Camera architecture & driver in MTK

Ov5642 used in MTK6235

林耕书

booklinbook@hotmail.com

Agenda

- 概述
- IRQ，FIQ介绍
- LCD介绍
- Camera 在mtk中的整体架构和驱动介绍
- 下期预告

概述

- 这个ppt主要把mtk中camera部分向大家做个尽量系统的汇报。ppt以camera的工作流程为线索，以及数据流为副线索，从mmi层开始经media中间层一直讲到camera sensor driver层，希望能给大家一个camera模块总体的介绍，为以后的开发提供一个资料参考。

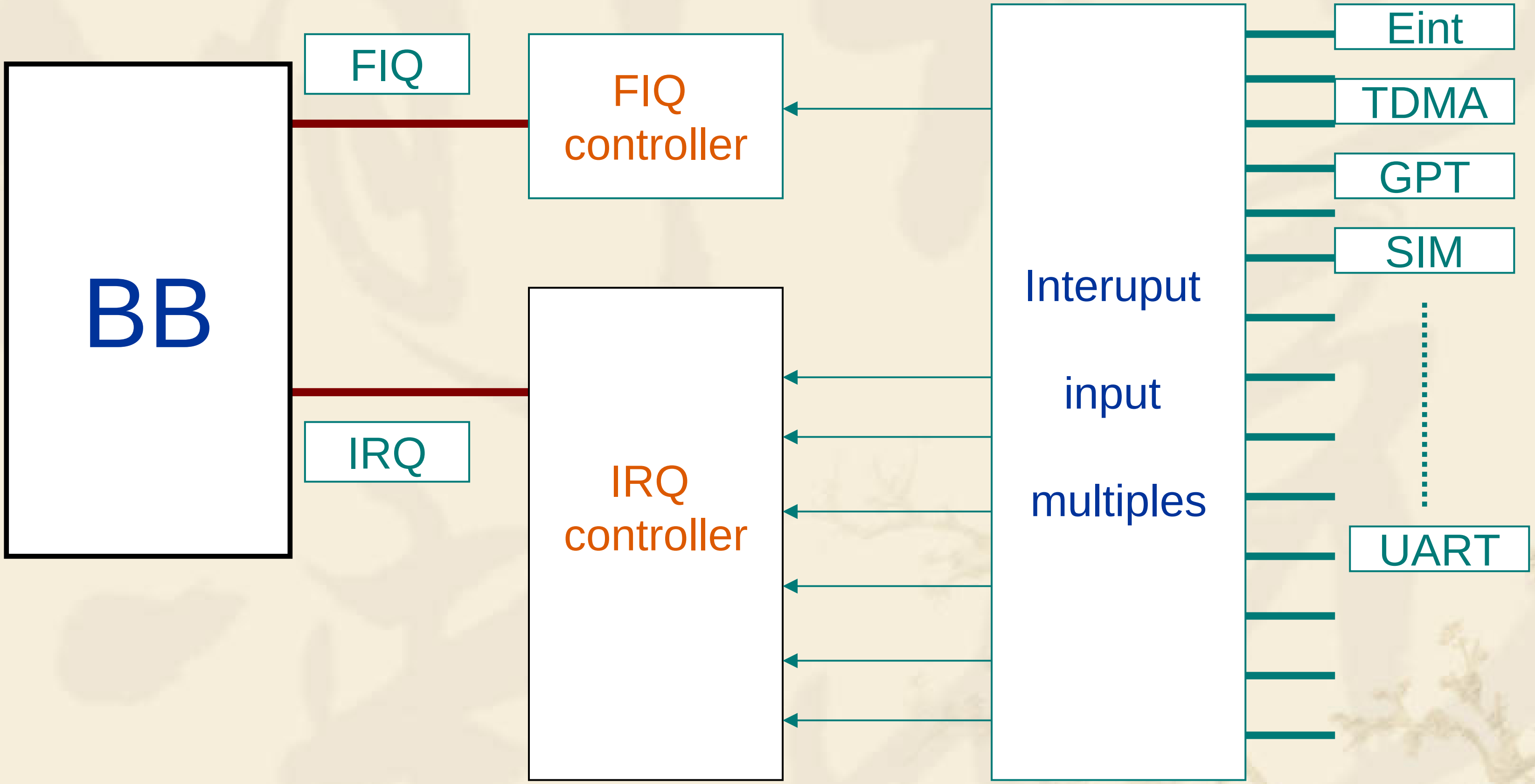
由于有的时候我的思路和mtk的烂代码一样会错乱，所以有些地方我们就图个不求甚解，我尽量把它说的清楚一些， 时间和水平有限，如果在ppt中有疏误的地方，请大家能指出一起讨论，谢谢

IRQ & FIQ

- 驱动的开发是以系统为基础的，功能实现过程中离不开系统服务和调用。在camera中也是这样，比如中断的使用就涉及到的很多，sensor什么时候传来一帧数据，什么时候传输完成，类似与这样的异同步的流程都会涉及到中断，而且也相对有些复杂，所以我单独把mtk中断部分先做个简要的介绍，为后续的讲解提供一些方便。

con

❖ 什么是中断，什么是IRQ，FIQ？（我来口述）



(1)FIQ 优先与IRQ。

(2)系统中只用一个FIQ和一个IRQ pin，那么多人用，就要加中断控制模块来仲裁和控制。

con

- 在程序里FIQ 和IRQ怎样体现，现在来看code。

1.中断向量表（..\init\src\bootarm.s）

```
    EXPORT    INT_Table
INT_Table

INT_Initialize_Addr   DCD    INT_Initialize
Undef_Instr_Addr      DCD    Undef_Instr_ISR
SWI_Addr              DCD    SWI_ISR
Prefetch_Abort_Addr   DCD    Prefetch_Abort_ISR
Data_Abort_Addr       DCD    Data_Abort_ISR
Undefined_Addr        DCD    0                  ; NO LONGER USED
IRQ_Handler_Addr      DCD    INT_IRQ_Parse
FIQ_Handler_Addr      DCD    INT_FIQ_Parse

INT_Table_END
```

INT_IRQ_Parse就是irq中断处理函数在汇编中，再往下看

con

- 在INT_IRQ_Parse中有一句BL isrC_Main 就跳到c 代码部分进行进一步的中断处理

```
    B         _tx_thread_context_save
__tx_irq_processing_return
    BL        _tx_thread_irq_nesting_start
    BL        isrC_Main
    BL        _tx_thread_irq_nesting_end
    B         _tx_thread_context_restore

    ENDIF
```

- isrC_Main()中

```
  ReEnableIRQ();
  lisr_dispatch_tbl[irqx].lisr_handler();
  DisableIRQ();
```

Irqx就是不同的**irq**的中断号，这句话就调用相应的**irq**的处理函数了

lisr_dispatch_tbl在IRQ_Register_LISR这个中断注册函数中被赋值，你看到了吗？

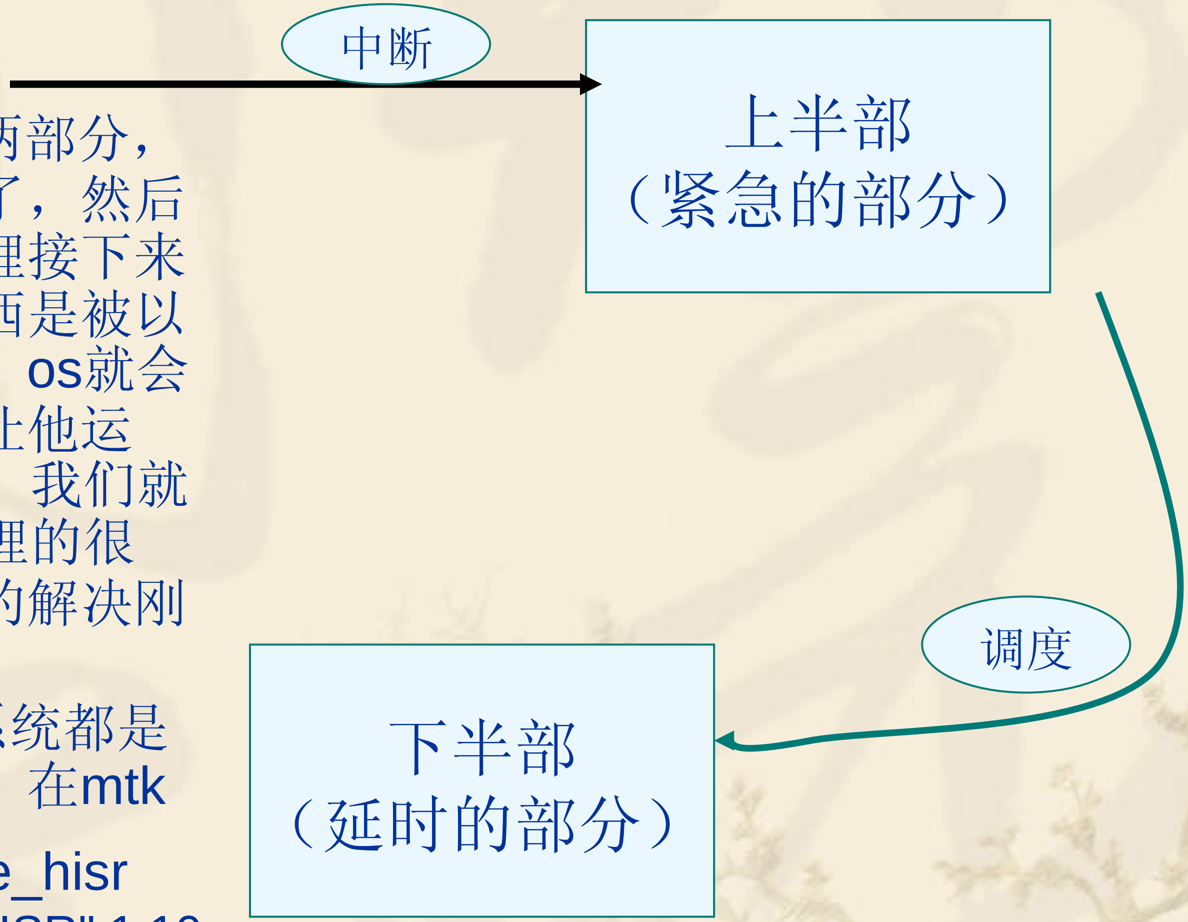
```
void IRQ_Register_LISR(kal_uint32 code, void (*reg_lisr)(void), char* description)
{
   kal_uint32 savedMask;
   kal_uint32 irqvector;

   savedMask = LockIRQ();
   irqvector = IRQCode2Line[code];
   lisr_dispatch_tbl[irqvector].vector = irqvector;
   lisr_dispatch_tbl[irqvector].lisr_handler = reg_lisr;
   lisr_dispatch_tbl[irqvector].description = description;
   RestoreIRQ(savedMask);
}
```

讲到这里好像已经把中断讲完了，硬件产生电位变化引发中断，经过中断控制器的仲裁，给bb发最终的irq或fiq中断，bb收到中断，中断当前手头上的事，跳到中断向量表处指定的处理函数，然后执行相应的irq之类的函数。一切看起来都很完整了，很完美了。好像是喔!

但是我们可能忽略了一个问题，就是中断不能太久，要很快就要回来，要短要快，但是如果我们中断中要处理的东西的确很多，就好比sensor传来一帧数据，我们要处理，要解码，要保存，要...,的确很多事情快不了怎么办？下面就讨论这个问题，完善中断处理机制。

❖ 中断处理函数架构



把中断处理分为两部分，先把紧急的部分处理了，然后通过调度的方式来处理接下来的事情。一旦一个东西是被以调度的方式远行的话，os就会根据相应的调度方式让他运行，这个是os的活儿，我们就不用担心了，os会处理的很好。这样就可以很好的解决刚才提出的问题。

其实几乎所有的系统都是这样解决这个问题的，在mtk

里我们用visual_active_hisr kal_create_hisr("VISUHISR",1,1024,VISUAL_HISR,NULL);这样的函数来实现，我们不妨叫他虚中断。下面分析一下。

先看虚中断相关的函数，我一个个解释说明

```
void visual_hisr_init(void)
{
   kal_uint32 index;
   for(index=0;index<MAX_VISUAL_DEVICE;index++)
   {
      VISUAL_HISR_TABLE[index].hisr_func = VISUAL_ERROR_HISR;
      VISUAL_HISR_TABLE[index].hisr_count = 0;
   }

   if (visual_hisr == NULL)
      visual_hisr = kal_create_hisr("VISUHISR",1,1024,VISUAL_HISR,NULL);
}
```

```
void VISUAL_Register_HISR(kal_uint8 visula_hisr_id, VOID_FUNCTION hisr_func)
{
   VISUAL_HISR_TABLE[visula_hisr_id].hisr_func = hisr_func;
}
```

```
#define visual_active_hisr(_id)   \
{\
   kal_uint32 _savedMask;\
   _savedMask = SaveAndSetIRQMask();\
   visual_hisr_status |= (1<<_id);\
   VISUAL_HISR_TABLE[_id].hisr_count++;\
   RestoreIRQMask(_savedMask);\
   kal_activate_hisr(visual_hisr);\
}
```

```
visual_active_hisr(VISUAL_CAMERA_HISR_ID);
```

```
visual_active_hisr(VISUAL_G2D_HISR_ID);
```

- 通过以上分析，不难看出，虚中断就是通过利用kal_create_hisr，kal_activate_hisr这样kal层提供的有一点点像信号机制一样的功能函数来实现中断下半部的调度。（kal还有些功能函数后面会介绍）
- 这样说可能还是有点抽象，没关系，下面我举个小小的例子把一个中断的上下部串起来讲一遍大家就清楚了。
- 看这个，我来讲解

```
void init_isp_if(void)
{
    ::
    ::
    ::
    IRQ_Register_LISR(IRQ_CAMERA_CODE, camera_isp_LISR,"Camera ISR");
    VISUAL_Register_HISR(VISUAL_CAMERA_HISR_ID,camera_isp_HISR);
    ::
    ::
    init_yuv_isp();/* include yuv init */
}
```

