CCITT SGXV
Working Party XV/1
Experts Group for ATM Video Coding

Document AVC-281
June 24 1992

Title: The comparison of number of operations between adaptive field/frame and simplified FAMC
Source : Japan
Purpose: Information

This document just informs the content of the contribution from Matsushita which will be submitted to CCITT/MPEG joint session at Rio on 6-10 July.

1. Introduction

Field-time Adjusted MC (FAMC MPEG92/24 (AVC-194), MPEG92/100 (AVC-232)) is defined as one of the frame structure prediction mode in TM1 draft (Appendix K). The coding performance of FAMC is superior to adaptive field/frame prediction, however the current FAMC specification needs floating calculations. We investigated the simplification of Motion Estimation (ME) and Motion Compensation (MC) of FAMC from the hardware point of view.

This document is concerned to the investigation of hardware complexity of simplified FAMC. And accompanyed document (MPEG92/ ,AVC-280) describes the coding performance of simplified FAMC.

2. Simplified FAMC

We simplify FAMC from the hardware point of view. The simplified points for MC are:

1) N = float(1/(2*Frame_Distance)) is truncated to 4 bits.
2) The coefficient for interpolation (a, b) is truncated to by 3 bits, and (a+b) is always 8.
3) The interpolation result is rounded to the nearest integer at once.

and simplified ME is below.

step 1 : Frame-based ME with 2(V)X1(H) accuracy
step 2 : Simplified FAMC-based ME with +-1.5(V)X ;-0.5(H) range and half pixel accuracy

The simplified points are indicated in the specification and it is attached as Annex A. Refer MPEG92/ or AVC-280 for the exact simplified FAMC specification.

3. Estimation results

The number of operations per second for adaptive field/frame and simplified FAMC is estimated under the following conditions.

(1) N=15, M=3
(2) Estimation is on Y only

The estimation result is listed is Table 1, and the detail is attached as Annex B and Annex C.

Table 1 Comparison of operation times between adaptive frame/field and FAMC

Prediction type	operation	ME MOPS (norm.)	MC(Encoder) MOPS (norm.)	MC(Decoder) MOPS (normalized)
Adaptive frame/field	add(8,10)	41,883 (1)	334.9 (1)	116 (1)
	add(8,16)	13,233 (1)	55.4 (1)	0 (1)
Simplified FAMC	add(8,10)	21,619 (.52)	196.0 (.59)	120.3 (1.0)
	add(8,16)	6,556 (.50)	32.4 (.58)	0 (1)
	mul(9, 3)	541 (—)	32.4 (—)	32.4 (—)

Note: add(x,y): (x bit) + (y bit) operation.
mul(x,y): (x bit) x (y bit) operation.
MOPS (Mega Operations Per Second)

The rough reason of the above figures are follows.

1) In ME, the operation rate depends on the number of candidate vectors. Simplified FAMC has about half number of vectors of adaptive field/frame prediction because of 2 line(V)x1 pixel(H) accuracy in the first step of ME. So, operation rate of simplified FAMC is also about half.

2) The most operation of MC(Encoder) is MAE (Mean Absolute Error) calculation for mode decision. Simplified FAMC has 4 modes (Forward, Backward, Interpolated, Intra) and adaptive field/frame has 7 modes (Forward Fr, Backward Fr, Interpolated Fr, Forward Fi, Backward Fi, Interpolated Fi, Intra) in B frame. So simplified FAMC about 60% operation rate of adaptive field/frame.

3) Get_FAMC_MB and Get_Fr(Fi)_MB_half has the almost same number of adder operations(Table C.1 in Annex C). So MC(Decoder)of adaptive field/frame and simplified FAMC has almost same adder operation rate.

4) Simplified FAMC needs multiplier for interpolation.

4. Conclusion

The simplified FAMC has about half number of adder operations of adaptive frame/field prediction in ME and MC(Encoder), and the same number in MC(Decoder). And FAMC needs multiplier operations for interpolation. The maximum memory bus load is still double of adaptive field/frame prediction and this problem remains unsolved.

