Hi3516A/Hi3516D AF 统计模块

使用说明

文档版本 **03**

发布日期 **2016-03-21**

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：深圳市龙岗区坂田华为基地华为电气生产中心 邮编：518129

网址：http://www.hisilicon.com

客户服务电话：+86-755-28788858

客户服务传真：+86-755-28357515

客户服务邮箱：support@hisilicon.com

前 言

概述

本文档主要介绍 Hi3516A 的统计模块及 FV 的融合方法、配置，方便用户进行 AF 开发。

说明

本文以 Hi3516A 描述为例，未有特殊说明，Hi3516D 与 Hi3516A 一致。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3516A	V100
Hi3516D	V100

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 03 (2016-03-21)

第 3 次正式发布。

2.1.1 小节涉及刷新。

文档版本 02 (2015-10-30)

第 2 次正式发布，修改第 2 章的相关内容。

文档版本 01 (2015-06-12)

第 1 次正式发布。

目 录

插图目录

1 海思 AF 统计模块概述

1.1 概述

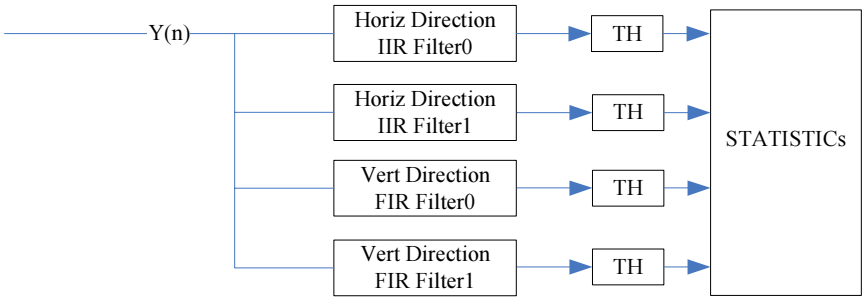
被动式自动对焦一般是通过分析图像特征得出图像清晰度值 FV(Focus Value)，通过驱动对焦马达调节焦点到最佳位置，获取图像清晰度算法有多种，如灰度梯度法，高频分量法等，Hi3516A 采用高频分量法来计算 FV，即图像越清晰的时候高频部分幅值越大，将图像通过高通滤波器便可以得到高频分量，Hi3516A 一共提供四个滤波器和亮度信息，分别为水平方向滤波 H1、H2，垂直方向滤波 V1、V2，以及 Y。

1.2 统计模块结构

1.2.1 概述

Hi3516A 使用 IIR 和 FIR 分别作为水平垂直方向滤波器，对滤波器输出进行 abs 后通过 threshold 门限判断，超过门限的计入统计。相应逻辑框图如图 1-1 所示。

图1-1 AF 统计模块框图



1.2.2 统计模式配置

在统计模块里还有一些模式设置，比如：

- 峰值 peak 模式

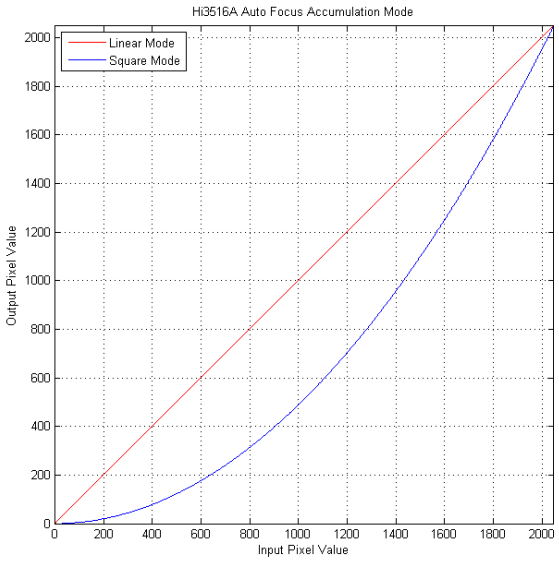
设置为 peak 模式 block 的统计值为滤波后图像每行最大值之和，否则直接将每个点的值求和。求和模式适用于噪声较大场景。

