深圳市灵明光子科技有限公司
Shenzhen Adaps Photonics Technology Co.,Ltd

Ads6401 dToF SDK for Linux User Guide

V3.0.0

Adaps Photonics

2025-8-28

目录

1. 目的

1.1 提供Linux系统中应用层，对下与驱动层接口以及对上与算法库接口API的介绍及示范代码

1.2 为客户在相关平台中的集成工作提供参照。

2. 系统架构

2.1 模组硬件架构

我司的 Ads6401 芯片目前已开发两种类型的模组：散点模组 和 小面阵模组。

- 散点模组使用OPN7020作为vcsel driver芯片，内置64K bytes的eeprom用于存储标定数据。
- 小面阵模组使用PhotonIC 5015作为vcsel driver芯片，内置32K bytes的eeprom用于存储标定数据，同时内置一个MCU用于控制vcsel driver芯片，温度采集和vop电压控制等。

3. Linux SDK

3.1 与 Linux 驱动层 v4l2 dToF sensor driver 通讯的接口

在 adaps_dtof_uapi.h 文件中定义的一些 ioctl 命令：

```
struct adaps_dtof_intial_param {
   AdapsEnvironmentType env_type;
   AdapsMeasurementType measure_type;
   AdapsPowerMode  power_mode;
   AdapsFramerateType framerate_type;
   AdapsVcselZoneCountType vcselzonecount_type;

   UINT8 rowOffset;
   UINT8 colOffset;
   UINT8 rowSearchingRange;
   UINT8 colSearchingRange;

   // The following config are for Advanced user only, just set them to 0 (the
default setting will be used in ads6401.c driver code) if you are not clear what
they are.
    UINT8 grayExposure;
    UINT8 coarseExposure;
    UINT8 fineExposure;
    UINT8 laserExposurePeriod;  // laser_exposure_period, register configure value
    bool roi_sram_rolling;
};

struct adaps_dtof_runtime_param{
   AdapsEnvironmentType env_type;
   AdapsMeasurementType measure_type;
   AdapsVcselMode vcsel_mode;
   bool env_valid;
   bool measure_valid;
   bool vcsel_valid;
};

struct adaps_dtof_exposure_param{
    __u8 ptm_coarse_exposure_value;//ptm_coarse_exposure_value, register configure
value
    __u8 ptm_fine_exposure_value;// ptm_fine_exposure_value, register configure
value
    __u8 pcm_gray_exposure_value;// pcm_gray_exposure_value, register configure
value
    __u8 exposure_period;  // laser_exposure_period, register configure value
};

struct adaps_dtof_runtime_status_param {
    bool test_pattern_enabled;
    __u32 inside_temperature_x100; //since kernel doesn't use float type, this is a
expanded integer value (x100), Eg 4515 means 45.15 degree
    __u32 expected_vop_abs_x100;
    __u32 expected_pvdd_x100;
};

struct adaps_dtof_module_static_data{
    __u32 module_type;  // refer to ADS6401_MODULE_SPOT/ADS6401_MODULE_FLOOD/... of
adaps_types.h file
    __u32 eeprom_capacity;      // unit is byte
    __u16 otp_vbe25;
    __u16 otp_vbd;       // unit is 10mv, or the related V X 100
    __u16 otp_adc_vref;
    __u8 chip_product_id[SWIFT_PRODUCT_ID_SIZE];
    __u8 sensor_drv_version[FW_VERSION_LENGTH];
    __u8 ready;
    __u8 eeprom_crc_matched;
```

```
};

struct adaps_dtof_update_eeprom_data{
    __u32 module_type;  // refer to ADS6401_MODULE_SPOT/ADS6401_MODULE_FLOOD/... of adaps_types.h file
    __u32 eeprom_capacity;      // unit is byte
    __u32 offset;           //eeprom data start offset
    __u32 length;              //eeprom data length
};

typedef struct {
    __u8 work_mode;
    __u16 sensor_reg_setting_cnt;
    __u16 vcsel_reg_setting_cnt;
} external_config_script_param_t;

typedef struct {
    __u32 roi_sram_size;
} external_roisram_data_size_t;

#define ADAPS_SET_DTOF_INITIAL_PARAM      \
    _IOW('T', ADAPS_DTOF_PRIVATE + 0, struct adaps_dtof_intial_param)

// This command has been deprecated; do not use it anymore.
#define ADAPS_UPDATE_DTOF_RUNTIME_PARAM      \
    _IOW('T', ADAPS_DTOF_PRIVATE + 1, struct adaps_dtof_runtime_param)

#define ADAPS_GET_DTOF_RUNTIME_STATUS_PARAM      \
    _IOR('T', ADAPS_DTOF_PRIVATE + 2, struct adaps_dtof_runtime_status_param)

#define ADAPS_GET_DTOF_MODULE_STATIC_DATA      \
    _IOR('T', ADAPS_DTOF_PRIVATE + 3, struct adaps_dtof_module_static_data)

#define ADAPS_GET_DTOF_EXPOSURE_PARAM      \
    _IOR('T', ADAPS_DTOF_PRIVATE + 4, struct adaps_dtof_exposure_param)

#define ADTOF_SET_DEVICE_REGISTER      \
    _IOW('T', ADAPS_DTOF_PRIVATE + 5, register_op_data_t *)

#define ADTOF_GET_DEVICE_REGISTER      \
    _IOR('T', ADAPS_DTOF_PRIVATE + 6, register_op_data_t *)

#define ADTOF_SET_EXTERNAL_CONFIG_SCRIPT      \
    _IOW('T', ADAPS_DTOF_PRIVATE + 7, external_config_script_param_t *)

// This command carries the risk of damaging the module calibration data and is restricted to internal use at adaps company only.
```

```
#define ADTOF_UPDATE_EEPROM_DATA      \
   _IOW('T', ADAPS_DTOF_PRIVATE + 8, struct adaps_dtof_update_eeprom_data)

#define ADTOF_SET_EXTERNAL_ROISRAM_DATA_SIZE      \
   _IOW('T', ADAPS_DTOF_PRIVATE + 9, external_roisram_data_size_t *)
```

重要数据结构

在 adaps_types.h 文件定义了一些数据结构:

```
enum adaps_work_mode {
   ADAPS_PTM_PHR_MODE   = 0,
   ADAPS_PCM_MODE = 1,
   ADAPS_PTM_FHR_MODE   = 2,
   ADAPS_PTM_DEBUG_PHR_MODE = 3,
   ADAPS_PTM_DEBUG_FHR_MODE=4,
   ADAPS_MODE_MAX,
};
```

该结构提供了 Swift 芯片的运行模式。

```
typedef enum
{
   AdapsMeasurementTypeUninitilized,
   AdapsMeasurementTypeNormal,
   AdapsMeasurementTypeShort,
   AdapsMeasurementTypeFull,
} AdapsMeasurementType;
```

测距范围类型：未指定/正常/近/全距离

```
typedef enum {
   AdapsEnvTypeUninitilized,
   AdapsEnvTypeIndoor,
   AdapsEnvTypeOutdoor,
} AdapsEnvironmentType;
```