Concerning to coding performance, performance of simplified FAMC is almost same as the original FAMC, and simplified FAMC can be kept higher than adaptive field/frame, which is reported in the accompanyed document. (MPEG92/ , AVC-280)

From those results, we propose to adopt the simplified FAMC for prediction.

Annex A Simplified FAMC

```
Get_FAMC_MB_for_Forward (Frame_Distance, Origin, FAMC_MV, FAMC_MB) {

N = float(1/(2*Frame_Distance))    /* =0.5    (0.100  (Bin)) When Frame_Distance=1  */
                                   /* =0.25   (0.010  (Bin)) When Frame_Distance=2  */
                                   /* =0.1875 (0.0011 (Bin)) When Frame_Distance=3  */
                                   ^^^^^^^^^^^^^^^^^^^^^^^^^^

Point 1  N is truncated to 4 bit.


for (yl=0; yl<16; ++yl) {
   yg = yl + yorigin
   for (xl=0; xl<16; ++xl) {
      xg = xl + xorigin

      if (yl == even) {                  /* For first(Even) Field   */

         x_even1 = xg + (2 * FAMC_MVx)/2                              /* Addressing X of pixel P      */
         x_even2 = xg + (2 * FAMC_MVx)//2                             /* Addressing X of pixel Q      */
         y_even = yg + Adjacent_Even_Line_for_Even_Field_for_Forward  /* Addressing Y of P & Q pixels */
                  (Frame_Distance, FAMC_MVy)

         x_odd1 = xg + ((4 * (FAMC_MVx - N*FAMC_MVx))//2)/2           /* Addressing X of R pixel      */
         x_odd2 = xg + ((4 * (FAMC_MVx - N*FAMC_MVx))//2)//2          /* Addressing X of S pixel      */
         y_odd = yg + Adjacent_Odd_Line_for_Even_Field_for_Forward    /* Addressing Y of R & S pixels */
                 (Frame_Distance, FAMC_MVy)
                                                                      /* Horizontal interpolation     */
         ref_even = ref_frame(x_even1, y_even) + ref_frame(x_even2, y_even)
         ref_odd  = ref_frame(x_odd1, y_odd)   + ref_frame(x_odd2, y_odd)
         FAMC_MB(xl, yl) = (a*ref_even + b*ref_odd)//(2*(a+b))            /* Vertical interpolation       */
                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Point 2:  a, b is truncated to 3 bit and (a+b) is always 8.
Point 3:  The interpolation results are rounded to the nearest integer
          at once.



         }


      else {

        < Same as first field >

         }
      }
   }
}
```

ANNEX B Estimation on number of operations for MC and ME

Note: The estimation of number of operations for functions
(Get_XXX_MB, Cal_XXX_MAE) is attached in Annex C.

Motion Estimation for adaptive frame/field

ME-range: +-15.5 x +-15.5 / Frame
ME-search: Telescopic search and 2 step search (integer, half accuracy)
ME-size 16x16

processing--- 1) Cal_each_candidate_MAE
2) Choose the the least MAE candidate

1)Cal_each_candidate_MAE

forward(backward) Frame based vector
-integer accuracy
[Get_Fr_MB_integer add(8,10)x274
Cal_Fr_MAE add(8,10)x256x2, add(8,16)x256
compare_MAE 0] x31 x31
-half accuracy
[Get_Fr_MB_half add(8,10)x1574
Cal_Fr_MAE add(8,10)x256x2, add(8,16)x256
compare_MAE 0] x3 x3

Total(/1MV/MB) add(8,10)x774120, add(8,16)x248320

forward(backward) Field based vector
(If the frame based vector is used as the offset vector for telescopic search, the search area for frame based and field based vector becomes the same. So we do not need to count up the operation for Get_Fi_MB_integer and Cal_fi_MAE, because these are counted up in frame based vector already.)