- 平方 squ 模式

  设置为 squ 模式后，会先对滤波器输出进行归一化后进行平方再做统计。

  平方模式的 FV 曲线在焦点附近会更加陡峭，相应的映射曲线如图 1-2 所示。

图1-2 映射曲线图



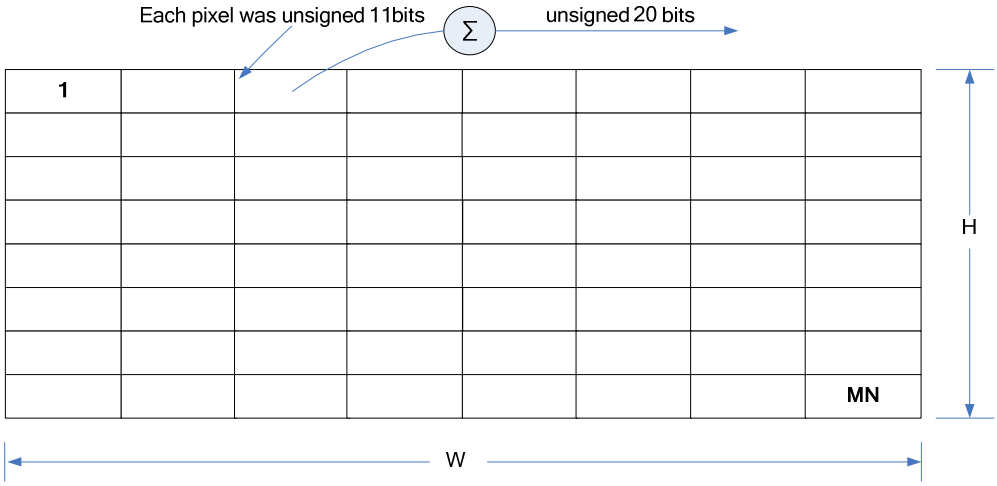
b. 输出 shift

对输出值 right shift 缩小，防止值溢出。

c. Block 大小的配置

对通过滤波器后的图像可以进行分块统计，如图 1-3 所示，块的大小可以设置，Hi3516A 最大支持 17*15 个 blocks，最终块的统计值是对每个 pixel 的累加或者对每行最大值累加。因为累加器输出为 unsigned 20bits，滤波器输出 pixel 宽度为 unsigned 11bits，所以当块内输出值都为 2047 的时候，最大支持累加的 pixel 个数为 512，否者会发生溢出，用户设置大小时应该注意。

图1-3 滤波器后的图像



2 FV 的融合

2.1 海思推荐的方法

2.1.1 统计模块配置

调用 HI_MPI_ISP_SetStatisticsConfig 接口配置 stFocusCfg 字段以下内容，以 1080p 为例。

```
{
    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}}
};
```

如果用户使用分辨更高的 sensor，输出可能会溢出，stFVParam 中的几个字段应该同时增加相同 shift 量，如变为{0, {1, 1,}, {4, 3}}，增加的大小需要对照每个 block 中的值，直到不发生溢出。

当图像噪声较大时，由于中高频频带 SNR 很小，所以推荐使用较低的 PassBand。并设置 enPeakMode = ISP_AF_STA_NORM，enSquMode = ISP_AF_STA_SUM_SQU。推荐滤波器系数如下。

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

2.1.2 统计值的获取

当一帧图像最后一个 pixel 通过 AF 模块后，统计值即更新，推荐用户通过 HI_MPI_ISP_GetVDTimeOut 同步获取统计值，然后 AutoFocus 和 ZoomTracking 算法完成目标 focus 和 zoom postion 的计算，需要注意的是因为 LINUX user space 任务调度不能保证一致的实时性，建议将需要保证实时性的驱动配置放在 kernel space 完成。ISP 提供同步回调接口的注册，可以实现与 VD 同步。在 mpp\extdrv\sample_ist 有相应的 sample，用户可以将实时性要求较高的任务放在同步回调里面，底层提供 HwIRQ，Tasklet，Workqueue 三种方式实现，可以选择相应的实现方式以确定实时级别。

2.1.3 FV 值计算

对于每一个 block，FV1，FV2 的计算方式为：

$$\begin{pmatrix} FV1\_n = \alpha * H1\_n + (1-\alpha) * V1\_n \\ FV2\_n = \beta * H2\_n + (1-\beta) * V2\_n \end{pmatrix}$$

