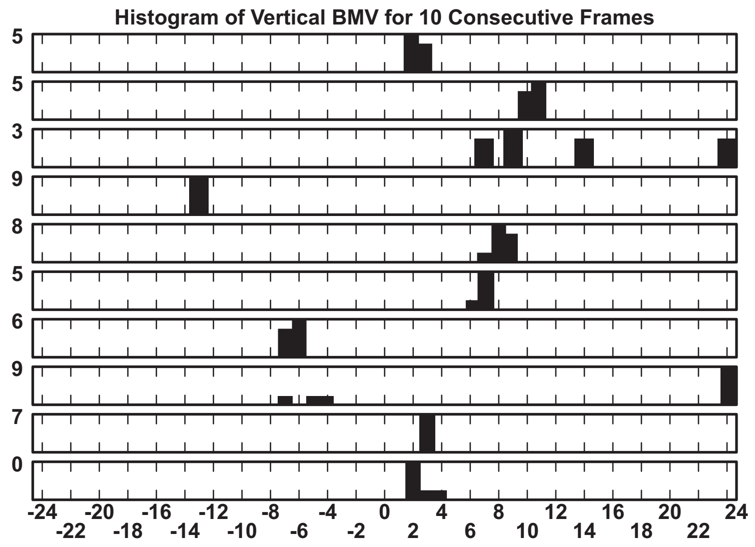
Figure 10. FME Block Diagram



3.2.1 Raw Histogram

The x-axis of the histogram is BMV values. The y-axis of the histogram is the number of blocks in the current frame that have the particular BMV value. A raw histogram example is given in Figure 11.

Figure 11. Raw Histogram Example

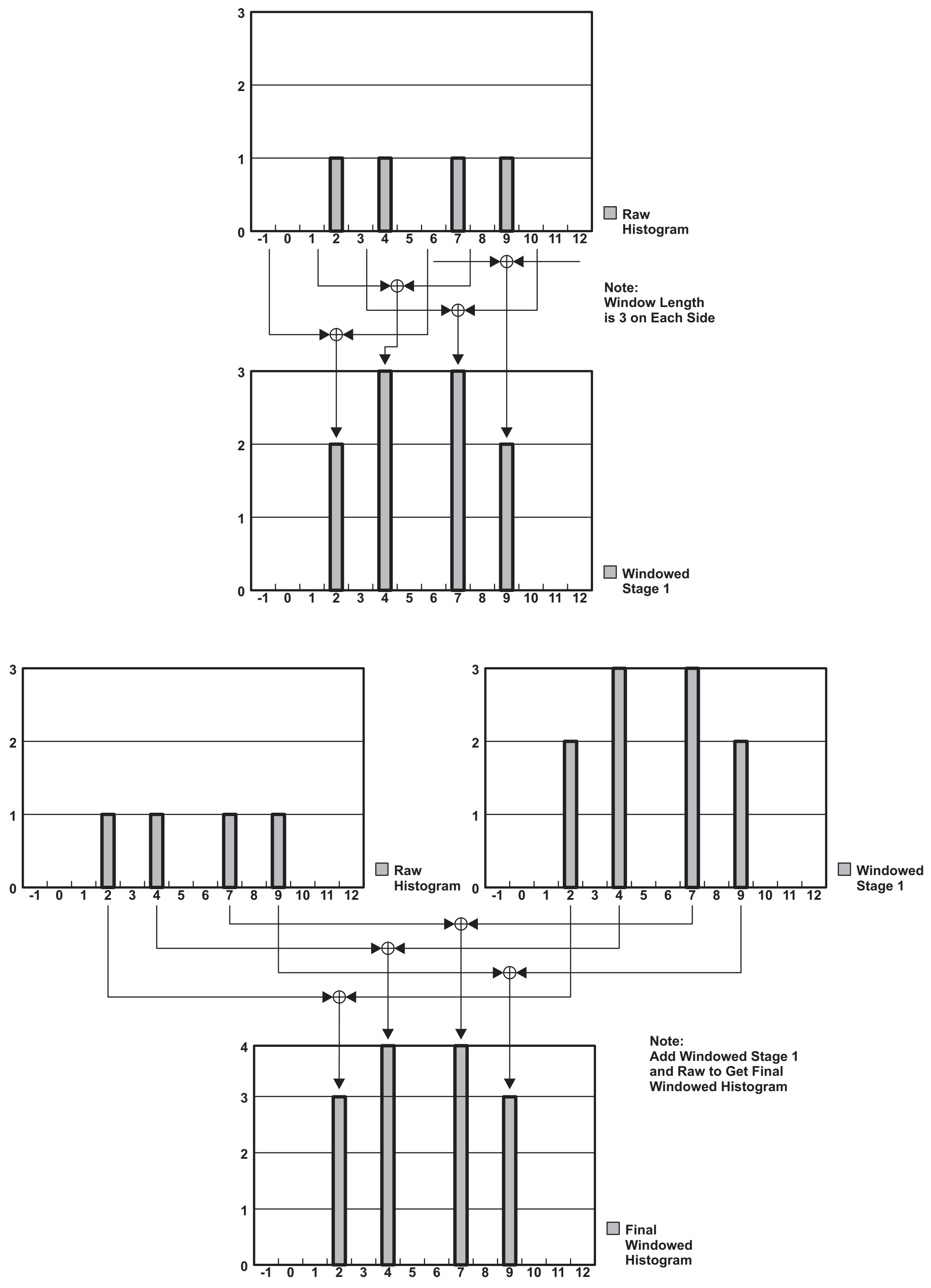


3.2.2 Windowed Histogram

In windowed histogram, a window of pre-determined length is slid across the raw histogram. The BMV position which is at the center of the window is accumulated with all the histogram bars within the window. The accumulated value is added with the center bar in the window, to avoid neighborhood bars of different height resulting in equal bar length with windowing. On completion of sliding across the raw histogram, a windowed histogram is generated.

Figure 12 is an example for the windowed histogram computation procedure.

Figure 12. Windowed Histogram Example



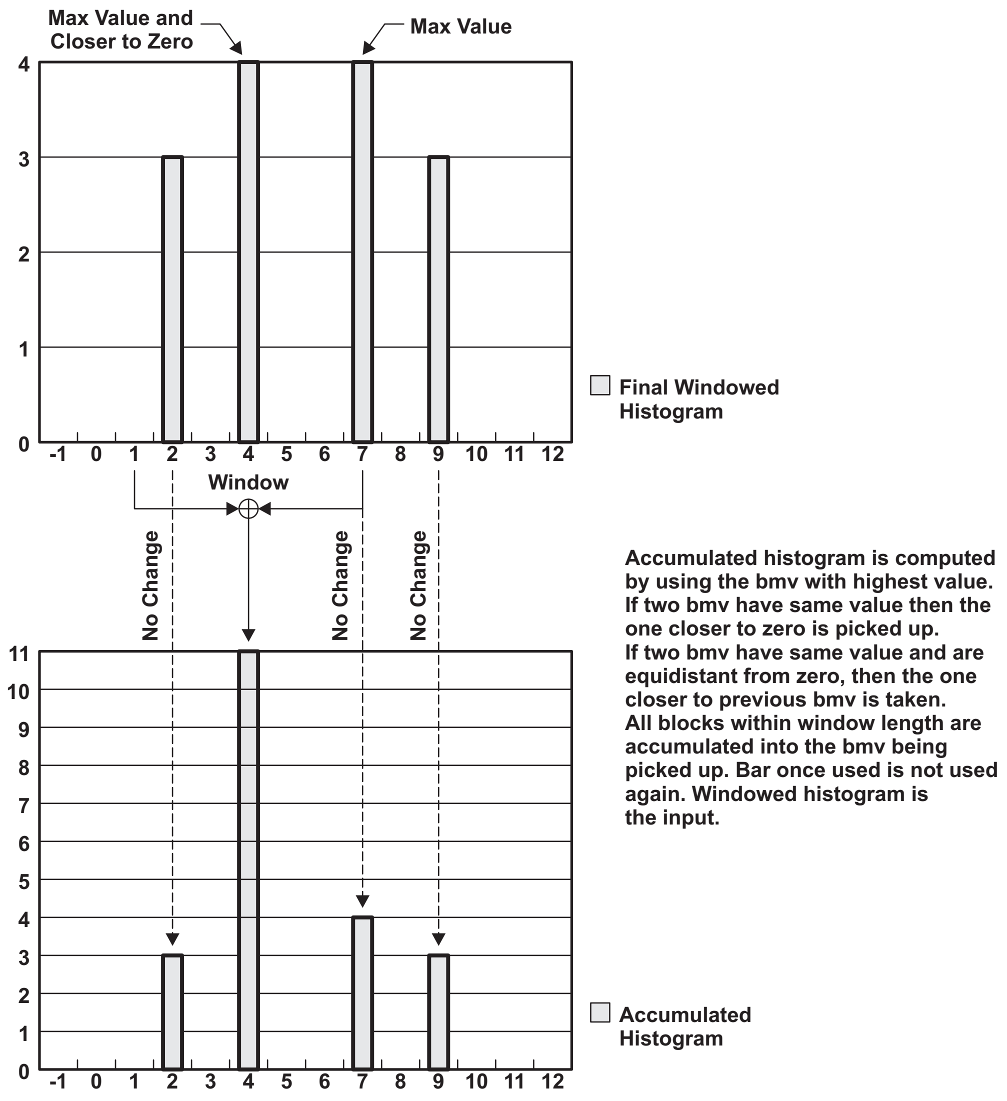
3.2.3 Accumulated Histogram

The largest bar in the windowed histogram is selected as the center. A window is established with this center. The window length is the same as in a windowed histogram. All the bars within the window are accumulated to this center. Once accumulation is completed, the bars used in this accumulation process are excluded from further accumulation. In other words, the bars contribute in only one accumulation. The above mentioned procedure for accumulation is repeated until all bars are used for accumulation.

When two or more bars have the same length and a center needs to be established, the bar closest to the previous valid FMV is selected as the center. When two bars are of the same distance from the previous valid FMV, the FMV closest to the BMV value of zero is selected as the center.

Figure 13 is an example for the accumulated histogram computation procedure.