-integer accuracy
Get_Fi_(16x8)_integer < already counted in Get_Fr_MB_integer >
Cal_Fi_MAE < already counted in Gal_Fr_MAE >
compare_MAE 0] x31 x31

-half accuracy
[Get_Fi_(16x8)_half add(8,10)x(1612/2)
/* Get_Fi_(16x8)_half is estimated the half of Get_Fi_MB_half */
Cal_Fi_MAE add(8,10)x128x2, add(8,16)x128
compare_MAE 0] x3 x3

Total(/1MV/MB) add(8,10)x 9558 ,add(8,16)x1152

1) frame add(8,10): 774120 x 44(h) x 30(v) x 40(vectors/sec) =40874 MOPS
add(8,16): 248320 x 44(h) x 30(v) x 40(vectors/sec) =13111 MOPS
2) field add(8,10): 9558 x 44(h) x 30(v) x 80(vectors/sec) = 1009 MOPS
add(8,16): 1152 x 44(h) x 30(v) x 80(vectors/sec) = 122 MOPS

| add(8,10) 41,883 MOPS |
| add(8,16) 13,233 MOPS |

Motion Estimation for simplified FAMC

ME-range: +-15.5 x +-15.5 / Frame
ME-search: Telescopic search and 2 step search
1st step (frame based ME)
accuracy: V; 2 line H; 1 pixel candidate 15x31
2nd step (FAMC based ME)
accuracy: V, 0.5 line H; 0.5 pel
candidate 7(v)x3(h)-1 = 20
ME-size 16x16

processing— 1) Cal_each_candidate_MAE
2) Choose the the least MAE candidate

---

1)Cal_each_candidate_MAE

forward(backward) vector
-integer accuracy
[Get_Fr_MB_integer add(8,10)x274
Cal_Fr_MAE add(8,10)x256x2, add(8,16)x256
compare_MAE 0] x15 x31
-half accuracy
[Get_FAMC_MB add(8,10)x1690 multi(9,3)x514
Cal_Fr_MAE add(8,10)x256x2, add(8,16)x256
compare_MAE 0] x20

Total(/1MV/MB) add(8,10)x409530, add(8,16)x124160, multi(9,3)x10280

==========

1) FAMC add(8,10)x409530 x 44(h) x 30(v) x 40(vectors/sec) =21619 MOPS
add(8,16)x124160 x 44(h) x 30(v) x 40(vectors/sec) = 6556 MOPS
multi(9,3)x10280 x 44(h) x 30(v) x 40(vectors/sec) = 541 MOPS

add(8,10)	22,159 MOPS
add(8,16)	6,556 MOPS
multi(9,3)	541 MOPS

MC(Encoder) for adaptive frame/field

processing— 1) Cal_each_mode_MAE (I: 1 mode, P: 3 mode, B:7mode)
2) Choose_the_least_MAE_mode
3) Cal_prediction_error

---

1)Cal_each_mode_MAE
I picture (1 mode): Intra
P picture (3 mode): For_Fr, For_Fi, Intra
B picture (7 mode): For_Fr, For_Fi, Back_Fr, Back_Fi, Int_Fr, Int_Fi, Intra

I picture no mode selection

P picture

1) intra	Get_intra_MB	add(8,10)x272	
	Cal_intra_MAE	add(8,10)x256,	add(8,16)x256
2) For_Fr	Get_Fr_MB_half	add(8,10)x1574	
	Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256
3) For_Fi	Get_Fi_MB_half	add(8,10)x1612	
	Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256
	Total(/MB)	add(8,10)x4738,	add(8,16)x768

B picture

1) intra	Get_intra_MB	add(8,10)x272		
	Cal_intra_MAE	add(8,10)x256,	add(8,16)x256	
2-3) For(Back)_Fr	[Get_Fr _MB_half	add(8,10)x1574		] x2
	[Cal_Fr _MAE	add(8,10)x256x2,	add(8,16)x256	] x2
4-5) For(Back)_Fi	[Get_Fi _MB_half	add(8,10)x1612		] x2
	[Cal_Fr _MAE	add(8,10)x256x2,	add(8,16)x256	] x2
6-7) Int_Fr(Fi)	[Get_Fr(Fi)_MB_half	<already calculated in For(Back)_Fr(Fi)>]x2		
	[rounding	add(8,10)x256		] x2
	[Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256	] x2
	Total(/MB)	add(8,10)x10484,	add(8,16)x1792	

