Hi3516A/Hi3516D AF Statistical Module

Description

Issue 03

Date 2016-03-25

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document describes the Hi3516A statistical module as well as the focus value (FV) blending method and configuration to facilitate automatic focus (AF) development.

NOTE

This document uses the Hi3516A as an example. Unless otherwise specified, this document applies to the Hi3516D and the Hi3516A.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516A	V100
Hi3516D	V100

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 03 (2016-03-25)

This issue is the third official release, which incorporates the following changes:

Section 2.1.1 is modified.

Issue 02 (2015-10-30)

This issue is the second official release, which incorporates the following changes:

The description in Chapter 2 is modified.

Issue 01 (2015-06-12)

This issue is the first official release.

Contents

Figures

1 HiSilicon AF Statistical Module

1.1 Introduction

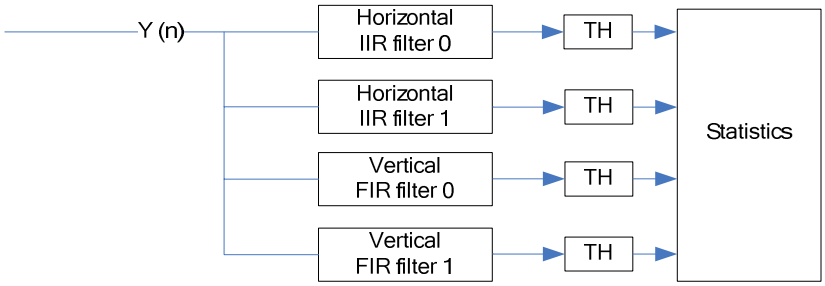
In passive AF mode, the focus is adjusted to the optimal position by driving the focus motor, and then the picture definition value (FV) is obtained by analyzing picture features. There are various algorithms for obtaining the FV, for example, the gray scale gradient method and high-frequency component method. The Hi3516A uses the high-frequency component method to calculate the FV. The higher the picture definition, the higher the amplitude of the high-frequency part. The high-frequency components are obtained after the picture is processed by the high-pass filter. The Hi3516A provide four filters and luminance information, that is, horizontal filters H1 and H2, vertical filters V1 and V2, and Y.

1.2 Module Structure

1.2.1 Overview

The Hi3516A uses the infinite impulse response (IIR) filters and finite impulse response (FIR) filters as the horizontal filters and vertical filters respectively. The absolute values of the filter outputs are obtained, and values that exceed the threshold are collected for statistics. Figure 1-1 shows the logical block diagram of the AF statistical module.

Figure 1-1 Logical block diagram of the AF statistical module



1.2.2 Mode Configuration

The AF statistical module supports the following two modes:

- Peak mode

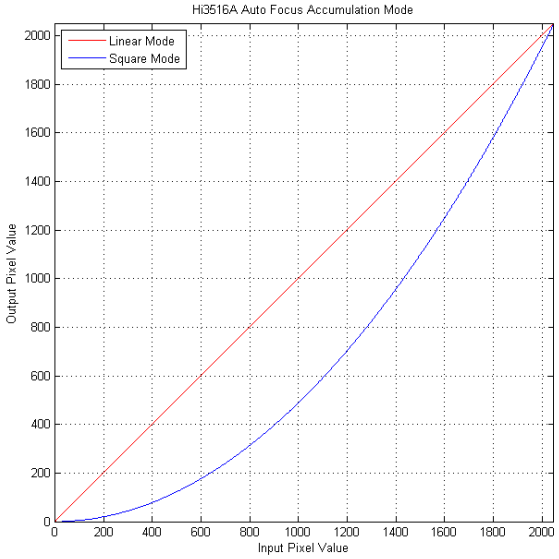
  In peak mode, the block statistics are the sum of the maximum value in each row after filtering. In non-peak mode, the block statistics are the sum of each pixel value. The summation mode applies to the scenarios with large noises.

- Square mode

  In square mode, the filter outputs are normalized, squared, and then collected for statistics.

  In square mode, the FV curve is steeper near the focus. Figure 1-2 shows the corresponding mapping curve.