这个是isp模块初始化时，其中就注册了一个中断上部份camera_isp_LISR

和注册了一个中断下半部camera_isp_HISR

isp(image signal processor)是图像处理模块,sensor传来的照片yuv格式的数据就是由isp处理的（这个后面详细讲），isp处理中就会用到中断。首先isp的硬件产生中断， camera_isp_LISR（）就被调用了，

```
void camera_isp_LISR(void)
{
    kal_uint32 int_status;
    kal_uint32 save_irq_mask;

    save_irq_mask=SaveAndSetIRQMask();
    IRQMask(IRQ_CAMERA_CODE);
    int_status=DRV_Reg32(ISP_INT_STATUS_REG);
    isp_int_status=0;
    RestoreIRQMask(save_irq_mask);
    isp_frame_count++;
    if (int_status & REG_CAMERA_INT_FRAME_READY_ENABLE_BIT)
    {
        isp_int_status |= ISP_INT_FRAME_READY;
    }
    :::
    :::
    visual_active_hisr(VISUAL_CAMERA_HISR_ID);
    :::
}
```

camera_isp_LISR中先把重要的一些事给做了，最后通过visual_active_hisr(VISUAL_CAMERA_HISR_ID);来激发调度isp中断的下半部camera_isp_HISR（这个函数里做的事情可多了）

再介绍一下后面会用到的几个异同步函数

kal_create_event_group(…), kal_set_eg_events(…)，kal_retrieve_eg_events(…), 举个例子给大家说明一下，什么原理，怎么用（口述）。

```
camera_isp_event_id=kal_create_event_group("ISP_EVT");
```

```
    kal_set_eg_events(camera_isp_event_id,0,KAL_AND);
    ENABLE_VIEW_FINDER_MODE;
    kal_retrieve_eg_events(camera_isp_event_id,CAMERA_ISP_VD_READY_EVENT,KAL_OR_CONSUME,
                          &event_group,KAL_SUSPEND);
  /****************************************/
```

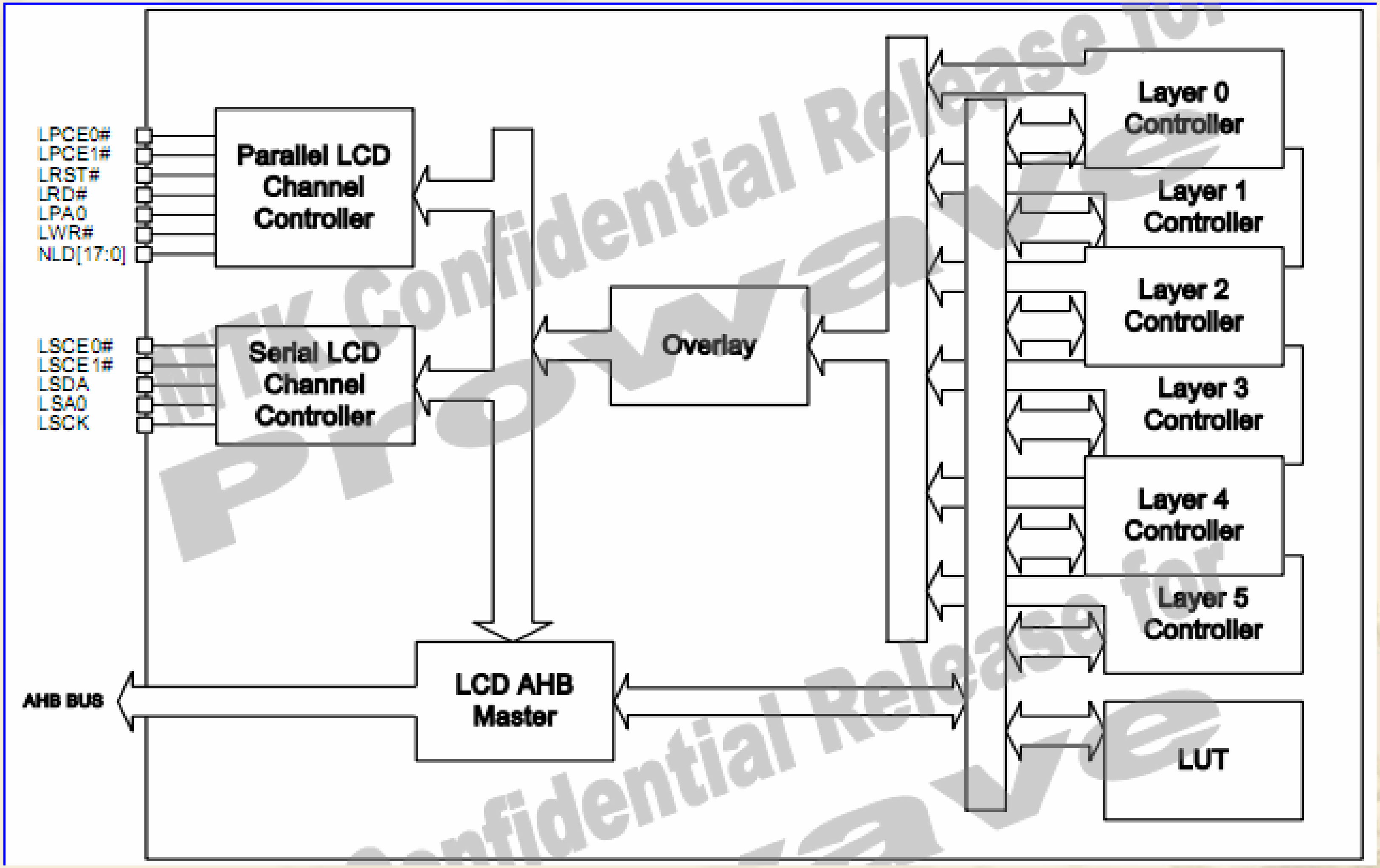
```
kal_set_eg_events(camera_isp_event_id, CAMERA_ISP_VD_READY_EVENT, KAL_OR);
```

```
#define CAMERA_ISP_IDLE_EVENT           0x00000001
#define CAMERA_ISP_FRAME_READY_EVENT    0x00000002
#define CAMERA_JPEG_ENCODE_EVENT            0x00000004
#define CAMERA_ISP_VD_READY_EVENT       0x00000008
```

Lcd相关部分的简介

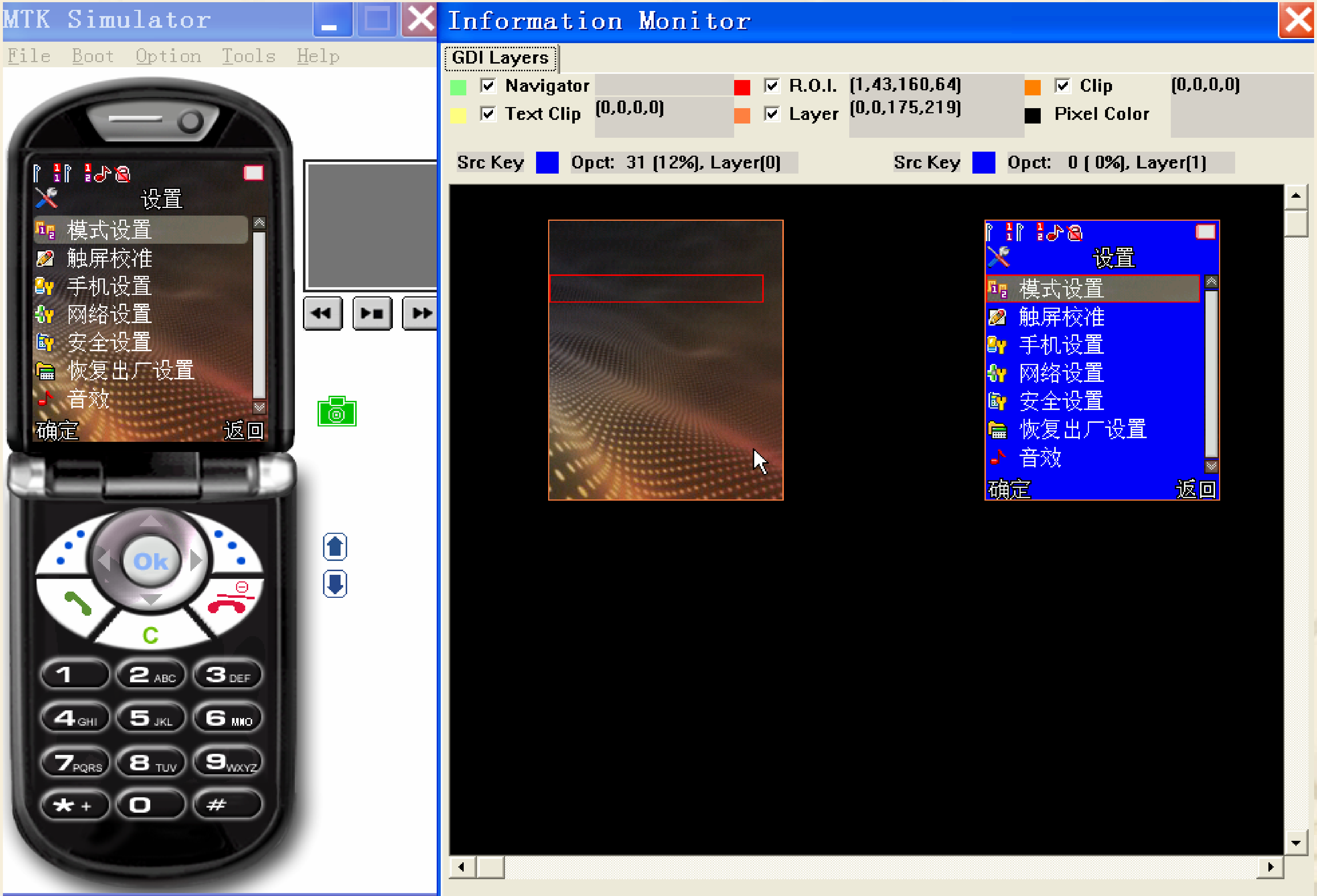
- 这里只介绍两个关于lcd的问题，一个是关于layer的问题，另一个是快速定时自动刷新的问题。

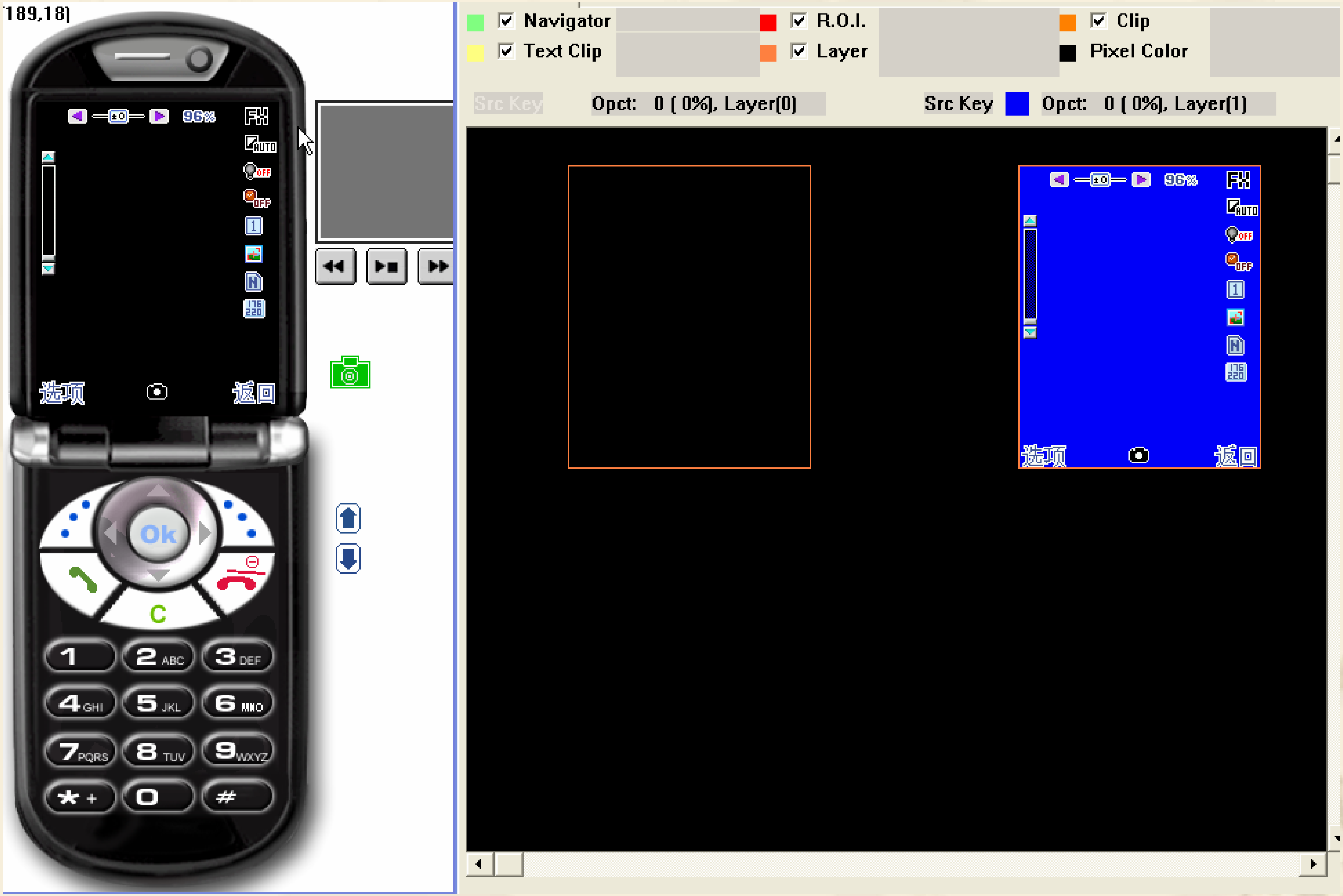
❖ 先看一幅截图，我来说明


LPCE0#
LPCE1#
LRST#
LRD#
LPA0
LWR#
NLD[17:0]
Parallel LCD Channel Controller
LSCE0#
LSCE1#
LSDA
LSA0
LSCK
Serial LCD Channel Controller
Overlay
Layer 0 Controller
Layer 1 Controller
Layer 2 Controller
Layer 3 Controller
Layer 4 Controller
Layer 5 Controller
LUT
AHB BUS
LCD AHB Master
MTK Confidential Release for Prowave