所处环境类型：未指定/室内/室外

```
typedef enum {
   AdapsVcselModeUninitilized,
   AdapsVcselModeOn,
   AdapsVcselModeOff,
} AdapsVcselMode;
```

Vcsel开关类型：未指定/开/关

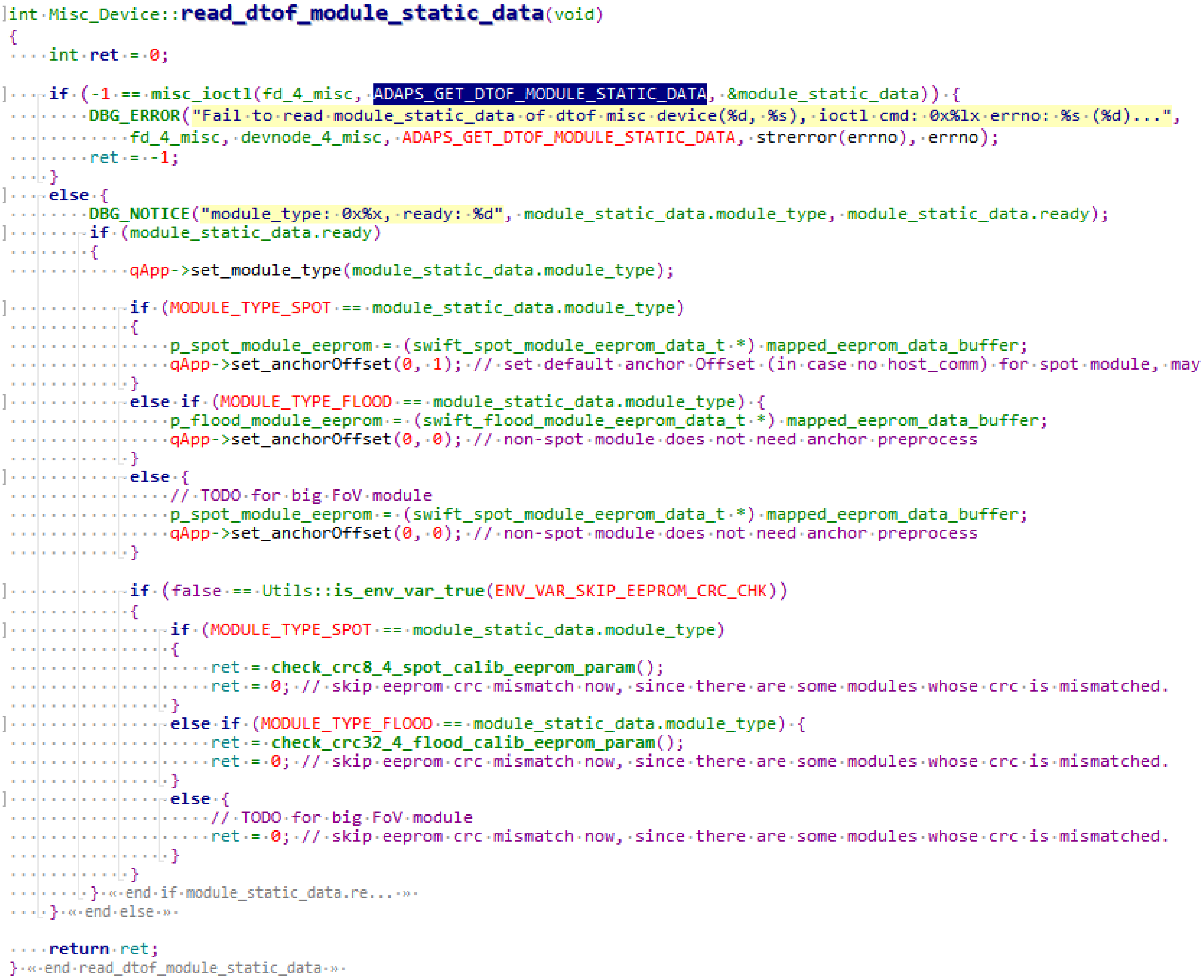
```
typedef enum
{
   AdapsVcselZoneCountUninitilized,
   AdapsVcselZoneCount1,
   AdapsVcselZoneCount4 = 4,
} AdapsVcselZoneCountType;
```

Vcsel分区类型：未指定/一分区/四分区

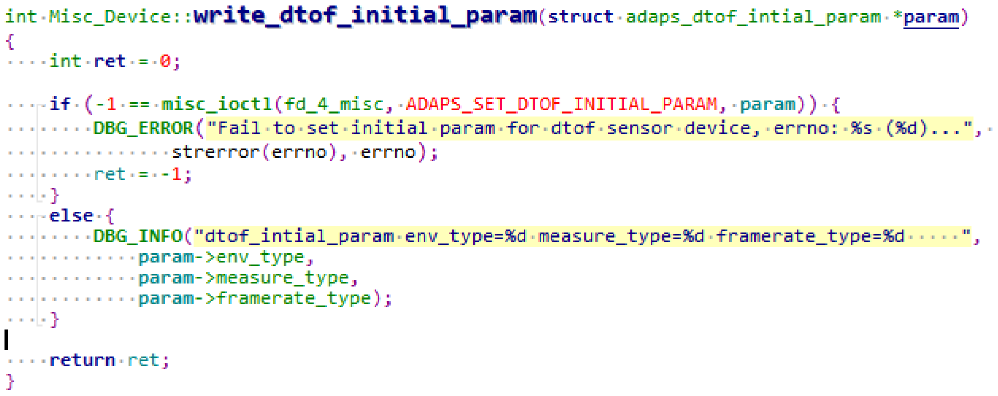
```
typedef enum
{
   AdapsFramerateTypeUninitilized,
   AdapsFramerateType15FPS,
   AdapsFramerateType25FPS,
   AdapsFramerateType30FPS,
   AdapsFramerateType60FPS,
} AdapsFramerateType;
```