Figure 13. Accumulated Histogram Example



3.2.4 Filtered Histogram

The filtered histogram relies on the past frame BMV validity status. A finite impulse response (FIR) filter with coefficients b0=2-1, b1=2-2, b2=2-3, b3=2-4, and b4=2-4 is applied on BMV validity of each block in the current frame and the past four frames. The number of past frames used in filtering process can be increased or decreased as needed and the coefficients may be adjusted accordingly. The filter equation is given by (9). The filter result will have a value between 0 and 1. The filter result of each block (blkId) is mapped against the corresponding BMV of the block in the current frame. Steps in accumulated histogram are performed on this mapped result to yield filtered histogram.

$$BMV_{valid}^{filt}[n][blkId] = \sum_{k=0}^{4} bk * BMV_{valid}[n-k][blkId],$$

where n is frame number and $blkId$ is blocks in frame(0..8) (9)

3.2.5 FMV Computation

If there are any valid BMV in the current frame, FMV can be estimated using the results of histogram stages. When an automatic exposure algorithm is active and the gain change is not adjusted during the SAD computation, FMV estimation is skipped since SAD will not have a minima at the BMV position.

Step 1:

If a raw histogram has a single maxima bar with the maxima value ($maxVal_{raw}$) greater than the threshold ($numBlk_{RawHistThr}$ = 3) and the maxima bar value is greater than or equal to the sum of all the other histogram bar values, then the BMV value on the maxima bar is picked as the FMV of the current frame.

If $maxVal_{raw}$ is less than or equal to one fourth the number of valid BMVs, no FMV is available from this step. If $maxVal_{raw}$ is greater than or equal to one fourth the number of valid BMVs, then the accumulated and filtered histogram based FMV detection in steps 3 and 4 are bypassed.

Step 2:

If the raw histogram failed to yield an FMV, the steps of FMV detection is repeated on the windowed histogram. The threshold value ($numBlk_{WinHistThr}$ = 6) is increased to accommodate for the windowing procedure of the histogram.

Step 3:

If the raw and windowed histogram does not yield a FMV, an accumulated histogram is used for FMV estimation. If there is a single maximum bar, the BMV corresponding to the maxima bar in the accumulated histogram is picked as the FMV. If the histogram bar value is below a limit ($numBlk_{AccHistThr}$=12), a flag ($spurious_{possible}$) is notified to the spurious FMV detector stage.

If there are two or more maxima, the BMV position closest to the zero vector at which the maxima histogram bar is detected is picked as the FMV. If two maxima are positioned at equal distance from zero BMV, the BMV position closest to the past FMV and nearest to the zero BMV is picked as the FMV from the histogram bars having maxima. If FMV is detected from multiple maxima, a flag ($spurious_{possible}$) is notified to the spurious FMV detector stage. In addition, the flag ($spurious_{possible}$) is notified if |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}.

Step 4:

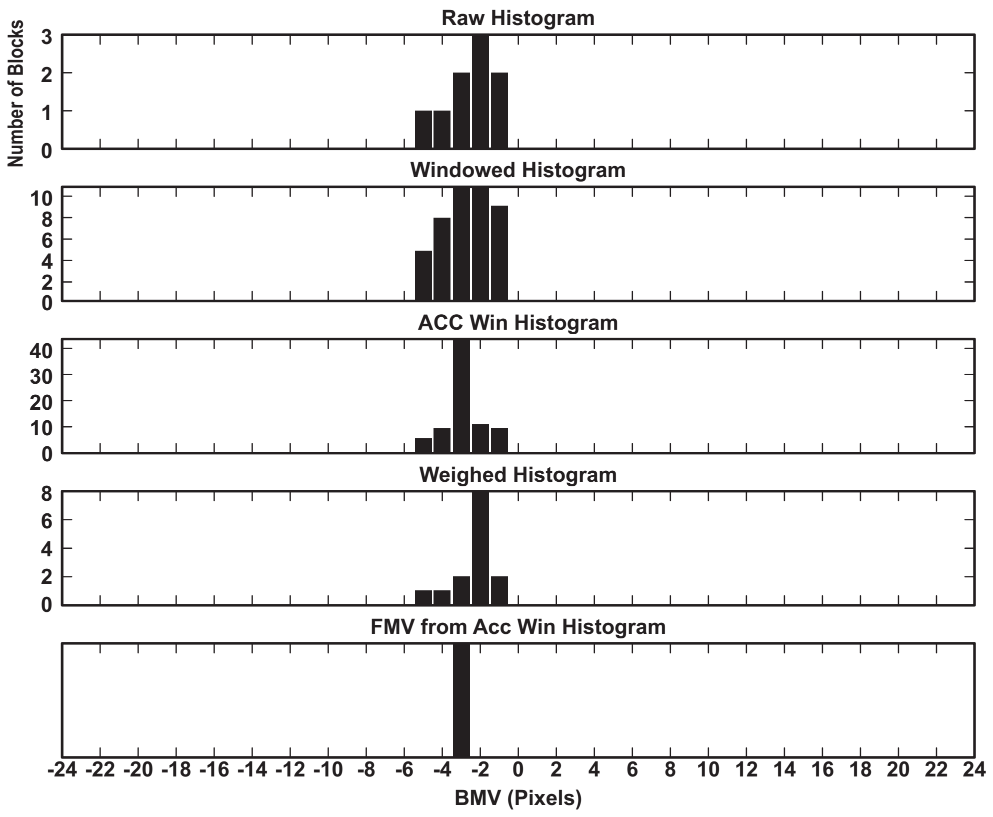
If the accumulated histogram based FMV detection failed to pick the FMV or the flag ($spurious_{possible}$) is set, then the filtered histogram based FMV detection is triggered. FMV is detected if the filtered histogram yields a single maxima with maxima value more than or equal to a dynamic threshold ($b_0+b_1 \times numBlk_{FiltHistThr}$), where $numBlk_{FiltHistThr}$ is the maximum of $numBlk_{RawHistThr}$ or $maxVal_{raw}$. If the filtered maximum filtered histogram bar value is less than or equal to max_{raw}, then the flag ($spurious_{possible}$) is set.

Step 5:

If flag (spurious$_{possible}$) is set by step 3 or step 4, or if the difference of FMV detected at step 3 (if available) and step 4 is more than threshold (BMV$_{max}$/12) then set flag (spurious$_{detected}$). If flag (spurious$_{possible}$) is set by step 3 and difference of FMV detected at step 3 and 4 is within threshold (BMV$_{max}$/12), use the FMV from filtered histogram (step 4).

Figure 14 provides an example of FMV estimation using histograms in a frame.

Figure 14. Example of FMV Estimation Using Histogram



Extreme motion jitter compensation can be disabled by treating the estimated FMV as zero. The extreme motion compensation disable facility passes FMV as zero during the attenuation period in FMV smoother. Extreme FMV values may result due to flicker introduced luma variation across successive frames. Thus, this feature may be used in poorly tuned cameras where there will be flicker in video being captured under artificial lighting conditions.

3.2.6 Spurious FMV Detector

If FMV is not detected from the histograms or no BMVs are detected in the current frame, the fame is marked as having no FMV (FMV_INVALID). In case of luminance variation on input frame, the boundary signals would have picked up the gain corresponding to Luma variation. Due to non-linearity in the SAD computation, this would result in a negative or positive slope SAD vector or a SAD vector having a pseudo minimum. To avoid spurious FMV due to luma variation, several low computational measures are used to detect spurious FMV.

1. If the BMVs within a frame have larger variation across blocks and if the majority of BMVs are not close to each other, the frame is marked as not having FMV (FMV_INVALID). Otherwise, the other methods described below are used for spurious FMV detection.

   If BMV variation within the frame (bmv_{range}) is more than the dynamic threshold (bmv_{Thr}) computed as in (10), and (11), BMV concentration is checked. If BMV concentration is not centered on estimated FMV, the frame is marked as not having FMV (FMV_INVALID). A window ($bmv_{DynamicThr}$) is established around FMV. The number of BMVs outside the window should be greater than the truncated value of the number of BMVs inside the window, scaled by 0.5 for the frame to be classified as not having FMV.

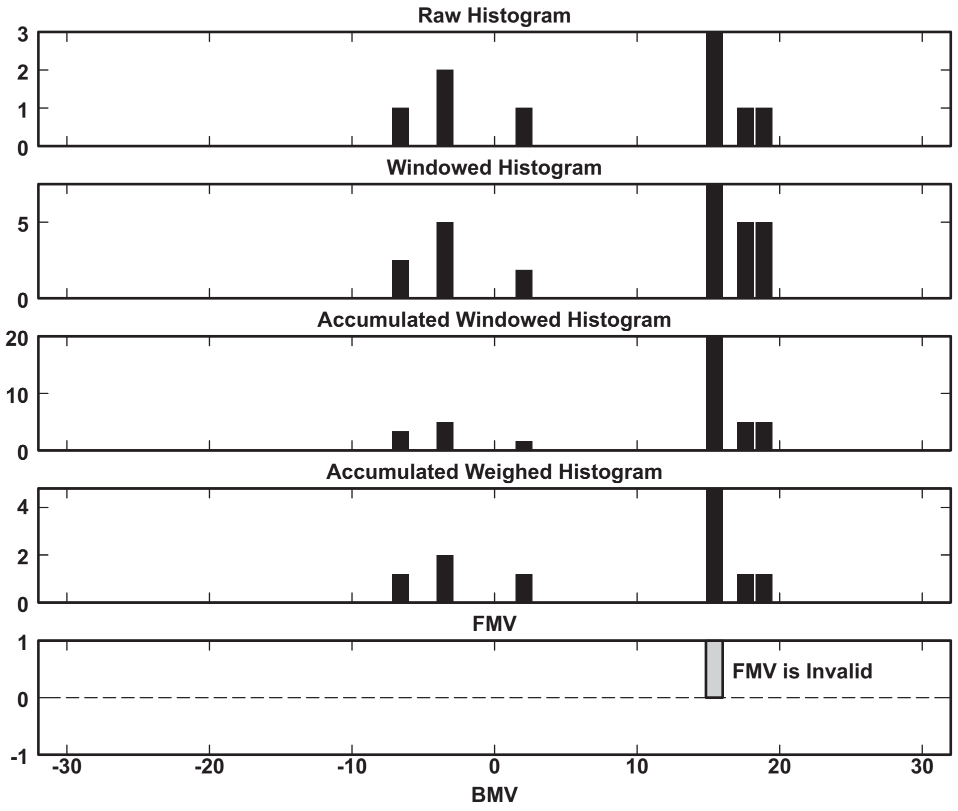
$$bmv_{DynamicThr}[n] = max\begin{bmatrix} bmv^{SF}_{DynamicThr} \text{ x } bmv_{DynamixThr}[n-1], \\ bmv^{SF}_{range} \text{ x } max(bmv_{range}[n-m]) \end{bmatrix},$$

$$\begin{aligned} &\text{where } m = 1,2,3,..., \text{MAX\_PAST}(=5) \\ &n = \text{currentframe number} \\ &bmv_{range}[n] = bmv_{max}[n] - bmv_{min}[n] \\ &bmv^{SF}_{DynamicThr} = 0.9 \\ &bmv^{SF}_{range} = 1.25 \end{aligned} \tag{10}$$

$$\begin{aligned} &bmv_{Thr}[n] = saturate\_round\,(bmv_{DynamicThr}[n], bmv^{max}_{Thr}\ bmv^{max}_{Thr}), \\ &\text{where } bmv^{max}_{Thr} = round(0.6 \text{ x } 2 \text{ x } BMV_{max}) \\ &bmv^{min}_{Thr} = round(0.35 \text{ x } 2 \text{ x } BMV_{max}) \end{aligned} \tag{11}$$

Figure 15 is an example for spurious FMV detection using this procedure.