2) Choose_the_least_MAE_mode
very small
3) Cal_prediction_error
add(8,10)x256

==========

Total (per MB)
1) for I : 0x 2/30
for P : (add(8,10)x 4738+add(8,16)x 768)x 8/30
for B : (add(8,10)x10484+add(8,16)x1792)x 20/30
2) 0
3) add(8,10)x256

add(8,10): 8458 x 44(h) x 30(v) x 30(t) = 334.9 MOPS
add(8,16): 1399 x 44(h) x 30(v) x 30(t) = 55.4 MOPS

MC(Encoder) for Simplified FAMC

processing— 1) Cal_each_mode_MAE (I: 1 mode, P: 2 mode, B:4mode)
2) Choose_the_least_MAE_mode
3) Cal_prediction_error

1)Cal_each_mode_MAE
I picture (1 mode): Intra
P picture (2 mode): For_FAMC, Intra
B picture (4 mode): For_FAMC, Back_FAMC, Int_FAMC, Intra

I picture no mode selection

P picture

1) intra	Get_intra_MB	add(8,10)x272		
	Cal_intra_MAE	add(8,10)x256,	add(8,16)x256	
2) For_FAMC	Get_FAMC_MB	add(8,10)x1686,		multi(9,3)x514
	Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256	
	Total(/MB)	add(8,10)x2726,	add(8,16)x512,	multi(9,3)x514

B picture

1) intra	Get_intra_MB	add(8,10)x272		
	Cal_intra_MAE	add(8,10)x256,	add(8,16)x256	
2-3) For(Back)_FAMC	[Get_FAMC_MB	add(8,10)x1686,		multi(9,3)x514] x2
	[Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256	] x2
4) Int_FAMC	Get_FAMC_MB	<already calculated in For(Back)_FAMC>		
	add & rounding	add(8,10)x256x2		
	Cal_Fr_MAE	add(8,10)x256x2,	add(8,16)x256	
	Total(/MB)	add(8,10)x5948 ,	add(8,16)x1024,	multi(9,3)x1024

2) Choose_the_least_MAE_mode
very small

3) Cal_prediction_error
add(8,10)x256

Total (per MB)
1) for I : 0x 2/30
for P : (add(8,10)x 2726 +add(8,16)x 512 +multi(9,3)x 514)x 8/30
for B : (add(8,10)x 5948 +add(8,16)x1024 +multi(9,3)x1028)x 20/30
2) 0
3) add(8,10)x256

add(8,10): 4948 x 44(h) x 30(v) x 30(t) = 196.0 MOPS
add(8,16): 820 x 44(h) x 30(v) x 30(t) = 32.4 MOPS
mul(9,3) : 820 x 44(h) x 30(v) x 30(t) = 32.4 MOPS

MC(Decoder) for adaptive frame/field

processing— 1) Cal_prediction_error

1) Cal_prediction_error (Estimate the most operation mode in each picture)

	prediction mode of the most operation
I picture :	Intra
P picture :	For_Fi
B picture :	Int_Fi

I picture	nothing to do	
P picture		
For_Fi	Get_Fi_MB_half	add(8,10)x1612
	Total(/MB)	add(8,10)x1612
B picture		
Int_Fi	[Get_Fi_MB_half	add(8,10)x1612] x 2
	add & rounding	add(8,10)x256x2
	Total(/MB)	add(8,10)x3736

Total (per MB)
for I : 0x 2/30
for P : (add(8,10)x1612)x 8/30
for B : (add(8,10)x3736)x 20/30

add(8,10) :2921 x 44(h) x 30(v) x 30(t) = 116 MOPS

MC(Decoder) for simplified FAMC

processing— 1) Cal_prediction_error

1) Cal_prediction_error (Estimate the most operation in each picture)

prediction of the most operation

I picture : Intra
P picture : For_FAMC
B picture : Int_FAMC

I picture nothing to do

P picture

For_FAMC	Get_FAMC_MB	add(8,10)x1686,	multi(9,3)x514
	Total(/MB)	add(8,10)x1686,	multi(9,3)x514