测距模式帧率类型：未指定/15/25/30/60 FPS，这里是指4合一后完整图像帧的帧率

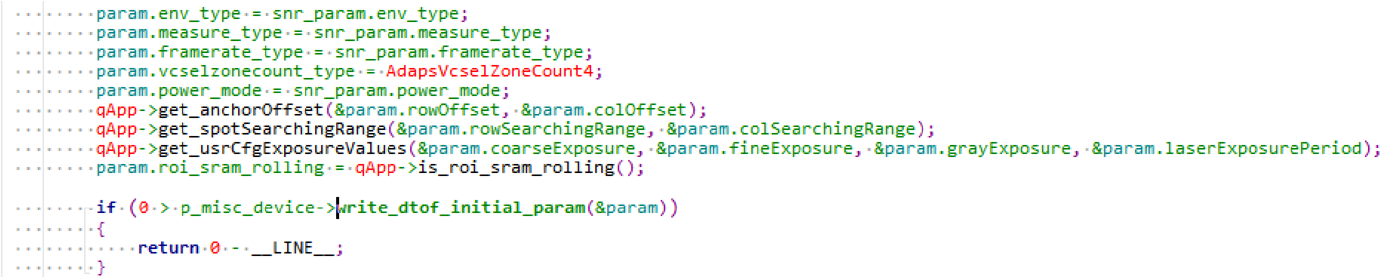
接口实际使用示范代码



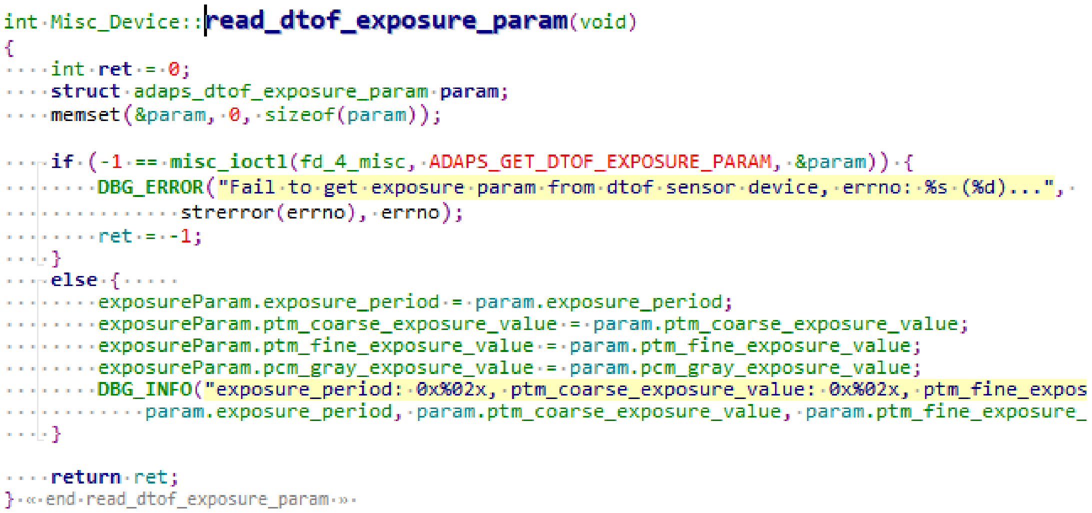
```
int Misc_Device::read_dtof_module_static_data(void)
{
    int ret = 0;

    if (-1 == misc_ioctl(fd_4_misc, ADAPS_GET_DTOF_MODULE_STATIC_DATA, &module_static_data)) {
        DBG_ERROR("Fail to read module_static_data of dtof misc device(%d, %s), ioctl cmd: 0x%lx errno: %s (%d)...",
            fd_4_misc, devnode_4_misc, ADAPS_GET_DTOF_MODULE_STATIC_DATA, strerror(errno), errno);
        ret = -1;
    }
    else {
        DBG_NOTICE("module_type: 0x%x, ready: %d", module_static_data.module_type, module_static_data.ready);
        if (module_static_data.ready)
        {
            qApp->set_module_type(module_static_data.module_type);

            if (MODULE_TYPE_SPOT == module_static_data.module_type)
            {
                p_spot_module_eeprom = (swift_spot_module_eeprom_data_t *) mapped_eeprom_data_buffer;
                qApp->set_anchorOffset(0, 1); // set default anchor Offset (in case no host_comm) for spot module, may
            }
            else if (MODULE_TYPE_FLOOD == module_static_data.module_type) {
                p_flood_module_eeprom = (swift_flood_module_eeprom_data_t *) mapped_eeprom_data_buffer;
                qApp->set_anchorOffset(0, 0); // non-spot module does not need anchor preprocess
            }
            else {
                // TODO for big FoV module
                p_spot_module_eeprom = (swift_spot_module_eeprom_data_t *) mapped_eeprom_data_buffer;
                qApp->set_anchorOffset(0, 0); // non-spot module does not need anchor preprocess
            }

            if (false == Utils::is_env_var_true(ENV_VAR_SKIP_EEPROM_CRC_CHK))
            {
                if (MODULE_TYPE_SPOT == module_static_data.module_type)
                {
                    ret = check_crc8_4_spot_calib_eeprom_param();
                    ret = 0; // skip eeprom crc mismatch now, since there are some modules whose crc is mismatched.
                }
                else if (MODULE_TYPE_FLOOD == module_static_data.module_type) {
                    ret = check_crc32_4_flood_calib_eeprom_param();
                    ret = 0; // skip eeprom crc mismatch now, since there are some modules whose crc is mismatched.
                }
                else {
                    // TODO for big FoV module
                    ret = 0; // skip eeprom crc mismatch now, since there are some modules whose crc is mismatched.
                }
            }
        } « end if module_static_data.re... »
    } « end else »

    return ret;
} « end read_dtof_module_static_data »
```



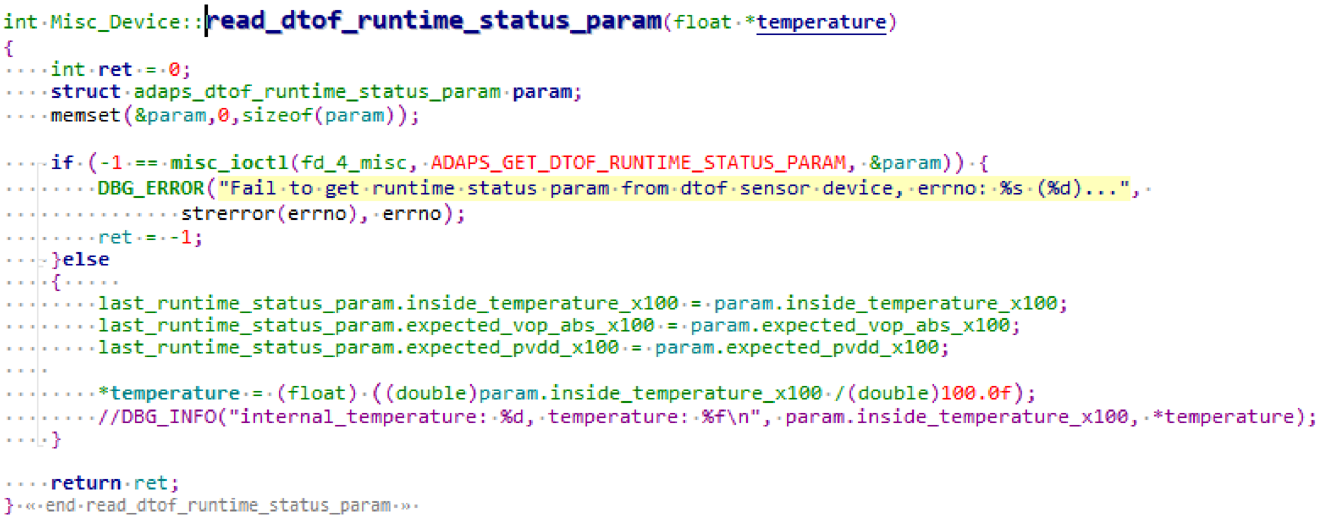
```
int Misc_Device::write_dtof_initial_param(struct adaps_dtof_intial_param *param)
{
    int ret = 0;

    if (-1 == misc_ioctl(fd_4_misc, ADAPS_SET_DTOF_INITIAL_PARAM, param)) {
        DBG_ERROR("Fail to set initial param for dtof sensor device, errno: %s (%d)...",
               strerror(errno), errno);
        ret = -1;
    }
    else {
        DBG_INFO("dtof_intial_param env_type=%d measure_type=%d framerate_type=%d     ",
            param->env_type,
            param->measure_type,
            param->framerate_type);
    }

    return ret;
}
```



```
        param.env_type = snr_param.env_type;
        param.measure_type = snr_param.measure_type;
        param.framerate_type = snr_param.framerate_type;
        param.vcselzonecount_type = AdapsVcselZoneCount4;
        param.power_mode = snr_param.power_mode;
        qApp->get_anchorOffset(&param.rowOffset, &param.colOffset);
        qApp->get_spotSearchingRange(&param.rowSearchingRange, &param.colSearchingRange);
        qApp->get_usrCfgExposureValues(&param.coarseExposure, &param.fineExposure, &param.grayExposure, &param.laserExposurePeriod);
        param.roi_sram_rolling = qApp->is_roi_sram_rolling();

        if (0 > p_misc_device->write_dtof_initial_param(&param))
        {
            return 0 - __LINE__;
        }
```