Figure 15. Example of Spurious FMV Elimination Using BMV Spread



2. If the deviation of BMV within a frame (bmvrange) is more than the dynamic threshold (BMV_{Thr}) and |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}, the FMV is marked as invalid (FMV_INVALID). This method checks for a positive or negative slope SAD vector with high BMV deviation within a frame to detect spurious FMV.
3. If any of the FMV in the past five frames is invalid and |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}, the FMV is marked as invalid (FMV_INVALID). If a frame is marked as invalid using this condition alone then the frame invalidity of this frame is not used for making the future frames invalid via this step.
4. If the number of BMVs outside the window is greater than the truncated value of the number of BMVs inside the window scaled by 0.5 and |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}, the FMV is marked as invalid (FMV_INVALID). This step checks for the number of BMVs closer to FMV and any high BMV deviation within a frame to detect spurious FMV.

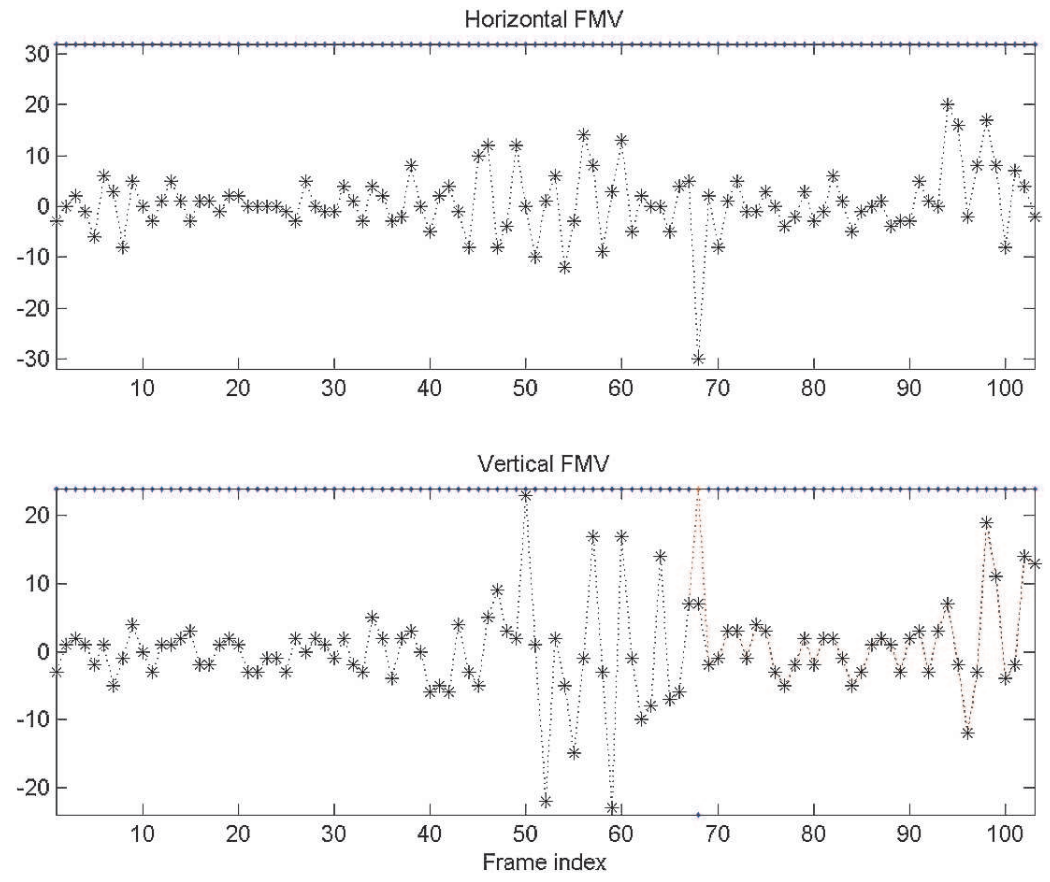
   If any frame in the MAX_PAST past frames (MAX_PAST=5) is invalid (FMV_INVALID) and |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}, the FMV is marked as invalid (FMV_INVALID). If this condition alone designated the frame as INVALID, a flag (SKIP_INVALID_HIST) is set to handle the state update. This step looks for any one of the recent frames declared as invalid and with high BMV deviation to designate the current frame as invalid (FMV_INVALID).
5. If the flag ($spurious_{possible}$) is set and |FMV| is positioned close to BMV_{max} or within the 15% limit of BMV_{max}, the FMV is marked as invalid (FMV_INVALID).
6. If the flag ($spurious_{detected}$) is set, the FMV is marked as invalid (FMV_INVALID). This is determined by the result of the filtered histogram and the accumulated histogram is the same (within a range) .

If the frame is not marked as invalid and an FMV is available for the current frame, the FMV is flagged as valid (FMV_VALID).

If any of the above steps forced the FMV to be marked as invalid, a counter is incremented. Otherwise, the counter is reset. If the counter is more than or equal to the threshold (value of 2), the BMVs within a frame have a smaller variation across blocks and the majority of the BMVs are close to each other, then the state variables holding history of $bmv_{range}[n-m]$ and $bmv_{DynamicThr}[n-1]$ are updated. Figure 16 shows the vertical and horizontal FMV with spurious FMV detection and smoothing at frame number 68 for vertical FMV.

Each of the conditions for detecting spurious FMV can be disabled or enabled independently, or collectively with any possible combination, depending on the nature of video input.

Figure 16. Example of FMV with Spurious Data Handling



3.2.7 FMV Smoother

The FMV smoother gradually increases attenuation of previous valid FMV for use as the current FMV where the current FMV is invalid. The attenuation is carried out until the FMV reaches 0. This step can possibly compensate jitter in an output stabilized frame when the current FMV is invalid and takes as 0. In addition, this logic gradually releases attenuation on transition from invalid FMV to valid FMV to avoid possible jerks.

The FMV smoother can be disabled if not required by the application scenario. By default, the smoother is disabled since the smoother is preferred to be present at output stabilization coordinate. The main purpose of the smoother at the disabled state is to detect spurious FMV which are not detected by the spurious FMV detector.

The computational steps are as follows:

1. If an FMV is invalid, the last valid FMV is gradually attenuated until the FMV reaches 0. The attenuations factors are 1, 0.75, 0.5, 0.25 and 0. The attenuation is gradually increased (attenuation factor is decreased) on each successive invalid frame until the FMV reaches 0.
2. If there is a valid FMV when the attenuation is in progress (FMV has not been fully attenuated to 0) **or** if any one of the past N (5) FMV was invalid, and if the FMV deviation (*fmv*$_{diff}$) between the current FMV and the previous FMV is above the dynamic threshold (fmv$_{DynamicThr}$) then the FMV attenuation is continued using the previous valid FMV. The FMV of the current frame is treated as invalid. The dynamic threshold is computed as in (12) and (13).

$$fmv_{DynamicThr}[n] = max\begin{bmatrix} fmv^{SF}_{DynamicThr} \times fmv_{DynamixThr}[n-1], \\ fmv^{SF}_{range} \times max(bmv_{diff}[n-m]) \end{bmatrix},$$

where m = 1,2,3,..., MAX_PAST(= 5)

n = currentframe number

$fmv_{diff}[n] = fmv[n] - fmv[n-1]$

$fmv^{SF}_{DynamicThr} = 0.9$

$fmv^{SF}_{range} = 1.5$ (12)

$$fmv_{Thr}[n] = saturate\_round\,(fmv_{DynamicThr}[n], fmv^{max}_{Thr}, fmv^{max}_{Thr}),$$

where $fmv^{max}_{Thr} = round(0.8 \times 2 \times FMV_{max})$

$fmv^{min}_{Thr} = round(0.4 \times 2 \times FMV_{max})$ (13)

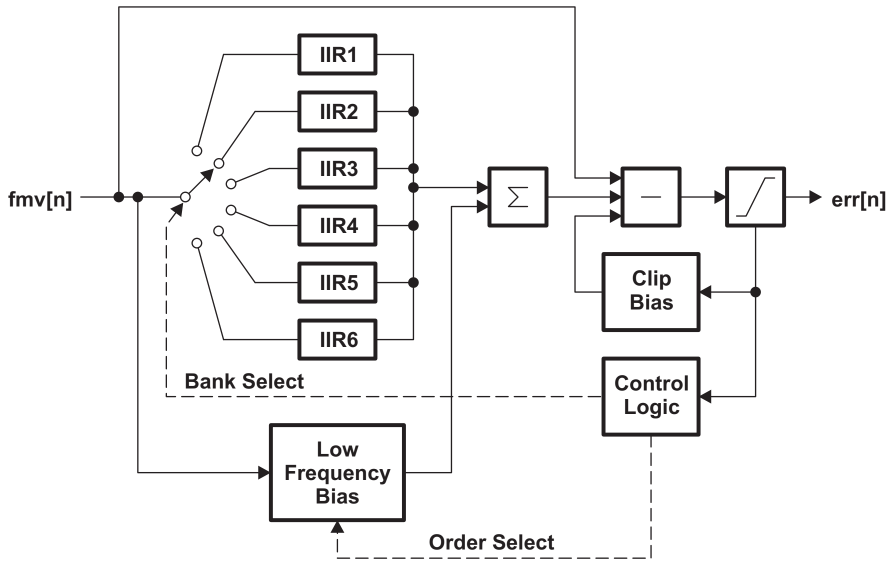
If a disabled state occurs, the FMV is marked as invalid (FMV_INVALID) if any of the past N FMV was invalid (note that the FMV marked as INVALID by the smoother does not contribute to this decision) and if the FMV deviation (*fmv*$_{diff}$) between the current FMV and the previous FMV is above the dynamic threshold (*fmv*$_{DynamicThr}$).