LCD + 0070h	LCD Layer 0 Window Control Register	32	LCD_L0WINCON
LCD + 0074h	LCD Layer 0 Source Color Key Register	32	LCD_L0WINSKEY
LCD + 0078h	LCD Layer 0 Window Display Offset Register	32	LCD_L0WINOFS
LCD + 007ch	LCD Layer 0 Window Display Start Address Register	32	LCD_L0WINADD
LCD + 0080h	LCD Layer 0 Window Size	32	LCD_L0WINSIZE
LCD + 0090h	LCD Layer 1 Window Control Register	32	LCD_L1WINCON
LCD + 0094h	LCD Layer 1 Source Color Key Register	32	LCD_L1WINSKEY
LCD + 0098h	LCD Layer 1 Window Display Offset Register	32	LCD_L1WINOFS
LCD + 009ch	LCD Layer 1 Window Display Start Address Register	32	LCD_L1WINADD
LCD + 00a0h	LCD Layer 1 Window Size	32	LCD_L1WINSIZE
LCD + 00b0h	LCD Layer 2 Window Control Register	32	LCD_L2WINCON
LCD + 00b4h	LCD Layer 2 Source Color Key Register	32	LCD_L2WINSKEY
LCD + 00b8h	LCD Layer 2 Window Display Offset Register	32	LCD_L2WINOFS
LCD + 00bch	LCD Layer 2 Window Display Start Address Register	32	LCD_L2WINADD
LCD + 00c0h	LCD Layer 2 Window Size	32	LCD_L2WINSIZE
LCD + 00d0h	LCD Layer 3 Window Control Register	32	LCD_L3WINCON
LCD + 00d4h	LCD Layer 3 Source Color Key Register	32	LCD_L3WINSKEY
LCD + 00d8h	LCD Layer 3 Window Display Offset Register	32	LCD_L3WINOFS
LCD + 00dch	LCD Layer 3 Window Display Start Address Register	32	LCD_L3WINADD
LCD + 00e0h	LCD Layer 3 Window Size	32	LCD_L3WINSIZE
LCD + 00f0h	LCD Layer 4 Window Control Register	32	LCD_L4WINCON
LCD + 00f4h	LCD Layer 4 Source Color Key Register	32	LCD_L4WINSKEY
LCD + 00f8h	LCD Layer 4 Window Display Offset Register	32	LCD_L4WINOFS
LCD + 00fch	LCD Layer 4 Window Display Start Address Register	32	LCD_L4WINADD
LCD + 0100h	LCD Layer 4 Window Size	32	LCD_L4WINSIZE
LCD + 0110h	LCD Layer 5 Window Control Register	32	LCD_L5WINCON
LCD + 0114h	LCD Layer 5 Source Color Key Register	32	LCD_L5WINSKEY
LCD + 0118h	LCD Layer 5 Window Display Offset Register	32	LCD_L5WINOFS
LCD + 011ch	LCD Layer 5 Window Display Start Address Register	32	LCD_L5WINADD
LCD + 0120h	LCD Layer 5 Window Size	32	LCD_L5WINSIZE

说白了layer就是一个memory buffer，要画在屏幕上的数据就写在这个buffer相应的位置上，最后buffer的数据就传给lcd，图像就被显示出来。多个layer无非就是多开了几个memory buffer，在不同buffer上画画，最后在把他们几个buffer进行叠加，有透明值部分的点就被过滤掉，叠加后再把他传到lcd中显示出来，这样做好处就是提高了显示能力和效率，像camera部分，preview的图像就被画在一个层上面，而那些图标（即osd）画在另一个层上，叠加后传到lcd中刷新显示。设想一下如果只有一个layer（一个buffer）的话，像这样的osd之类的功能就很难实现了。


MTK Simulator
File Boot Option Tools Help
设置
模式设置
触屏校准
手机设置
网络设置
安全设置
恢复出厂设置
音效
确定
返回
Information Monitor
GDI Layers
Navigator
R.O.I. [1,43,160,64]
Clip [0,0,0,0]
Text Clip [0,0,0,0]
Layer [0,0,175,219]
Pixel Color
Src Key
Opct: 31 [12%], Layer[0]
Src Key
Opct: 0 [0%], Layer[1]


189,18]
Navigator
R.O.I.
Clip
Text Clip
Layer
Pixel Color
Src Key
Opct: 0 (0%), Layer(0)
Src Key
Opct: 0 (0%), Layer(1)
96%
选项
返回
Ok
C
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* +
0
#

❖ 具体看一下代码中的几个部分，我来讲解

```
static void _lcd_fb_update_fw(kal_uint8 lcd_layer, kal_uint32 addr)
{
    ::
    ::
    ::
   LCD_config_fw_layer_address(addr);
   if ((LCD_FW_CMD_QUEUE_STATE == main_lcd_operation_state)
      && (KAL_FALSE == lcd_hw_trigger_flag) )
   {
    ::
    ::
         lcd_fb_update_internal((lcd_frame_update_struct *) &main_lcd_fb_update_para);
    ::
    ::

   }
   else
   {
    ::
    ::
         START_LCD_TRANSFER;
    ::
    ::
   }

}
```

这个是lcd刷新显示的底层的一个函数了，其两个参数：lcd_layer 是表示第几层如LCD_LAYER0，LCD_LAYER1. addr 就是具体的buffer地址，buffer可以是系统默认的，也可以是动态开的（mmi的开发过程中就会常常碰到这个东西）

接下来看LCD_config_fw_layer_address(addr);在这个函数里buffer的地址就被写入相应的layer地址寄存器中去了如LCD_LAYER0_SIZE_REG，(刚才说过)最后开始传输START_LCD_TRANSFER

```
void LCD_config_fw_layer_address(kal_uint32 addr)
{
   #if defined(DRV_LCD_FW_UPDATE_SUPPORT)
      if(lcd_debug==KAL_TRUE) {
         kal_uint32 lcd_layer_base_addr=LCD_LAYER0_BASE_ADDR;

         lcd_layer_base_addr+=lcd_fw_update_layer*LCD_LAYER_ADDR_OFFSET;
         #if !defined(DRV_LAYER_COLOR_FORMAT)
            DRV_WriteReg32((lcd_layer_base_addr+0x08),addr);
         #else
            DRV_WriteReg32((lcd_layer_base_addr+0x0c),addr);
         #endif
      }
   #endif
}
```

赋值

```
#define LCD_LAYER0_BASE_ADDR          (LCD_base+0x0070)
#define LCD_LAYER0_CTRL_REG           (LCD_base+0x0070)
#define LCD_LAYER0_SRC_KEY_REG        (LCD_base+0x0074)
#define LCD_LAYER0_OFFSET_REG         (LCD_base+0x0078)
#define LCD_LAYER0_BUFF_ADDR_REG      (LCD_base+0x007C)
#define LCD_LAYER0_SIZE_REG           (LCD_base+0x0080)
#define LCD_LAYER1_CTRL_REG           (LCD_base+0x0090)
#define LCD_LAYER1_SRC_KEY_REG        (LCD_base+0x0094)
#define LCD_LAYER1_OFFSET_REG         (LCD_base+0x0098)
#define LCD_LAYER1_BUFF_ADDR_REG      (LCD_base+0x009C)
#define LCD_LAYER1_SIZE_REG           (LCD_base+0x00A0)
#define LCD_LAYER2_CTRL_REG           (LCD_base+0x00B0)
#define LCD_LAYER2_SRC_KEY_REG        (LCD_base+0x00B4)
#define LCD_LAYER2_OFFSET_REG         (LCD_base+0x00B8)
#define LCD_LAYER2_BUFF_ADDR_REG      (LCD_base+0x00BC)
#define LCD_LAYER2_SIZE_REG           (LCD_base+0x00C0)
#define LCD_LAYER3_CTRL_REG           (LCD_base+0x00D0)
#define LCD_LAYER3_SRC_KEY_REG        (LCD_base+0x00D4)
#define LCD_LAYER3_OFFSET_REG         (LCD_base+0x00D8)
#define LCD_LAYER3_BUFF_ADDR_REG      (LCD_base+0x00DC)
#define LCD_LAYER3_SIZE_REG           (LCD_base+0x00E0)
#define LCD_LAYER4_CTRL_REG           (LCD_base+0x00F0)
```

相应寄存器地址

Lcd快速定时自动刷新问题

- 在某些情况下（如camera preview时）对lcd的刷新速度是要求很高的。因此mtk的为lcd度身定制了一个自动刷新，原理很简单，大家只要知道有这回事情就可以了，不用深究，这里只做一个简单的介绍。先看一下代码的几个片段，我来讲述说明。

```
typedef enum
{
   LCD_SW_TRIGGER_MODE       =   0,      /* LCD SW trigger with frame buffer */
   LCD_HW_TRIGGER_MODE       =   1,      /* LCD HW trigger with frame buffer */
   LCD_DIRECT_COUPLE_MODE    =   2       /* LCD HW trigger without frame buffer, direct couple */
}  LCD_UPDATE_MODE_ENUM;
```

```
typedef struct
{
    /// module ID that request frame buffer update
    /// LCD_UPDATE_MODULE_MMI, LCD_UPDATE_MODULE_MEDIA, LCD_UPDAET_MODULE_EXT_CAMERA,
    kal_uint8 module_id;
    ::
    kal_uint8 fb_update_mode;
    ::
    kal_uint32 update_layer;
    /// which layer will be applied by hw trigger or direct couple
    kal_uint32 hw_update_layer;
    /// rotate select for hardware update layer
    kal_uint8 rotate_value;
    ::

}  lcd_frame_update_struct;
```

```c
static void lcd_fb_update_17_23_25_series(lcd_frame_update_struct *lcd_para)
{       ::
        switch(lcd_para->hw_update_layer)
        {
            case LCD_LAYER0_ENABLE: lcd_fw_update_layer=LCD_LAYER0; break;
            case LCD_LAYER1_ENABLE: lcd_fw_update_layer=LCD_LAYER1; break;
            case LCD_LAYER2_ENABLE: lcd_fw_update_layer=LCD_LAYER2; break;
            case LCD_LAYER3_ENABLE: lcd_fw_update_layer=LCD_LAYER3; break;
        }
        ::
        lcd_power_ctrl(KAL_TRUE);
        ::
        SET_LCD_ROI_WINDOW_OFFSET(lcd_para->roi_offset_x, lcd_para->roi_offset_y);
        SET_LCD_ROI_WINDOW_SIZE(lcd_para->lcm_end_x-lcd_para->lcm_start_x+1,
                                lcd_para->lcm_end_y-lcd_para->lcm_start_y+1);
        SET_LCD_ENABLE_LAYER(lcd_para->update_layer);
        ::
        SET_LCD_ROI_CTRL_OUTPUT_FORMAT(MAIN_LCD_OUTPUT_FORMAT);
        DRV_WriteReg32(LCD_ROI_CMD_ADDR_REG,MAIN_LCD_CMD_ADDR);
        DRV_WriteReg32(LCD_ROI_DATA_ADDR_REG,MAIN_LCD_DATA_ADDR);
           // to protect when LCD and MSDC share pins
        LSD_TakeControl(LSD_LCDARB_LCD);
           MainLCD->BlockWrite(lcd_para->lcm_start_x,lcd_para->lcm_start_y,
                                  lcd_para->lcm_end_x,lcd_para->lcm_end_y);
        ::
        #if defined(DRV_LCD_FW_UPDATE_SUPPORT)/*support LCD_FW_UPDATE_STATE*/
        if((LCD_HW_TRIGGER_MODE == lcd_para->fb_update_mode) ||
        (LCD_DIRECT_COUPLE_MODE == lcd_para->fb_update_mode))
        {
        lcd_save_hw_trigger_roi_setting(lcd_para->hw_update_layer);
        }
        ::
        lcd_power_ctrl(KAL_FALSE);
}
```

```c
void lcd_save_hw_trigger_roi_setting(kal_uint32 hw_layer)
{
#if defined(DRV_LCD_FW_UPDATE_SUPPORT)
   kal_uint32 i=0;
   lcd_hw_trigger_layer=hw_layer;
   lcd_hw_trigger_roi_offset=DRV_Reg32(LCD_ROI_OFFSET_REG);
   lcd_hw_trigger_roi_size=DRV_Reg32(LCD_ROI_SIZE_REG);
   lcd_hw_trigger_roi_ctrl=DRV_Reg32(LCD_ROI_CTRL_REG);
   lcd_hw_trigger_flag=KAL_FALSE;
   for (i=0;i<LCD_CMD_QUEUE_LENGTH;i++)
   {
      lcd_hw_trigger_para[i]=DRV_Reg32(LCD_CMD_PARAMETER_ADDR+(i<<2));
   }
#endif
}
```

一般用到定时自动刷新的，都是重复的刷新，比如camera poreview，所以刷新显示的区域都是相对固定的，只是刷新的内容在变，所以在上面的这个函数里，它要刷新的参数都保存在了lcd_hw_trigger_para[]数组中，它将在lcd_HISR()中被用到。看下图

```c
void lcd_HISR(void)
{
    ::
    ::
    lcd_fw_update_done_hdlr();
    ::
    DRV_WriteReg32(LCD_ROI_CTRL_REG,lcd_hw_trigger_roi_ctrl);
    DRV_WriteReg32(LCD_ROI_OFFSET_REG,lcd_hw_trigger_roi_offset);
    DRV_WriteReg32(LCD_ROI_SIZE_REG,lcd_hw_trigger_roi_size);
    ::
    for (i=0;i<LCD_CMD_QUEUE_LENGTH;i++)
        DRV_WriteReg32(LCD_CMD_PARAMETER_ADDR+(i<<2),lcd_hw_trigger_para[i]);
    ::

    START_LCD_TRANSFER;

}
```