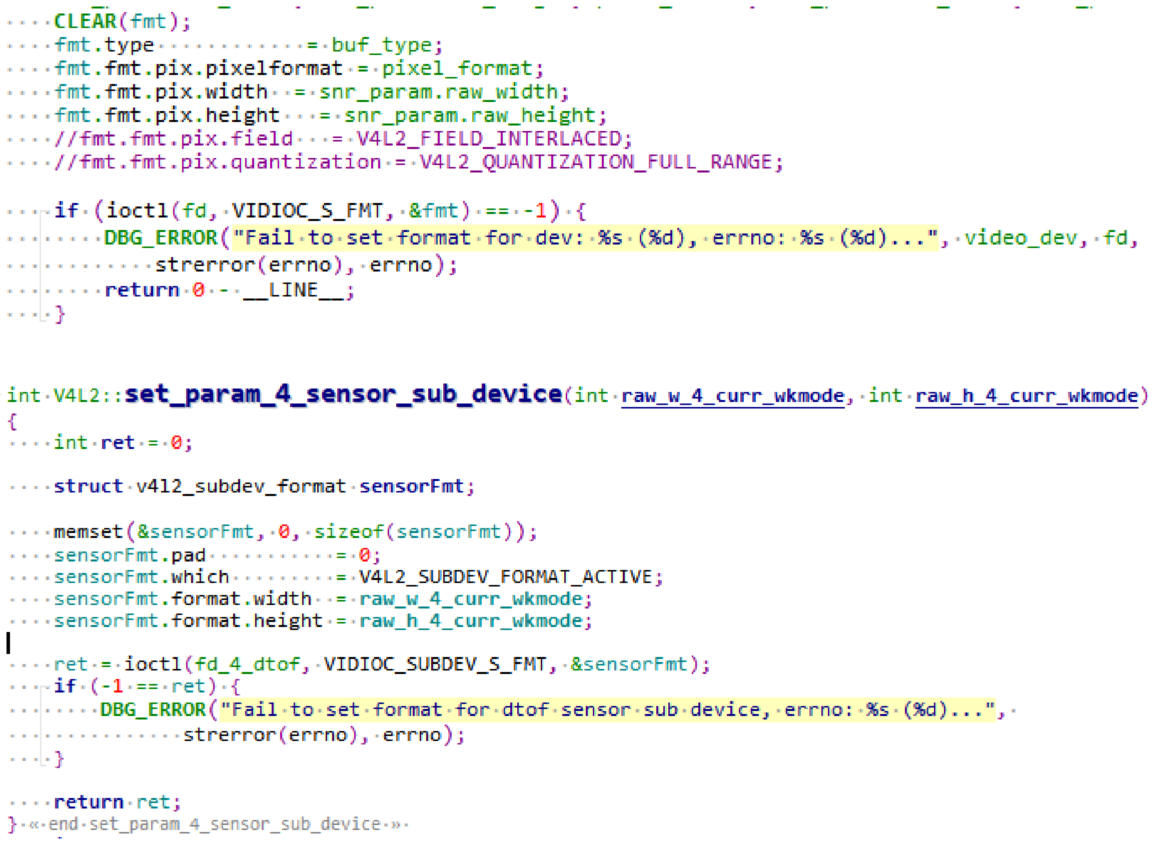
```
int Misc_Device::read_dtof_exposure_param(void)
{
    int ret = 0;
    struct adaps_dtof_exposure_param param;
    memset(&param, 0, sizeof(param));

    if (-1 == misc_ioctl(fd_4_misc, ADAPS_GET_DTOF_EXPOSURE_PARAM, &param)) {
        DBG_ERROR("Fail to get exposure param from dtof sensor device, errno: %s (%d)...", 
                strerror(errno), errno);
        ret = -1;
    }
    else {     
        exposureParam.exposure_period = param.exposure_period;
        exposureParam.ptm_coarse_exposure_value = param.ptm_coarse_exposure_value;
        exposureParam.ptm_fine_exposure_value = param.ptm_fine_exposure_value;
        exposureParam.pcm_gray_exposure_value = param.pcm_gray_exposure_value;
        DBG_INFO("exposure_period: 0x%02x, ptm_coarse_exposure_value: 0x%02x, ptm_fine_expos
            param.exposure_period, param.ptm_coarse_exposure_value, param.ptm_fine_exposure_
    }

    return ret;
} « end read_dtof_exposure_param » 
```



```
int Misc_Device::read_dtof_runtime_status_param(float *temperature)
{
    int ret = 0;
    struct adaps_dtof_runtime_status_param param;
    memset(&param,0,sizeof(param));

    if (-1 == misc_ioctl(fd_4_misc, ADAPS_GET_DTOF_RUNTIME_STATUS_PARAM, &param)) {
        DBG_ERROR("Fail to get runtime status param from dtof sensor device, errno: %s (%d)...", 
                strerror(errno), errno);
        ret = -1;
    }else
    {     
        last_runtime_status_param.inside_temperature_x100 = param.inside_temperature_x100;
        last_runtime_status_param.expected_vop_abs_x100 = param.expected_vop_abs_x100;
        last_runtime_status_param.expected_pvdd_x100 = param.expected_pvdd_x100;
    
        *temperature = (float) ((double)param.inside_temperature_x100 /(double)100.0f);
        //DBG_INFO("internal_temperature: %d, temperature: %f\n", param.inside_temperature_x100, *temperature);
    }

    return ret;
} « end read_dtof_runtime_status_param » 
```