Figure 1-2 Mapping curve



The configurations of the output shift and the block size are described as follows:

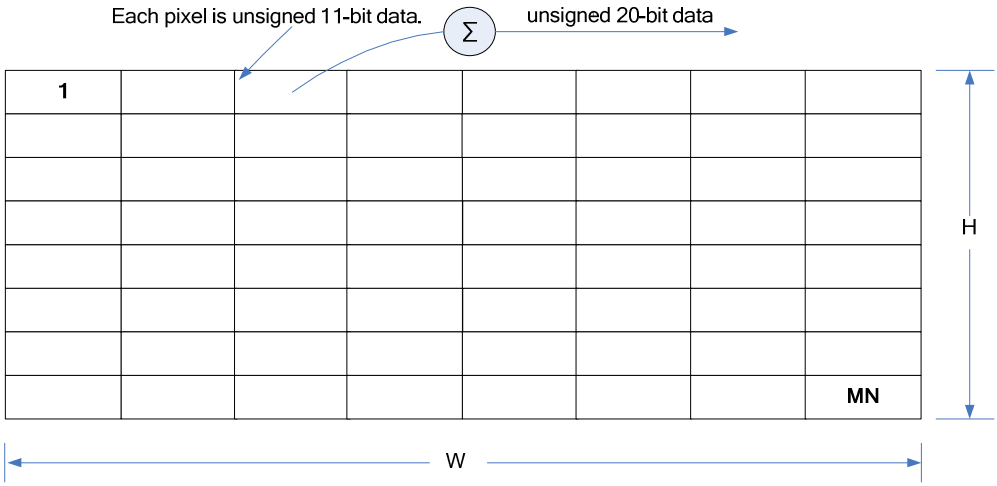
- Output shift

  The output shift can be configured to decrease the right shift of the output value to avoid value overflow.

- Block size

  After filtering, the picture can be divided into blocks, as shown in Figure 1-3. The block size can be configured. The Hi3516A supports at most 17 x 15 blocks. The final block statistics are the accumulated value of each pixel or the accumulated value of the maximum value in each row. The accumulator output is 20-bit unsigned data, and the filter output pixel width is 11 bits (unsigned number). Therefore, when the block output value is 2047, the maximum number of pixels that can be accumulated is 512. The block size must be set to an appropriate value; otherwise, overflow occurs.

Figure 1-3 Picture that is divided into blocks after filtering



2 FV Blending

2.1 Configuration and Calculation Method Recommended by HiSilicon

2.1.1 Configuring the Statistical Module

To configure the statistical module, call HI_MPI_ISP_SetStatisticsConfig to configure the **stFocusCfg** field as follows (taking 1080p as an example):

```
{
    ISP_AF_CFG_S        stConfig;
    ISP_AF_H_PARAM_S    stHParam_IIR0;
    ISP_AF_H_PARAM_S    stHParam_IIR1;
    ISP_AF_V_PARAM_S    stVParam_FIR0;
    ISP_AF_V_PARAM_S    stVParam_FIR1;
    ISP_AF_FV_PARAM_S   stFVParam;
}


{
     {1, 8, 8, 1920, 1080, 1, 0},
     {{0, 1, 1}, {188,  0,   0,    414, -165, 400, -164}, {7, 0, 3, 2}, 127},
     {{0, 1, 0}, {200,  0,   0,    200,   -55,  0,   0},    {6, 0, 1, 0}, 31},
     {{-6,  12, 22, 12, -6}, 511},
     {{-16, 21, 0, -21, 16}, 10},
     {0, {0, 0}, {3, 2}}
};
```

If the sensor with higher resolution is used, output overflow may occur. In this case, the same shift value should be added to the fields in **stFVParam**, for example, change the configuration from {0, {0, 0,}, {3, 2}} to {0, {1, 1,}, {4, 3}}. The added value needs to be determined based on the value of each block to avoid overflow.