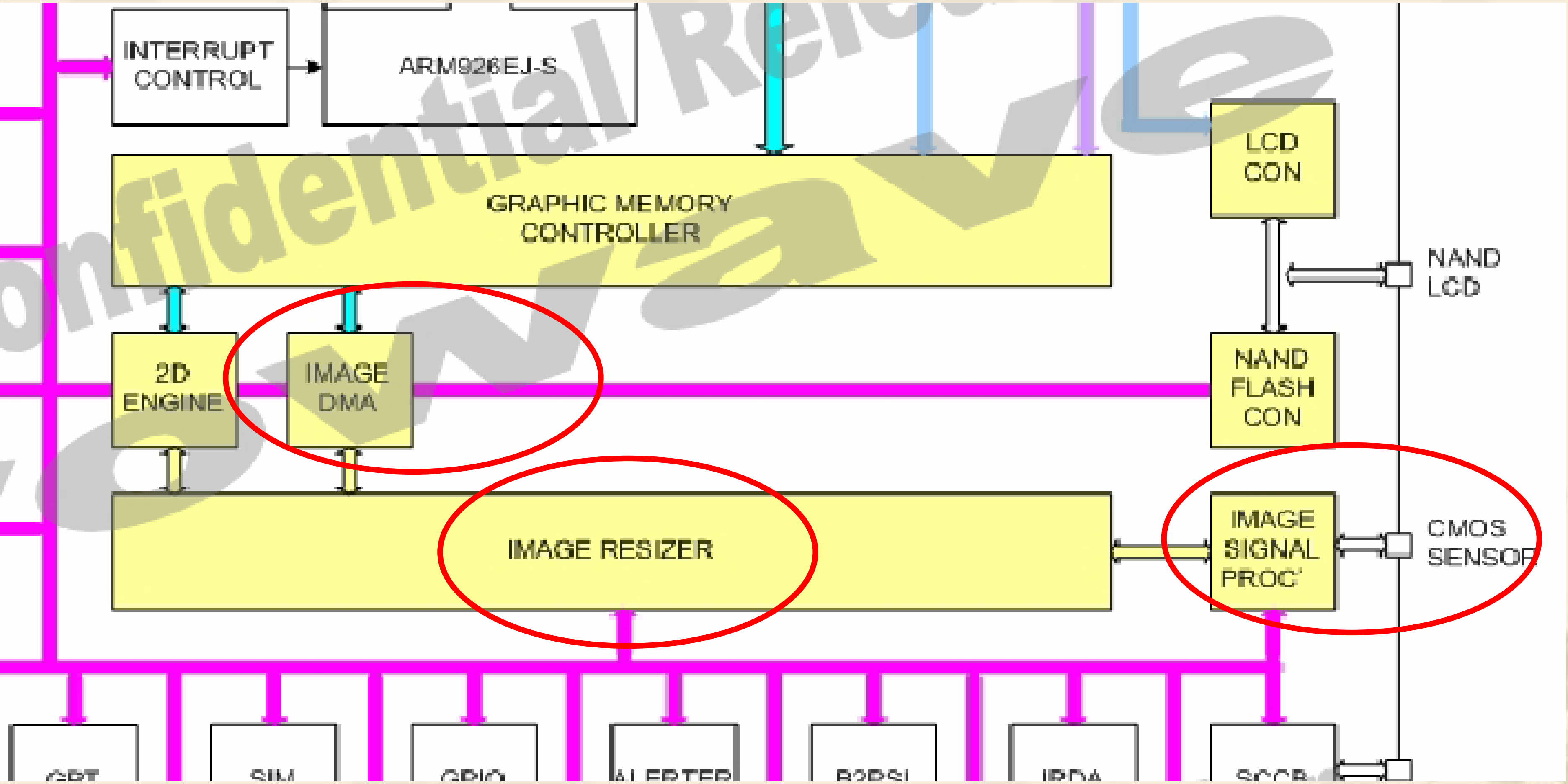
```c
void lcd_fw_update_done_hdlr(void)
{
#if defined(DRV_LCD_FW_UPDATE_SUPPORT)||defined(__DIRECT_SENSOR_SUPPORT__)
  LCD_OPERATION_STATE_ENUM lcd_state_check=NULL;

  if (KAL_FALSE==lcd_hw_trigger_flag)
     return;
```

- lcd_HISR()就是中断处理函数在lcd_system_init()被注册，lcd_hw_trigger_flag就是开关变量，这个一目了然。
- 讲这个的目的就是为下面的camera preview时刷屏问题做个铺垫，当然camera preview的刷屏和这个还有一点点不同，还涉及到resizer的问题，这是后话，暂且不表。

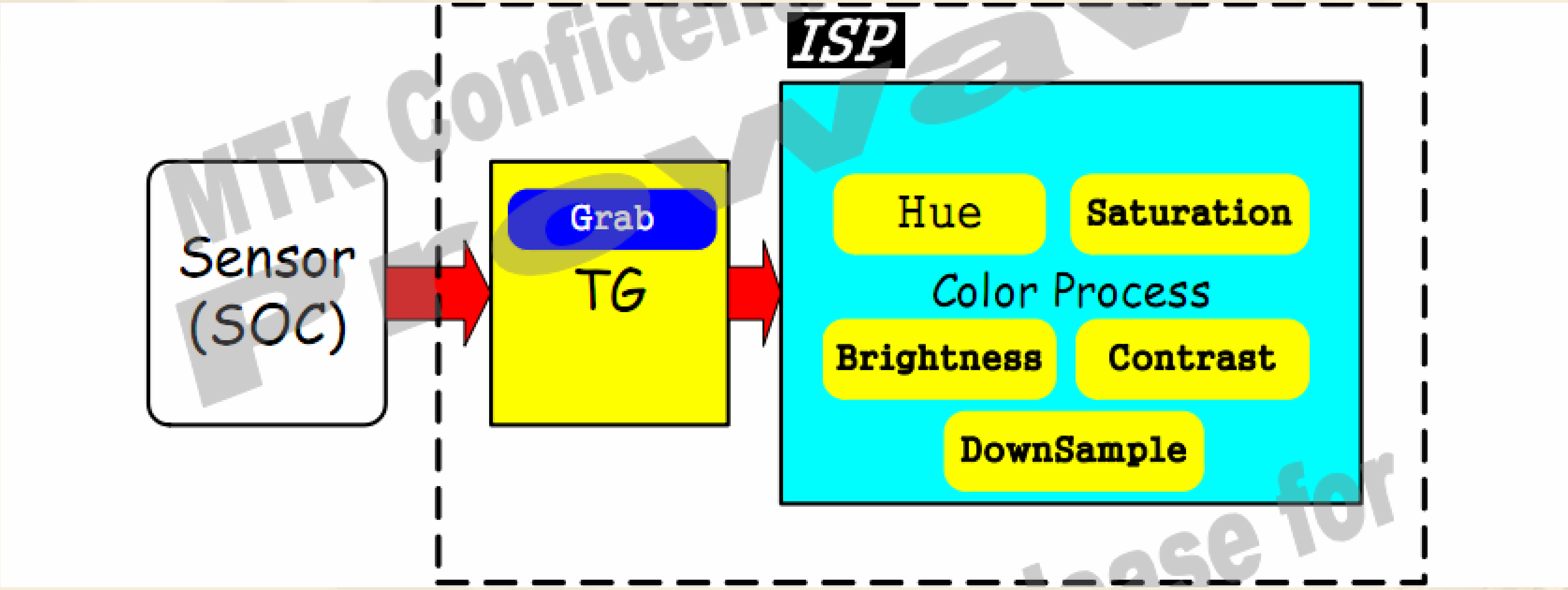
camera

- 终于可以开始讲camera了，让大家久等了。先看几副图，听我一个个讲解。

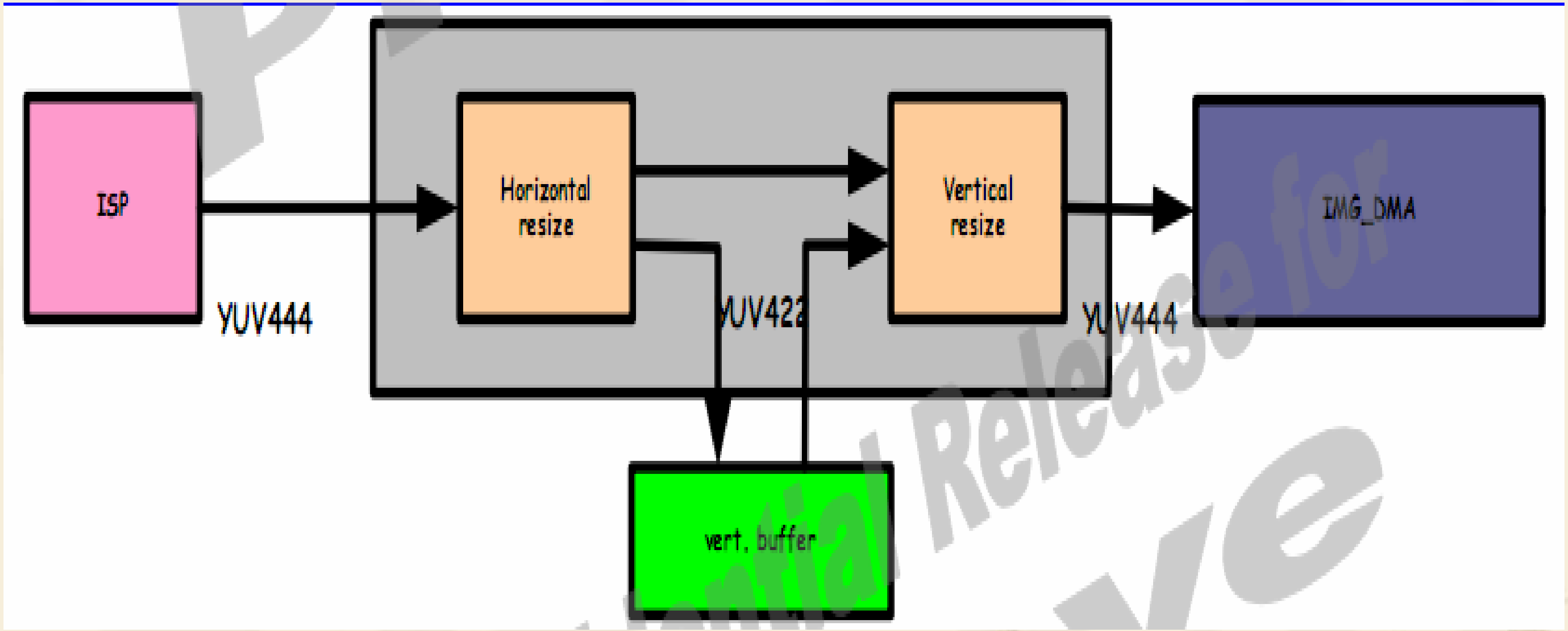


- ISP (image singal proccessor)顾名思义图像数据的处理。
- Image Resizer 图像数据的压缩。
- Image DMA 把数据传给lcd显示。

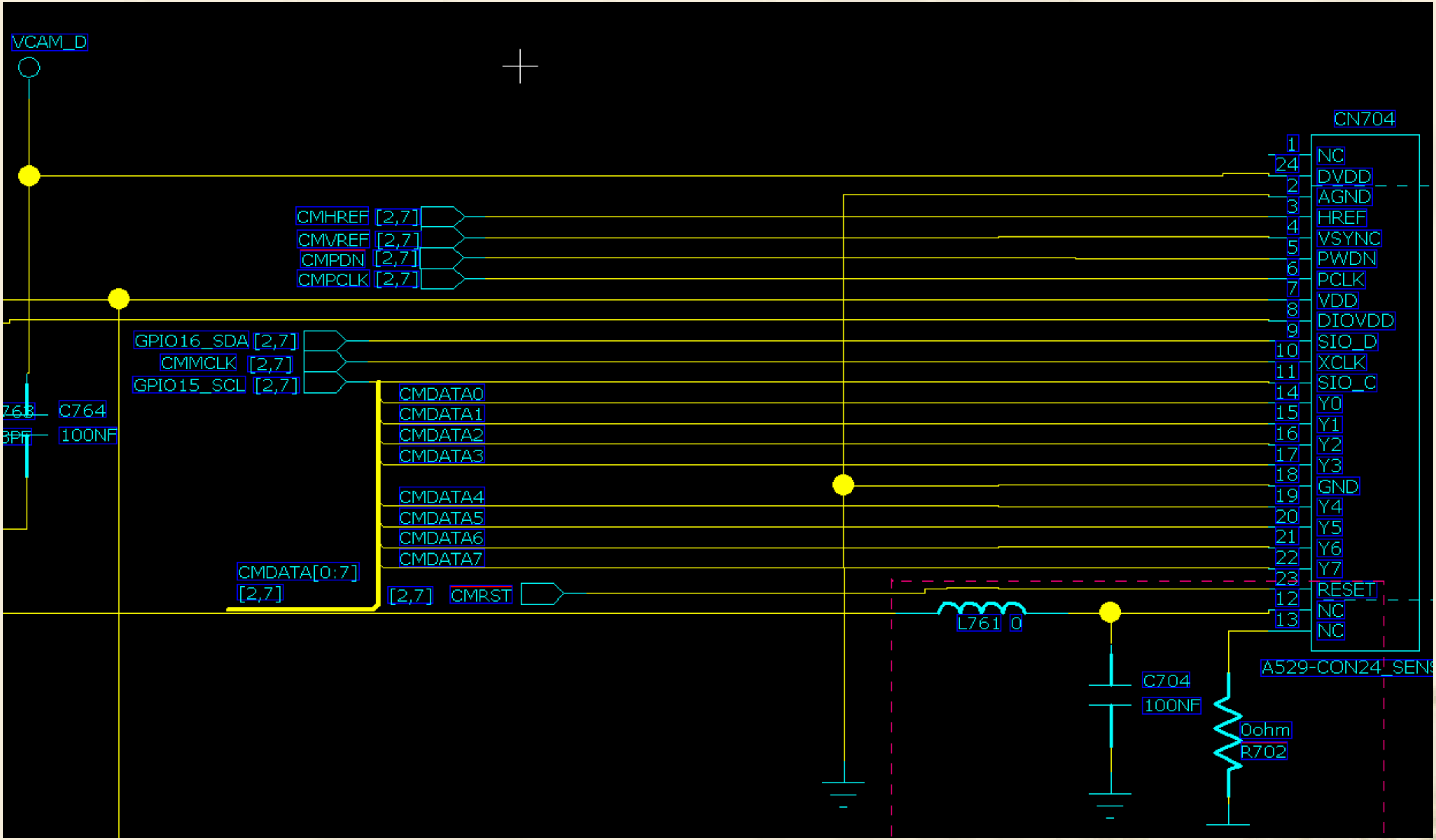
- ISP的组成部分



RESIZER的大致组成



Sensor 部分原理图



- DVDD，VDD，DIOVDD这3个电源，要单独说明一下，听我口述。

- 最低层的sensor driver配置部分大家都很熟悉了，不是要介绍的重点，我们看看，就一笔带过

```
image_sensor_func_struct image_sensor_func_OV5642=
{
    init_OV5642,
    get_OV5642_id,
    get_OV5642_size,
    get_OV5642_period,
    OV5642_preview,
    OV5642_capture,
    write_OV5642_reg,
    read_OV5642_reg,
    set_OV5642_shutter,
    OV5642_night_mode,
    power_off_OV5642,
    set_OV5642_gain
#if (!defined(DRV_ISP_6219_SERIES))
    ,set_OV5642_flashlight
#endif
#if (defined(YUV_SENSOR_SUPPORT))
    ,OV5642_yuv_sensor_setting
#endif
};  /* image_sensor_func_OV5642 */
```

```
void cis_module_power_on(kal_bool on)
{
   if(on==KAL_TRUE)
   {
      #ifdef __CUST_NEW__
      //GPIO_InitIO(1, MODULE_POWER_PIN);
      //GPIO_ModeSetup(MODULE_POWER_PIN, 0);

      pmu_set_vcam_d_sel(VCAM_D_SEL_1_5);//pmu_set_vcamera_sel(2); // select VCAM_D= 1.8 V

      pmu_set_vcam_d_en(KAL_TRUE);//pmu_set_vcamera_en(KAL_TRUE); //enable Vcamera voltage
      pmu_set_vcam_a_sel(VCAM_A_SEL_2_8);//pmu_set_vsw_a_sel(3); // select VCAM_A = 2.8V
      pmu_set_vcam_a_en(KAL_TRUE);//pmu_set_vsw_a_en(KAL_TRUE);//enable Vsw voltage
      #endif /* __CUST_NEW__ */
//       GPIO_WriteIO(1, MODULE_POWER_PIN);
      sccb_setDelay(0);
      sccb_config(SCCB_SW_8BIT, 0x42, 0x43, NULL);  // Default 300KHz
   }
   else
   {
      #ifdef __CUST_NEW__
```