3.2 与 Linux 驱动层 v4l2 framework 的接口

基于 Linux v4l2 framework 的 camera sensor 驱动和应用，需要通过 VIDIOC_S_FMT 及 VIDIOC_SUBDEV_S_FMT ioctl 命令来配置当前工作模式的参数，以便驱动层 v4l2 framework core 层申请适当大小的 buffer，sensor 驱动里获悉应用使用的 work mode 从而配置相应的寄存器。



```
    CLEAR(fmt);
    fmt.type             = buf_type;
    fmt.fmt.pix.pixelformat = pixel_format;
    fmt.fmt.pix.width  = snr_param.raw_width;
    fmt.fmt.pix.height   = snr_param.raw_height;
    //fmt.fmt.pix.field   = V4L2_FIELD_INTERLACED;
    //fmt.fmt.pix.quantization = V4L2_QUANTIZATION_FULL_RANGE;

    if (ioctl(fd, VIDIOC_S_FMT, &fmt) ==  -1) {
        DBG_ERROR("Fail to set format for dev: %s (%d), errno: %s (%d)...", video_dev, fd,
            strerror(errno), errno);
        return 0 - __LINE__;
    }


int V4L2::set_param_4_sensor_sub_device(int raw_w_4_curr_wkmode, int raw_h_4_curr_wkmode)
{
    int ret = 0;

    struct v4l2_subdev_format sensorFmt;

    memset(&sensorFmt, 0, sizeof(sensorFmt));
    sensorFmt.pad           = 0;
    sensorFmt.which         = V4L2_SUBDEV_FORMAT_ACTIVE;
    sensorFmt.format.width  = raw_w_4_curr_wkmode;
    sensorFmt.format.height = raw_h_4_curr_wkmode;

    ret = ioctl(fd_4_dtof, VIDIOC_SUBDEV_S_FMT, &sensorFmt);
    if (-1 == ret) {
        DBG_ERROR("Fail to set format for dtof sensor sub device, errno: %s (%d)...",
            strerror(errno), errno);
    }

    return ret;
} «end set_param_4_sensor_sub_device »
```

以上 raw_w_4_curr_wkmode 和 raw_h_4_curr_wkmode 帧的宽和高，其中帧宽是以 bytes 为单位，表示一行数据需要的内存空间大小。pixel_format 表示单个像素存储的格式，对于 ads6401 dToF sensor 而言，pixel_format 固定为 V4L2_PIX_FMT_SBGGR8，也就是 mipi 协议里的 RAW8。

以下是 ads6401 dToF sensor 常见 work mode 的分辨率信息：

Work mode	Raw_Width	Raw_Height	Depth_Width	Depth_Height
PCM (gray scale)	2560	32	210	160
FHR	4104	32	210	160
PHR	1032	32	210	160

现代计算机的 CPU 在访问内存时，对于按照特定字节边界对齐的数据，能够更快地进行读写操作。例如，某些 CPU 可能要求 4 字节或 8 字节对齐的数据

访问，这样可以减少内存访问的延迟。因此在 Linux 系统下，我们会经常发现 v4l2 framework 申请的帧 buffer 大小可能并不等于以上 Raw_Width * Raw_Height，而是会更大一些，这时每一行的占用的实际大小等于帧 buffer 的大小除以 Raw_Heigh（也就是 32），这个值减去 Raw_Width * Raw_Height 就是每一行末尾的 padding 字节大小。

另外，ads6401 dToF sensor 输出的深度图点阵虽然是 210 * 160，但是实际上只是部分点是有深度的，其余的点深度为 0，每个 zone 最多有 240 个点是具有有效深度的，所以 4 个 zone 总共最多有 960 个点有深度，这个是跟普通 RGB camera 不同的，请留意。

3.3 与算法库的接口

算法库主要有三个外部接口，位于 depthmapwrapper.h 文件中：

算法库初始化，将返回一个实例句柄存在 handler 指针里

```
int  DepthMapWrapperCreate(
   void** handler,
   WrapperDepthInitInputParams  inputParams,
   WrapperDepthInitOutputParams outputParams
);
```

对每一帧 mipi raw data 进行解码+深度运算，每 4 个 mipi 帧为一组，前 3 帧返回 false，第 4 帧返回 true 表示已生成一个完整的深度图像帧

```
bool DepthMapWrapperProcessFrame(
   void* handler,
   WrapperDepthInput in_image,
   WrapperDepthCamConfig *wrapper_depth_map_config,
   uint32_t num_outputs,
   WrapperDepthOutput outputs[]
   );
```

算法库销毁（释放资源），当停止出图时调用

```
void DepthMapWrapperDestroy(
    void * handler
);
```

算法库的重要数据结构

```
typedef enum {
    DEPTH_OUT_NORMAL,       ///< No change
    DEPTH_OUT_MIRROR,       ///< Mirror(horizontal)
    DEPTH_OUT_FLIP,         ///< Flip(vertical)
    DEPTH_OUT_MIRROR_FLIP,  ///< Mirror/Flip(h/v)
} RotateConfig;
```

```
typedef struct ADAPS_MIRROR_FRAME_SET
{
    UINT8 mirror_x;
    UINT8 mirror_y;
}AdapsMirrorFrameSet;
```

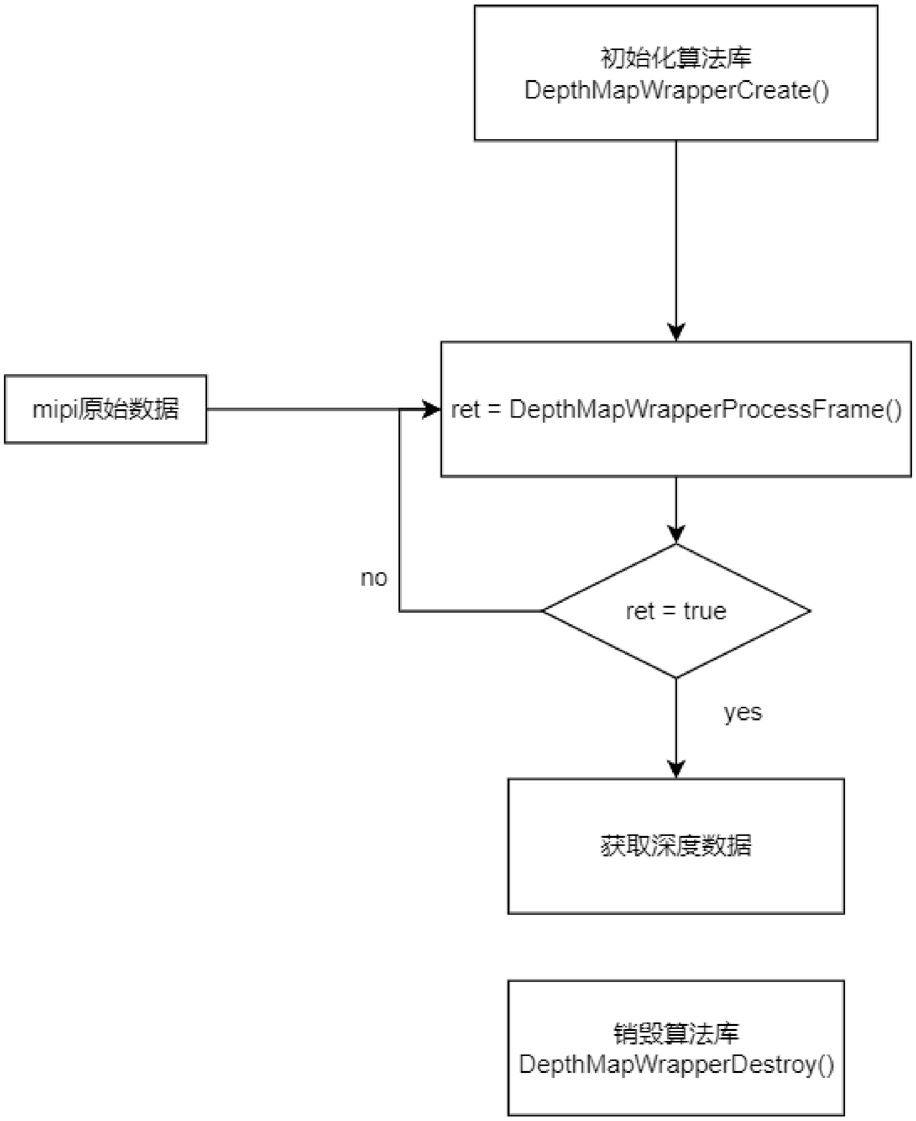
```
typedef struct {
    uint8_t work_mode;
    bool compose_subframe;
    bool expand_pixel;
    bool walkerror;
    AdapsMirrorFrameSet mirror_frame;
    float* adapsLensIntrinsicData;         // 9xsizeof(float)
    float* adapsSpodOffsetData;           // 4x240xsizeof(float)
    float* accurateSpotPosData;           // 4x240xsizeof(float)x2
    uint8_t ptm_fine_exposure_value;       // fine exposure value, 0 - 255
    uint8_t exposure_period;              // exposure_period, 0 - 255
    float cali_ref_tempe[2];  //[0] for indoor, [1] for outdoor
    float cali_ref_depth[2];  //[0] for indoor, [1] for outdoor
    AdapsEnvironmentType env_type;  // value 0-->indoor, value 1 -->outdoor
    AdapsMeasurementType  measure_type;  //value  0-->normal  distance,  1-->short
distance
    uint8_t *proximity_hist; //256 bytes for eeprom
    uint8_t roiIndex;   // Only zoom focus Camx version support the "roiIndex"
    // TODO - after v1.2.0
    uint8_t *OutAlgoVersion;  // OutAlgoVersion[AdapsAlgoVersionLength];
    uint8_t zone_cnt;
    uint8_t peak_index;
    uint8_t* spot_cali_data;//add 2023-11-7
} SetWrapperParam;
```