When the picture noise is large, a smaller **PassBand** value is recommended because the SNR of the IF and high-frequency bands is low. In addition, set **enPeakMode** to

ISP_AF_STA_NORM and **enSquMode** to **ISP_AF_STA_SUM_SQU**. The recommended setting of the filter coefficients is as follows:

stHParam_IIR = {{0, 1, 1}, {168, 0, 0, 0, 0, 484, –230}, {6, 0, 1, 1}, 127}

2.1.2 Obtaining Statistics

The statistics are updated immediately after the last pixel of a frame picture is processed by the AF module. It is recommended that HI_MPI_ISP_GetVDTimeOut be called to synchronously obtain the statistics. Then the AutoFocus and ZoomTracking algorithms calculate the target focus and zoom position. It is recommended that the driver configuration that requires real time be implemented in the kernel space because task scheduling in the Linux user space cannot guarantee real time. The ISP supports the registration for the synchronous callback interface, which can synchronize the VD. There are samples in **mpp\extdrv\sample_ist**. It is recommended that a task that requires high real-time performance be implemented in the synchronous callback. The bottom layer provides three ways for implementation: HwIRQ, Tasklet, Workqueue. The corresponding way can be used to define the real-time level.

2.1.3 Calculating FVs

For each block, FV1 and FV2 are calculated as follows:

FV1_n = α * H1_n + (1 – α) * V1_n

FV2_n = β * H2_n + (1 – β) * V2_n

where H1 and H2 indicate two horizontal filters, and V1 and V2 indicate two vertical filters.

To obtain the final FV, the weighted operation needs to be performed on each block. FV1 and FV2 are calculated as follows:

$$FV = \frac{\sum_{n=1}^{BLOCKS}(FV_n * Weight_n)}{\sum Weight_n}$$

$\frac{\sum_{n=1}^{BLOCKS}(FV_n*Weight_n)}{\sum Weight_n}$

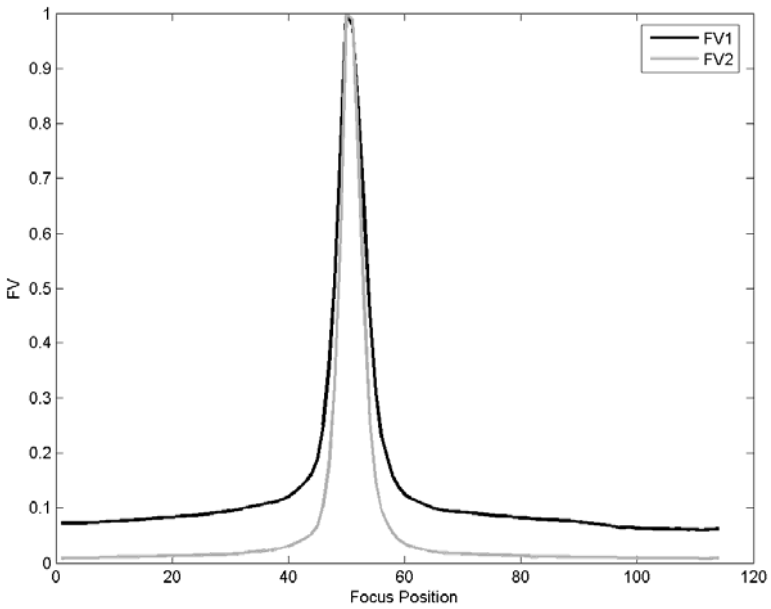
The recommended weight table is as follows:

$$\mathbf{Weight\_n} = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

For details about the sample code, see section 2.3 "Reference Code for Calculating FVs."

