Using the flag **int emv_enable** for the no_emv first disable the extra candidate in the **mvpredict.c** then , disable the zero vector in the mv.c , considering the median is also based on the neighboring candidate and it is based on smaller luma rather than 16x16 macroblock, so it is kept.
The codes are seen below:

```
/* This just improves encoder performance, it's not part of the spec */
void x264_mb_predict_mv_ref16x16( x264_t *h, int i_list, int i_ref, int16_t mvc[9][2], int *i_mvc )
{
    int16_t (*mvr)[2] = h->mb.mvr[i_list][i_ref];
    int i = 0;


    if (emv_enable==1)
    {

        #define SET_MVP(mvp) \
        { \
            CP32( mvc[i], mvp ); \
            i++; \
        }

      #define SET_IMVP(xy) \
        if( xy >= 0 ) \
        { \
            int shift = 1 + MB_INTERLACED - h->mb.field[xy]; \
            int16_t *mvp = h->mb.mvr[i_list][i_ref<<1>>shift][xy]; \
            mvc[i][0] = mvp[0]; \
            mvc[i][1] = mvp[1]<<1>>shift; \
            i++; \
        }

            /* b_direct */
        if( h->sh.i_type == SLICE_TYPE_B
            && h->mb.cache.ref[i_list][x264_scan8[12]] == i_ref )
        {
            SET_MVP( h->mb.cache.mv[i_list][x264_scan8[12]] );
        }

        if( i_ref == 0 && h->frames.b_have_lowres )
        {
```

```
        int idx = i_list ? h->fref[1][0]->i_frame-h->fenc->i_frame-1
                         : h->fenc->i_frame-h->fref[0][0]->i_frame-1;
        if( idx <= h->param.i_bframe )
        {
            int16_t (*lowres_mv)[2] = h->fenc->lowres_mvs[i_list][idx];
            if( lowres_mv[0][0] != 0x7fff )
            {
                M32( mvc[i] ) = (M32( lowres_mv[h->mb.i_mb_xy] )*2)&0xfffefffff;
                i++;
            }
        }
    }

    /* spatial predictors */
    if( SLICE_MBAFF )
    {
        SET_IMVP( h->mb.i_mb_left_xy[0] );
        SET_IMVP( h->mb.i_mb_top_xy );
        SET_IMVP( h->mb.i_mb_topleft_xy );
        SET_IMVP( h->mb.i_mb_topright_xy );
    }
    else
    {
        SET_MVP( mvr[h->mb.i_mb_left_xy[0]] );
        SET_MVP( mvr[h->mb.i_mb_top_xy] );
        SET_MVP( mvr[h->mb.i_mb_topleft_xy] );
        SET_MVP( mvr[h->mb.i_mb_topright_xy] );
    }
#undef SET_IMVP
#undef SET_MVP

    /* temporal predictors */
    if( h->fref[0][0]->i_ref[0] > 0 )
    {
        x264_frame_t *l0 = h->fref[0][0];
        int field = h->mb.i_mb_y&1;
        int curpoc = h->fdec->i_poc + h->fdec->i_delta_poc[field];
        int refpoc = h->fref[i_list][i_ref>>SLICE_MBAFF]->i_poc;
        refpoc += l0->i_delta_poc[field^(i_ref&1)];

#define SET_TMVP( dx, dy ) \
```

```
        { \
            int mb_index = h->mb.i_mb_xy + dx + dy*h->mb.i_mb_stride; \
            int scale = (curpoc - refpoc) * l0->inv_ref_poc[MB_INTERLACED&field]; \
            mvc[i][0] = (l0->mv16x16[mb_index][0]*scale + 128) >> 8; \
            mvc[i][1] = (l0->mv16x16[mb_index][1]*scale + 128) >> 8; \
            i++; \
        }

        SET_TMVP(0,0);
        if( h->mb.i_mb_x < h->mb.i_mb_width-1 )
            SET_TMVP(1,0);
        if( h->mb.i_mb_y < h->mb.i_mb_height-1 )
            SET_TMVP(0,1);
    #undef SET_TMVP
    }

}
else
{

    #define SET_MVP(mvp) \
        { \
            CP32( mvc[i], mvp ); \
        i++; \
        }

    #define SET_IMVP(xy) \
    if( xy >= 0 ) \
    { \
        int shift = 1 + MB_INTERLACED - h->mb.field[xy]; \
        int16_t *mvp = h->mb.mvr[i_list][i_ref<<1>>shift][xy]; \
        mvc[i][0] = mvp[0]; \
        mvc[i][1] = mvp[1]<<1>>shift; \
        i++; \
    }

        /* spatial predictors */
    if( SLICE_MBAFF )
    {
        SET_IMVP( h->mb.i_mb_left_xy[0] );
        SET_IMVP( h->mb.i_mb_top_xy );
```

```
            SET_IMVP( h->mb.i_mb_topleft_xy );
            SET_IMVP( h->mb.i_mb_topright_xy );
        }
        else
        {
            SET_MVP( mvr[h->mb.i_mb_left_xy[0]] );