- 设置a，d电压大小,cis_module_power_on()是对sensor进行上电，不是对camera 在bb中的模块上电。cis_module_power_on()在，sensor initial时被调用。

从camera power on开始讲起

- 下面我们以camera的使用流程为线索开始分析，流程分为启动，预览，抓拍，保存，回显几个部分。先说启动（是启动camera 在bb的相关module即isp，不是sensor，sensor要到后面启动init，前面提过）。

mmi_camera_app_screen(void) ← CameraApp.c

↓

mmi_camera_entry_app_screen_internal() ← CameraApp.c

↓

mdi_camera_power_on() ← mdi_camera.c

↓

media_cam_power_up() ← cam_api.c

↓

Send MSG_ID_MEDIA_CAM_POWER_UP_REQ ← cam_ilm.c

↓

cam_power_up_req_hdlr(ilm_ptr); ← cam_main.c cam_msg_handler.c

↓

init_isp_if（）和**init_yuv_isp**（）关键 ← camera_process_v2.c

```
void init_isp_if(void)
{
    ::
    GPIO_ModeSetup(0,1);     GPIO_InitIO(1,0);    //Reset
    GPIO_ModeSetup(1,1);     GPIO_InitIO(1,1);    //PWR Down
    GPIO_ModeSetup(2,1);     GPIO_InitIO(0,2);    //V-sync
    GPIO_ModeSetup(3,1);     GPIO_InitIO(0,3);    //H-sync
    GPIO_ModeSetup(4,1);     GPIO_InitIO(1,4);    //CMPCLK
    GPIO_ModeSetup(5,1);     GPIO_InitIO(1,5);    //CMMCLK
    GPIO_ModeSetup(6,1);    GPIO_InitIO(1,6);     //Data7
    GPIO_ModeSetup(7,1);    GPIO_InitIO(1,7);     //Data6
    GPIO_ModeSetup(8,1);    GPIO_InitIO(1,8);     //Data5
    GPIO_ModeSetup(9,1);    GPIO_InitIO(1,9);     //Data4
    GPIO_ModeSetup(10,1);    GPIO_InitIO(1,10);   //Data3
    GPIO_ModeSetup(11,1);    GPIO_InitIO(1,11);   //Data2
    GPIO_ModeSetup(12,1);    GPIO_InitIO(1,12);   //Data1
    GPIO_ModeSetup(13,1);    GPIO_InitIO(1,13);   //Data0
    ::
    /* configure TG phase counter register */
    SET_CMOS_FALLING_EDGE(1);           //set to HW default
    SET_TG_PIXEL_CLK_DIVIDER(1);        //set to HW default
    ENABLE_CAMERA_TG_PHASE_COUNTER;
    ENABLE_CAMERA_CLOCK_OUTPUT_TO_CMOS;
    ::
    ::
    IRQ_Register_LISR(IRQ_CAMERA_CODE, camera_isp_LISR,"Camera ISR");
    VISUAL_Register_HISR(VISUAL_CAMERA_HISR_ID,camera_isp_HISR);
    init_yuv_isp();/* include yuv init */

}
```

```
void init_yuv_isp(void)
{

    exposure_window.isp_hsub_factor=1;
    exposure_window.isp_vsub_factor=1;
    RESET_CMOS_SENSOR;
    POWER_ON_CMOS_SENSOR;
    ENABLE_CMOS_SENSOR;
    ::
    load_camera_para();
    image_sensor_func_config();
    sensor_err_check=image_sensor_func->sensor_init();
    image_sensor_func->get_sensor_size(&image_sensor_width,&image_sensor_height);
    ::
    IRQSensitivity(IRQ_CAMERA_CODE,LEVEL_SENSITIVE);
    IRQUnmask(IRQ_CAMERA_CODE);
    ::
}    /* i
```

Sensor驱动部分
initial

- 这部分了解就可以。而后面接下去要讲的preview和capture就是关键了。

- 一启动，接下来就直接preview了。我来讲解下面流程

mmi_camera_entry_app_screen_internal()

↓

CAMERA_ENTER_STATE(CAMERA_STATE_PREVIEW);
即mmi_camera_enter_state(CAMERA_STATE_PREVIEW；

↓

mmi_camera_entry_preview_state()

↓

mmi_camera_preview_start()

↓

mdi_camera_preview_start(&camera_preview_data, &camera_setting_data);

↓

mdi_camera_preview_start_internal()

↓

media_cam_preview()

media_cam_preview()

↓

cam_send_preview_req()
Send MSG_ID_MEDIA_CAM_PREVIEW_REQ

↓

cam_preview_req_hdlr(ilm_ptr);

↓

camera_preview_process()

↓

yuv_preview_process()这个函数是要重点介绍的

- 以上是函数流程图供大家以后跟踪代码时参考，现在我先用白话文描述一下这个preview的全过程，然后分别例举代码片段，重点讨论几个问题。

  isp 初始化启动以后，就马上进入preview状态，首先通过数据流设置，告诉isp 相应模块预览的屏幕有多宽多长，sensor传来的数据是什么格式yuv422啊还是yuv444，或者是raw data，图片效果啊，亮度啊，对比度啊，传来的数据要放在哪个buffer中，要多大，外部存还是内部存memory，给lcd刷新显示的数据放在哪个buffer中，用哪个layer来刷，等等。

  这些事情做好后，就通过sensor driver 的preivew函数告诉sensor，你开始传数据（如yuv格式的）吧！我等着收呢。于是sensor就按照要求

- 源源不断的开始发数据了，发给谁，发给isp，一个个pixel地发，若干个pixel组成一行，若干行组成一个帧，等isp收到一个帧之后，先做数据图像处理，让后告诉Resizer “我收好一帧了，也处理好了，给你了”，resizer接到指示把数据收过来，进行横向，纵向压缩。压好后通过dma往lcd上传，我们就看到了preview的图像，就这样sensor不断的发数据过来，isp，resizer，dma，lcd周而复始的皆同工作，就得到了我们camera preview的效果。

- 下面我们通过代码讨论几个问题：

  1.先看一下yuv_preview_process()函数，这个函数是preview的关键。我们截取关键的部分介绍。

```
kal_uint8 yuv_preview_process(camera_preview_process_struct *isp_data)//调这个
{
    ::
    yuv_sensor_parameter_config(ISP_PREVIEW_STATE);
    image_sensor_func->sensor_preview_setting(&exposure_window,&sensor_config_data);
    yuv_isp_grab_size_config(ISP_PREVIEW_STATE,isp_preview_config_data.target_width,
                             isp_preview_config_data.target_height,&exposure_window);
    ::
    lcd_data.module_id=LCD_UPDATE_MODULE_MEDIA;
    lcd_data.lcd_id=isp_preview_config_data.lcd_id;
    lcd_data.fb_update_mode=LCD_HW_TRIGGER_MODE;
    lcd_data.lcm_start_x=isp_preview_config_data.lcm_start_x;
    lcd_data.lcm_start_y=isp_preview_config_data.lcm_start_y;
    lcd_data.lcm_end_x=isp_preview_config_data.lcm_end_x;
    lcd_data.lcm_end_y=isp_preview_config_data.lcm_end_y;
    lcd_data.roi_offset_x=isp_preview_config_data.roi_offset_x;
    lcd_data.roi_offset_y=isp_preview_config_data.roi_offset_y;
    lcd_data.update_layer=isp_preview_config_data.update_layer;
    lcd_data.hw_update_layer=isp_preview_config_data.hw_update_layer;
    ::
    config_pixel_resizer(SCENARIO_CAMERA_PREVIEW_ID);
    ::
    yuv_image_setting(CAM_PARAM_WB,isp_preview_config_data.wb_mode);
    yuv_image_setting(CAM_PARAM_EXPOSURE,isp_preview_config_data.ev_value);
    yuv_image_setting(CAM_PARAM_BANDING,isp_preview_config_data.banding_freq);
    yuv_image_setting(CAM_PARAM_NIGHT_MODE,isp_preview_config_data.night_mode);

    /* start isp engine */
    start_yuv_isp(ISP_PREVIEW_STATE);
    if(isp_preview_config_data.lcd_update==KAL_TRUE)
    {
        lcd_fb_update(&lcd_data);
    }
    ::
}
```

- lcd_data.fb_update_mode=LCD_HW_TRIGGER_MODE;这个就是告诉lcd我要定时自动刷新。这个前面有讲过，这里用到了。
- lcd_data.update_layer=isp_preview_config_data.update_layer;

  告诉lcd，一共要刷哪几层，camera preview会用到layer0(画影像)，layer1(画osd icon)。

  lcd_data.hw_update_layer=isp_preview_config_data.hw_update_layer;

  这个是告诉lcd，用hardware layer(layer0)来刷影像。

  以上两个设置都会通过lcd_fb_update(&lcd_data)函数来起作用。
- config_pixel_resizer(SCENARIO_CAMERA_PREVIEW_ID);这个函数很关键，用来设置resizer模块，它会调到

  ```
  RESZ_Config((RESZ_CFG_STRUCT *) &resize_struct);
  RESZ_Start(scenario_id);
  ```

  这两个函数，并会产生resizer irq中断，调用RESZ_LISR()， RESZ_HISR()来负责lcd刷新，这个后面还会拎出来介绍。
- start_yuv_isp(ISP_PREVIEW_STATE);最关键的就是这个函数了，它让isp可以开始收取数据了。下面具体说明一下。

```c
void start_yuv_isp(ISP_OPERATION_STATE_ENUM isp_state)
{
::
    wait_first_frame_flag=KAL_TRUE;
    isp_preview_frame=0;
    kal_set_eg_events(camera_isp_event_id,0,KAL_AND);
    SET_CAMERA_PREVIEW_MODE;
    SET_CAMERA_FRAME_RATE(0);
    ENABLE_VIEW_FINDER_MODE;
    kal_retrieve_eg_events(camera_isp_event_id,CAMERA_ISP_VD_READY_EVENT,KAL_OR_CONSUME,
                           &event_group,KAL_SUSPEND);
::
}
```

- Start_yuv_isp()中最关键的就是这个宏了，ENABLE_VIEW_FINDER_MODE就是让isp开始收取数据。 ENABLE_VIEW_FINDER_MODE把cam+0018h这个寄存器的第6个bit位置（如右图所示）后面讲到的capture也会用到这个。 跟在它后面的是一个信号等待，(这个机制我前面有讲过)，这个CAMERA_ISP_VD_READY_EVENT信号量是当isp一帧收到处理好后出发。具体在camera_isp_HISR()中可见。

CAM+0018h View Finder Mode Control Register CAM_VFCON

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Name	AV_SYNC_SEL				AV_SYNC_LINENO[11:0]											
Type	R/W				R/W											
Reset	0				0											

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name						SP_DELAY			SP_MODE	TAKE_PIC				FR_CON		
Type						R/W			R/W	R/W				R/W		
Reset						0			0	0				0		

AV_SYNC_SEL		Av_sync start point selection
	0	Start from AV_SYNC_LINENO
	1	Start from vsync
AV_SYNC_LINENO		Av_sync start point line counts
SP_DELAY		Still Picture Mode delay
SP_MODE		Still Picture Mode
TAKE_PIC		Take Picture Request
FR_CON		Frame Sampling Rate Control
	000	Every frame is sampled
	001	One frame is sampled every 2 frames
	010	One frame is sampled every 3 frames
	011	One frame is sampled every 4 frames
	100	One frame is sampled every 5 frames
	101	One frame is sampled every 6 frames
	110	One frame is sampled every 7 frames
	111	One frame is sampled every 8 frames

- 2.camera preview过程中刷屏问题。先把流程画出来给大家看一下。

RESZ_HISR()

↓

RESZ_update_lcd_buffer()

↓

RESZ_preview_update_lcd_buffer()

↓

LCD_preview()

↓

_lcd_fb_update_fw()

- _lcd_fb_update_fw()这个函数在前面提过，用到了和前面讲过的lcd快速定时自动刷新大致一样，唯一不同的是这里是通过resizer中断每次产生来刷屏。压一帧刷一幅，这个也好理解。

```
DRV_WriteReg32(LCD_ROI_OFFSET_REG,lcd_hw_trigger_roi_offset);
DRV_WriteReg32(LCD_ROI_SIZE_REG,lcd_hw_trigger_roi_size);
for (i=0;i<LCD_CMD_QUEUE_LENGTH;i++) {
    *((volatile unsigned int *) (LCD_CMD_PARAMETER_ADDR+(i<<2)))=lcd_hw_trigger_para[i];
}
START_LCD_TRANSFER;
lcd_hw_trigger_flag=KAL_FALSE;
```