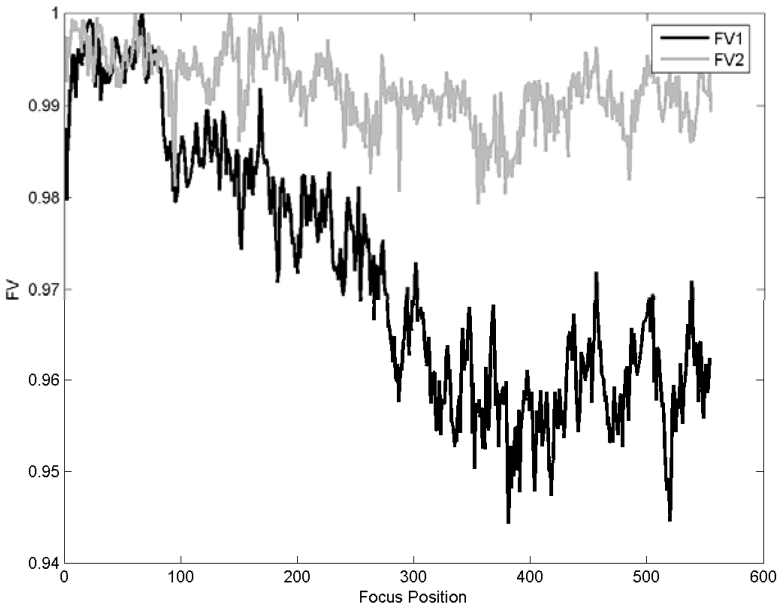
Figure 2-1 shows the FV curves after blending. Compared with the FV2 curve, the FV1 curve better reflects the variation trend of the picture definition. However, the FV2 curve has better selectivity and sensitivity near the focus. The FV1 curve applies to scenarios with large noises. The FV2 curve shows high immunity to interference under the normal illumination, and has sharper peaks near the focus. Therefore, the FV2 curve can be used to perform over-peak judgment and reduce overshoot. You can use the two FV curves flexibly as required.

Figure 2-1 FV curves



More than 500 FV sampling points are captured after the focus is fixed on the outdoor leave scenario, as shown in Figure 2-2. The capture lasts for about one minute, during which the sunlight intensity gradually becomes lower. The fluctuation stability of only about 2% of the FV2 curve is better than that of the FV1 curve. Therefore, the FV2 curve is recommended when the illumination is normal.

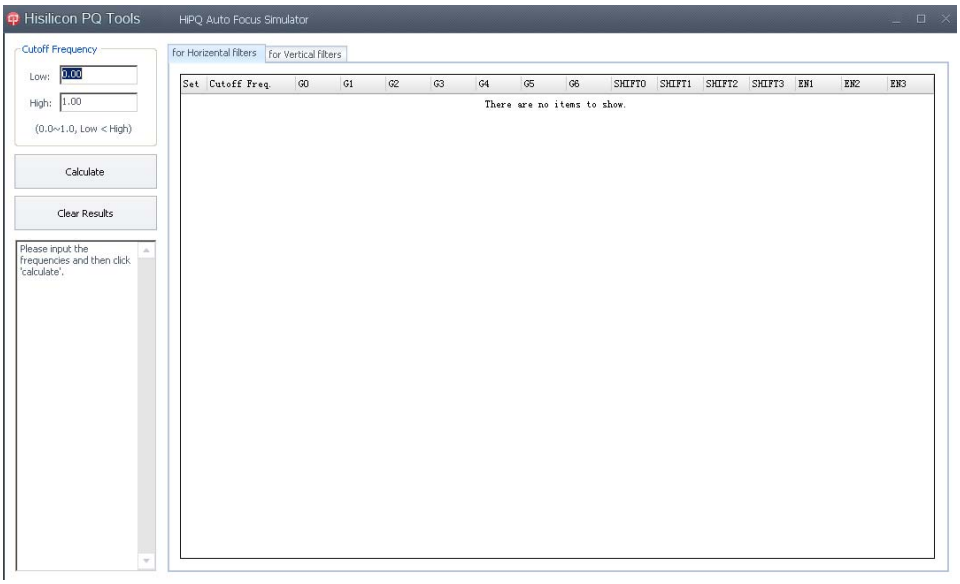
Figure 2-2 FV curves in a fixed scenario



2.2 User Implementation

The HiSilicon PQ Tools integrates the HiPQ Auto Focus Simulator, as shown in Figure 2-3. You can set the high and low filter cutoff frequencies to values that fall within [0, 1]. Then the configuration parameters of the AF statistical module are generated. Note that the frequencies are the digital angular frequencies normalized to π. The precision of the IIR filter coefficients is S10Q8 (8-bit fixed-point decimal), and the precision of the FIR filter coefficients is S6Q2.

Figure 2-3 HiPQ Auto Focus Simulator



You can flexibly use the statistics obtained by calling the corresponding MPIs and perform the calculation using you own method.