B picture

Int_FAMC [	Get_FAMC_MB	add(8,10)x1686,	multi(9,3)x514] x2
	add & rounding	add(8,10)x256x2	
	Total(/MB)	add(8,10)x3886 ,	multi(9,3)x1028

Total (per MB)
for I : 0x 2/30
for P : (add(8,10)x 1686 +multi(9,3)x514)x 8/30
for B : (add(8,10)x 3886 +multi(9,3)x1028)x 20/30

add(8,10) :3039 x 44(h) x 30(v) x 30(t) = 120.3 MOPS
multi(9,3): 820 x 44(h) x 30(v) x 30(t) = 32.4 MOPS

ANNEX C Number of operations for the functions (Get_XXX_MB, Cal_XXX_MAE)

The following estimation is assumed ;
1) One rounding calculation "//" needs add(8,10)x1.
2) One absolute calculation "|a|" needs add(8,10)x1.

Table C.1 The number of operations for Get_XXX_MB

Function	# MB type	operation	Add. Gen. /MB	Interpo. /MB	# Total /MB
Get_intra_MB	# Intra MB	add(8,10)	272	0	272
Get_Fr_MB_half	# Frame based MC half pel MV	add(8,10)	550	1024	1574
Get_Fr_MB_integer	# Frame based MC Integer pel MV	add(8,10)	274	0	274
Get_Fi_MB_half	# Field based MC half pel MV	add(8,10)	558	1024	1612
Get_Fi_MB_integer	# Field based MC Integer pel MV	add(8,10)	292	0	292
Get_FAMC_MB	# Simplified FAMC	add(8,10)	662	1024	1686
		mul(9,3)		514	514

Table C.2 The number of operations for Cal_XXX_MAE

Functions	# type	Num. of sum.	operation	# /loop	Num of op. Total
Cal_intra_MAE	# |a|	16x16	add(8,10)	1	256
			add(8,16)	1	256
Cal_Fr_MAE	# |a-b|	16x16	add(8,10)	2	512
			add(8,16)	1	256
Cal_Fi_MAE	# |a-b|	16x 8	add(8,10)	2	256
			add(8,16)	1	128

```
Get_Intra_MB (Origin, Intra_MB) {

for (x1=0; x1<16; ++x1) {
   xg = x1 + xorigin                    /* add(8,10) x1 x16 times */

   for (y1=0; y1<16; ++y1) {
      yg = y1 + yorigin                 /* add(8,10) x1  x (16x16) times*/

  intra_MB(x1,y1) = ref_frame(xg,yg)
 }
}
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB))
add(8,10)	1x16+1x16x16=272	0	272

```
Get_Fr_MB_half (Origin, Fr_MV, Fr_MB) {

xorigin_1 = xorigin + (2 * Fr_MVx)/2    /* add(8,10) x1 */
xorigin_2 = xorigin + (2 * Fr_MVx)//2   /* add(8,10) x2 */
yorigin_1 = yorigin + (2 * Fr_MVy)/2    /* add(8,10) x1 */
yorigin_2 = yorigin + (2 * Fr_MVy)//2   /* add(8,10) x2 */

for (x1=0; x1<16; ++x1) {
   x_1 = x1 + xorigin_1                 /* add(8,10) x1 x16 times */
   x_2 = x1 + xorigin_2                 /* add(8,10) x1 x16 times */