3. If the FMV deviation (*fmv*$_{diff}$) is less than or the same as the dynamic threshold (*fmv*$_{DynamicThr}$) and attenuation is in progress, then the attenuation is gradually removed until the attenuation factor reaches unity. The attenuation factor is applied on the current frame's valid FMV. The FMV of the current frame is treated as valid in this case.

3.3 Unwanted Motion Estimation (UME)

In this stage, the unwanted motion of the frame is estimated. The unwanted motion is converted into (x, y) coordinates of the top-left corner of the current frame that is to be used for cropping by FMC. The block diagram is shown in Figure 17.

Figure 17. UME Block Diagram



The UME module consists of a bank of biquad filters: low frequency bias, clip removal bias, control logic, and output formatter. The biquad filters are used for smoothing the FMV signal. The smoothed signal is used for estimating motion jitter. Different orders of filter are used so as to filter out the jitter to different levels. For example, panning (motion is intentional) video sequences need to track fast and steady shots (no desired motion) require no tracking.

As shown in Figure 18, the low frequency bias module, which consists of an averaging filter, is used for compensating any delay introduced by the biquad filters that may result in errors in tracking/compensation. Control logic is used for tuning the biquad filter parameters (filter selection) and low frequency bias module (order of averaging filter). Since allowable motion compensation is limited by the amount pixels excluded from the original frame during cropping, the jitter to be compensated is limited to a threshold. To minimize the effect of clipped jitter compensation, a steady bias component is generated using the clip removal bias module.

Figure 18. UME Flowchart

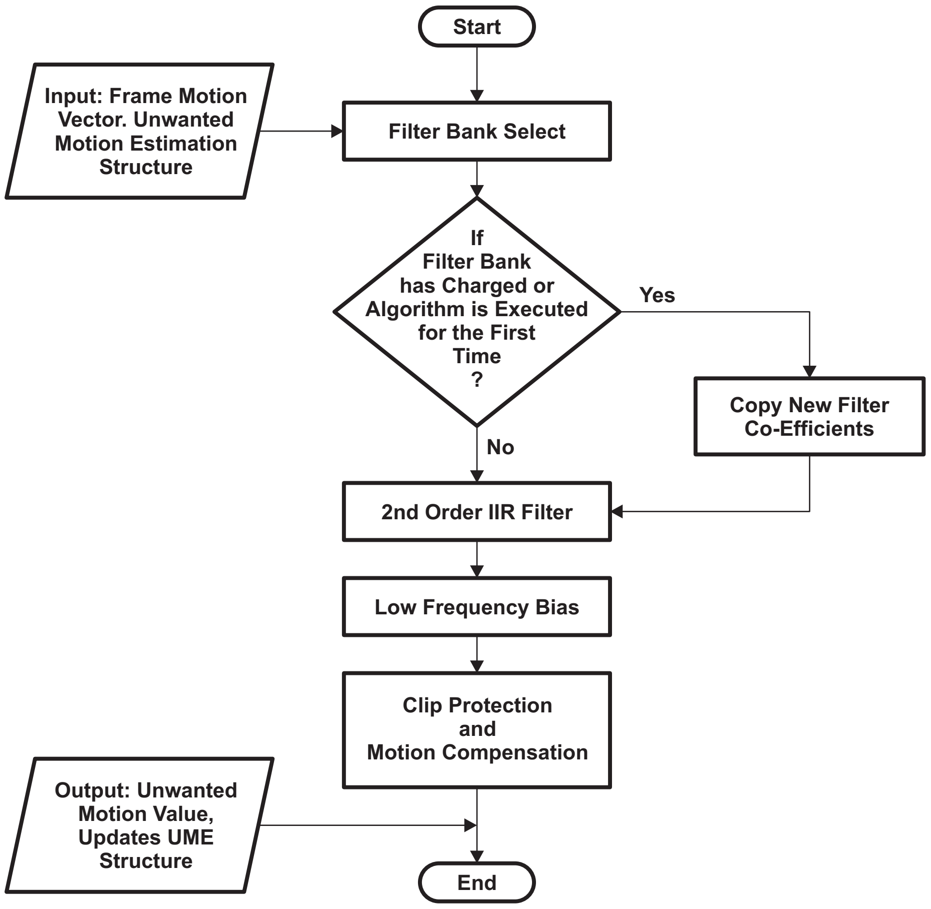
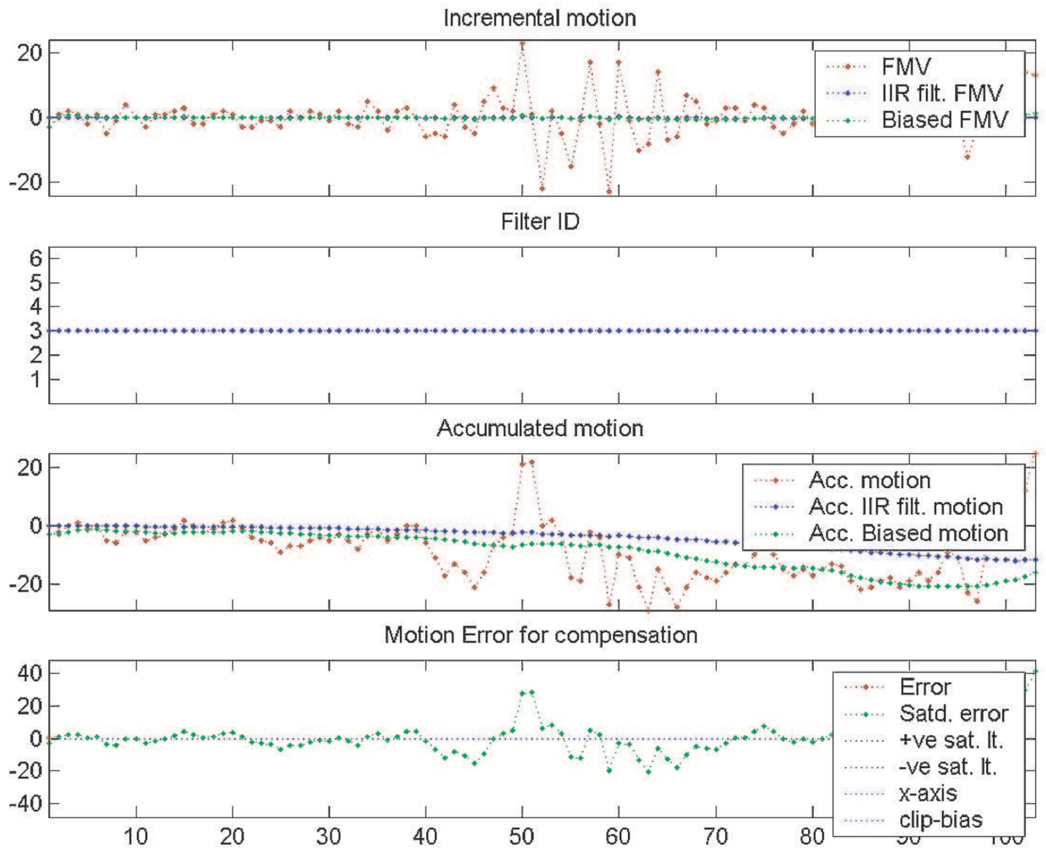

Start
Input: Frame Motion Vector. Unwanted Motion Estimation Structure
Filter Bank Select
If Filter Bank has Charged or Algorithm is Executed for the First Time ?
Yes
Copy New Filter Co-Efficients
No
2nd Order IIR Filter
Low Frequency Bias
Clip Protection and Motion Compensation
Output: Unwanted Motion Value, Updates UME Structure
End

Figure 19 illustrates the functioning of jitter motion compensation using all the waveforms processed inside UME.

Figure 19. UME Waveforms Example



The computational steps in the UME flowchart as shown in Figure 18 are:

1. Estimate if the filter bank needs to be changed.
2. If the filter bank changes, the filter coefficients are copied from the filter bank.
3. The frame motion vector obtained from the previous stage is passed through a second order IIR filter.
4. Low frequency bias is computed.
5. The motion to be compensated is computed. The coordinates for cropping the window are computed. Clip protection is done for the coordinates if the cropping window is large.

If FMV is invalid, the UME stage is bypassed. The UME stage retains the state so that the previously used compensation coordinate is retained. This is essential to avoid any jerks introduced by the algorithm when FMV estimation is inaccurate. That is, the inherent jitter in stream is passed through when FMV is invalid.

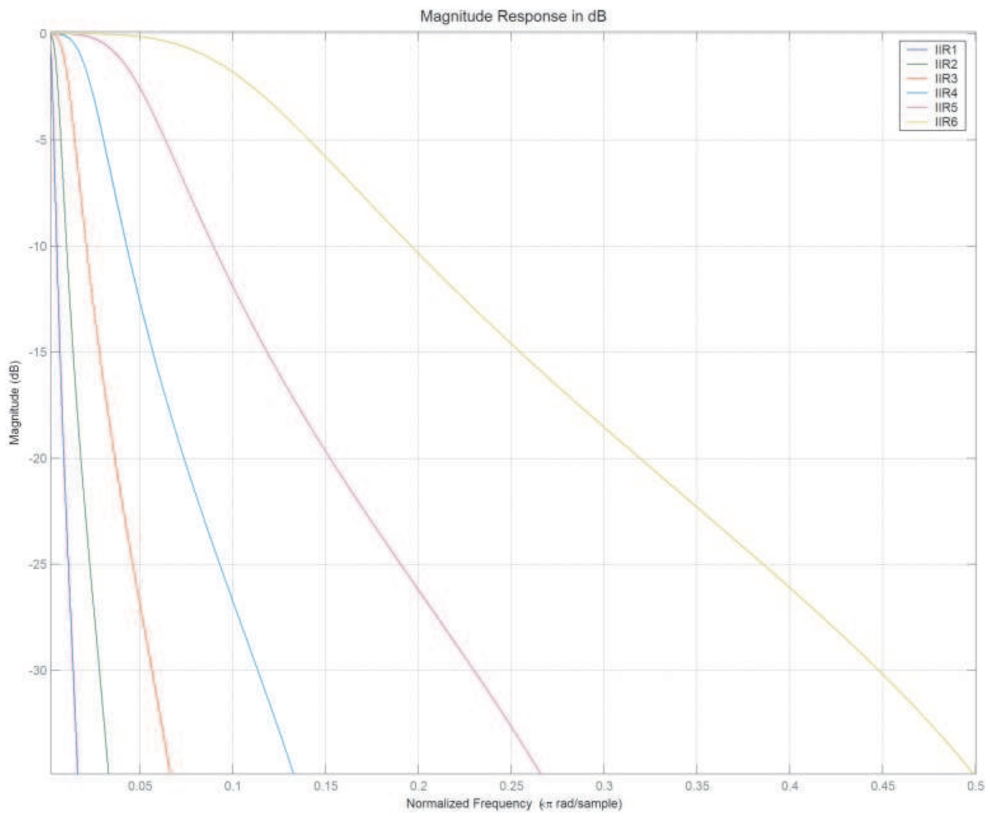
If FMV is invalid for two consecutive frames, the UME IIR filter state and settling timer are reset. All other parameters like filter selection, low frequency bias and clip protection are retained as that of the most recent frame having valid FMV.