该结构体定义了运算函数所需的参数设置

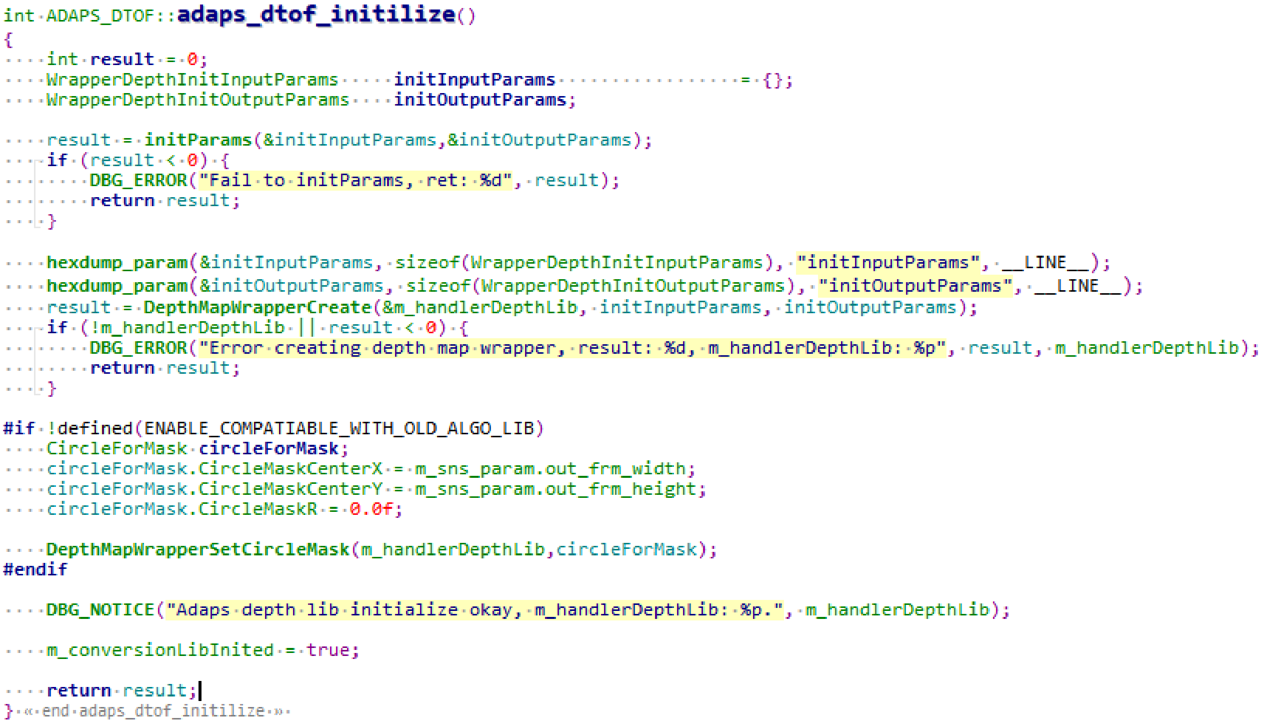
```
typedef struct {
    const char*    configFilePath;
    int32_t       width;
    int32_t       height;
    int32_t       dm_width;
    int32_t       dm_height;
    uint8_t*      pRawData;
    uint32_t      rawDataSize;
    RotateConfig   rotateConfig;
    uint32_t      outputPlaneCount;
    uint32_t      outputPlaneFormat[WRAPPER_CAM_FORMAT_MAX];
    SetWrapperParam setparam;
} WrapperDepthInitInputParams;
```

```
typedef struct {
    uint64_t* exposure_time;
    int32_t*  sensitivity;
} WrapperDepthInitOutputParams;
```

算法库的工作流程


初始化算法库
DepthMapWrapperCreate()
mipi原始数据
ret = DepthMapWrapperProcessFrame()
no
ret = true
yes
获取深度数据
销毁算法库
DepthMapWrapperDestroy()