❖ 3.简单介绍一下preview过程中数据流的问题(这部份太复杂了，加上mtk写的很乱，非常乱，所以我把主要的几个参数数据列出来供大家参考)

很多camera preview的数据是在mdi_camera_fillin_preview_data()设置的，如尺寸大小，buffer开得大小。下面关键的看几个。

```
cam_preview_data->preview_offset_x = 0; /* not used */
cam_preview_data->preview_offset_y = 0; /* not used */

cam_preview_data->image_buffer_p = (void*)buf_ptr;
```

gdi_layer_get_buffer_ptr(&buf_ptr); 当前acitve layer 的buffer ← 就是mmi buffer在external

```
cam_preview_data->image_width = (kal_uint16) camera_setting_p->image_width;
cam_preview_data->image_height = (kal_uint16) camera_setting_p->image_height;
```

照片大小

```
cam_preview_data->buffer_width = (kal_uint16) layer_width;
cam_preview_data->buffer_height = (kal_uint16) layer_height;
```

屏宽高

```
cam_preview_data->image_buffer_size =
    (kal_uint32) ((camera_setting_p->preview_width * camera_setting_p->preview_height *
                  GDI_LAYER.act_bit_per_pixel) >> 3);
```

即为除以8

preview 刷屏buffer的大小

```
/* set settings parameters */
cam_preview_data->effect = (kal_uint16) camera_setting_p->effect;
cam_preview_data->WB = (kal_uint16) camera_setting_p->wb;
cam_preview_data->exposure = (kal_uint16) camera_setting_p->ev;
cam_preview_data->zoom_factor = (kal_uint16) camera_setting_p->zoom;
cam_preview_data->banding_freq = (kal_uint8) camera_setting_p->banding;
cam_preview_data->flash_mode = (kal_uint16) camera_setting_p->flash;

cam_preview_data->brightness = (kal_uint16) camera_setting_p->brightness;
cam_preview_data->contrast = (kal_uint16) camera_setting_p->contrast;
cam_preview_data->saturation = (kal_uint16) camera_setting_p->saturation;
cam_preview_data->hue = (kal_uint16) camera_setting_p->hue;
cam_preview_data->night_mode = (kal_bool) camera_setting_p->night;
cam_preview_data->snapshot_number = 1;
```

效果参数

```
cam_preview_data->continue_capture = (kal_uint8) camera_setting_p->continue_capture;
```

- 在cam_preview_req_hdlr()中，也有几个地方对重要数据进行进一步的设置，比如终于刷新buffer的大小，在在内部还是外部就在这个函数中决定。篇幅比较大，分两个屏来说明，我来讲解

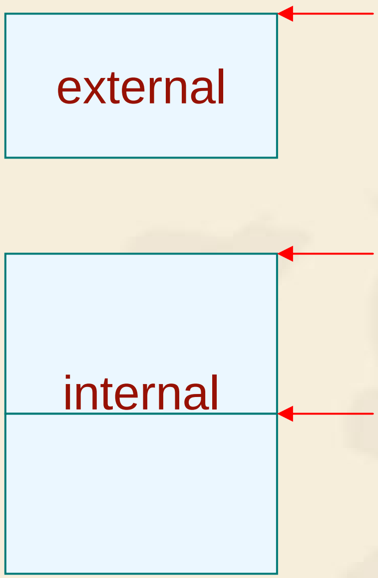
```
req_p = (media_cam_preview_req_struct*) ilm_ptr->local_para_ptr;
cam_context_p->lcd_update = preview_param.lcd_update = req_p->lcd_update;
cam_context_p->lcd_id = preview_param.lcd_id = req_p->lcd_id;
cam_context_p->lcd_start_x = preview_param.lcm_start_x = req_p->lcd_start_x;
cam_context_p->lcd_start_y = preview_param.lcm_start_y = req_p->lcd_start_y;
cam_context_p->lcd_end_x = preview_param.lcm_end_x = req_p->lcd_end_x;
cam_context_p->lcd_end_y = preview_param.lcm_end_y = req_p->lcd_end_y;
cam_context_p->roi_offset_x = preview_param.roi_offset_x = req_p->roi_offset_x;
cam_context_p->roi_offset_y = preview_param.roi_offset_y = req_p->roi_offset_y;
cam_context_p->update_layer = preview_param.update_layer = req_p->update_layer;
cam_context_p->hw_update_layer = preview_param.hw_update_layer = req_p->hw_update_layer;
cam_context_p->preview_offset_x = preview_param.preview_offset_x = req_p->preview_offset_x;
cam_context_p->preview_offset_y = preview_param.preview_offset_y = req_p->preview_offset_y;
cam_context_p->intmem_start_address = preview_param.intmem_start_address =
                                        (kal_uint32) med_alloc_int_mem(cam_preview_mem[0]);
cam_context_p->intmem_size = preview_param.intmem_size = (kal_uint32) cam_preview_mem[0];
cam_context_p->extmem_start_address = preview_param.extmem_start_address = 0;
cam_context_p->extmem_size = preview_param.extmem_size = 0;
cam_context_p->preview_width = preview_param.target_width = req_p->preview_width;
cam_context_p->preview_height = preview_param.target_height = req_p->preview_height;
```

intmem_start_address被赋了动态开的一个内部memory空间头地址，这个空间用来给isp接收处理sensor传来的数据。yuv_preview_process()中就会用到他们，如

```
intmem_init((kal_uint32 *) isp_preview_config_data.intmem_start_address,
                        isp_preview_config_data.intmem_size);
extmem_init((kal_uint32 *) isp_preview_config_data.extmem_start_address,
                        isp_preview_config_data.extmem_size);
```

- 还有一个memory空间很重要，就是供lcd刷屏buffer，而且根据当前系统的内外存剩余情况，再判断是开两个还是开一个，开在哪里。下面我来介绍。

```
 /* use internal memory for preview buffer if enough */
 if (((req_p->image_buffer_size * 2) + MEM_OVERHEAD) < med_int_left_size())
 {
 /* internal memory enough for double buffer */
cam_context_p->frame_buffer_p = (kal_uint32) req_p->image_buffer_p;
cam_context_p->frame_buffer_size = cam_context_p->extmem_size = preview_param.frame_buffer_size =
                                        preview_param.frame_buffer_size1 = req_p->image_buffer_size;
cam_context_p->int_frame_buffer_p = preview_param.frame_buffer_address =
                                (kal_uint32) med_alloc_int_mem(preview_param.frame_buffer_size * 2);
preview_param.frame_buffer_address1 =
                           preview_param.frame_buffer_address + preview_param.frame_buffer_size;
cam_context_p->ext_frame_buffer_p = 0;
 }
 else if (req_p->image_buffer_size + MEM_OVERHEAD < med_int_left_size())
 {
```



req_p->image_buffer_p,
cam_context_p->frame_buffer_p
Cam_preview_data-> image_buffer_p,
Current active layer buffer mmi用的 buffer

Preview_param->frame_buffer_address,
cam_context_p->int_frame_buffer_p
用完要free掉

Preview_param->frame_buffer_address1,
用完要free掉

- 看这个判断语句，如果internal memory空间能够容纳两个image大小的话，就在internal memory中开两个大小的空间，以后就用这两个internal memory来放lcd刷新数据，而原先的external memory也就是mdi_camera_fillin_preview_data()中得到的current active layer buffer，即mmi用的那个buffer，就不去用它了。

- 为什么用两个buffer，这里打断一下，做个说明。大家看一下lcd_preview()函数就明白了，用两个buffer轮流刷新，提高速度。

```
void LCD_preview(kal_uint32 layer, kal_uint32 buffer1, kal_uint32 buffer2 )
{
   :|:
   ::
   if(lcd_preview_count==0) {
      #ifdef MT6225_IDP_DEBUG
      kal_prompt_trace(MOD_BMT,"buf 1=%d",dbg_time2);
      #endif
      _lcd_fb_update_fw(lcd_fw_update_layer, buffer1);
      lcd_preview_count=1;
   } else {
      #ifdef MT6225_IDP_DEBUG
      kal_prompt_trace(MOD_BMT,"buf 2=%d", dbg_time2);
      #endif
      _lcd_fb_update_fw(lcd_fw_update_layer, buffer2);
      lcd_preview_count=0;
   }

}
```

回来继续讲memory空间问题。

```
else if (req_p->image_buffer_size + MEM_OVERHEAD < med_int_left_size())
{
    CAM_BUFFER_TRACE(2);
    /* internal memory enough for single buffer */
    cam_context_p->frame_buffer_p = (kal_uint32) req_p->image_buffer_p;
    cam_context_p->frame_buffer_size = cam_context_p->extmem_size = preview_param.frame_buffer_size
                                       preview_param.frame_buffer_size1 = req_p->image_buffer_size;
    /* one internal buffer */
    cam_context_p->int_frame_buffer_p = preview_param.frame_buffer_address =
                            (kal_uint32) med_alloc_int_mem(preview_param.frame_buffer_size);
    /* one external buffer */
    preview_param.frame_buffer_address1 = cam_context_p->frame_buffer_p;
    cam_context_p->ext_frame_buffer_p = 0;

}
```

external

req_p->image_buffer_p,
cam_context_p->frame_buffer_p
Preview_param->frame_buffer_address1,
Current active layer buffer mmi用的buffer

internal

Preview_param->frame_buffer_address,
cam_context_p->int_frame_buffer_p
用完要free掉

,

- 如果internal memory空间只能够容纳1个image大小的话，就在internal memory中开1个大小的空间，以后就用这1个internal memory和一个原先的external memory来放lcd刷新数据。

```c
else if (req_p->image_buffer_size + MEM_OVERHEAD < med_ext_left_size())
{
    CAM_BUFFER_TRACE(3);
    cam_context_p->frame_buffer_p = (kal_uint32) req_p->image_buffer_p;
    cam_context_p->frame_buffer_size = cam_context_p->extmem_size = preview_param.frame_buffer_size =
                                      preview_param.frame_buffer_size1 = req_p->image_buffer_size;
    /* one external buffer given from MMI */
    preview_param.frame_buffer_address = cam_context_p->frame_buffer_p;
    /* one more external buffer */
    if (cam_context_p->frame_buffer_size %16 != 0)
    {
        cam_context_p->frame_buffer_size = ((cam_context_p->frame_buffer_size + 15) >> 4) << 4;
        preview_param.frame_buffer_size = cam_context_p->frame_buffer_size;
    }
    cam_context_p->ext_frame_buffer_p = preview_param.frame_buffer_address1 =
                                        (kal_uint32) med_alloc_ext_mem(preview_param.frame_buffer_size);
    cam_context_p->int_frame_buffer_p = 0;

}
else
```

External(原先)

req_p->image_buffer_p,
cam_context_p->frame_buffer_p
Preview_param->frame_buffer_address,
Current active layer buffer mmi用的buffer

external

Preview_param->frame_buffer_address1,
cam_context_p->ext_frame_buffer_p
用完要free掉

- 如果internal空间不够的话，就在external上开，看external还能不能再开一个，原先已有一个在external上了。

```
else
{/* use single external buffer from MMI */
    cam_context_p->frame_buffer_p = preview_param.frame_buffer_address1
                                  = preview_param.frame_buffer_address
                                  = (kal_uint32) req_p->image_buffer_p;
    cam_context_p->frame_buffer_size = preview_param.frame_buffer_size1
                                     = preview_param.frame_buffer_size
                                     = req_p->image_buffer_size;
    cam_context_p->int_frame_buffer_p = 0;
    cam_context_p->ext_frame_buffer_p = 0;
}
```

- 内外空间都不够了，那只能用原先mmi上的external了。

External(原先)

req_p->image_buffer_p,
cam_context_p->frame_buffer_p
Preview_param->frame_buffer_address1,
Preview_param->frame_buffer_address2,
Current active layer buffer mmi用的buffer

- 讲到这里,preview的几个细节，大体上就介绍完了，我再口头总结一下，大家不清楚的地方，可以提出来，我们一起探讨一下，一起启迪一下智慧，陶冶一下情操。

capture

- 一句话Capture就是把sensor传来的数据，处理后保存下来。

　对于sensor来说，preview和capture都是在传数据来，如果是yuv数据格式的话，那么数据尺寸也是一样。比如30w的sensor，传来的数据就176*220的尺寸，preview通过resizer压缩成屏的大小或想要显示的大小尺寸供刷屏，而capture根据相应保存尺寸压缩和decode后保存在文件里，当然如果capture要保存的尺寸大小和刚刚preview的尺寸大小一样，也是屏幕大小的话，那么就可以直接用preview的最后一帧数据，这个数据就在刷屏的layer buffer中。所以在capture时，数据有两个地方可以得来，一个是sensor重新传给isp处理后得来，一个就是从layer buffer中直接得来。这里我们主要讨论第一种 capture from isp。先看流程图

Mmi_camera_capture()

mdi_camera_capture_to_memory_direct_couple()

media_cam_capture()

cam_send_capture_req()   send singnal
cam_capture_req_hdlr(ilm_ptr);

if(req_p->source_device == CAM_SRC_ISP)我们讨论from isp这种情况
cam_capture_from_isp(req_p);

camera_capture_jpeg_process(&capture_isp_param);
yuv_capture_jpeg_process(isp_data);
收到yuv数据存成jpeg文件。

mdi_camera_capture_to_memory_direct_couple()这个函数我先提一下，couple有一对的意思，这里有点给人一种“咖啡加伴侣的意思”，的确它与mdi_camera_capture_to_memory()的区别就是除了capture数据到memory供保存外，还另外又准备了一张供capture后回显的图像，这个我在回显章节会介绍。

前面讲过capture(from isp)和preview在很流程上有很多地方是一样的，所不同的是capture 对数据的处理多了一些decode 成jpeg并保存的步骤。