            SET_MVP( mvr[h->mb.i_mb_top_xy] );
            SET_MVP( mvr[h->mb.i_mb_topleft_xy] );
            SET_MVP( mvr[h->mb.i_mb_topright_xy] );
        }
        #undef SET_IMVP
        #undef SET_MVP
    }



    *i_mvc = i;
}
if (1)
{
    if( h->mb.i_subpel_refine >= 3 )
    {


        /* Calculate and check the MVP first */
        int  bpred_mx = x264_clip3( m->mvp[0], SPEL(mv_x_min), SPEL(mv_x_max) );
        int  bpred_my = x264_clip3( m->mvp[1], SPEL(mv_y_min), SPEL(mv_y_max) );
         pmv = pack16to32_mask( bpred_mx, bpred_my );
         pmx = FPEL( bpred_mx );
         pmy = FPEL( bpred_my );

         COST_MV_HPEL( bpred_mx, bpred_my, bpred_cost );
        int pmv_cost = bpred_cost;


    if (emv_enable==1) num_thred=0;
      else num_thred=3; //Only use four neighboring vector
```

```
        //fprintf(imvc_file,"%d\n",i_mvc); //Outut motion_x
        //fclose(imvc_file);


        if( i_mvc > num_thred ) //Only use four neighboring vector
        {

           /* Clip MV candidates and eliminate those equal to zero and pmv. */
                int valid_mvcs = x264_predictor_clip( mvc_temp+2, mvc, i_mvc, h->mb.mv_limit_fpel,
pmv );

           if( valid_mvcs > 0 )
           {
              int i = 1, cost;
              /* We stuff pmv here to branchlessly pick between pmv and the various
               * MV candidates. [0] gets skipped in order to maintain alignment for
               * x264_predictor_clip. */
                if (emv_enable==1)   //the median vecor
                {
                  M32( mvc_temp[1] ) = pmv;
                  bpred_cost <<= 4;
                }
                else bpred_cost = COST_MAX;

              do
              {
                 int mx = mvc_temp[i+1][0];
                 int my = mvc_temp[i+1][1];
                 COST_MV_HPEL( mx, my, cost );
                 COPY1_IF_LT( bpred_cost, (cost << 4) + i );
              } while( ++i <= valid_mvcs );
              bpred_mx = mvc_temp[(bpred_cost&15)+1][0];
              bpred_my = mvc_temp[(bpred_cost&15)+1][1];
              bpred_cost >>= 4;
           }
        }



        /* Round the best predictor back to fullpel and get the cost, since this is where
         * we'll be starting the fullpel motion search. */
```

```
    bmx = FPEL( bpred_mx );
    bmy = FPEL( bpred_my );
    bpred_mv = pack16to32_mask(bpred_mx, bpred_my);
    if( bpred_mv&0x00030003 ) /* Only test if the tested predictor is actually subpel... */
       COST_MV( bmx, bmy );
    else                      /* Otherwise just copy the cost (we already know it) */
       bcost = bpred_cost;

    /* Test the zero vector if it hasn't been tested yet. */
    if (emv_enable==1)  //Do not use zero vector
    {
      if( pmv )
        {
           if( bmx|bmy ) COST_MV( 0, 0 );
        }
        /* If a subpel mv candidate was better than the zero vector, the previous
         * fullpel check won't have gotten it even if the pmv was zero. So handle
         * that possibility here. */
        else
        {
           COPY3_IF_LT( bcost, pmv_cost, bmx, 0, bmy, 0 );
        }
    }

  }
  else
```

For the using ETAR_enable to disabel the function.
Codeseen below :

```
     if (ETAR_enable==1) //Flag of ETAR
           {
              ----
        }
    if( i_mvc && ETAR_enable==1 )//Flag of ETAR
           {
         ----
        }
```

For the time record , using the function , Code seen below :

```
        struct timeval  tv1, tv2;  //THE bigin time and stop time
          gettimeofday(&tv1, NULL);//Record the begin time
```

```
/*------serach process---*/
gettimeofday(&tv2, NULL);//Record the stop for  time
time_spent+= (double) (tv2.tv_usec - tv1.tv_usec) / 1000000 + (double) (tv2.tv_sec – tv1.tv_sec);//calculate the time interval
 printf("the time for search = %f\n",time_spent);  //print the time for motion search
```