   for (y1=0; y1<16; ++y1) {
      y_1 = y1 + yorigin_1              /* add(8,10) x1  x (16x16) times */
      y_2 = y1 + yorigin_2              /* add(8,10) x1  x (16x16) times */
      Fr_MB(x1,y1) = (ref_frame(x_1,y_1) + ref_frame(x_2,y_1) +
                      ref_frame(x_1,y_2) + ref_frame(x_2,y_2) )//4
                                        /* add(8,10) x 4 x (16x16) times*/
 }
}
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB))
add(8,10)	6+2x16+2x16x16=550	4x16x16=1024	1574

```
Get_Fr_MB_integer (Origin, Fr_MV, Fr_MB) {

xorigin_1 = xorigin + Fr_MVx            /* add(8,10) x1 */
yorigin_1 = yorigin + Fr_MVy            /* add(8,10) x1 */

for (x1=0; x1<16; ++x1) {
   xg = x1 + xorigin_1                  /* add(8,10) x1 x16 times */

   for (y1=0; y1<16; ++y1) {
      yg = y1 + yorigin_1               /* add(8,10) x1  x (16x16) times */
      Fr_MB(x1,y1) = ref_frame(xg,yg)
 }
}
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB))
add(8,10)	2+1x16+1x16x16=274	0	274

```
Get_Fi_MB_half (Origin, Fi_MV1, Fi_MV2, Fi_MB) {


      /* First field */
xorigin_1 = xorigin + (2 * Fi_MV1x)/2   /* add(8,10) x1 */
xorigin_2 = xorigin + (2 * Fi_MV1x)//2  /* add(8,10) x2 */
yorigin_1 = yorigin + (2 * Fi_MV1y)/2   /* add(8,10) x1 */
yorigin_2 = yorigin + (2 * Fi_MV1y)//2  /* add(8,10) x2 */

for (x1=0; x1<16; ++x1) {
   x_1 = x1 + xorigin_1                 /* add(8,10) x1 x16 times */
   x_2 = x1 + xorigin_2                 /* add(8,10) x1 x16 times */

   for (y1=0; y1<16; y1=y1+2) {
      y_1 = y1 + yorigin_1              /* add(8,10) x1  x (16x8) times */
      y_2 = y1 + yorigin_2              /* add(8,10) x1  x (16x8) times */
      Fi_MB(x1,y1) = (ref_frame(x_1,y_1) + ref_frame(x_2,y_1) +
                      ref_frame(x_1,y_2) + ref_frame(x_2,y_2) )//4
                                        /* add(8,10) x 4 x (16x8) times*/
      }
   }
      /* Second field */
xorigin_1 = xorigin + (2 * Fi_MV2x)/2   /* add(8,10) x1 */
xorigin_2 = xorigin + (2 * Fi_MV2x)//2  /* add(8,10) x2 */
yorigin_1 = yorigin + (2 * Fi_MV2y)/2   /* add(8,10) x1 */
yorigin_2 = yorigin + (2 * Fi_MV2y)//2  /* add(8,10) x2 */

for (x1=0; x1<16; ++x1) {
   x_1 = x1 + xorigin_1                 /* add(8,10) x1 x16 times */
   x_2 = x1 + xorigin_2                 /* add(8,10) x1 x16 times */

   for (y1=1; y1<16; y1=y1+2) {
      y_1 = y1 + yorigin_1              /* add(8,10) x1  x (16x8) times */
      y_2 = y1 + yorigin_2              /* add(8,10) x1  x (16x8) times */
      Fi_MB(x1,y1) = (ref_frame(x_1,y_1) + ref_frame(x_2,y_1) +
                      ref_frame(x_1,y_2) + ref_frame(x_2,y_2) )//4
                                        /* add(8,10) x 4 x (16x8) times*/
      }
   }
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,10)	[6+2x1x6+2x16x8]x2=558	[4x16x8]x2=1024	1612

```
Get_Fi_MB_integer (Origin, Fi_MV1, Fi_MV2, Fi_MB) {

       /* First Field */
xorigin_1 = xorigin + Fi_MV1x          /* add(8,10) x1 */
yorigin_1 = yorigin + Fi_MV1y          /* add(8,10) x1 */
for (x1=0; x1<16; ++x1) {
   xg = x1 + xorigin_1                 /* add(8,10) x1 x16 times */

   for (y1=0; y1<16; y1=y1+2) {
      yg = y1 + yorigin_1              /* add(8,10) x1  x (16x8) times */
      Fi_MB(x1,y1) = ref_frame(xg,yg)
 }
}
       /* Second Field */
xorigin_1 = xorigin + Fi_MV2x          /* add(8,10) x1 */
yorigin_1 = yorigin + Fi_MV2y          /* add(8,10) x1 */
for (x1=0; x1<16; ++x1) {
   xg = x1 + xorigin_1                 /* add(8,10) x1 x16 times */

   for (y1=1; y1<16; y1=y1+2) {
      yg = y1 + yorigin_1              /* add(8,10) x1  x (16x8) times */
      Fi_MB(x1,y1) = ref_frame(xg,yg)
 }
}
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,10)	[2+1x16+1x16x8]x2=292	0	292