In cases where FMV is invalid for more than a specified number of frames (N=6) consecutively, the states of BME and FME are reset to the initial values. Additionally, default filter bank settings are chosen on UME and UME is bypassed. No other parameters in UME are altered under such conditions so that the compensation coordinate is retained as the same as that of last frame that contained valid FMV, until the next valid FMV. If the consecutive frame invalid is due to disabling of extreme jitter compensation, FME parameters only are reset.

3.3.1 IIR Filter Bank

Second order (biquad) IIR filters with frequency responses as shown in Figure 20 are used for building the filter bank.

Figure 20. Magnitude Response of Filter Bank



The filter is implemented as in (14).

$$a0 \times y[n] = b0 \times x(n) + b1 \times x(n-1) + b2 \times x(n-2) - a1 \times y(n-1) - a2 \times y(n-2),$$
where b0, b1, b2, a0, a1, a2 are filter coefficients. (14)

The coefficients {b0, b1, b2, a0, a1, a2} of the filters are as follows:

```
{1.768435e-002,   -3.527182e-002,    1.768435e-002,
1.0000000e+000,   -1.986156e+000,    9.862525e-001}, // IIR1

{1.763447e-002,   -3.488359e-002,    1.763447e-002,
1.0000000e+000,   -1.972291e+000,    9.726761e-001}, // IIR2

{1.767785e-002,   -3.382677e-002,    1.767785e-002,
1.0000000e+000,   -1.944431e+000,    9.459598e-001}, // IIR3

{1.833034e-002,   -3.058438e-002,    1.833034e-002,
1.0000000e+000,   -1.887696e+000,    8.937721e-001}, // IIR4

{2.207892e-002,   -1.908483e-002,    2.207892e-002,
1.0000000e+000,   -1.765662e+000,    7.907348e-001}, // IIR5

{4.104093e-002,    2.736051e-002,    4.104093e-002,
1.0000000e+000,   -1.483873e+000,    5.933153e-001} // IIR6
```

The cut-off frequencies of the low pass filters are at [0.5/30; 1/30; 2/30; 4/30; 8/30; 15/30], with 1.0 corresponding to half the sample rate. Chebyshev type II order filter with attenuation of 35 dB is used to achieve faster roll-off [12], [13].

Filter Bank Switching

The settling time of the filters are [680; 340; 170; 84; 42; 21] samples, respectively. The IIR1 filter is ideal for steady shots with motional jitter and IIR6 is best suited for high panning motion with jitter. When the pass band is narrow, filter response is slow and settling time is large. When the pass band is wide, filter response is fast and settling time is low. Generally, the filter is switched to new parameters only when the current filter is in a steady state.

If the filter has reached steady state and if there was any clipping of jitter compensation during the filter transience period, the filter cutoff frequency is increased. For example, the selection is changed from IIR3 to IIR4. During the cut-off frequency change, the previous filter stage state is retained to minimize the transience state in the new filter stage. If the current filter is IIR1 or IIR2 during which clipping occurred, and if the filter has exceeded the settling time of the IIR3 filter then the filter is switched directly to IIR4. This is done to avoid a large period of clipping caused by tight control. For example, a steady state shot may suddenly transition to panning motion and the filter switch is required sooner to minimize the amount of clipping caused by the start of panning due to tight tolerances of the IIR1 and IIR2.

If the filter has reached steady state and if there was no clipping during the filter transience, the cut-off frequency of the filter is decreased. For example, the filter selection is changed from IIR3 to IIR2. The filter states are unaltered during the switching.

This filter switching procedure is repeated until the least or highest cut-off frequency is reached. The filter switching logic is active throughout the duration of motion jitter stabilization.

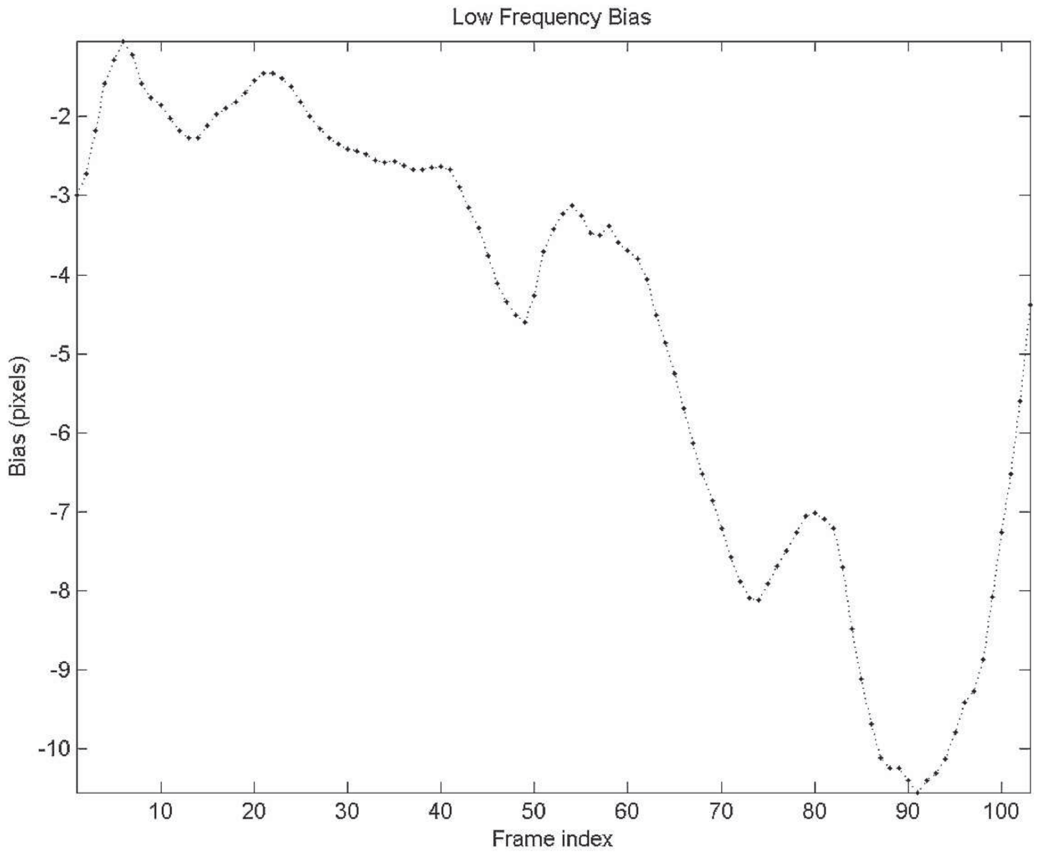
3.3.2 Low Frequency Bias

The low frequency bias is estimated using an averaging filter on the difference between the actual motion (accumulated FMV) and the absolute motion (accumulated IIR output). The difference is a low frequency bias caused by lag in IIR filter and filter switching. When this difference exceeds a limit, it will cause saturation of motion jitter compensation. The purpose of low frequency bias is to avoid saturation while being able to provide effective jitter removal. The stage reduces the motion lag caused by the second order filter during panning sequences and intentional motion of the camera.

The averaging filter order is dynamically computed based on whether motion compensation is saturated. On start up the filter order gradually increases from 1 to max order (32). In case the absolute motion compensation vector is more than the threshold (BMV_{max}/16) and the counter tracking this event reaches a limit (16), the filter order is decreased by 1 on each frame. The minimum order is limited to a minimum limit (4). Similarly, if the counter is 0, the order is increased until the order reaches the maximum allowed (32). The counter is incremented each time the motion compensation vector has the same sign and it exceeds the threshold (BMV_{max}/16), or if there is no clipping of the motion compensation vector. The counter is less than the limit (16) and if the previous and current frame motion compensation factor were more than threshold (BMV_{max}/16) on opposite directions, the counter is set to 0. Otherwise, the counter is decremented by 1.

Figure 21 is an example of low frequency bias waveform.

Figure 21. Low Frequency Bias Waveform Example



The computational steps are as follows:

- The absolute motion from FME stage (*AbsoluteMotion*) is obtained by summing all FMVs as shown in (15). Similarly the absolute motion of IIR output (*AbsoluteFilterMotion*) is obtained as shown in (16).

$$AbsoluteMotion = \sum_{0}^{Present} FMV, \quad \text{where } FMV \text{ is input to IIR filter} \tag{15}$$

$$AbsoluteFilterMotion = \sum_{0}^{Present} FiltOut, \quad \text{where } FiltOut \text{ is output from IIR filter} \tag{16}$$

- The absolute motion and absolute filter motion are prevented from overflowing over time by subtracting a common factor (*cf*) from both variables as shown in (17). The subtraction factor is the minimum integer value among both variables.

$$\begin{aligned} cf &= \min(AbsoluteMotion, \text{floor}(AbsoluteFilterMotion)) \\ AbsoluteMotion &= AbsoluteMotion - cf \\ AbsoluteFilterMotion &= AbsoluteFilterMotion - cf \end{aligned} \tag{17}$$

- The difference (*Diff*) between *AbsoluteMotion* and *AbsoluteFilterMotion* is stored in a circular buffer of length equal to the maximum order of the averaging filter (32) as shown in (18).

$$Diff[n] = AbsoluteMotion - AbsoluteFilterMotion \tag{18}$$

- The low frequency bias is computed by taking the average of available samples in the LFB array and averaging the obtained value with the previous value of the low frequency bias as shown in (19). On start-up, use the FMV of the current frame for initializing *LFB[n-1]*.