很眼熟吧，和yuv_preview_process()很像吧，我来口头讲一下

```
kal_uint32 yuv_capture_jpeg_process(camera_capture_jpeg_struct *isp_data)
{
    ::
    intmem_init((kal_uint32 *) isp_capture_jpeg_data.intmem_start_address,
                isp_capture_jpeg_data.intmem_size);
    extmem_init((kal_uint32 *) isp_capture_jpeg_data.extmem_start_address,
                isp_capture_jpeg_data.extmem_size);
    yuv_sensor_parameter_config(ISP_CAPTURE_JPEG_STATE);
    image_sensor_func->sensor_capture_setting(&exposure_window,&sensor_config_data);
    /* calculate digital zoom step */
    yuv_isp_grab_size_config(ISP_CAPTURE_JPEG_STATE,isp_capture_jpeg_data.target_width,
        isp_capture_jpeg_data.target_height,&exposure_window);
    ::
    yuv_image_setting(CAM_PARAM_EFFECT,isp_preview_config_data.image_effect);
    yuv_image_setting(CAM_PARAM_WB,isp_preview_config_data.wb_mode);
    ::
    yuv_wait_sensor_capture_stable_frame();
    config_pixel_resizer(SCENARIO_CAMERA_CAPTURE_JPEG_ID);
    /* start isp engine */
    start_yuv_isp(ISP_CAPTURE_JPEG_STATE);
    /* check  recapture fail return */
    while (RESZ_CheckBusy())
    {
       ::
    }
       ::
    stop_yuv_isp(ISP_CAPTURE_JPEG_STATE);
    RESZ_Stop(SCENARIO_CAMERA_CAPTURE_JPEG_ID);
    RESZ_Close(SCENARIO_CAMERA_CAPTURE_JPEG_ID);
    intmem_deinit();
    extmem_deinit();
       ::
    return result;
}
```

```
sw_jpeg_encode_config_data.jpeg_yuv_data[0]=(kal_uint8 *)isp_capture_jpeg_data.y_address;
sw_jpeg_encode_config_data.jpeg_yuv_data[1]=(kal_uint8 *)isp_capture_jpeg_data.u_address;
sw_jpeg_encode_config_data.jpeg_yuv_data[2]=(kal_uint8 *)isp_capture_jpeg_data.v_address;
sw_jpeg_encode_config_data.jpeg_yuv_size[0]=isp_capture_jpeg_data.y_size;
sw_jpeg_encode_config_data.jpeg_yuv_size[1]=isp_capture_jpeg_data.u_size;
sw_jpeg_encode_config_data.jpeg_yuv_size[2]=isp_capture_jpeg_data.v_size;
```

- 把其中yuv_wait_sensor_capture_stable_frame()函数单独拿出来说一下，这个函数就是用来，在capture前，把几个不稳定的帧滤掉，要滤几个我们可以设。一般2，3帧，看一下代码，我来说明一下。

```
void yuv_wait_sensor_capture_stable_frame(void)
{
    kal_uint32 i=0;
    kal_uint32 event_group;
    ENABLE_CAMERA_IDLE_INT;
    SET_CAMERA_CAPTURE_MODE;
    SET_CAMERA_FRAME_RATE(0);
    for(i=0;i<camera_oper_data.capture_delay_frame;i++)
    {
        DISABLE_VIEW_FINDER_MODE;
        kal_set_eg_events(camera_isp_event_id,0,KAL_AND);
        ENABLE_VIEW_FINDER_MODE;
        kal_retrieve_eg_events(camera_isp_event_id,CAMERA_ISP_IDLE_EVENT,KAL_OR_CONSUME,&event_group,KAL
    }
    DISABLE_VIEW_FINDER_MODE;
    DISABLE_CAMERA_IDLE_INT;

}   /* yuv_wait_sensor_capture_stable_frame() */
```

看到什么啊，好眼熟啊，对了，和preview中的是同一个，我前面有提到过的

- 再看一下紧接下来的start_yuv_isp(ISP_CAPTURE_JPEG_STATE);这个函数一调，真的就开始拍照了

```
void start_yuv_isp(ISP_OPERATION_STATE_ENUM isp_state)
{
    case ISP_CAPTURE_JPEG_STATE:
    case ISP_CAPTURE_MEM_STATE:
        SET_CAMERA_CAPTURE_MODE;
        SET_CAMERA_FRAME_RATE(0);
        ENABLE_VIEW_FINDER_MODE;
}
```

ENABLE_VIEW_FINDER_MODE 这个宏是camera的精髓，用它来定位很方便的，我管它叫“大爷”。

- 前面介绍了capture的一个大体流程，现在就结合capture过程中数据流的设置，再介绍一下对收到的数据我们怎样decode和怎样保存成文件。

先看数据流,大部分在cam_capture_from_isp()里

```
kal_uint32 cam_capture_from_isp(media_cam_capture_req_struct *req_p)
{

    capture_isp_param.target_width = cam_context_p->image_width = req_p->image_width ;//1600;//1280;
    capture_isp_param.target_height = cam_context_p->image_height = req_p->image_height ;//1200;//960;
    capture_isp_param.intmem_start_address
        = cam_context_p->intmem_start_address =(kal_uint32) med_alloc_int_mem(cam_capture_mem[0]);
    capture_isp_param.intmem_size = cam_context_p->intmem_size = (kal_uint32) cam_capture_mem[0];
    capture_isp_param.extmem_start_address = 0;
    capture_isp_param.extmem_size = 0;
    capture_isp_param.image_quality = cam_context_p->image_quality = (kal_uint8) req_p->image_quality;
    capture_isp_param.jpeg_gray_mode = 0;    /* alway color mode */
    capture_isp_param.target_buffer_start_address = (kal_uint32) cam_context_p->capture_buffer_p;
    capture_isp_param.target_buffer_size = MAX_CAM_FILE_BUFFER_LEN;
    /* allocate memory for YUV channels */
    img_width = (req_p->image_width%16==0)? req_p->image_width : 16 - (req_p->image_width%16) + req_p->i
    img_height = (req_p->image_height%16==0)? req_p->image_height : 16 - (req_p->image_height%16) + req_
    cam_context_p->channel_size = img_width*img_height;

    if (capture_isp_param.jpeg_gray_mode ==0)
    {
        /* color mode */
        capture_isp_param.y_address = cam_context_p->y_address =
            (kal_uint32) med_alloc_ext_mem(cam_context_p->channel_size);
        capture_isp_param.u_address = cam_context_p->u_address =
            (kal_uint32) med_alloc_ext_mem(cam_context_p->channel_size/4);
        capture_isp_param.v_address = cam_context_p->v_address =
            (kal_uint32) med_alloc_ext_mem(cam_context_p->channel_size/4);

       ::
    }
    else
```

- 这里的intmem_start_address和preview中的是一样的，也是给isp接收处理sensor传来的数据。
- capture_isp_param.target_buffer_start_address = (kal_uint32) cam_context_p->capture_buffer_p;处理好的数据就放在这里，再经过cam_encode_layer()函数encode后以jpeg格式被保存成文件，这个后面还会再介绍。
- y_address，u_address，v_address；这3个是分别开出的yuv channel，也就是另外开出的3个buffer 空间，从得到的数据中分别提取出y，u，v，这3个值，这三个值经过下图所示的这个函数，被转换成

```
iul_yuv420_non_interleave_to_rgb565_general_resize((unsigned char *)cam_context_p->y_address,
                                (unsigned char *)cam_context_p->u_address,
                                (unsigned char *)cam_context_p->v_address,
                                capture_isp_param.target_width,
                                capture_isp_param.target_height,
                                (unsigned short*)cam_context_p->memory_output_buffer_address,
                                cam_context_p->memory_output_width,
                                cam_context_p->memory_output_height);
```

RGB565格式，存在cam_context_p->memory_output_buffer_address中，用来回显用的，这个后面回显章节，还会在介绍。

以上也可以看出mdi_camera_capture_to_memory_direct_couple特点所在，不但把数据capture到capture_buffer_p中，还同时通过把yuv值转换成rgb565格式，放在memory_output_buffer_address中，将来直接用来回显。

顺便插一句，简要说明一个yuv。

- YUV从何而来呢？在现代彩色电视系统中，通常采用三管彩色摄像机或彩色CCD摄像机进行摄像，然后把摄得的彩色图像信号经分色、分别放大校正后得到RGB，再经过矩阵变换电路得到亮度信号Y和两个色差信号R－Y（即U）、B－Y（即V），最后发送端将亮度和色差三个信号分别进行编码，用同一信道发送出去。这种色彩的表示方法就是所谓的YUV色彩空间表示。

  采用YUV色彩空间的重要性是它的亮度信号Y和色度信号U、V是分离的。如果只有Y信号分量而没有U、V分量，那么这样表示的图像就是黑白灰度图像。彩色电视采用YUV空间正是为了用亮度信号Y解决彩色电视机与黑白电视机的兼容问题，使黑白电视机也能接收彩色电视信号。

  YUV与RGB相互转换的公式如下（RGB取值范围均为0-255）：

$$Y = 0.299R + 0.587G + 0.114B$$
$$U = -0.147R - 0.289G + 0.436B$$
$$V = 0.615R - 0.515G - 0.100B$$
$$R = Y + 1.14V$$
$$G = Y - 0.39U - 0.58V$$
$$B = Y + 2.03U$$

❖ 接收下来在看一下，capture到的数据是怎样被encode成jpeg并被保存的。我来口头讲解

```
void cam_save_req_hdlr(ilm_struct *ilm_ptr)
{
    ::                                          encode成jpeg
    cam_context_p->file_size = cam_encode_layer();
    result = cam_close_image_file(cam_context_p->file_size);
    ::                         保存成file
}
```

```
kal_uint32 cam_encode_layer(void)
{
    jpg_encode.image_buffer_address = NULL;
    jpg_encode.input_type = IMAGE_FORMAT_YUV420;
    jpg_encode.output_type = IMAGE_FORMAT_YUV420;
    jpg_encode.jpeg_yuv_data[0] = (kal_uint8*) cam_context_p->y_address;
    jpg_encode.jpeg_yuv_data[1] = (kal_uint8*) cam_context_p->u_address;
    jpg_encode.jpeg_yuv_data[2] = (kal_uint8*) cam_context_p->v_address;
    jpg_encode.jpeg_yuv_size[0] = cam_context_p->channel_size;
    jpg_encode.jpeg_yuv_size[1] = cam_context_p->channel_size/4;
    jpg_encode.jpeg_yuv_size[2] = cam_context_p->channel_size/4;
    jpg_encode.intmem_size = img_context_p->intmem_size = MAX_IMG_ENC_INT_MEM_SIZE;
    jpg_encode.intmem_start_address = img_context_p->intmem_start_address =
            (kal_uint32) med_alloc_int_mem(MAX_IMG_ENC_INT_MEM_SIZE);
    jpg_encode.image_width = cam_context_p->image_width;
    jpg_encode.image_height = cam_context_p->image_height;
    jpg_encode.image_quality = cam_context_p->image_quality;
    jpg_encode.jpeg_file_start_address = (kal_uint32) cam_context_p->capture_buffer_p;
    jpg_encode.jpeg_file_buffer_size = MAX_CAM_FILE_BUFFER_LEN;
    DCM_Load(DYNAMIC_CODE_JPEG_ENC,NULL,NULL);
    img_context_p->file_size = jpeg_encode_process(&jpg_encode);
    DCM_Unload(DYNAMIC_CODE_JPEG_ENC);
    return img_context_p->file_size;

}
```

```
kal_int32 cam_close_image_file(kal_uint32 size)
{
::
result = FS_Write(cam_context_p->file_handle, (void*)cam_context_p->capture_buffer_p, size, &len);
FS_Close(cam_context_p->file_handle);
::
}
```