```
Get_FAMC_MB_for_Forward (Frame_Distance, Origin, FAMC_MV, FAMC_MB) {

N = float(1/(2*Frame_Distance))    /* =0.5    (0.100  (Bin)) When Frame_Distance=1  */
                                   /* =0.25   (0.010  (Bin)) When Frame_Distance=2  */
                                   /* =0.1875 (0.0011 (Bin)) When Frame_Distance=3  */

       /* For first(Even) Field */
xorigin_even1 = xorigin +(2 * FAMC_MVx)/2                                  /* add(8,10) x1 */
xorigin_even2 = xorigin +(2 * FAMC_MVx)//2                                 /* add(8,10) x1 */
xorigin_odd1  = xorigin + ((4 * (FAMC_MVx - N*FAMC_MVx))//2)/2             /* add(8,10)x3 mul(8,4)x1 */
xorigin_odd2  = xorigin + ((4 * (FAMC_MVx - N*FAMC_MVx))//2)//2            /* add(8,10)x4 mul(8,4)x1 */
yorigin_even  = yorigin + Adjacent_Even_Line_for_Even_Field_for_Forward  /* add(8,10) x1 */
                          (Frame_Distance, FAMC_MVy)
yorigin_odd   = yorigin + Adjacent_Odd_Line_for_Even_Field_for_Forward  /* add(8,10) x1 */
                          (Frame_Distance, FAMC_MVy)

for (x1=0; x1<16; ++x1) {
   x_even1 = x1 + xorigin_even1        /* add(8,10) x1 x16 times */  /* Addressing X of pixel P      */
   x_even2 = x1 + xorigin_even2        /* add(8,10) x1 x16 times */  /* Addressing X of pixel Q      */
   x_odd1  = x1 + xorigin_odd1         /* add(8,10) x1 x16 times */  /* Addressing X of R pixel      */
   x_odd2  = x1 + xorigin_odd2         /* add(8,10) x1 x16 times */  /* Addressing X of S pixel      */

   for (y1=0; y1<16; y1=y1+2) {
      y_even = y1 + yorigin_even       /* add(8,10) x1 x(16x8) times */ /* Addressing Y of P & Q pixels */
      y_odd  = y1 + yorigin_odd        /* add(8,10) x1 x(16x8) times */ /* Addressing Y of R & S pixels */

            /* Horizontal interpolation     */
         ref_even =  ref_frame(x_even1,y_even) + ref_frame(x_even2,y_even)  /* add(8,10) x1 x(16x8)times */
         ref_odd  =  ref_frame(x_odd1,y_odd)   + ref_frame(x_odd2,y_odd)    /* add(8,10) x1 x(16x8)times */
            /* Vertical interpolation       */
         FAMC_MB(x1,y1) = (a*ref_even + b*ref_odd)//(2*(a+b))  /* a,b: 3bit  /(2*(a+b)): just shift */
```