2.3 Reference Code for Calculating FVs

```
#include <stdio.h>
#include <string.h>
#include "hi_type.h"
#include "mpi_isp.h"
#include "mpi_ae.h"

#define BLEND_SHIFT  6
#define ALPHA        64          // 1
#define BELTA        54          // 0.85

#define AF_BLOCK_8X8
```

```
#ifdef AF_BLOCK_8X8
static int AFWeight[8][8] = {{1,1,1,1,1,1,1,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,2,2,2,2,2,2,1,},
                             {1,1,1,1,1,1,1,1,},
                            };
#else
static int AFWeight[15][17] = {{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1},
                             {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                            };
#endif

static int MapISO(int iso);
static __inline int iMin2(int a, int b) {{ if (a > b) a = b; }; return a; }
static __inline int iMax2(int a, int b) {{ if (a > b) b = a; }; return b; }

int main(int argc, char* argv[])
{
    HI_S32 s32Ret = HI_SUCCESS;
    HI_U32 u32FrmCnt = 0;
    ISP_DEV IspDev;
    HI_U32 u32Iso;
    ISP_EXP_INFO_S stExpInfo;
    ISP_STATISTICS_CFG_S stIspStaticsCfg;
    ISP_STATISTICS_S stIspStatics;
    ISP_VD_INFO_S stVdInfo;
```

```
    ISP_PUB_ATTR_S stPubattr;
#ifdef AF_BLOCK_8X8
    ISP_FOCUS_STATISTICS_CFG_S  stFocusCfg =
    {
        {1, 8, 8, 1920, 1080, 1, 0},
        {{0, 1, 1}, {188,  0,   0,    414, -165, 400, -164}, {7, 0, 3, 2}, 127},
        {{0, 1, 0}, {200,  0,   0,    200,   -55,  0,   0},    {6, 0, 1, 0}, 31},
        {{-6,  12, 22, 12, -6}, 511},
        {{-16, 21, 0, -21, 16}, 10},
        {0, {0, 0}, {3, 2}}
    };
#else
    ISP_FOCUS_STATISTICS_CFG_S  stFocusCfg =
    {
        {1, 17, 15, 1920, 1080, 1, 0},
        {{0, 1, 1}, {188,  0,   0,    414, -165, 400, -164}, {7, 0, 3, 2}, 127},
        {{0, 1, 0}, {200,  0,   0,    200,   -55,  0,   0},    {6, 0, 1, 0}, 31},
        {{-6,  12, 22, 12, -6}, 511},
        {{-16, 21, 0, -21, 16}, 10},
        {0, {0, 0}, {3, 2}}
    };
#endif
    if (argc < 2)
    {
        printf("use like: ./sample_af A    ---> auto threshold\n");
        printf("......... ./sample_af M    ---> manual threshold\n");
        return HI_FAILURE;
    }

    //step1: set AF static window to 8x8 block
    IspDev = 0;
    s32Ret = HI_MPI_ISP_GetStatisticsConfig(IspDev, & stIspStaticsCfg);
    s32Ret |= HI_MPI_ISP_GetPubAttr(IspDev, &stPubattr);

    stFocusCfg.stConfig.u16Vsize = stPubattr.stWndRect.u32Height;
    stFocusCfg.stConfig.u16Hsize = stPubattr.stWndRect.u32Width;

    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_ISP_GetStatisticsConfig error!(s32Ret = 0x%x)\n",
s32Ret);
        return HI_FAILURE;
    }
    memcpy(&stIspStaticsCfg.stFocusCfg, &stFocusCfg,
```

```
sizeof(ISP_FOCUS_STATISTICS_CFG_S));
    s32Ret = HI_MPI_ISP_SetStatisticsConfig(IspDev, & stIspStaticsCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_ISP_SetStatisticsConfig error!(s32Ret = 0x%x)\n",
s32Ret);
        return HI_FAILURE;
    }

    //step2:calculate FV for every frame
    while (1)
    {
        s32Ret = HI_MPI_ISP_GetVDTimeOut(IspDev, &stVdInfo, 5000);
        s32Ret |= HI_MPI_ISP_GetStatistics(IspDev, &stIspStatics);