接口实际使用示范代码



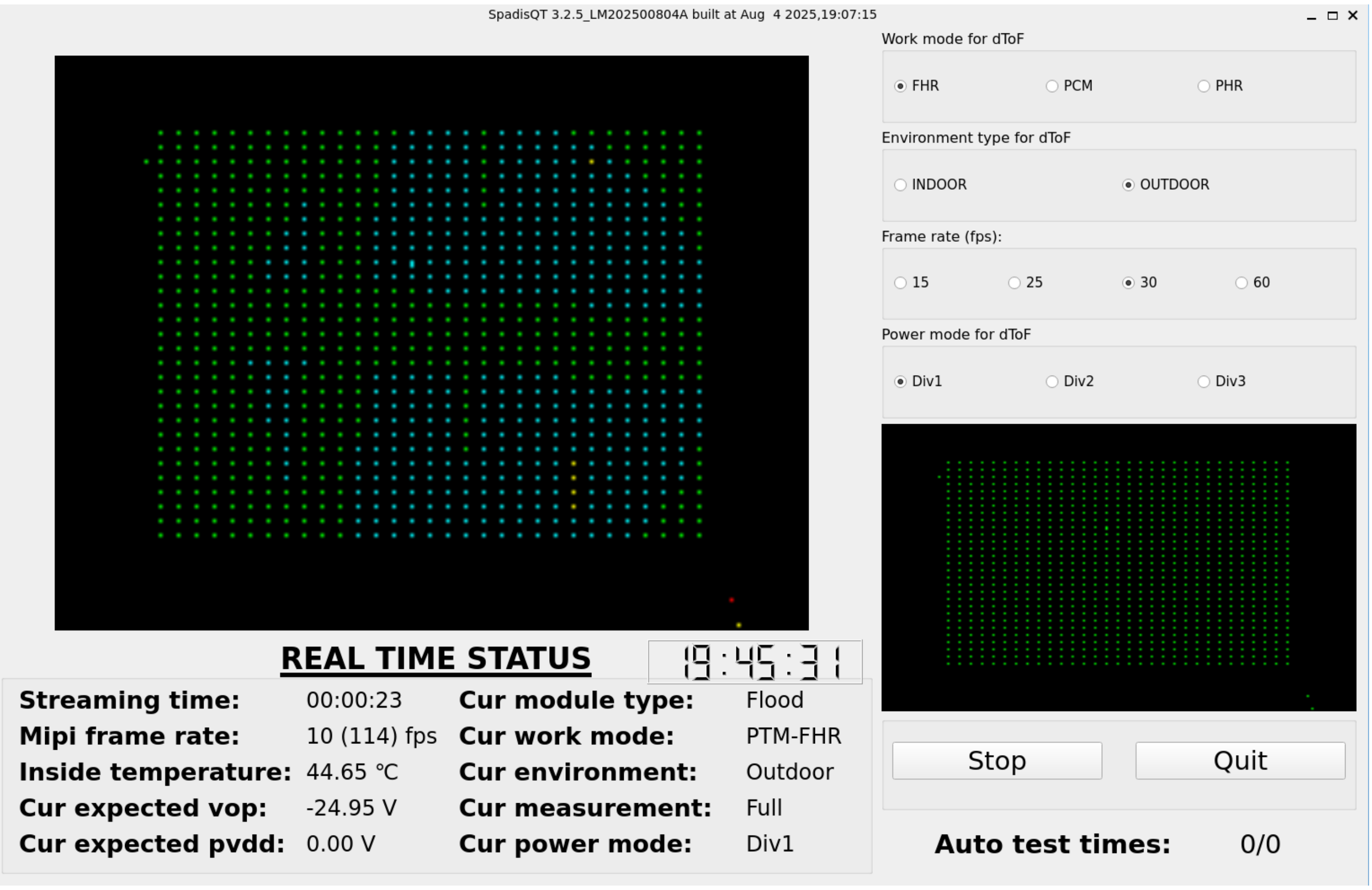
```
int ADAPS_DTOF::adaps_dtof_initilize()
{
    int result = 0;
    WrapperDepthInitInputParams     initInputParams                 = {};
    WrapperDepthInitOutputParams    initOutputParams;

    result = initParams(&initInputParams,&initOutputParams);
    if (result < 0) {
        DBG_ERROR("Fail to initParams, ret: %d", result);
        return result;
    }

    hexdump_param(&initInputParams, sizeof(WrapperDepthInitInputParams), "initInputParams", __LINE__);
    hexdump_param(&initOutputParams, sizeof(WrapperDepthInitOutputParams), "initOutputParams", __LINE__);
    result = DepthMapWrapperCreate(&m_handlerDepthLib, initInputParams, initOutputParams);
    if (!m_handlerDepthLib || result < 0) {
        DBG_ERROR("Error creating depth map wrapper, result: %d, m_handlerDepthLib: %p", result, m_handlerDepthLib);
        return result;
    }

#if !defined(ENABLE_COMPATIABLE_WITH_OLD_ALGO_LIB)
    CircleForMask circleForMask;
    circleForMask.CircleMaskCenterX = m_sns_param.out_frm_width;
    circleForMask.CircleMaskCenterY = m_sns_param.out_frm_height;
    circleForMask.CircleMaskR = 0.0f;

    DepthMapWrapperSetCircleMask(m_handlerDepthLib,circleForMask);
#endif

    DBG_NOTICE("Adaps depth lib initialize okay, m_handlerDepthLib: %p.", m_handlerDepthLib);

    m_conversionLibInited = true;

    return result;
} « end adaps_dtof_initilize »
```

4. Linux 应用开源项目 SpadisQT

SpadisQT 是一款针对 ADAPS Photonics 公司 ADS6401 dToF（直接飞行时间）传感器的演示应用，旨在嵌入式 Linux 系统上运行。该应用通过 V4L2 框架采集传感器的原始 MIPI 数据，经专有算法库处理后转换为深度或灰度数据，并以 RGB 色彩可视化深度信息，方便用户直观解读。

- **适用传感器**：ADAPS ADS6401 dToF 传感器，支持两种模组类型（SPOT 散点模组、FLOOD 面阵模组）。
- **测试平台**：已在 RK3568 开发板（Linux 5.10 内核）上验证。
- **开发框架**：基于 QT 5.x 构建，依赖 V4L2 接口（因此仅支持 Linux 系统，不支持 Windows）。

下载链接:
https://github.com/David1934/SpadisQT
项目介绍 wiki:
https://github.com/David1934/SpadisQT/wiki



SpadisQT 3.2.5_LM202500804A built at Aug 4 2025,19:07:15

Work mode for dToF

● FHR ○ PCM ○ PHR

Environment type for dToF

○ INDOOR ● OUTDOOR

Frame rate (fps):