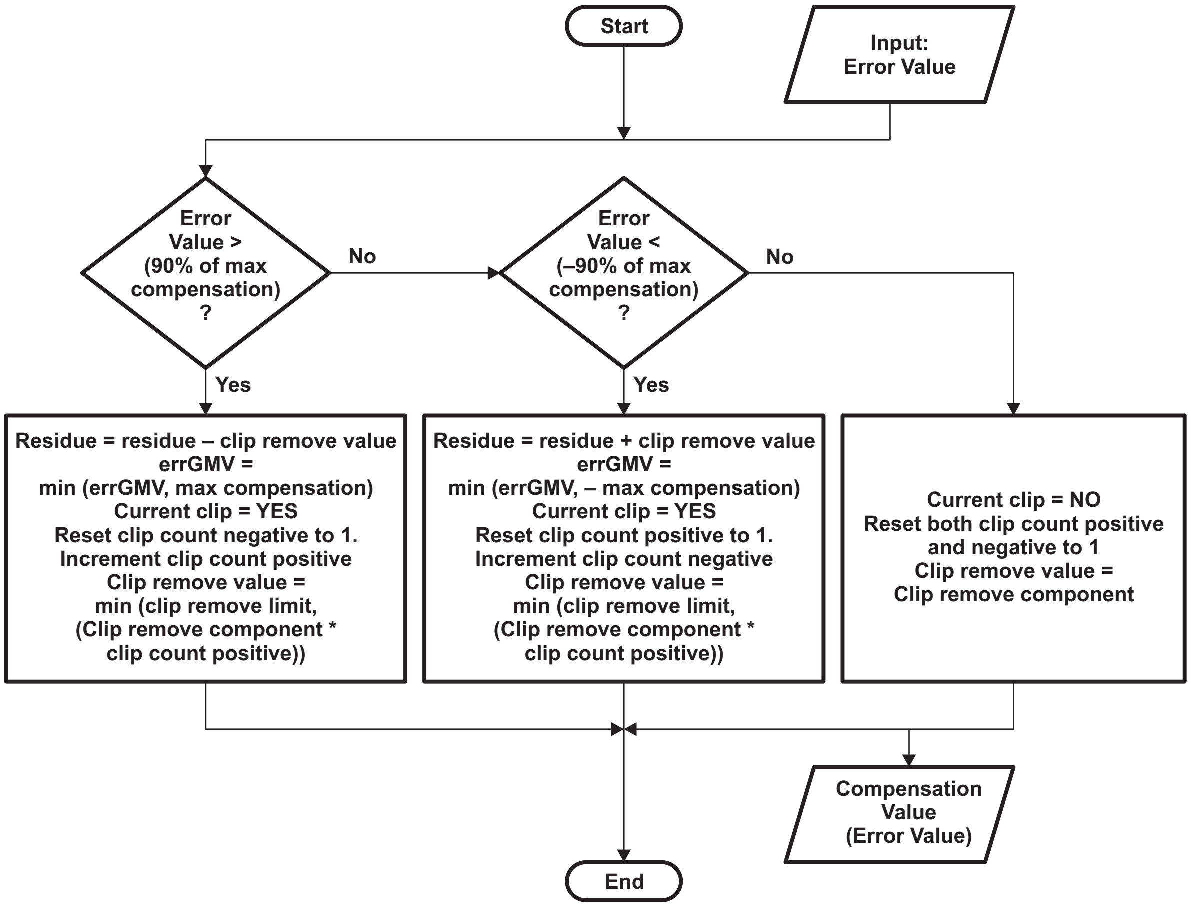
$$LFB[n] = \frac{\frac{1}{order} \times \sum_{m=0}^{m=order-1} Diff[n-m] + LFB[n-1]}{2}$$

where order = order of averaging filter (19)

3.3.3 Clip Bias

If the motion compensation vector is more than 90% of the compensation limit, clip removal bias is used to minimize the effect of clipping as shown in Figure 22.

Figure 22. Compensation Coordinate Estimation and Clip Bias Estimation


Start
Input: Error Value
Error Value > (90% of max compensation) ?
No
Error Value < (–90% of max compensation) ?
No
Yes
Yes
Residue = residue – clip remove value
errGMV = min (errGMV, max compensation)
Current clip = YES
Reset clip count negative to 1.
Increment clip count positive
Clip remove value = min (clip remove limit, (Clip remove component * clip count positive))
Residue = residue + clip remove value
errGMV = min (errGMV, – max compensation)
Current clip = YES
Reset clip count positive to 1.
Increment clip count negative
Clip remove value = min (clip remove limit, (Clip remove component * clip count positive))
Current clip = NO
Reset both clip count positive and negative to 1
Clip remove value = Clip remove component
Compensation Value (Error Value)
End

The minimum and maximum threshold clip removal bias are 0 and $2 \times BMV_{max}/12$, respectively. The clip removal bias least count is $2 \times BMV_{max}/48$. On each consecutive frame that needs clip removal bias in the same direction, the clip removal bias (*ClipBias*) is linearly increased as shown in (20). The sign of *ClipBias* (*sign*) is opposite in sign of the motion compensation vector. For example, if there is negative clipping on motion compensation, apply positive clip removal bias.

$$ClipBias = sign \times max \left[numClips \times \frac{2 \times BMV_{max}}{48}, \frac{2 \times BMV_{max}}{12} \right],$$

where *numClips* is the number of consecutive frames exceeding motion compensation limit in the same direction (20)

3.3.4 Unwanted Motion Computation

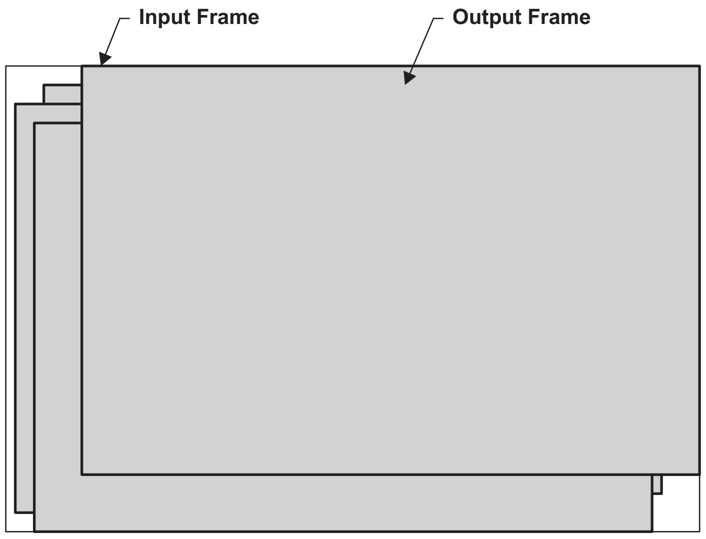
The error value or compensation vector (*Err*) is computed as shown in (21). The error or compensation vector in a vertical and horizontal direction can be mapped to the compensation coordinate for stabilization.

$$Err[n] = AbsoluteMotion[n] - AbsoluteFiltMotion[n] - LFB[n] + ClipBias[n] \tag{21}$$

3.4 Frame Motion Compensation (FMC)

The simplest FMC can be cropping of the window for display from the captured frame. The captured frame is larger than the window being recorded or displayed, large enough to allow for FMC as shown in Figure 23. By changing the coordinate (h, v) for the window start position, the unwanted motion can be compensated. The window is positioned in the opposite direction of the jitter for compensation. The horizontal window position can be adjusted to the nearest even pixel to avoid chroma sample reversal, as can be seen from the chroma sample arrangement $YC_bYC_rYC_bYC_r$… in YCbCr 422 format.

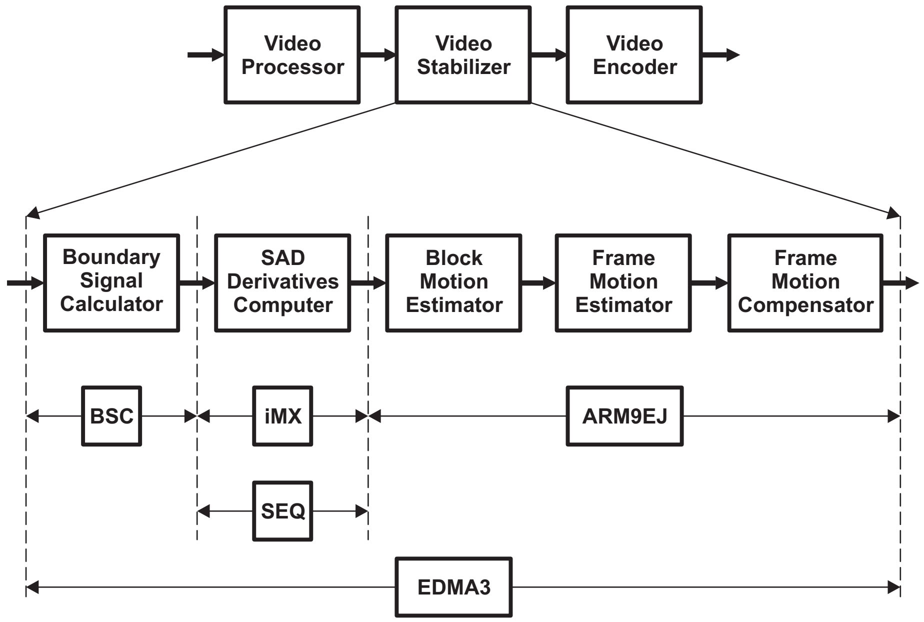
Figure 23. Motion Compensation by Cropping



4 Hardware, Software Partition

In an ARM9EJ processor performing fixed point computations, the BSC amounts to 69% of the computational load, SAD computation is 25%, and BME and FME computation are the remaining 6%. The BSC, SAD and motion estimation (ME) computations require 51 MHz and 200 MHZ for (Quarter VGA) QVGA and (Video Graphics Array) VGA, respectively [14], [15]. This necessitates hardware acceleration for performing BSC and SAD computation as shown in Figure 24.