```
                                                    /* mul(9,3)x2x(16x8) add(8,10)x2x(16x8)*/
    }
  }
      /* For Second(Odd) Field */
xorigin_odd1  = xorigin +(2 * FAMC_MVx)/2                              /* add(8,10) x1 */
xorigin_odd2  = xorigin +(2 * FAMC_MVx)//2                             /* add(8,10) x1 */
xorigin_even1 = xorigin + ((4 * (FAMC_MVx + N*FAMC_MVx))//2)/2         /* add(8,10) x3 */
xorigin_even2 = xorigin + ((4 * (FAMC_MVx + N*FAMC_MVx))//2)//2        /* add(8,10) x4 */
yorigin_odd   = yorigin + Adjacent_Odd_Line_for_Odd_Field_for_Forward  /* add(8,10) x1 */
                        (Frame_Distance, FAMC_MVy)
yorigin_even  = yorigin + Adjacent_Even_Line_for_Odd_Field_for_Forward /* add(8,10) x1 */
                        (Frame_Distance, FAMC_MVy)

for (x1=0; x1<16; ++x1) {
  x_odd1  = x1 + xorigin_odd1         /* add(8,10) x1 x16 times */  /* Addressing X of R pixel      */
  x_odd2  = x1 + xorigin_odd2         /* add(8,10) x1 x16 times */  /* Addressing X of S pixel      */
  x_even1 = x1 + xorigin_even1        /* add(8,10) x1 x16 times */  /* Addressing X of pixel P      */
  x_even2 = x1 + xorigin_even2        /* add(8,10) x1 x16 times */  /* Addressing X of pixel Q      */

  for (y1=1; y1<16; y1=y1+2) {
    y_odd  = y1 + yorigin_odd         /* add(8,10) x1 x(16x8) times */ /* Addressing Y of R & S pixels */
    y_even = y1 + yorigin_even        /* add(8,10) x1 x(16x8) times */ /* Addressing Y of P & Q pixels */

          /* Horizontal interpolation     */
      ref_odd  = ref_frame(x_odd1,y_odd)   + ref_frame(x_odd2,y_odd)     /* add(8,10) x1 x(16x8)times */
      ref_even = ref_frame(x_even1,y_even) + ref_frame(x_even2,y_even)   /* add(8,10) x1 x(16x8)times */
           /* Vertical interpolation       */
      FAMC_MB(x1,y1) = (a*ref_odd + b*ref_even)//(2*(a+b))  /* a,b: 3bit  /(2*(a+b)): just shift */
                                                            /* multi(9,3)x2x(16x8) add(8,10)x2x(16x8)*/
    }
  }
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,10)	[11+4x16+2x16x8]x2=662	[4x(16x8)]x2=1024	1686
mulit(9,3)		2+ [2x(16x8)]x2= 514	514

Cal_Intra_MAE (MB_a, MAE) {

```
    /* Address generation is counted in Get_XXX__MB, so not counted hear */

MAE=0
for (y1=0; y1<16; ++y1) {
   for (x1=0; x1<16; ++x1) {
        MAE = MAE + MB_a(x1,y1)          /* add(8,16) x1 x(16x16) times */
                                         /* MB_a(x1,y1) >= 0 */
 }
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,16)	It is counted in Get_XXX_MB	1x(16x16)	256

Cal_Fr_MAE (MB_a, MB_b, MAE) {

```
    /* Address generation is counted in Get_XXX__MB, so not counted hear */

MAE=0
for (y1=0; y1<16; ++y1) {
   for (x1=0; x1<16; ++x1) {
        MAE = MAE + |MB_a(x1,y1)-MB_b(x1,y1)|    /* add(8, 9) x2 x(16x16) times */
                                                 /* add(8,16) x1 x(16x16) times */
 }
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,10)	It is counted in Get_XXX_MB	2x(16x16)	512
add(8,16)		1x(16x16)	256

Cal_Fi_MAE (MB_a, MB_b, MAE) {

```
    /* Address generation is counted in Get_XXX__MB, so not counted hear */

MAE=0
for (y1=0; y1<16; y1=y1+2) {
   for (x1=0; x1<16; ++x1) {
        MAE = MAE + |MB_a(x1,y1)-MB_b(x1,y1)|    /* add(8, 9) x2 x(16x8) times */
                                                 /* add(8,16) x1 x(16x8) times */
 }
}
```

Operation	Add. Generation(/MB)	interpolation(/MB)	Total(/MB)
add(8,10)	It is counted in Get_XXX_MB	2x(16x8)	256
add(8,16)		1x(16x8)	128