❖ 讲到这里capture的大致过程就讲完了，而且也顺带把保存部分介绍了，所以下面就把保存部分的流程图画出来给大家，其他的就不细讲了。

Mdi_camera_save_capture_image

↓

Media_cam_save_image

↓

Cam_send_save_req

↓

Cam_send_save_hdlr

↓

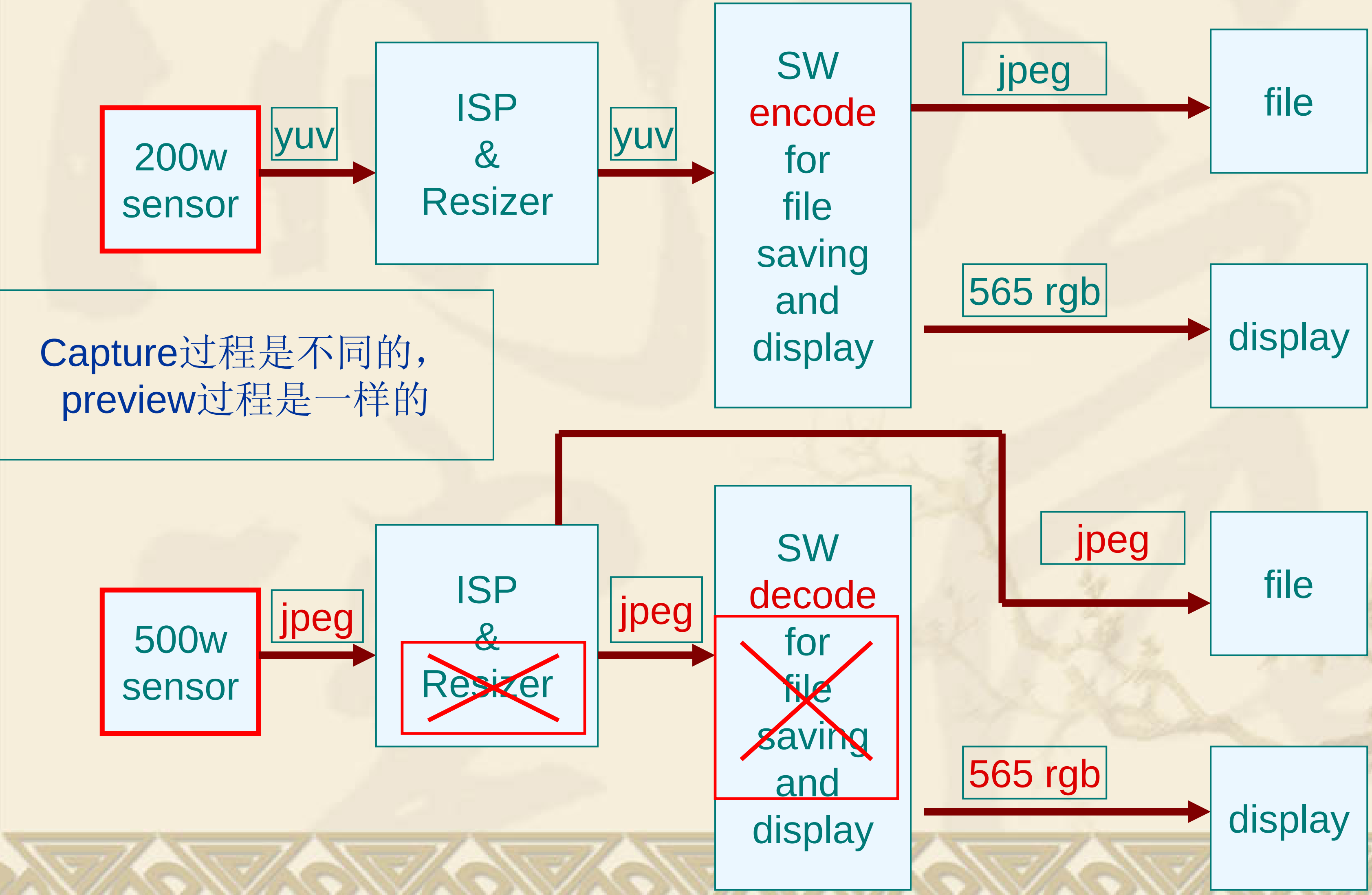
Cam_close_image_file

- 现在讲一下回显。

刚才在capture的时候，特别对mdi_camera_capture_to_memory_direct_couple()函数做了说明，他除了capture要保存成file的数据的同时，也encode了一张供屏幕回显的图片，下面看一下代码片段，我讲解。

```
static void mmi_camera_capture(void)
{
 ::
 g_camera_cntx.last_error
 = mdi_camera_capture_to_memory_direct_couple(&g_camera_cntx.capture_buf_ptr,
                    &g_camera_cntx.capture_size,(U32) g_camera_cntx.direct_couple_buffer,
  ((g_camera_cntx.resized_width *g_camera_cntx.resized_height*GDI_MAINLCD_BIT_PER_PIXEL)>>3),
     g_camera_cntx.resized_width,g_camera_cntx.resized_height,g_camera_cntx.captured_filepath);
    if(g_camera_cntx.last_error == MDI_RES_CAMERA_SUCCEED)
    {
        if(g_camera_cntx.is_direct_couple == TRUE)
        {
            PU8 src;PU8 dest;U16 src_width, src_height, src_pitch;
            S32 src_offset_x=0, src_offset_y=0;S32 dest_pitch, dest_offset_x, dest_offset_y;
            gdi_rect_struct dest_clip;
            gdi_layer_get_buffer_ptr(&dest);
            gdi_layer_clear(GDI_COLOR_BLACK);
            src = g_camera_cntx.direct_couple_buffer;
            src_pitch = g_camera_cntx.resized_width;
            dest_pitch = g_camera_cntx.osd_UI_device_width;
            dest_clip.x1 = g_camera_cntx.resized_offset_x;
            dest_clip.y1 = g_camera_cntx.resized_offset_y;
            dest_clip.x2 = g_camera_cntx.resized_offset_x + g_camera_cntx.resized_width - 1;
            dest_clip.y2 = g_camera_cntx.resized_offset_y + g_camera_cntx.resized_height - 1;
            src_width = g_camera_cntx.resized_width;
            src_height = g_camera_cntx.resized_height;
            dest_offset_x = g_camera_cntx.resized_offset_x;
            dest_offset_y = g_camera_cntx.resized_offset_y;
            /* use 2D copy from cache layer to thumbnail layer */
        gdi_2d_memory_blt_without_transpant_check(src,src_pitch,src_offset_x,src_offset_y,
                                    src_width,src_height,dest,dest_pitch,dest_offset_x,
                                    dest_offset_y,dest_clip,GDI_MAINLCD_BIT_PER_PIXEL);
        gdi_layer_set_blt_layer(g_camera_cntx.base_layer_handle,g_camera_cntx.osd_layer_handle0, 0, 0);
        }
}
```

❖ 讲到这里camera部分的几大模块也大体介绍完了，现在我来口头总结一下，接下来，就利用上面所介绍的知识，我们一起探讨一下，在只支持200万sensor的mtk6235上怎样让ov 500万sensor跑起来。口头白板上简介。



- 通过以上分析，可知Ov5642的调试关键就是在capture的过程，所以这里把capture拿出来单独解释一下。

```
#ifdef PWV_DRV_CAMERA_OV5642_500W
   capture_isp_param.memory_output = req_p->memory_output;
   capture_isp_param.memory_output_width = req_p->memory_output_width;
   capture_isp_param.memory_output_height = req_p->memory_output_height;
   capture_isp_param.memory_output_buffer_address = req_p->memory_output_buffer_address;
   capture_isp_param.memory_output_buffer_size = req_p->memory_output_buffer_size;
   file_size = camera_capture_direct_jpeg_process(&capture_isp_param);

#else
  file_size = camera_capture_jpeg_process(&capture_isp_param);
#endif
```

我来一句句解释给大家听

```
kal_uint32 camera_capture_direct_jpeg_process(camera_capture_jpeg_struct *isp_data)
{
    kal_uint32 zte_jpeg_file_size;
#if (defined(ISP_SUPPORT)&&defined(YUV_SENSOR_SUPPORT))
    kal_uint32 jpeg_file_size=1;
    volatile kal_uint8 i;
    kal_uint32 temp_grab_size;
    MMDI_SCENERIO_ID scene_id=SCENARIO_CAMERA_CAPTURE_JPEG_ID;
    kal_uint32 event_group;
    kal_uint8* pImage;
    static char continue_capture=0;
    kal_int32 nErr =0;
    capture_from_jpeg_format = KAL_TRUE;
    //SET_CAMERA_INPUT_TYPE(INPUT_BAYER);  //this coiuld be a problem  book
    ENABLE_CMOS_SESNOR;
    DISABLE_VIEW_FINDER_MODE;
    DISABLE_CAMERA_IDLE_INT;
    DISABLE_CAMERA_VD_DONE_INT;

    exposure_window.image_target_width=isp_data->target_width;
    exposure_window.image_target_height=isp_data->target_height;
    exposure_window.digital_zoom_factor=isp_digital_zoom_factor;
    exposure_window.night_mode = isp_preview_config_data.night_mode;
    exposure_window.banding_freq = isp_preview_config_data.banding_freq;
    exposure_window.exposure_window_width = 1024; //default

    sensor_config_data.meta_mode=KAL_FALSE;
    sensor_config_data.enable_flashlight_tansfer=KAL_FALSE;
    sensor_config_data.image_quality=isp_data->image_quality;
```

```
if(isp_data->continue_capture==1)
{
    sensor_config_data.enable_shutter_tansfer=KAL_TRUE;
}
else
{
    sensor_config_data.enable_shutter_tansfer=KAL_FALSE;
}
image_sensor_func->sensor_capture_setting(&exposure_window,&sensor_config_data);
isp_grab_width  =  exposure_window.exposure_window_width;
isp_grab_height = exposure_window.exposure_window_height;
isp_grab_start_x=exposure_window.grab_start_x;
isp_grab_start_y=exposure_window.grab_start_y;
SET_YUV_TG_GRAB_PIXEL(isp_grab_start_x,isp_grab_width);
SET_YUV_TG_GRAB_LINE(isp_grab_start_y,isp_grab_height);
DISABLE_VERTICAL_SUB_SAMPLE;
DISABLE_HORIZONTAL_SUB_SAMPLE;
ENABLE_REZ_DISCONN;
DISABLE_REZ_LPF;
SET_OUTPUT_PATH_TYPE(0);
kal_mem_set((void*)isp_data->target_buffer_start_address,0xff,isp_data->target_buffer_size);
REG_ISP_OUTPUT_ADDR=(kal_uint32)isp_data->target_buffer_start_address;
SET_CAMERA_CAPTURE_MODE;
ENABLE_CAMERA_VD_DONE_INT;
SET_CAMERA_FRAME_RATE(0);
```

不要resize

用来接收的内存地址

jpeg数据就存在这里

```
for(i=0;i<3;i++)
  {
      DISABLE_VIEW_FINDER_MODE;
      ENABLE_CAMERA_OUTPUT_TO_MEM;
      kal_set_eg_events(camera_isp_event_id,0,KAL_AND);
      ENABLE_VIEW_FINDER_MODE;
      kal_retrieve_eg_events(camera_isp_event_id,CAMERA_ISP_VD_READY_EVENT,KAL_OR_CONSUME,
                          &event_group,KAL_SUSPEND);
  }
      DISABLE_VIEW_FINDER_MODE;
      DISABLE_CAMERA_IDLE_INT;
/*capturing*/
      isp_operation_state=ISP_CAPTURE_JPEG_STATE;
      ENABLE_CAMERA_OUTPUT_TO_MEM;
      SET_CAMERA_CAPTURE_MODE;
      SET_CAMERA_FRAME_RATE(0);
      kal_set_eg_events(camera_isp_event_id,0,KAL_AND);
      ENABLE_VIEW_FINDER_MODE;
      kal_retrieve_eg_events(camera_isp_event_id,CAMERA_ISP_VD_READY_EVENT,KAL_OR_CONSUME,
                            &event_group,KAL_SUSPEND);
    /*****************************************/
    /* Disable capture path */
    DISABLE_CMOS_SESNOR;
    DISABLE_VIEW_FINDER_MODE;
    DISABLE_CAMERA_OUTPUT_TO_MEM;
    ENABLE_REZ_LPF;
    DISABLE_REZ_DISCONN;
```

过滤掉3帧，不用。

真正开始抓拍

```
isp_operation_state=ISP_STANDBY_STATE;
pImage = (void*)isp_data->target_buffer_start_address;
while(jpeg_file_size<isp_data->target_buffer_size){
    if((*pImage)==0xff)
    {
        pImage++;
        jpeg_file_size++;
        if((*pImage)==0xD9)//0xD9
        {
            break;
        }
        if((*pImage)==0xff)//0xD9
        {
            (*pImage)=0xD9;
            break;
        }
    }
    else
    {
        pImage++;
        jpeg_file_size++;
    }
   if(jpeg_file_size>=isp_data->target_buffer_size)
    {
        jpeg_file_size =isp_data->target_buffer_size;
        break;
    }
};
temp_grab_size = (jpeg_file_size & 0x0f);
temp_grab_size = 0x10 - temp_grab_size;
jpeg_file_size += temp_grab_size;
jpeg_file_size = (isp_data->target_buffer_size > jpeg_file_size)? jpeg_file_size:isp_data->target_
```

对jpeg数据进行简单尾巴处理

```
//decode process  jpeg->rgb 565
    sw_jpeg_decode_config_data.jpeg_file_handle          = 0;
    sw_jpeg_decode_config_data.jpeg_file_buffer_address = isp_data->target_buffer_start_address;
    sw_jpeg_decode_config_data.jpeg_file_size            = jpeg_file_size;
    sw_jpeg_decode_config_data.jpeg_file_buffer_size     = isp_data->target_buffer_size;
    sw_jpeg_decode_config_data.intmem_start_address      = isp_data->intmem_start_address;
    sw_jpeg_decode_config_data.intmem_size               = isp_data->intmem_size;
    sw_jpeg_decode_config_data.extmem_start_address      = (kal_uint32) med_alloc_ext_mem(MAX_PROG_JPG_D
    if (sw_jpeg_decode_config_data.extmem_start_address)
    {
        sw_jpeg_decode_config_data.extmem_size = MAX_PROG_JPG_DEC_EXT_MEM_SIZE;
    }
    else
    {
        if ((sw_jpeg_decode_config_data.extmem_start_address =
            (kal_uint32) med_alloc_ext_mem(MAX_IMG_DEC_EXT_MEM_SIZE)) == 0)
        {
            ASSERT(sw_jpeg_decode_config_data.extmem_start_address > 0);
        }
        sw_jpeg_decode_config_data.extmem_size = MAX_IMG_DEC_EXT_MEM_SIZE;
    }
    sw_jpeg_decode_config_data.image_buffer_address      = isp_data->memory_output_buffer_address;
    sw_jpeg_decode_config_data.image_buffer_size         = isp_data->memory_output_buffer_size;
    sw_jpeg_decode_config_data.image_width               = isp_data->memory_output_width;
    sw_jpeg_decode_config_data.image_height              = isp_data->memory_output_height;
    sw_jpeg_decode_config_data.image_clip_x1             = 0;
    sw_jpeg_decode_config_data.image_clip_x2             = isp_data->memory_output_width-1;
    sw_jpeg_decode_config_data.image_clip_y1             = 0;
    sw_jpeg_decode_config_data.image_clip_y2             = isp_data->memory_output_height-1;
    sw_jpeg_decode_config_data.image_data_format         = IMGDMA_IBW_OUTPUT_RGB565;
    sw_jpeg_decode_config_data.image_pitch_mode          = KAL_FALSE;
    sw_jpeg_decode_config_data.jpeg_decode_cb            = camera_capture_jpeg_decode_cb;
    sw_jpeg_decode_config_data.memory_output = 0;
    nErr = jpeg_decode_process(&sw_jpeg_decode_config_data);
/*
   med_free_ext_mem((void **)&sw_jpeg_decode_config_data.extmem_start_address);
   camera_process_stage|=LEAVE_CAMERA_CAPTURE_JPEG;
   zte_jpeg_file_size=jpeg_file_size+10;
#endif
   return zte_jpeg_file_size;
```

把jpeg decode成屏幕大小的565rbg格式，用来回显

- 现在讲一下我在调试中的一些心得和注意点。

1.先确认硬件的正确性。这个很关键，在调试ov5642的过程中，就遇到dvdd供电流不足的问题，导致拍照一直有问题，折腾了好久。我们在写driver的时候也要把搞硬件的哥们儿拖下水。

2.还有就是要把camera模块的整个流程和相应的寄存器，认认真真顺一遍查一遍，磨刀不误砍柴功，在对整体框架了解的情况下，再开发，而不能直接porting FAE提供的代码，他们的代码有时是有问题的。因为毕竟不是简简单单调试camera 效果的问题。

3.注重细节，对于RD来说，细节是关键。

4.至于在调试ov5642过程中的具体细节问题我在前面都讲的很清楚了，就不再次罗列出来了。

下期预告

Q&A

Thank you