○ 15 ○ 25 ● 30 ○ 60

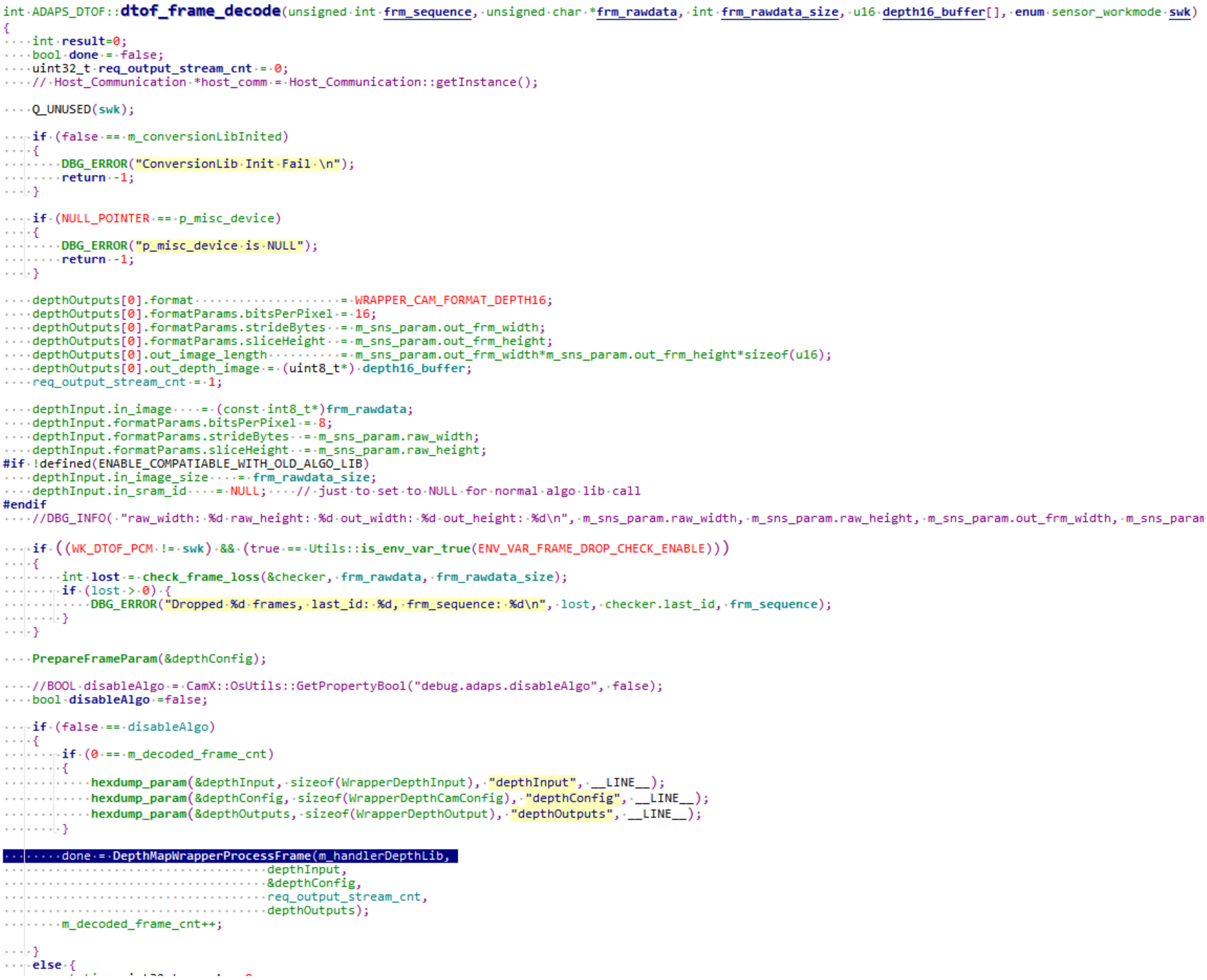
Power mode for dToF

● Div1 ○ Div2 ○ Div3

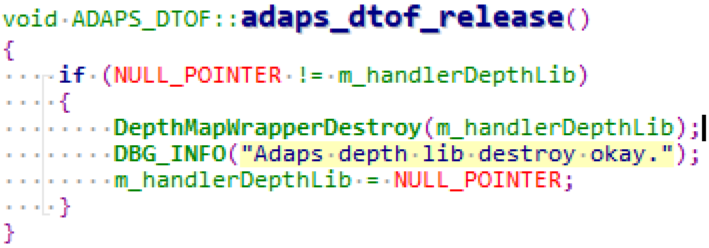
REAL TIME STATUS 19:45:31

Streaming time:	00:00:23	**Cur module type:**	Flood
Mipi frame rate:	10 (114) fps	**Cur work mode:**	PTM-FHR
Inside temperature:	44.65 ℃	**Cur environment:**	Outdoor
Cur expected vop:	-24.95 V	**Cur measurement:**	Full
Cur expected pvdd:	0.00 V	**Cur power mode:**	Div1

Auto test times: 0/0



```
int ADAPS_DTOF::dtof_frame_decode(unsigned int frm_sequence, unsigned char *frm_rawdata, int frm_rawdata_size, u16 depth16_buffer[], enum sensor_workmode swk)
{
    int result=0;
    bool done = false;
    uint32_t req_output_stream_cnt = 0;
    // Host_Communication *host_comm = Host_Communication::getInstance();

    Q_UNUSED(swk);

    if (false == m_conversionLibInited)
    {
        DBG_ERROR("ConversionLib Init Fail \n");
        return -1;
    }

    if (NULL_POINTER == p_misc_device)
    {
        DBG_ERROR("p_misc_device is NULL");
        return -1;
    }

    depthOutputs[0].format                     = WRAPPER_CAM_FORMAT_DEPTH16;
    depthOutputs[0].formatParams.bitsPerPixel = 16;
    depthOutputs[0].formatParams.strideBytes  = m_sns_param.out_frm_width;
    depthOutputs[0].formatParams.sliceHeight  = m_sns_param.out_frm_height;
    depthOutputs[0].out_image_length          = m_sns_param.out_frm_width*m_sns_param.out_frm_height*sizeof(u16);
    depthOutputs[0].out_depth_image = (uint8_t*) depth16_buffer;
    req_output_stream_cnt = 1;

    depthInput.in_image    = (const int8_t*)frm_rawdata;
    depthInput.formatParams.bitsPerPixel = 8;
    depthInput.formatParams.strideBytes  = m_sns_param.raw_width;
    depthInput.formatParams.sliceHeight  = m_sns_param.raw_height;
#if !defined(ENABLE_COMPATIABLE_WITH_OLD_ALGO_LIB)
    depthInput.in_image_size    = frm_rawdata_size;
    depthInput.in_sram_id    = NULL;    // just to set to NULL for normal algo lib call
#endif
    //DBG_INFO( "raw_width: %d raw_height: %d out_width: %d out_height: %d\n", m_sns_param.raw_width, m_sns_param.raw_height, m_sns_param.out_frm_width, m_sns_param

    if ((WK_DTOF_PCM != swk) && (true == Utils::is_env_var_true(ENV_VAR_FRAME_DROP_CHECK_ENABLE)))
    {
        int lost = check_frame_loss(&checker, frm_rawdata, frm_rawdata_size);
        if (lost > 0) {
            DBG_ERROR("Dropped %d frames, last_id: %d, frm_sequence: %d\n", lost, checker.last_id, frm_sequence);
        }
    }

    PrepareFrameParam(&depthConfig);

    //BOOL disableAlgo = CamX::OsUtils::GetPropertyBool("debug.adaps.disableAlgo", false);
    bool disableAlgo =false;

    if (false == disableAlgo)
    {
        if (0 == m_decoded_frame_cnt)
        {
            hexdump_param(&depthInput, sizeof(WrapperDepthInput), "depthInput", __LINE__);
            hexdump_param(&depthConfig, sizeof(WrapperDepthCamConfig), "depthConfig", __LINE__);
            hexdump_param(&depthOutputs, sizeof(WrapperDepthOutput), "depthOutputs", __LINE__);
        }

        done = DepthMapWrapperProcessFrame(m_handlerDepthLib,
                                           depthInput,
                                           &depthConfig,
                                           req_output_stream_cnt,
                                           depthOutputs);
        m_decoded_frame_cnt++;

    }
    else {
```



```
void ADAPS_DTOF::adaps_dtof_release()
{
    if (NULL_POINTER != m_handlerDepthLib)
    {
        DepthMapWrapperDestroy(m_handlerDepthLib);
        DBG_INFO("Adaps depth lib destroy okay.");
        m_handlerDepthLib = NULL_POINTER;
    }
}
```