H1，H2，V1，V2 分别为水平垂直四组滤波器，最终的 FV 值还需要对每个 block 进行加权，即 FV1，FV2 采用如下公式计算。

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

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

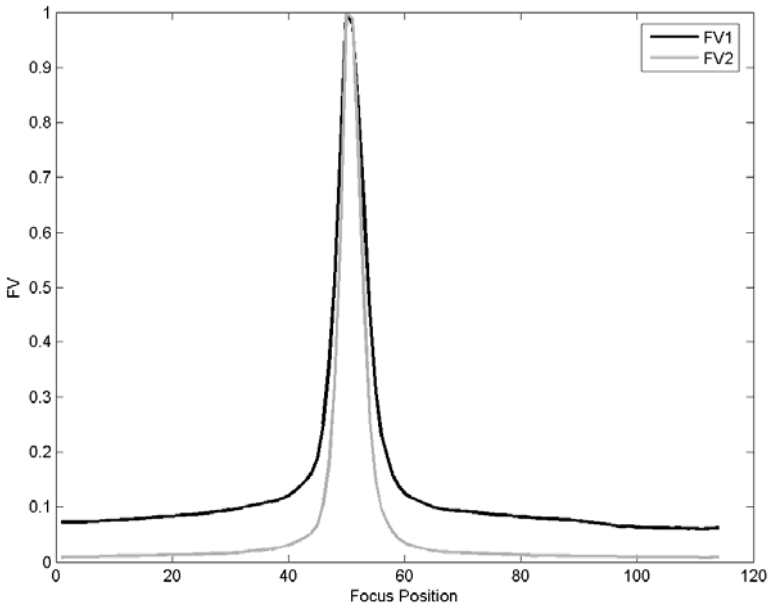
推荐权重表如下：

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

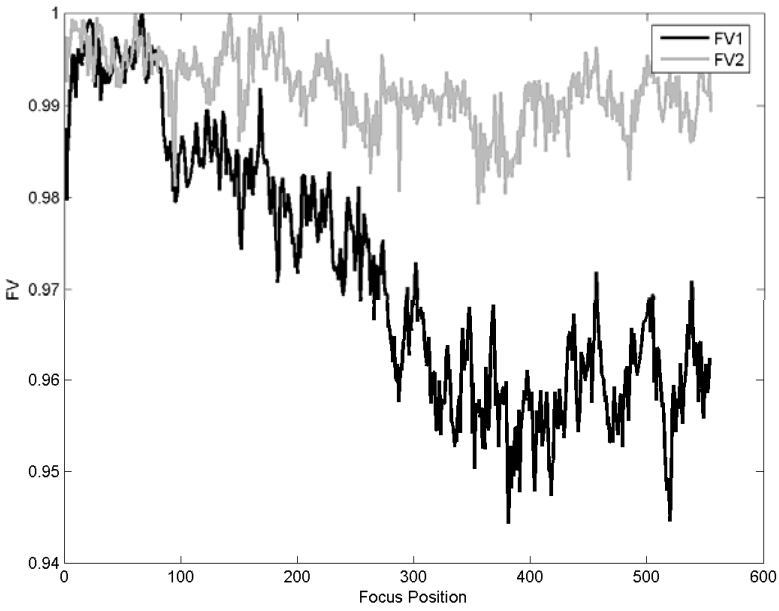
相应的示例代码可以在附录中找到。

融合后的 FV 曲线如图 2-1 所示，FV1 较 FV2 在失焦区域更能反映图像清晰变化趋势，FV2 较 FV1 在焦点附近有更好的选择性和灵敏度，FV1 更适合在噪声较大的场景使用，FV2 则在良好照度下具有较好的抗干扰性，并且在焦点附近有更加尖锐的 peak，使用它来作过峰判断能减小过冲，用户应该根据实际需求灵活应用这两个 FV。

图2-1 FV 曲线

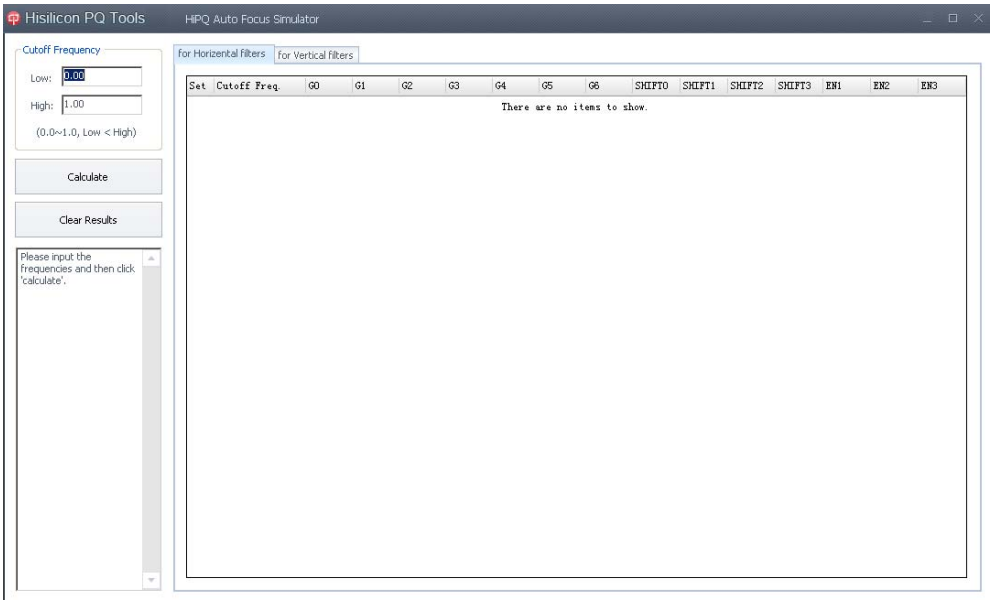


下图是对着一固定场景（室外树叶）固定 focus 收集了 500 多个 FV 采样点，时间约一分钟，期间日照强度缓慢变化变暗，可以看出 FV2 只有 2%左右的波动稳定性优于 FV1，因此推荐在照度较好的情况下，优先参考 FV2。



2.2 用户实现

PQtools 中集成了 HiPQ Auto Focus Simulator 插件，用户可以在[0, 1]之间设置滤波器截止频率即可生成统计模块配置参数，需要注意的是此处的频率是数字角频率对 π 的归一化。IIR 滤波器系数的精度为 S10Q8 即定点 8bit 小数，FIR 滤波器的精度为 S6Q2。



用户可以灵活的选择使用 MPI 接口获取的统计值，按照自己的方法进行计算。

2.3 FV 计算参考代码

```
#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 widnow 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 场景 FV 采集

```
//-----------------------------------------------------------------------//
Sensor:         Aptina AR0230
Lens:           60mm
```

```
Focus Range:    1m ~ inf
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}}
};
//---------------------------------------------------------------------//
```