Figure 24. Hardware-Software Partition



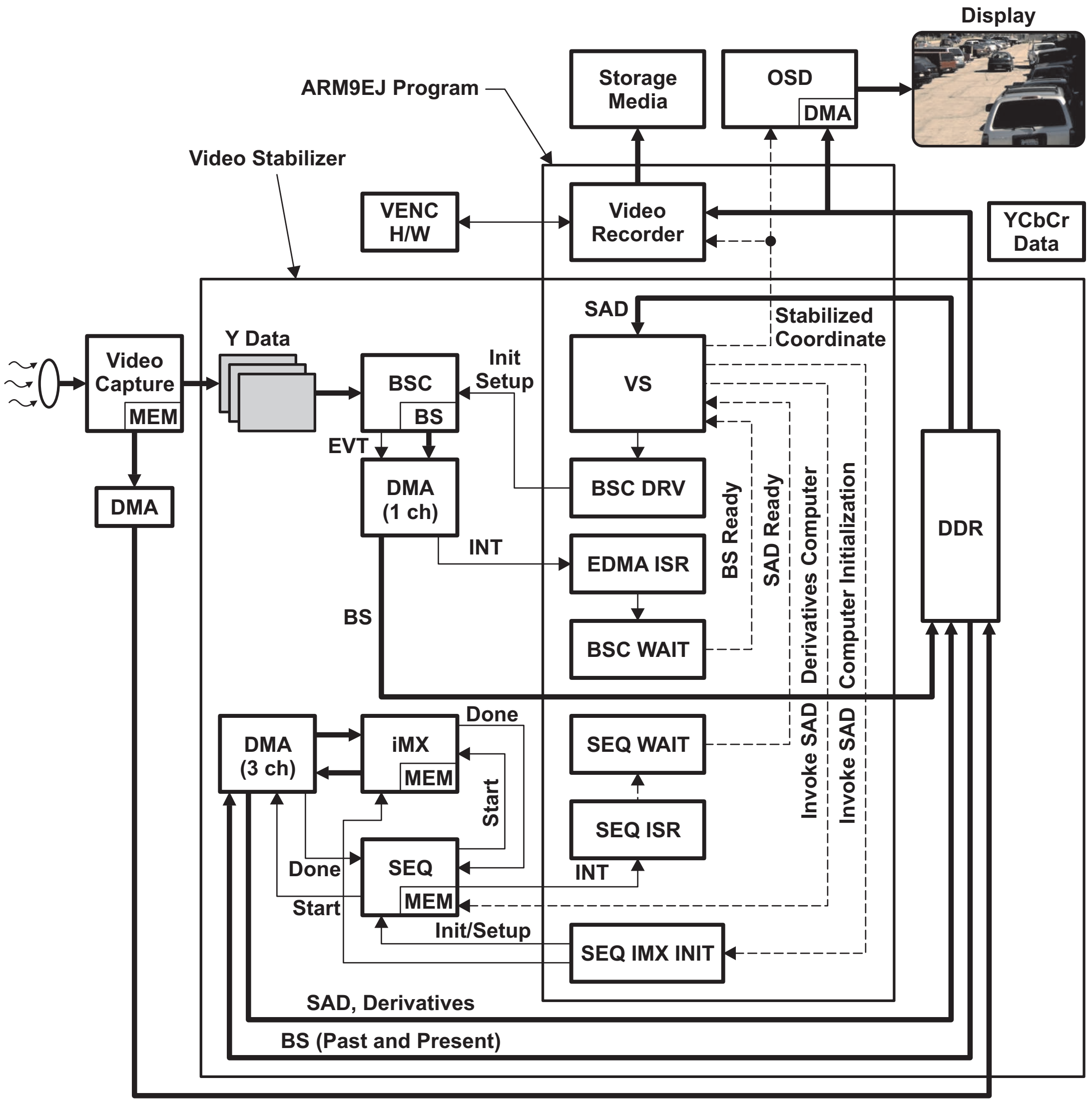
The DM355 System on a Chip (SoC) for DSC provides Boundary Signal Calculator (BSC) hardware accelerator and programmable SIMD image processing engine (iMX) for BS and SAD computations, respectively [16], [17], [18]. Though BS computation can be performed using iMX, BSC is efficient and offloads the iMX engine. The task of scheduling and triggering iMX execution and the direct memory access (DMA) peripheral is handled by programmable sequencer (SEQ) to offload control code from the ARM9EJ involved in managing the coprocessors and accelerators. The DM355 includes the ARM9EJ core for BME and FME. The ARM9EJ is efficient in performing control-related code like BME and FME. If cropping is required for FMC, iMX can be used for the rearrangement of data as may be required in the YC_bC_r 422 format. If there is window position adjustment in the YC_bC_r 422 format video sequence, the odd horizontal pixel position will reverse the C_b and C_r position in $YC_bYC_rYC_bYC_r$… sample sequence. Interpolation of adjacent pixels, on iMX, can be used to estimate the chroma samples.

The video stabilizer apparatus is shown in Figure 25. The Y samples of the video frames are advanced one pixel at a time to the BSC. The BSC performs the accumulation of Y samples horizontally and vertically to generate BS in memory residing in the BSC module. The BSC can be programmed to generate one or more boundary signals vertically and horizontally. The generated boundary signal can be divided horizontally or vertically for vertical and horizontal boundary signals, respectively, to produce N×N block division. The computation of BSC is stopped for the last few lines of the video frame in order to allow for the copying of BS to DDR. The BSC generates the completion INT, which triggers the DMA transfer of BS to DDR. On transfer completion, the DMA generates INT to the ARM9EJ.

The ISR triggers the video stabilization algorithm to initiate the SAD computation using iMX. Once the SAD computation is initiated, the ARM9EJ is free to do other functions. An RTOS scheduler can schedule the next task. One SAD vector is computed per execution of the iMX program. The first and second derivatives are computed for each SAD vector. In addition, the minimum SAD position, minimum and maximum SAD value, and maximum second derivative position are computed using the iMX. The positions are used for estimating BMV. The maximum SAD value is used for estimating the dynamic right shift value to be applied to the SAD vector for the next frame.

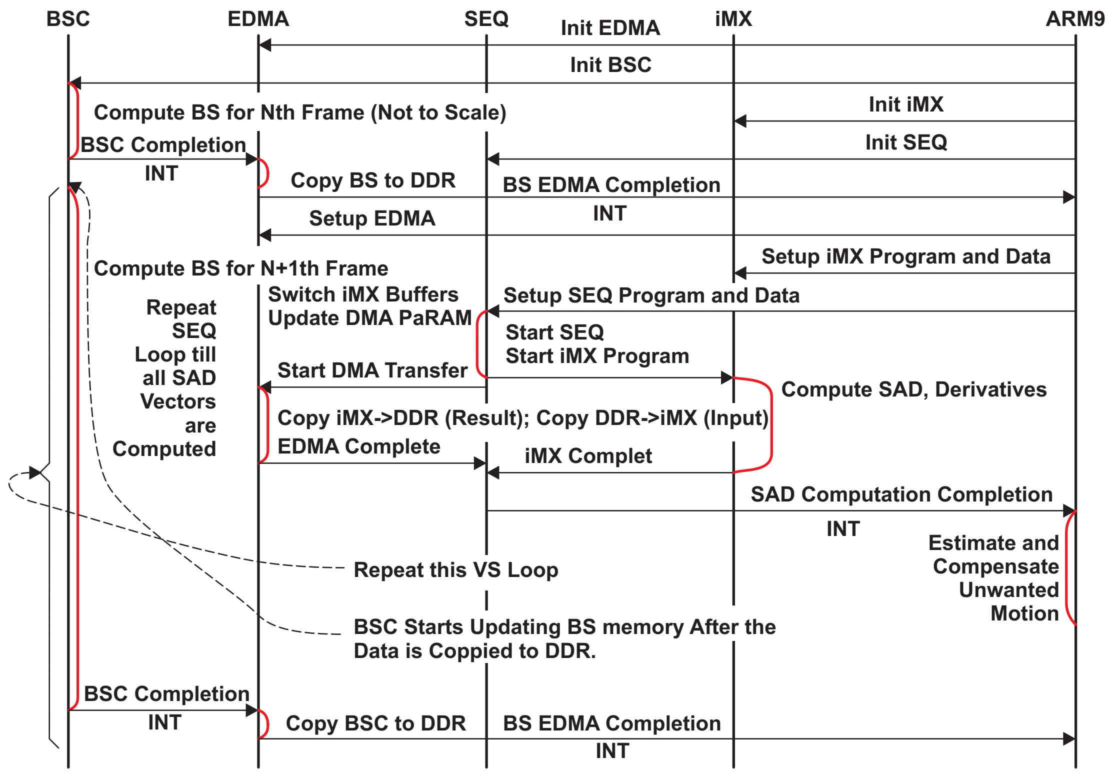
When the iMX is processing data in one image buffer, the DMA is fetching input for the next SAD vector computation into the second iMX buffer. In the mean time, the previous result is transferred to DDR from the third iMX buffer. The buffers cyclically switch to avoid any waits for input data fetch or output data store before executing the next iMX program. Thus, the input BS data fetch for the next SAD computation, current SAD computation, and past SAD result store are happening concurrently. When the DMA and iMX are performing their functions, SEQ is waiting for completion sync of DMA and iMX. On receiving completion sync, SEQ switches the buffers cyclically and initiates DMA transfers and starts the iMX program. On completion of the SAD computation, the SEQ INT ARM9EJ and VS algorithm are ready to run again. Now the rest of the computation is carried out in the ARM9EJ to find the video stabilization coordinates. The coordinates are passed onto the video recorder and video display threads.

Figure 25. Video Stabilizer Apparatus



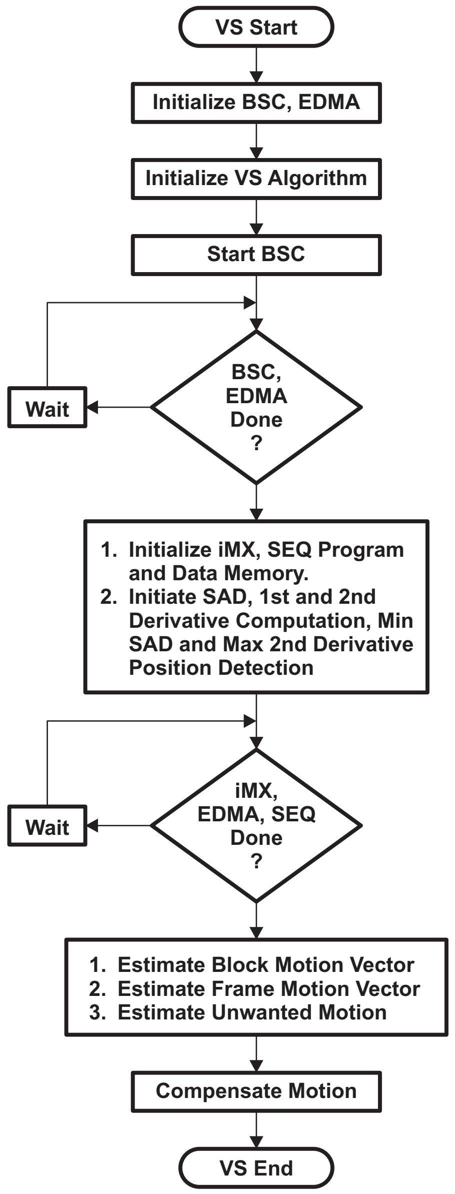
[Figure 26](#) illustrates the sequence of events during the stabilization progress. The BSC hardware is active for most of the time except when the BS data is being copied to DDR. The SEQ and iMX programs are briefly active to perform SAD computation. The SEQ and iMX hardware can be multiplexed to perform video encode, noise filter, etc when not in use by the video stabilizer. ARM9EJ is active for a short time to perform block motion estimation, frame motion estimation and coordinate computation for compensating unwanted motion.