        if (HI_SUCCESS != s32Ret)
        {
            printf("HI_MPI_ISP_GetStatistics error!(s32Ret = 0x%x)\n",
s32Ret);
            return HI_FAILURE;
        }

        HI_U32 i, j;
        HI_U32 u32SumFv1 = 0;
        HI_U32 u32SumFv2 = 0;
        HI_U32 u32WgtSum = 0;
        HI_U32 u32Fv1_n, u32Fv2_n, u32Fv1, u32Fv2;

        if ((++u32FrmCnt % 2))
        {
            continue;
        }

        for ( i = 0 ; i < stFocusCfg.stConfig.u16Vwnd; i++ )
        {
            for ( j = 0 ; j < stFocusCfg.stConfig.u16Hwnd; j++ )
            {

                HI_U32 u32H1 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16h1;
                HI_U32 u32H2 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16h2;
                HI_U32 u32V1 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16v1;
```

```
                HI_U32 u32V2 =
stIspStatics.stFocusStat.stZoneMetrics[i][j].u16v2;

                u32Fv1_n = (u32H1 * ALPHA + u32V1 * ((1<<BLEND_SHIFT) - ALPHA)) >>
BLEND_SHIFT;
                u32Fv2_n = (u32H2 * BELTA + u32V2 * ((1<<BLEND_SHIFT) - BELTA)) >>
BLEND_SHIFT;
                u32SumFv1 += AFWeight[i][j] * u32Fv1_n;
                u32SumFv2 += AFWeight[i][j] * u32Fv2_n;
                u32WgtSum += AFWeight[i][j];
            }
        }

        u32Fv1 = u32SumFv1 / u32WgtSum;
        u32Fv2 = u32SumFv2 / u32WgtSum;

        printf("%4d    %4d\n", u32Fv1, u32Fv2);

        /* update AF parameters depend on noise intensity */
        if ('M' != *argv[1])
        {
            HI_MPI_ISP_QueryExposureInfo(IspDev, &stExpInfo);
            u32Iso =  (HI_U64)stExpInfo.u32AGain * stExpInfo.u32DGain *
stExpInfo.u32ISPDGain * 100 >> 30;

            stIspStaticsCfg.stFocusCfg.stHParam_IIR0.u16IIRThd =
MapISO(u32Iso) << 3;
            s32Ret |= HI_MPI_ISP_SetStatisticsConfig(IspDev,
&stIspStaticsCfg);
        }
    }

    return HI_SUCCESS;
}


static int MapISO(int iso)
{
  int   j,  i = (iso >= 200);
i += ( (iso >= (200 << 1)) + (iso >= (400 << 1)) + (iso >= (400 << 2)) + (iso >=
(400 << 3)) + (iso >= (400 << 4)) );
i += ( (iso >= (400 << 5)) + (iso >= (400 << 6)) + (iso >= (400 << 7)) + (iso >=
(400 << 8)) + (iso >= (400 << 9)) );
j  = ( (iso >  (112 << i)) + (iso >  (125 << i)) + (iso >  (141 << i)) + (iso >
```

```
(158 << i)) + (iso >  (178 << i)) );
return (i * 6 + j + (iso >= 25) + (iso >= 50) + (iso >= 100));
}
```

2.4 FV Capture in a Specific Scenario

```
//-----------------------------------------------------------------------//
Sensor: Aptina AR0230
Lens: 60 mm
Focus range: 1 m-infinite
Image format: YUV
//-----------------------------------------------------------------------//


//---------------- AF Logic Configuration ------------------//
   ISP_FOCUS_STATISTICS_CFG_S  stFocusCfg =
   {
       {1, 8, 8, 1920, 1080, 1, 0},
       {{0, 1, 1}, {188,  0,   0,    414, -165, 400, -164}, {7, 0, 3, 2}, 127},
       {{0, 1, 0}, {200,  0,   0,    200,   -55,  0,   0},  {6, 0, 1, 0}, 31},
       {{-6, 12, 22, 12, -6}, 511},
       {{-16, 21, 0, -21, 16}, 10},
       {0, {0, 0}, {3, 2}}
};
//-----------------------------------------------------------------------//
```