Figure 26. Video Stabilizer Sequence Diagram


BSC
EDMA
SEQ
iMX
ARM9
Init EDMA
Init BSC
Init iMX
Init SEQ
Compute BS for Nth Frame (Not to Scale)
BSC Completion
INT
Copy BS to DDR
BS EDMA Completion
INT
Setup EDMA
Setup iMX Program and Data
Compute BS for N+1th Frame
Switch iMX Buffers
Update DMA PaRAM
Setup SEQ Program and Data
Repeat SEQ Loop till all SAD Vectors are Computed
Start SEQ
Start iMX Program
Start DMA Transfer
Compute SAD, Derivatives
Copy iMX->DDR (Result); Copy DDR->iMX (Input)
EDMA Complete
iMX Complet
SAD Computation Completion
INT
Estimate and Compensate Unwanted Motion
Repeat this VS Loop
BSC Starts Updating BS memory After the Data is Coppied to DDR.
BSC Completion
INT
Copy BSC to DDR
BS EDMA Completion
INT

The flowchart in Figure 27 provides a top level view of the ARM9 program required in the video stabilizer. The program initializes the BSC, DMA, SEQ and iMX hardware. Once the video stabilization is enabled, the BSC hardware is started. On receiving the EDMA transfer completion of BS signal, the SEQ and iMX program and data memory are initialized. Then, ARM transfers control to SEQ. On completion of the SAD computation, SEQ signals the VS algorithm in the ARM9EJ to estimate unwanted motion. ARM9EJ performs block and frame motion estimation, and compensates unwanted motion by outputting the start coordinates of the top-left corner of the frame. In addition, the stabilizer will output the FMV and maximum BMV of each frame. These output parameters may be provided as input for image stabilization for detecting any object or frame movement.

Figure 27. Top Level Flowchart for VS Program


VS Start
Initialize BSC, EDMA
Initialize VS Algorithm
Start BSC
BSC, EDMA Done ?
Wait
1. Initialize iMX, SEQ Program and Data Memory.
2. Initiate SAD, 1st and 2nd Derivative Computation, Min SAD and Max 2nd Derivative Position Detection
iMX, EDMA, SEQ Done ?
Wait
1. Estimate Block Motion Vector
2. Estimate Frame Motion Vector
3. Estimate Unwanted Motion
Compensate Motion
VS End

5 Results

Table 1 illustrates the stabilization quality measurement of videos containing known motion jitter. The video stream is a well illuminated scene with automobiles and a train moving in a small area of the scene. One frame of the 640×480 video stream is shown in Figure 28.

Table 1. Video Stabilization Quality

Jitter Type	Stream Description	Jitter Magnitude (pixels/frame)	Stabilization Error (22) (pixels/frame)	Stabilization Ratio (23) (%)
Constant	Horizontal jitter	+-31	0	0
Constant	Vertical jitter	+-23	0	0
Random	Jitter in both directions	Varying	1	0
Random	Vertical panning with jitter in both directions. Varying pane of 4 pixels.	+-12 (max)	3	4
Random	Horizontal panning with jitter in both directions.	+-16 (max)	2	0
Random	Diagonal panning with jitter in both directions. Varying pane of 4 pixels.	+-12 (max) V +-16 (max) H	3	5
Constant	Vertical jitter	+-24	1	0
Constant	Horizontal jitter	+-32	1	0

$$\text{*Error} = \frac{1}{200}\sum_{1}^{200}(jitter - (-\,compensati\ on)); \text{ Total number of frames = 200.} \quad (22)$$

$$\text{**Stabilization ratio = 100 x } \frac{\text{Number of frames not stabilizes}}{\text{Total number of frames (=200)}} \quad (23)$$

Figure 28. Video stream used in quality measurement in Table 1

Table 2 illustrates the improvement in performance of the DM355 video stabilizer with respect to the ARM9EJ with a data cache and instruction cache. The use of BSC and iMX offloads the ARM9EJ processing load by 69% and 27 % respectively. The ARM device is free to execute any task while iMX is performing the SAD computation.

Table 2. Video Stabilization Performance ARM9EJ vs DM355

Resolution	ARM9EJ (MHz)	DM355	
		ARM9EJ (MHz)	iMX, SEQ (MHz)
640×480 @ 30 fps	200	5.184	5.054

6 Summary

The histogram of all the block motion vector in a frame, sliding window on raw histogram, neighborhood accumulated histogram on windowed, and weighted accumulated histogram based on past frame histograms aid in better global motion estimation. The spurious data handling minimizes the global motion estimation induced jitter that is caused due to scene changes, luma variations, and object motion effectively. Thus, the precise control using variable cut-off low pass filter aids in maximum motion jitter removal. The averaging filter of variable order eliminates the delay introduced by the low pass IIR filter and minimizes the control lag and clipping. The effect of clipping even if present is minimized by the use of dynamically computed steady bias. This helps in gaining a better stabilization result when the difference of the cropped area to the frame area is smaller. The tests carried out with synthetic and natural videos (about 175 video streams of average 500 frames per stream) validate the above theory.

The computational efficiency is achieved by the use of hardware accelerator/coprocessors for projection vector (boundary signal), and a SAD vector and its derivative computations. With the aid of sequencer (SEQ) hardware and the use of an enhanced DMA, the ARM9 is freed up from control and data transfer activities. The ARM9 is used for a very small amount of time for control algorithms and filter. The use of accelerators allows the clock frequency requirement to be lower and thus power usage to be less. In addition, ARM9 (GPP) is offloaded for carrying out other application tasks when accelerators are functioning. Since the accelerators execution time is constant for the boundary signal generation, the algorithm execution speed is the same for all resolution videos. In case of frames of resolution higher than 640×480, the accelerator down-samples the spatial data before computing projection vector. This means the stabilization quality may be sacrificed while maintaining the same clock frequency requirement for video resolutions greater than 640×480.

7 Acknowledgment

The author thanks Jayanth R. Rai (project trainee) for his contributions in implementation of the C-model of the algorithm, and providing inputs for algorithm enhancements.

8 References

1. *Real-Time Digital Video Stabilization for Multi-Media Applications* by Ratakonda, K. , Circuits and Systems, 1998. ISCAS '98.
2. *Digital Video Stabilization Architecture for Low Cost Devices* by Auberger, S.; Miro, C.; Image and Signal Processing and Analysis, 2005. ISPA 2005.
3. *Full-Frame Video Stabilization* by Matsushita, Y.; Ofek, E.; Tang, X.; Shum, H., Computer Vision and Pattern Recognition, 2005. CVPR 2005.
4. *Circular Block Matching Based Video Stabilization* by Xu, L.; Fu, F.; Lin, X., Visual Communications and Image Processing 2005. Proceedings of the SPIE, Volume 5960, pp. 1307-1314 (2005).
5. *Robust method of digital image stabilization* by Tico, M.; Vehvilainen, M., Communications, Control and Signal Processing, 2008. ISCCSP 2008.
6. *Video Stabilization as a Variational Problem and Numerical Solution with the Viterbi Method* by Pilu, M., 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04) - Volume 1 pp. 625-630.
7. *Automatic Image Stabilizing System by Full-Digital Signal Processing* by Uomori, K.; Morimura, A.; Ishii, H.; Sakaguchi, T.; Kitamura, Y.; IEEE Transactions on Consumer Electronics, Volume 36, pp. 510 - 519 (Aug 1990).
8. *Probabilistic Video Stabilization using Kalman Filtering and Mosaicking* by Litvin, A.; Konrad,J.; Karl, W.C; (http://citeseer.ist.psu.edu/litvin03probabilistic.html)

9. *Image Sequence Stabilisation: Motion Vector Integration (MVI) Versus Frame Position Smoothing (FPS)* by Ertuk, S.; Image and Signal Processing and Analysis, 2001. ISPA 2001.
10. *Camera Stabilization Based on 2.5D Motion Estimation and Inertial Motion Filtering* by Zhu, Z.; Xu, G.; Yang, Y.; Jin, J.S, (http://citeseer.ist.psu.edu/391305.html)
11. *Fast Local and Global Projection-Based Methods for Affine Motion Estimation* by Robinson, D.; Milanfar, P.; Journal of Mathematical Imaging and Vision 18: 35–54, 2003.
12. *Introduction to Digital Signal Processing* by J. R. Johnson, Prentice Hall, Inc., 1992, ch 5.
13. *Discrete-Time Signal Processing* by A. V. Oppenheim, R. W.Schafer, Prentice-Hall, Inc., 1992, ch11, appendix b.
14. *Fixed-Point Arithmetic: An Introduction* by R. Yates, (http://www.digitalsignallabs.com/fp.pdf)
15. *ARM926EJ-S Technical Reference Manual* (http://www.arm.com/pdfs/DDI0198D_926_TRM.pdf)
16. *TMS320DM355 Digital Media System-on-Chip Datasheet* (SPRS463)
17. *TMS320DM35x Digital Media System-on-Chip Video Processing Front End (VPFE) Reference Guide* (SPRUF71)
18. *TMS320DM35x DMSoC Enhanced DMA (EDMA) User's Guide* (SPRUEE4A)

Fitzgerald Archibald was born in Nagercoil, Tamilnadu, India in 1975. He earned his B.E in electronics and communication engineering from PSG College of Technology, Coimbatore, Tamilnadu, India in 1996. He is pursuing MS in electronics and computer engineering at University of California, Santa Barbara, USA starting from Sep, 2008.

He worked on control systems software development for geo-synchronous satellites from 1996 to 1999 in ISRO Satellite Centre, Bangalore, India. In 2001-2002, he worked on speech decoder, real-time kernel, and audio algorithms for the DVD audio team at Sony Electronics, San Jose, USA. While at Philips Semiconductors (Bangalore, India, and Sunnyvale, USA) in 1999-2001 and 2002-2004, he worked on audio algorithms, codecs and systems software for set-top-box, analog/digital TV, and internet audio player. He is part of the DSPS group in Texas Instruments (TI) Inc, Bangalore, India from 2004 to present working on audio/speech/video/imaging algorithms, codecs and systems software development for portable audio/media players, voice recorders, digital still cameras, security cameras, and broad market software development kits. He has five pending US patent applications and has published four technical papers. He received the Manufacturing Incentive Award from TI for his work on Multi-Format Audio Player in Jul, 2008. Interests include multimedia and control algorithms and systems.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless