# FUllHAN/Application Engineer

## 1. peripherals

### 1.1 cache一致性

核心：CPU

手段：DMA

Start：外设，这里为ALC5616 To：SRAM

将数据从I/O读入SRAM：先将cache进行invalid操作，DMA进行数据搬运，搬运结束后CPU再进行访问，不会由于hit读取过时数据；

将数据从SRAM写入I/O：先将cache进行clean操作，CPU将数据写入cache中，DMA再将数据搬运回I/O中；

CPU需要访问变量A：

cache中含有变量A，发生hit，此时A在I/O中发生变化，产生一致性问题（脏数据），解决措施：invalid cache，由DMA重新从I/O搬运数据；

cache中不含变量A，发生miss，需要从MEM中读取数据，MEM中没有就从I/O将数据读入MEM再传回给CPU；

CPU需要写回变量A：

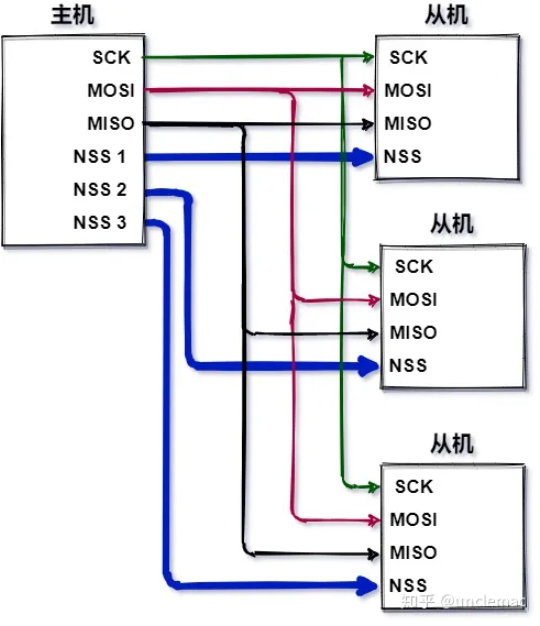
cache中含有变量A，发生hit，写回I/O中，与CPU中的新数据不一致，解决措施：clean cache，CPU将数据写入cache中，由DMA将数据从cache搬运至I/O中；

cache中不含变量A，直接CPU写数据至cache，由DMA将数据搬运至I/O；

### 1.2 SPI时序

SPI是Serial Peripheral Interface的缩写，是一种高速的全双工通信总线，主要应用在EEPROM、FLASH、实时时钟等。有主Master、从Slave两种工作模式。一个SPI通讯可以包含单个Master、多个Slave，其中SCK由Master进行提供，全双工。

一般有四根线，分别是SCK时钟线，MOSI主设备输出从设备输入、MISO主设备输入从设备输出、CS/SS片选信号线，连接方式如下：

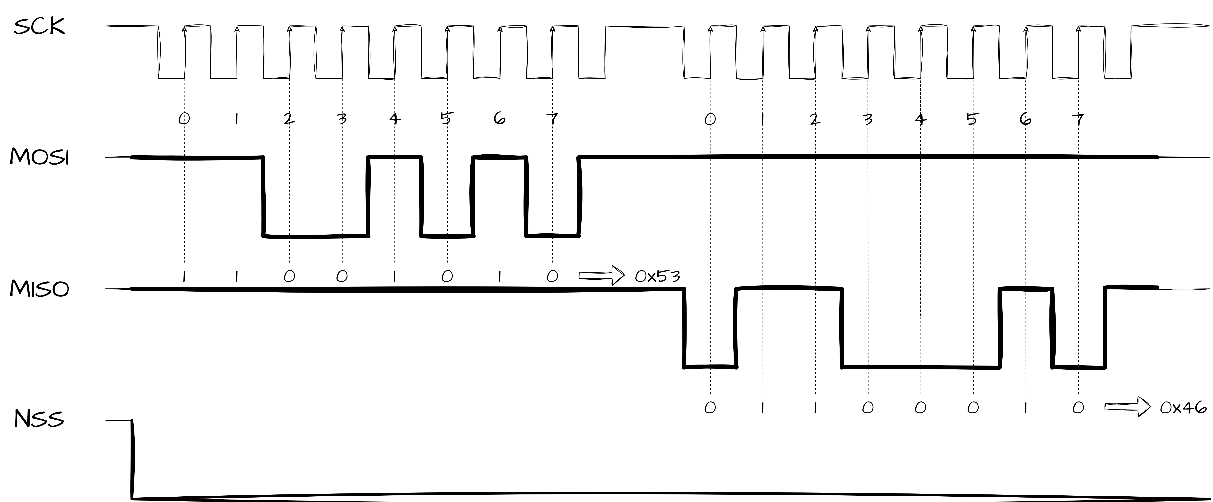
.

SPI多从机连接方式

SPI工作模式：

1. Master先将需要操作的Slave对应的NSS线电平拉低，保证一对一通讯；
2. 如果是MOSI，即主机输出从机输入，主机产生相应时钟信号，主机将数据按bit发送给从机；
3. 如果是MISO，即主机输入从机输出，主机产生相应时钟信号，从机将数据按bit发送给主机；

这里注意是按bit进行数据读写，先发送的数据先接收，如下图所示：



SPI数据读写

时钟极性和相位极性

时钟极性CKP：

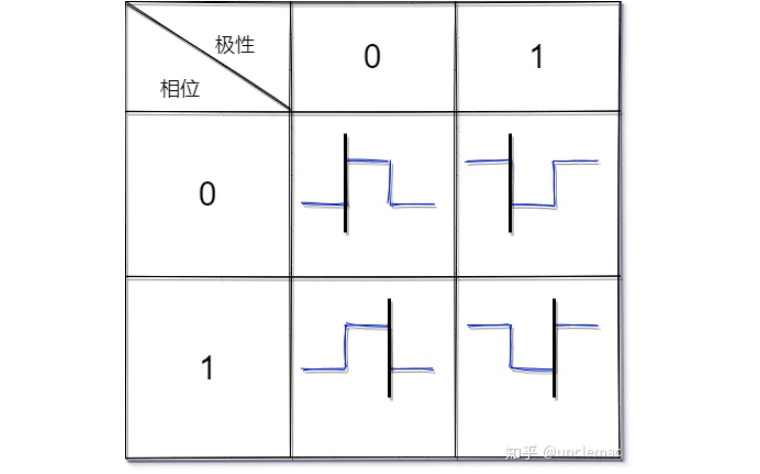
CKP = 0：时钟空闲IDLE为低电平 0；

CKP = 1：时钟空闲IDLE为高电平1；

相位极性CKE：

CKE = 0：在时钟信号SCK的第一个跳变沿采样；

CKE = 1：在时钟信号SCK的第二个跳变沿采样；



时钟极性和相位极性的配置称为SPI模式

|  |  |  |
| --- | --- | --- |
| SPI Mode | CPOL/时钟 | CPHA/相位 |
| 0[00] | 0 | 0 |
| 1[01] | 0 | 1 |
| 2[10] | 1 | 0 |
| 3[11] | 1 | 1 |

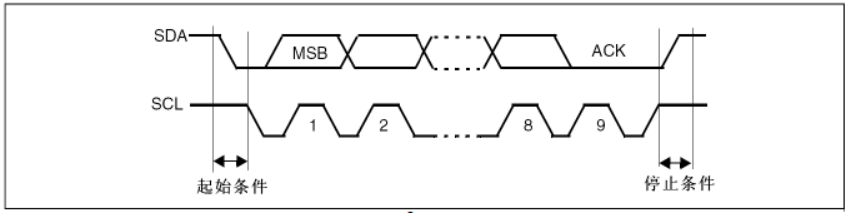
根据表格及上图，可以确定上图的 SPI数据读写 的工作模式为3

### 1.3 I2C时序

I2C（Inter-Integrated Circuit）两线式串行总线，属于一个master，多个slave的总线结构，每个设备都有一个特定的DEVICE\_ID。I2C两线为双向，分别为串行时钟线SCL，串行数据线SDA，通信方式均是由master发起，slave响应。高速I2C总线一般传输速度400k bps。

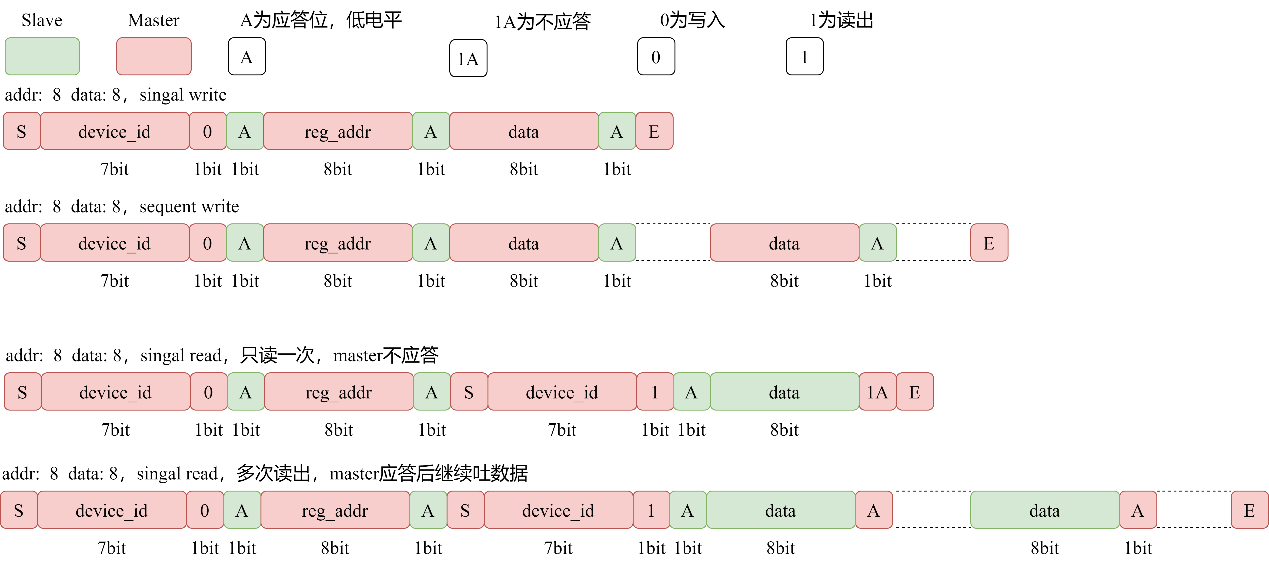
开始信号：sda电平由高变低，scl电平由高变低

结束信号：sda电平有低变高，scl电平由低变高



I2C总线时序图\_开始及结束标志

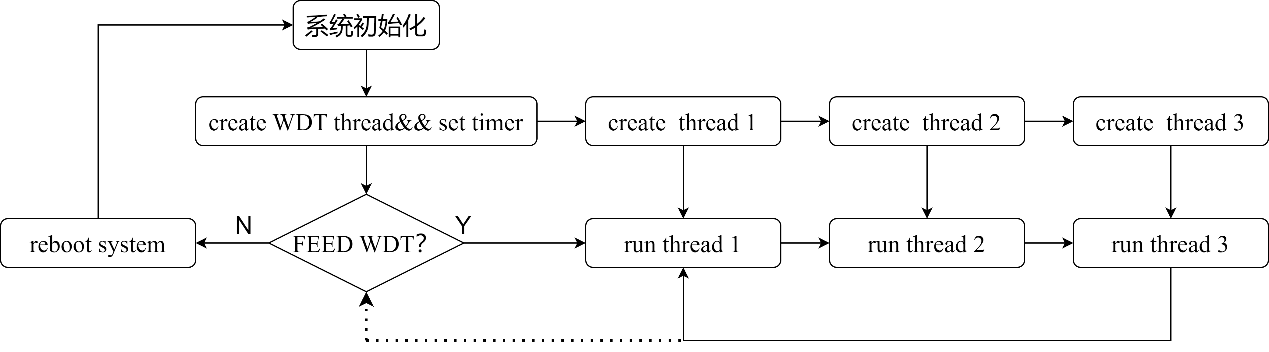
根据sensor的spec文档或者datasheet，查看I2C的读写逻辑，一般对于res：8，data：8来说



### 1.4 WDT

WTD的全称是看门狗定时器（watchdog timer），主要分为两类，一类是软件看门狗，一类是硬件看门狗。软件看门狗利用系统给出的定时器加以封装实现，硬件看门狗依赖于专门的硬件电路。

看门狗的主要作用为：确保整个系统正常运行，以及发生错误时的应对措施。



以上图为例，对于单核多线程系统来说，其实整个系统还是趋向于单片机模式，系统执行步骤为：

1）系统初始化，初始化系统级参数；

2）创建线程，本例中为创建看门狗线程并设置喂狗时间、创建线程1并运行、创建线程2并运行、创建线程3并运行，共创建4个线程，且多线程执行顺序为WDT、Thread1、Thread2、Thread3；

3）当系统分配的时间片（以50ms为例）被Thread3用完后，系统分配新的时间片给到WTD，并执行WatchDog Timer Feed动作，然后WDT重置内部“滴答数”，若50ms并用完，WDT线程自己放弃CPU，CPU分配时间片给到Thread1；

4）当发生一次异常时，例如在执行Thread1时整个系统崩溃了（比如内存泄漏或者物理资源耗尽），此时程序不能进行喂狗，当设置的喂狗时间达到之后，看门狗线程未检测到喂狗动作，“滴答数”耗尽，默认系统崩溃，此时执行reboot动作，重启整个系统。

注意，看门狗一般情况下是独立于整个系统之外。WDT对于保障整个系统的稳定性和自动恢复能力至关重要，尤其是复杂系统。

## 2. pipeline & 3A

### **2.1 RAW data**

//解决sensor自身硬件导致问题，如黑电平（Black Level Correction，BLC）、固定噪声（Fixed-Pattern Noise，FPN）、镜头阴影矫正（Lens Shield Correction，LSC）、镜头畸变矫正（Lens Distortion Correction，LDC）、坏点矫正（Defective Pixel Correction，DPC ）

2.1.1 黑电平（Black Level Correction，BLC）

2.1.2 坏点矫正（Defective Pixel Correction，DPC ）

2.1.3 绿通道平衡（Green Balance， GB）

2.1.4 镜头阴影矫正（Lens Shield Correction，LSC）

2.1.5 固定噪声（Fixed-Pattern Noise，FPN）

2.1.6 GAIN

2.1.7 3DNR/2DNR

2.1.8 WB

去马赛克Demosaic/CFA插值（Color Filter Array，CFA）

### 2.2 RGB data //色度域降噪 CGAMMA

2.2.1 高光恢复（，HLR）

2.2.2 CCM色彩矩阵

2.2.3 CGAMMA色度域降噪

RGB2YC（422）

### 2.3 YUV data //亮度域降噪 YGAMMA、YIE、CIE

2.3.1 亮度降噪YNR

2.3.2 锐化 APC

2.3.3 YGAMMA

2.3.4 YIE

2.3.5 CNR

2.3.6 PURPLE

2.3.7 CHROMA

2.3.8 CIE

2.3.9 LDC

2.3.10 SCALER

## 3. sensor bright up and debug

### 3.1 sensor bright up

1. 与客户沟通需求，向sensor FAE申请配置邮件，主要包括：

LINEAR or WDR：linear

Bit depth：10/12

CIS\_CLK：24M/27M

HTS \* VTS：？？ \* 1125

Active\_W \* Active\_H：1920 \* 1080

Frame rate：25/30

2. sensor接入，新建对应sensor的文件夹，将邮件中的配置写入其中，将AE相关函数写为固定曝光，看是否正常出图。

### 3.2 debug

1. 检查setting及硬件是否存在问题：度信抓RAW

2. 检查IIC及配置时钟是否有误，device\_id，mode(addr\_16\_data\_8，addr\_8\_data\_8)

3. 前两步后sensor输出就应该是正常的，通过示波器确定输出波形是否正常

4. ISP状态寄存器查看：active\_pixel\_per\_line，total\_active\_pixel\_per\_frame

比如检测到sensor输出的active\_w数据多一个或者少一个，vi对sensor的数据上升沿采样还是下降沿采样导致

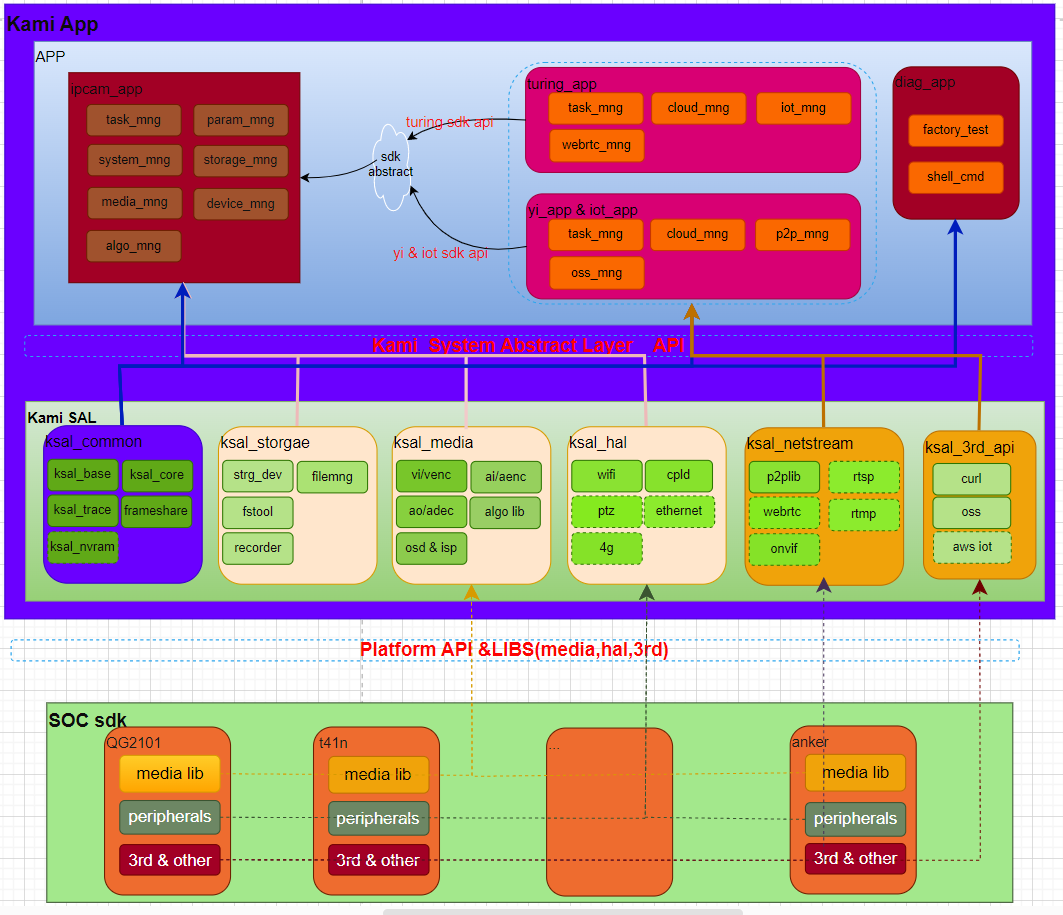
5. 通过示波器发现sensor有输出，isp这边的vi发现没有检测到输入，mipi时序

### 3.3 D-PHY & MIPI

### 4. coding & Tuning

# KAMI/Firmware Engineer

kami的turing coding组织架构由三块组成，分别是平台层（芯片的SDK），抽象层（类似于芯片的driver层），应用层（上层应用代码）。对于应用层来说，主要是两个大的APP，ipcam\_app和turing\_app。ipcam\_app主要包含一个总线程（task\_mng）和六个子线程（param\_mng、system\_mng、storage\_mng、media\_mng、device\_mng、algo\_mng），turing\_app主要包含一个总线程（task\_mng）和三个子线程（cloud\_mng、iot\_mng、webrtc\_mng）。下面将分别介绍。



## 1. APP

### 1.1 ipcam\_app

#### 1.1.1 task\_mng

task\_mng的主要作用：

1. 线程激活与终止；

2. WDT的喂狗。

|  |
| --- |
| void TaskMng\_Initialization(void)  {  const KSAL\_Task\_Attr\_S \*thread\_attributes;  KSAL\_Config\_S sal\_configuration;  bool\_t status\_ok;  KSAL\_Task\_Id\_T thread\_id;  Ipcam\_Get\_KSAL\_Config(&sal\_configuration);    KSAL\_Set\_Message\_Fault\_Callback(App\_Msg\_Fault\_Callback);  status\_ok = KSAL\_Init(&sal\_configuration);  KSAL\_Ensure(status\_ok, "KSAL\_Init() failed.");  thread\_attributes = KSAL\_Get\_Task\_Attr(TASKMNG\_TASK\_ID);    thread\_id = KSAL\_Create\_Task(TaskM\_Task, NULL, thread\_attributes);  KSAL\_Ensure((thread\_attributes->id == thread\_id), "KSAL\_Create\_Task() failed: %d", (int) thread\_id);  } |

|  |
| --- |
| void TaskM\_Task(void \*arg)  {  const KSAL\_Message\_T \*message;  size\_t num\_bytes;  bool\_t status\_ok;  const KSAL\_Event\_Id\_T subscribe\_list[] =  {  /\* Note: If new published messages are added to the list, then you \*might\*  \* need to update the KSAL\_Create\_Queue statement a little farther down.  \* Currently, APM\_EVG\_START\_STOP is the largest message. \*/  //SOH\_EVG\_STATUS\_QUERY,  };  KSAL\_Timer\_Id\_T wdtT;  KSAL\_Create\_Timer(TASKM\_EV\_FEED\_WDT\_CMD, &wdtT);  Ipcam\_Get\_App\_Configuration(&taskM\_Config);    KSAL\_Ensure(taskM\_Config != NULL, "NULL App Configuration");  KSAL\_Ensure(taskM\_Config->num\_tasks > 0, "No applications declared");  num\_bytes = BIT\_ARRAY\_NUM\_BYTES(taskM\_Config->num\_tasks);    TaskM\_Activated\_Tasks = KSAL\_Alloc(num\_bytes);  KSAL\_Ensure(TaskM\_Activated\_Tasks != NULL, "Could not allocate memory to manage applications");  memset(TaskM\_Activated\_Tasks, 0, num\_bytes);    TaskM\_Terminated\_Tasks = KSAL\_Alloc(num\_bytes);  KSAL\_Ensure(TaskM\_Terminated\_Tasks != NULL, "Could not allocate memory to manage applications");  memset(TaskM\_Terminated\_Tasks, 0, num\_bytes);    TaskM\_Activate\_Called = KSAL\_Alloc(num\_bytes);  KSAL\_Ensure(TaskM\_Activate\_Called != NULL, "Could not allocate memory to manage applications");  memset(TaskM\_Activate\_Called, 0, num\_bytes);  TaskM\_Terminate\_Called = KSAL\_Alloc(num\_bytes);  KSAL\_Ensure(TaskM\_Terminate\_Called != NULL, "Could not allocate memory to manage applications");  memset(TaskM\_Terminate\_Called, 0, num\_bytes);  KSAL\_Subscribe(subscribe\_list, Num\_Elems(subscribe\_list));  KSAL\_Signal\_Ready();  KSAL\_Start\_Timer(wdtT, WDT\_FEED\_PERIOD\_S\*1000, true);  TaskMng\_WDT\_Ctrl\_Init();  APM\_Ready\_Status\_Target = TASKM\_TASK\_START\_READY;  Set\_Task\_Ready\_Status(TASKM\_TASK\_START\_READY,0);  status\_ok = true;  while (status\_ok)  {  message = KSAL\_Receive\_Timeout(TASKM\_KSAL\_RECEIVE\_TIMEOUT\_MS);  if (NULL != message)  {  switch (message->event\_id)  {  case TASKM\_EV\_POWEROFF\_CMD:  APP\_SHUTDOWN\_TYPE\_E shut\_type = \*((APP\_SHUTDOWN\_TYPE\_E \*)(message->data));  Kami\_Info("TASKM\_EV\_POWEROFF\_CMD ---%d\n",shut\_type);  APM\_Ready\_Status\_Target = TASKM\_TASK\_STOP\_READY;  Set\_Task\_Ready\_Status(TASKM\_TASK\_STOP\_READY, 0);  break;  case TASKM\_EV\_START\_CMD:  if(APM\_Ready\_Status\_Target == TASKM\_TASK\_START\_READY)  {  if (TaskM\_Activate\_Apps())  Set\_Task\_Ready\_Status(TASKM\_TASK\_ALL\_READY, 0);  else  Set\_Task\_Ready\_Status(TASKM\_TASK\_START\_READY, 0);  }  break;  case TASKM\_EV\_STOP\_CMD:  if(0) TaskM\_Dump\_App\_Status();  if (TaskM\_Terminate\_Apps())  {  Set\_Task\_Ready\_Status(TASKM\_TASK\_ALL\_STOP, 100);  status\_ok = false;  }  else  Set\_Task\_Ready\_Status(TASKM\_TASK\_STOP\_READY, 100);  break;  case TASKM\_EV\_ACTIVATED:  TaskM\_Handle\_Activated(\*((APM\_EV\_ACTIVATED\_T \*) message->data));  break;  case TASKM\_EV\_TERMINATED:  TaskM\_Handle\_Terminated(\*((APM\_EV\_TERMINATED\_T \*) message->data));  break;  case TASKM\_EV\_HEART\_BEART\_CMD:  kvs\_app\_alive = 8;  break;  case TASKM\_EV\_ALIVE\_MONITOR\_CMD:  kvs\_app\_alive--;  if(kvs\_app\_alive < 0)  {  Kami\_Fault("kvs\_app not exist\n");  Set\_Task\_Ready\_Status(TASKM\_TASK\_STOP\_READY, 0);  kvs\_app\_alive = 8;  }  break;  case TASKM\_EV\_FEED\_WDT\_CMD:  TaskMng\_WDT\_Ctrl\_Feed\_WDT();  break;  case TASKM\_EV\_CONTROL\_WDT\_CMD:  int \* arry = ((int\*)message->data);  int op\_type = arry[0];//op\_type: 0: enable wdt 1: disable wdt 2: stop feeding wdt  int timeout\_s = arry[1];  Kami\_Info("op\_type:%d, timeout:%d\n", op\_type, timeout\_s);  if(0 == op\_type || 1 == op\_type)  {  if (OK == TaskMng\_WDT\_Ctrl\_Set\_WDT(op\_type,timeout\_s))  Kami\_Info("set wdt successfully, op\_type:%d,timeout\_s:%d\n", op\_type, timeout\_s);  else  Kami\_Info("set wdt failed, op\_type:%d,timeout\_s:%d\n", op\_type, timeout\_s);  }  else if(2 == op\_type)  {  Kami\_Info("stop feeding wdt\n");  KSAL\_Stop\_Timer(wdtT);  }  break;  default:  Kami\_Warn("Unexpected message received: %d\n", (int) message->event\_id);  break;  }  }  }  KSAL\_Exit(0);  } |

#### 1.1.2 param\_mng

参数管理线程（param\_mng）主要负责参数获取和设置，包括media\_media，wifi\_device，falldetect\_algo，主要处理以下事件：  
1）PARAMM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）PARAMM\_EV\_MEDIA\_CONF\_GET\_CMD 获取媒体配置，从NVRAM获取；

3）PARAMM\_EV\_PERIPH\_CONF\_GET\_CMD 获取外设配置，从NVRAM获取；

4）PARAMM\_EV\_BOARD\_CONF\_GET\_CMD 获取板级配置，从NVRAM获取；

5）PARAMM\_EV\_WIFI\_CONF\_GET\_CMD 获取wifi配置，从NVRAM获取；

6）PARAMM\_EV\_WIFI\_CONF\_SET\_CMD 设置wifi配置，通过NVRAM设置；

7）PARAMM\_EV\_COMMON\_CONF\_GET\_CMD 获取常规配置，从NVRAM获取；

8）PARAMM\_EV\_COMMON\_CONF\_SET\_CMD 设置常规配置，通过NVRAM设置；

9）PARAMM\_EV\_SAFEZONE\_SET\_CMD 设置安全区域，通过NVRAM设置；

10）PARAMM\_EV\_FALLZONE\_SET\_CMD 设置摔倒区域，通过NVRAM设置；

11）DEVM\_EV\_DEV\_RESET\_NOTIFY 设备reset通知，恢复NVRAM；

|  |
| --- |
| static bool\_t Param\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case PARAMM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" FLASH\_DATA\_EV\_THREAD\_SHUTDOWN received\n");  thread\_alive = false;  break;  case PARAMM\_EV\_MEDIA\_CONF\_GET\_CMD:  Media\_Cfg\_S media\_cfg = {0};  ksal\_nvram\_mediacfg\_get(&media\_cfg);  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id,  PARAMM\_EV\_MEDIA\_CONF\_GET\_ACK, &media\_cfg,sizeof(Media\_Cfg\_S));  break;  case PARAMM\_EV\_PERIPH\_CONF\_GET\_CMD:  KSal\_Peripheral\_Cfg\_S peri\_cfg = {0};  ksal\_nvram\_periphcfg\_get(&peri\_cfg);  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id,  PARAMM\_EV\_PERIPH\_CONF\_GET\_ACK, &peri\_cfg,sizeof(KSal\_Peripheral\_Cfg\_S));  break;  case PARAMM\_EV\_BOARD\_CONF\_GET\_CMD:  KSal\_Board\_Cfg\_S board\_cfg = {0};  ksal\_nvram\_boardcfg\_get(&board\_cfg);  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id,  PARAMM\_EV\_BOARD\_CONF\_GET\_ACK, &board\_cfg,sizeof(KSal\_Board\_Cfg\_S));  break;  case PARAMM\_EV\_WIFI\_CONF\_GET\_CMD:  KSal\_Wifi\_Config\_S wifi\_info = {0};  ksal\_nvram\_wificfg\_get(&wifi\_info);  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id,  PARAMM\_EV\_WIFI\_CONF\_GET\_ACK, &wifi\_info,sizeof(KSal\_Wifi\_Config\_S));  break;  case PARAMM\_EV\_WIFI\_CONF\_SET\_CMD:  KSal\_Wifi\_Config\_S \*wifi\_cfg = (KSal\_Wifi\_Config\_S \*)msg->data;  if( ksal\_nvram\_wificfg\_set(wifi\_cfg) == OK)  param\_context.need\_save = 1;  break;    case PARAMM\_EV\_COMMON\_CONF\_GET\_CMD:  int idx = \*((int \*)msg->data);  param\_ev\_common\_conf\_s com\_item;  com\_item.com\_idx = idx;    param\_comm\_get(&com\_item);  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id,  PARAMM\_EV\_COMM\_CONF\_GET\_ACK, &com\_item.value,sizeof(com\_item.value));  break;  case PARAMM\_EV\_COMMON\_CONF\_SET\_CMD:  param\_ev\_common\_conf\_s \*com\_ev = (param\_ev\_common\_conf\_s \*)msg->data;  param\_comm\_set(com\_ev);  break;  case PARAMM\_EV\_SAFEZONE\_SET\_CMD:  KSal\_Algo\_Zone\_Setting\_S\* zone\_cfg = (KSal\_Algo\_Zone\_Setting\_S\*) msg->data;  if(ksal\_nvram\_algoSafeZone\_set(zone\_cfg) == OK)  {  KSAL\_Publish\_Notify(PARAMM\_EV\_SAFEZONE\_CHANGE\_NOTIFY, zone\_cfg, sizeof(KSal\_Algo\_Zone\_Setting\_S));  param\_context.need\_save = 1;  }  break;  case PARAMM\_EV\_FALLZONE\_SET\_CMD:  KSal\_Algo\_Zone\_Setting\_S\* zone\_cfg = (KSal\_Algo\_Zone\_Setting\_S\*) msg->data;  if(ksal\_nvram\_algoFallZone\_set(zone\_cfg) == OK)  {  KSAL\_Publish\_Notify(PARAMM\_EV\_FALLZONE\_CHANGE\_NOTIFY, zone\_cfg, sizeof(KSal\_Algo\_Zone\_Setting\_S));  param\_context.need\_save = 1;  }  break;  case DEVM\_EV\_DEV\_RESET\_NOTIFY:  Kami\_Info("DEVM\_EV\_DEV\_RESET\_NOTIFY : %d\n",DEVM\_EV\_DEV\_RESET\_NOTIFY);  ksal\_nvram\_conf\_restore();  param\_context.need\_save = 0;  App\_TaskMng\_PowerOff\_Callout(SHUTDOWN\_BY\_RESTORE\_FACTORY);  break;  case PARAMM\_EV\_SYS\_SET\_CMD:  default:  break;  } /\* switch (msg->event\_id) \*/  }  else  {  if(param\_context.need\_save == 1)  {  ksal\_nvram\_conf\_save();  param\_context.need\_save = 0;  }  }  return(thread\_alive);  } |

#### 1.1.3 system\_mng

系统管理线程（system\_mng）负责OTA升级相关，主要处理以下事件：

1）SYSTEMMNG\_EV\_SHUTDOWN\_CMD终止自身线程，通过跳出while(1)结束；

2）SYSTEMMNG\_EV\_OTA\_STATTUS\_NOTIFY OTA状态通知，在OTA期间下载文件、下载失败、下载完成升级等音频播放；

|  |
| --- |
| static bool SystemMng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_fast thread\_alive = true;  bool ssm\_process = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case SYSTEMMNG\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" SYSTEMMNG\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case SYSTEMMNG\_EV\_OTA\_STATTUS\_NOTIFY:  int otaStatus =\*((int \*)msg->data);  if(otaStatus < 0)  App\_MediaMng\_PlayAudio\_callout(AUDIO\_FILE\_ID\_DOWLOAD\_FAIL);  else if( otaStatus == 1)  App\_MediaMng\_PlayAudio\_callout(AUDIO\_FILE\_ID\_DOWLOADING);  else if(otaStatus == 2)  {  App\_ParamMng\_CommConfigSet\_Callout(COMM\_PARAM\_ota\_status, 1);  App\_MediaMng\_PlayAudio\_callout(AUDIO\_FILE\_ID\_UPDATING);  }  ssm\_process = false;  break;  case STRGM\_EV\_DIAG\_FLAG\_NOTIFY:  default:  break;  } /\* switch (msg->event\_id) \*/    if(ssm\_process)  System\_State\_Handle(msg->event\_id,msg->data);  }  return(thread\_alive);  } |

#### 1.1.4 storage\_mng

存储管理线程（storage\_mng）主要负责对于SD卡状态检测、录像以及web端查看相关视频，主要处理以下事件：

1）STRGM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）STRGM\_EV\_SD\_STATE\_CMD SD卡状态切换，挂载状态、需要格式化等

3）STRGM\_EV\_SD\_FORMAT\_CMD 对需要格式化的SD卡进行格式化

4）SYSTEMMNG\_EV\_OTA\_STATTUS\_NOTIFY OTA状态通知，停止录像

5）STRGM\_EV\_SD\_STATUS\_NOTIFY SD卡状态通知，录像等相关事件

|  |
| --- |
| static bool\_t Strg\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case STRGM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" STRGM\_EV\_SHUTDOWN\_CMD received\n");  if(strgm\_context.sd\_status == STRGM\_SD\_STATUS\_FSOK)  {  //recorder\_stop();  }  thread\_alive = false;  break;  case STRGM\_EV\_SD\_STATE\_CMD:  STRG\_CARD\_STATE\_E\* sd\_state = ((STRG\_CARD\_STATE\_E \*)msg->data);  strgm\_sd\_state\_handle(\*sd\_state);  break;  case STRGM\_EV\_SD\_FORMAT\_CMD:  Kami\_Info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*STRGM\_EV\_SD\_FORMAT\_CMD received\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  int ret = SD\_Card\_Format\_Process();  if(ret == FAIL)  Kami\_Info("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SD Card Format fail\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  break;  case SYSTEMMNG\_EV\_OTA\_STATTUS\_NOTIFY:  stgm\_record\_stop(0);  break;  case STRGM\_EV\_SD\_STATUS\_NOTIFY:  if(strgm\_context.sd\_status == STRGM\_SD\_STATUS\_FSOK)  {  #if App\_Cap\_Is\_On(APP\_RECORDER\_CAP)  char RecordMode[10];  if(xaccess(SD\_CONF\_FILE\_PATH, F\_OK) == OK)  {  if(GetIniKeyString("App","RecordMode", SD\_CONF\_FILE\_PATH, RecordMode) == OK)  Kami\_Info("file %s found RecordMode, set RecordMode is %s \n", SD\_CONF\_FILE\_PATH, RecordMode);  else  Kami\_Warn("file %s not found RecordMode \n", SD\_CONF\_FILE\_PATH);  }  else  Kami\_Error("Check file %s does not exist\n ", SD\_CONF\_FILE\_PATH);  #endif  strgm\_mode\_check();  //recorder\_start();  if(xaccess("/tmp/sd/identifydevice", F\_OK) == 0)  {  char did[32],key[32],both[64];    ksal\_nvram\_did\_get(did);  ksal\_nvram\_keyId\_get(key);  ksal\_write\_file("/tmp/sd/did.txt", (uint8\_t\*)did, Strlen(did));  ksal\_write\_file("/tmp/sd/key.txt", (uint8\_t\*)key, Strlen(key));  ksal\_snprintf(both,64,"%s,%s\n",did,key);  ksal\_write\_file\_append("/tmp/sd/keycollectfile.txt",(uint8\_t\*)both,Strlen(both));  }  #if App\_Cap\_Is\_On(APP\_RECORDER\_CAP)  if(strgm\_context.record\_mode == 1 || xaccess("/tmp/sd/recorder\_enable",F\_OK) == 0)  {  filemng\_space\_monitor\_start();  stgm\_record\_start();  }  #endif  }  else  {  #if App\_Cap\_Is\_On(APP\_RECORDER\_CAP)  if(strgm\_context.record\_mode == 1 || xaccess("/tmp/sd/recorder\_enable",F\_OK) == 0)  {  filemng\_space\_monitor\_stop();  stgm\_record\_stop(-1);  }  #endif  }  break;  default:  break;  } /\* switch (msg->event\_id) \*/  }  return(thread\_alive);  } |

#### 1.1.5 media\_mng

媒体管理线程（media\_mng）主要负责音视频相关，其中包括H264主、子码流编码，抓图，flip/mirror，日夜两种状态互相切换，播放对应音频，主要处理以下事件：

1）MEDIAM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）MEDIAM\_EV\_SNAP\_JPEG\_CMD web端抓图；

3）MEDIAM\_EV\_ISP\_SWITCH\_CMD ISP的color和gray相互转换；

4）PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY 常规配置修改通知，用于翻转flip/mirror；

5）MEDIAM\_EV\_BOX\_OSD\_UPDATE\_CMD 更新主码流和子码流；

6）MEDIAM\_EV\_AUDIO\_PLAY\_CMD 音频播放；

7）SYSTEMMNG\_EV\_SYS\_STATE\_NOTIFY 系统状态通知，音、视频（主码流、子码流）编码；

8）DEVM\_EV\_NIGHT\_VISION\_STATUS\_NOTIFY 设备夜视状态通知，对于夜视媒体数据进行编码；

|  |
| --- |
| static bool\_t Media\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  int ret = 0 ;  if(msg != NULL)  {  switch (msg->event\_id)  {  case MEDIAM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" MEDIAM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;    case MEDIAM\_EV\_SNAP\_JPEG\_CMD:  App\_Media\_SnapJpg\_S \*snapCmd = (App\_Media\_SnapJpg\_S \*)msg->data;  Kami\_Info("Snap %s \n",snapCmd->jpgName);  ret = media\_venc\_jpeg(snapCmd->jpgName);  Kami\_Info("Snap %s ret = %d\n",snapCmd->jpgName,ret);  if(snapCmd->sync == 1)  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id, MEDIAM\_EV\_SNAP\_JPEG\_ACK, &ret, sizeof(ret));  break;  case MEDIAM\_EV\_ISP\_SWITCH\_CMD:  int isGraySwitch = \*((int\*)msg->data);  media\_isp\_switch\_config(isGraySwitch);  break;  case PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY:  param\_ev\_common\_conf\_s \*com\_ev = (param\_ev\_common\_conf\_s \*)msg->data;  if(com\_ev->com\_idx == COMM\_PARAM\_sensor\_invert)  {  Kami\_Info("COMM\_PARAM\_sensor\_invert: %d\n",com\_ev->value);  if(com\_ev->value == 1)  media\_vi\_set\_mirror\_flip(true,true);  else  media\_vi\_set\_mirror\_flip(false,false);  }  break;  case MEDIAM\_EV\_BOX\_OSD\_UPDATE\_CMD:  Media\_Venc\_OSD\_Box\_S \*osd\_box\_info = (Media\_Venc\_OSD\_Box\_S \*)(msg->data);  if(Media\_Cap\_Is\_On(MEDIA\_VIDEO\_LIVE\_USE\_MAIN))  media\_venc\_osd\_box\_update(H264\_MAIN\_STREAM,osd\_box\_info);  else  media\_venc\_osd\_box\_update(H264\_SUB\_STREAM,osd\_box\_info);  break;  case MEDIAM\_EV\_AUDIO\_PLAY\_CMD:  MEDIAM\_AUDIO\_PLAY\_ID\_E audio\_idx = \*((MEDIAM\_AUDIO\_PLAY\_ID\_E \*)msg->data);  audio\_play\_file( audio\_idx);  break;  case SYSTEMMNG\_EV\_SYS\_STATE\_NOTIFY:  SystemMng\_Ev\_SysState\_Notify\_E sysState = \*((SystemMng\_Ev\_SysState\_Notify\_E\*)msg->data);  if(sysState == SYS\_STATE\_NORMAL || sysState == SYS\_STATE\_PRIVACY)  mediam\_bussiness\_start(&mediam\_contex.media\_cfg);  break;  case DEVM\_EV\_NIGHT\_VISION\_STATUS\_NOTIFY:  int isGraySwitch = \*((int\*)msg->data);  media\_venc\_day\_night\_switch(isGraySwitch);  Kami\_Info("DEVM\_EV\_NIGHT\_VISION\_STATUS\_NOTIFY : %d\n",isGraySwitch);  break;  case DEVM\_EV\_NET\_STATUS\_NOTIFY:  default:  break;  }  }  return(thread\_alive);  } |

#### 1.1.6 device\_mng

设备管理线程（device\_mng）主要负责硬件相关，包括network（wifi），ptz（步进电机），led（状态），来自消息的重启系统，主要处理以下事件：

1）DEVM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）DEVM\_EV\_WIFI\_START\_CMD 启动wifi命令，；

3）DEVM\_EV\_PTZ\_CTL\_CMD ISP ptz控制命令，控制步进电机转动方向和等级；

4）DEVM\_EV\_LED\_STATUS\_SET\_CMD 设置led状态命令，这里是灯亮颜色；

5）DEVM\_EV\_RESET\_BY\_MSG\_CMD 消息重启命令；

6）DEVM\_EV\_DAY\_NIGHT\_CMD 系统状态通知，私有模式下通过PTZ转动；

7）PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY 常规配置改变通知；

|  |
| --- |
| static bool\_t Device\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case DEVM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" STRGM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case DEVM\_EV\_WIFI\_START\_CMD:  DevMng\_WIFI\_Start\_S \*wifi\_conf = (DevMng\_WIFI\_Start\_S \*)msg->data;  netctrl\_wifi\_conf(wifi\_conf->ssid,wifi\_conf->pwd);  break;  case DEVM\_EV\_PTZ\_CTL\_CMD:  #if Hal\_Cap\_Is\_On(HAL\_PTZ\_CAP)  DevMng\_Ptz\_Ctrl\_S \*ptzCtrlInfo = (DevMng\_Ptz\_Ctrl\_S \*)msg->data;  App\_DevMng\_Ptz\_DirMove(ptzCtrlInfo->dir,ptzCtrlInfo->move\_percent);  #endif  break;  case DEVM\_EV\_LED\_STATUS\_SET\_CMD:  LED\_STATUS\_E \*pStatus = (LED\_STATUS\_E\*) msg->data;  DeviceMng\_Led\_Set\_Status(\*pStatus);  break;  case DEVM\_EV\_RESET\_BY\_MSG\_CMD:  Kami\_Info("reset device my msg\n");  DeviceMng\_Reset\_Ctrl\_Reset\_By\_Msg();  break;  case DEVM\_EV\_DAY\_NIGHT\_CMD:  int \*day\_night\_mode = (int\*) msg->data;  DeviceMng\_Day\_Night\_ModeSet(\*day\_night\_mode);  break;  case SYSTEMMNG\_EV\_SYS\_STATE\_NOTIFY:  #if PRIVACY\_MODE  SystemMng\_Ev\_SysState\_Notify\_E sysState = \*((SystemMng\_Ev\_SysState\_Notify\_E\*)msg->data);  if(sysState == SYS\_STATE\_PRIVACY || sysState == SYS\_STATE\_NORMAL)  {  int x, y;  if(PTZ\_DEF\_TYPE == PTZ\_TYPE\_AONI)  x = 50, y = 0;  else  x = 50, y = 100;    App\_ParamMng\_CommConfigGet\_Callout(COMM\_PARAM\_ptz\_x\_pos, &x);  if(sysState == SYS\_STATE\_NORMAL)  {  App\_ParamMng\_CommConfigGet\_Callout(COMM\_PARAM\_ptz\_y\_pos, &y);  nvctrl\_set\_power\_mode(POWER\_MODE\_ON\_E);  }  else  nvctrl\_set\_power\_mode(POWER\_MODE\_OFF\_E);  App\_DevMng\_Ptz\_MoveToXY(x,y);  Kami\_Info("------sysState---=----%d--------\n",sysState);  Kami\_Info("--App\_DevMng\_Ptz\_MoveToXY(x,y)x---=---%d-App\_DevMng\_Ptz\_MoveToXY(x,y)y---=----%d--------\n",x, y);  }  #endif  break;  case PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY:  param\_ev\_common\_conf\_s \*com\_ev = (param\_ev\_common\_conf\_s \*)msg->data;  if(com\_ev->com\_idx == COMM\_PARAM\_start\_reset)  netctrl\_set\_device\_mode(com\_ev->value);  break;  default:  break;  } /\* switch (msg->event\_id) \*/  }  return(thread\_alive);  } |

#### 1.1.7 algo\_mng

算法管理线程（algo\_mng）主要是算法相关，其中包括falldetect、bedexit、bedexit的sensitivity、避免频繁报警的cooltime时间，两个算法的使能与否，使能之后区域参数变化等，用于和web端的变化进行交互。处理以下事件：

1）AIM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）AIM\_EV\_ALERT\_COOLTIME\_UPDATE\_CMD 冷却时间更新警告通知命令；

3）PARAMM\_EV\_SAFEZONE\_CHANGE\_NOTIFY 安全区域变化通知；

4）PARAMM\_EV\_FALLZONE\_CHANGE\_NOTIFY 摔倒区域变化通知；

5）PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY 常规配置变化通知，调整falldetect，bedexit以及bedexit的敏感度等级；

6）SYSTEMMNG\_EV\_SYS\_STATE\_NOTIFY 系统状态通知，falldetect和bedexit的启动；

7）DEVM\_EV\_NIGHT\_VISION\_STATUS\_NOTIFY 设备夜视状态通知；

|  |
| --- |
| static bool\_t AI\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case AIM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" STRGM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case AIM\_EV\_ALERT\_COOLTIME\_UPDATE\_CMD:  uint32\_t time = \*((uint32\_t \*)(msg->data));  //aimng\_context.cooldown\_time = time;  aimng\_context.fall\_cooldown\_timeout = ksal\_get\_cur\_time\_sec() + time;  break;  case PARAMM\_EV\_SAFEZONE\_CHANGE\_NOTIFY:  Kami\_Info(" PARAMM\_EV\_SAFEZONE\_CHANGE\_NOTIFY received\n");  #if AI\_Cap\_Is\_On(AI\_ENTRYEXIT\_DETECT\_CAP)  KSal\_Algo\_Zone\_Setting\_S\* safe\_zone = (KSal\_Algo\_Zone\_Setting\_S\*)msg->data;  ksal\_algo\_safeZone\_set(safe\_zone);  #endif  break;  case PARAMM\_EV\_FALLZONE\_CHANGE\_NOTIFY:  Kami\_Info(" PARAMM\_EV\_FALLZONE\_CHANGE\_NOTIFY received\n");  KSal\_Algo\_Zone\_Setting\_S\* fall\_zone = (KSal\_Algo\_Zone\_Setting\_S\*)msg->data;  #if AI\_Cap\_Is\_On(AI\_FALL\_DETECT\_CAP)  ksal\_algo\_fallZone\_set(fall\_zone);  #endif  break;  case PARAMM\_EV\_COMMON\_CONF\_CHANGE\_NOTIFY:  param\_ev\_common\_conf\_s \*com\_ev = (param\_ev\_common\_conf\_s \*)msg->data;  if(com\_ev->com\_idx == COMM\_PARAM\_fallDetect\_enable)  {  aimng\_context.fall\_det\_enable = com\_ev->value;  ksal\_algo\_enable(KSAL\_ALGO\_TYPE\_FALL\_DETECT,com\_ev->value);  }  else if(com\_ev->com\_idx == COMM\_PARAM\_bedExitDetect\_enable)  {  Kami\_Info(" KSAL\_ALGO\_TYPE\_ENTRYEXIT\_DETECT received\n");  aimng\_context.bed\_exit\_enable = com\_ev->value;  ksal\_algo\_enable(KSAL\_ALGO\_TYPE\_ENTRYEXIT\_DETECT,com\_ev->value);  }  else if(com\_ev->com\_idx == COMM\_PARAM\_bedExitDetect\_sens)  {  Kami\_Info(" COMM\_PARAM\_bedExitDetect\_sens set %d\n",com\_ev->value);  Pilot\_BedExitSens\_Setting(com\_ev->value);  }  break;  case SYSTEMMNG\_EV\_SYS\_STATE\_NOTIFY:  SystemMng\_Ev\_SysState\_Notify\_E sysState = \*((SystemMng\_Ev\_SysState\_Notify\_E\*)msg->data);  if(sysState == SYS\_STATE\_NORMAL)  {  #if AI\_Cap\_Is\_On(AI\_FALL\_DETECT\_CAP)  App\_ParamMng\_CommConfigGet\_Callout(COMM\_PARAM\_fallDetect\_enable,&aimng\_context.fall\_det\_enable);  Kami\_Info("aimng\_context.fall\_det\_enable = %d\n",aimng\_context.fall\_det\_enable );  if(aimng\_context.fall\_det\_enable == 1)  ksal\_algo\_enable(KSAL\_ALGO\_TYPE\_FALL\_DETECT,1);  #endif  #if AI\_Cap\_Is\_On(AI\_ENTRYEXIT\_DETECT\_CAP)  App\_ParamMng\_CommConfigGet\_Callout(COMM\_PARAM\_bedExitDetect\_enable,&aimng\_context.bed\_exit\_enable);  if(aimng\_context.bed\_exit\_enable == 1)  ksal\_algo\_enable(KSAL\_ALGO\_TYPE\_ENTRYEXIT\_DETECT,1);  #endif  }  break;  case DEVM\_EV\_NIGHT\_VISION\_STATUS\_NOTIFY:  aimng\_context.night\_mode = \*((int \*)(msg->data));  break;  default:  break;  } /\* switch (msg->event\_id) \*/  }    return(thread\_alive);  } |

### 1.2 turing\_app

#### 1.2.1 cloud\_mng

算法管理线程（algo\_mng）主要负责绝大多数web端事件，包括账号登录、上线、绑定、log、时间同步、上传jpg、警报、视频验证、上传、刷新token、kvs token、ota、处理以下事件：

1）CLOUDM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）CLOUDM\_EV\_TOKEN\_REFRESH\_TIMER\_CMD 云刷新token定时器

3）CLOUDM\_EV\_SYNC\_TIME\_CMD云同步时间命令；

4）CLOUDM\_EV\_DO\_BIND\_CMD 云绑定命令；

5）CLOUDM\_EV\_DO\_ONLINE\_CMD 云上线命令；

6）CLOUDM\_EV\_KVS\_TOKEN\_REFRESH\_CMD 云刷新kvs token命令；

7）CLOUDM\_EV\_JPG\_UPLOAD\_CMD 云上传jpg命令；

8）CLOUDM\_EV\_ALERT\_EVENT\_CMD 云警报事件命令；

9）CLOUDM\_EV\_VIDEO\_VERIFY\_CMD 云视频验证命令；

10）CLOUDM\_EV\_VIDEO\_UPLOAD\_CMD 云视频上传命令；

11）CLOUDM\_EV\_DO\_OTA\_CMD 云OTA命令；

12）CLOUDM\_EV\_LOG\_ON\_CMD 云登录命令；

13）CLOUDM\_EV\_LOG\_TIMER\_CMD 云日志定时器命令；

|  |
| --- |
| static bool\_t Cloud\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case CLOUDM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" STRGM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case CLOUDM\_EV\_TOKEN\_REFRESH\_TIMER\_CMD:  int errorCode = CloudRefreshToken();  Kami\_Info("############CloudRefreshToken() errorCode = [%d]############\n", errorCode);  if( errorCode != OK )  reAuthOnRefreshTokenExpire();  break;  case CLOUDM\_EV\_SYNC\_TIME\_CMD:  Kami\_Info("CLOUDM\_EV\_SYNC\_TIME\_CMD rcv\n");  Cloud\_Do\_SyncTime();  break;  case CLOUDM\_EV\_DO\_BIND\_CMD:  Kami\_Info("CLOUDM\_EV\_DO\_BIND\_CMD rcv\n");  if(cloudm\_context.do\_onlining == 0)  {  cloudm\_context.do\_onlining = 1;  Cloud\_Do\_Bind();  }  break;  case CLOUDM\_EV\_DO\_ONLINE\_CMD:  Kami\_Info("CLOUDM\_EV\_DO\_ONLINE\_CMD rcv\n");  if(cloudm\_context.do\_onlining == 0)  {  cloudm\_context.do\_onlining = 1;  Cloud\_Do\_Online();  }  break;  case CLOUDM\_EV\_KVS\_TOKEN\_REFRESH\_CMD:  int ret = CloudFetchKvsAwsCliCredentials();  KSAL\_Send(msg->sender\_app\_id, msg->sender\_thread\_id, CLOUDM\_EV\_KVS\_TOKEN\_REFRESH\_ACK,&ret,sizeof(ret));  break;  case CLOUDM\_EV\_JPG\_UPLOAD\_CMD:  CloudMng\_UploadJpeg\_S \*uploadInfo = (CloudMng\_UploadJpeg\_S \*)(msg->data);  ksal\_memcpy(&cloudm\_context.uploadJpgInfo, uploadInfo, sizeof(CloudMng\_UploadJpeg\_S));  xremove(IMAGE\_SNAP\_FILENAME);  turing\_sdk\_iot\_snapJpg(IMAGE\_SNAP\_FILENAME, Strlen(IMAGE\_SNAP\_FILENAME) + 1);  ksal\_sleep(1000);  //App\_MediaMng\_SnapJpeg\_Callout(IMAGE\_SNAP\_FILENAME);  KSAL\_Schedule\_Task(Cloud\_UploadJpeg\_Tasklet,&cloudm\_context.uploadJpgInfo,0);  break;  case CLOUDM\_EV\_ALERT\_EVENT\_CMD:  Algo\_Alert\_Event\_Info\_S \*fall\_ev = (Algo\_Alert\_Event\_Info\_S \*)msg->data;  int ev\_id = CLoud\_Get\_FreeEventId();  if(ev\_id < MAX\_PUSH\_EVENT\_NUM )  {  ksal\_memset(&cloudm\_context.pushEvt[ev\_id],0,sizeof(Cloud\_PushEvent\_t));  Safe\_Strncpy(cloudm\_context.pushEvt[ev\_id].jpegFilename, fall\_ev->jpg\_name, 64);  cloudm\_context.pushEvt[ev\_id].ev\_id = ev\_id;  cloudm\_context.pushEvt[ev\_id].event\_type = (fall\_ev->ev\_type == 1) ? PLAT\_EVENT\_FALL\_DETECT: PLAT\_EVENT\_BED\_EXIT;  cloudm\_context.pushEvt[ev\_id].eventTimeSec = (time\_t)((fall\_ev->ev\_time -500) / 1000);  KSAL\_Schedule\_Task(Cloud\_AlarmEventHandle\_Tasklet,&cloudm\_context.pushEvt[ev\_id],0);  }  break;  case CLOUDM\_EV\_VIDEO\_VERIFY\_CMD:  CloudMng\_VideoVerification\_Cmd\_S \*veriCmd = (CloudMng\_VideoVerification\_Cmd\_S \*)msg->data;  int ev\_id = CLoud\_Get\_FreeEventId();  if(ev\_id < MAX\_PUSH\_EVENT\_NUM )  {  ksal\_memset(&cloudm\_context.pushEvt[ev\_id],0,sizeof(Cloud\_PushEvent\_t));  cloudm\_context.pushEvt[ev\_id].ev\_id = ev\_id;  ksal\_memcpy(&cloudm\_context.verifyCmd, veriCmd, sizeof(CloudMng\_VideoVerification\_Cmd\_S));  KSAL\_Schedule\_Task(Cloud\_VideoVerification\_Tasklet,&cloudm\_context.pushEvt[ev\_id],0);  }  break;  case CLOUDM\_EV\_VIDEO\_UPLOAD\_CMD:  CloudMng\_VideoUpload\_Cmd\_S \*uploadCmd = (CloudMng\_VideoUpload\_Cmd\_S \*)msg->data;  int ev\_id = CLoud\_Get\_FreeEventId();  if(cloudm\_context.do\_uploading == 0 && ev\_id < MAX\_PUSH\_EVENT\_NUM )  {  ksal\_memset(&cloudm\_context.pushEvt[ev\_id],0,sizeof(Cloud\_PushEvent\_t));  cloudm\_context.pushEvt[ev\_id].ev\_id = ev\_id;  cloudm\_context.do\_uploading =1;  ksal\_memcpy(&cloudm\_context.uploadCmd, uploadCmd, sizeof(CloudMng\_VideoUpload\_Cmd\_S));  KSAL\_Schedule\_Task(Cloud\_VideoUpload\_Tasklet,&cloudm\_context.pushEvt[ev\_id],0);  }  else  {  char callbackUrl[1024];  ksal\_shm\_vdoUploadUrl\_get(callbackUrl);  CloudStartVideoAck\_error(callbackUrl);  }  break;  case CLOUDM\_EV\_DO\_OTA\_CMD:  KSAL\_Schedule\_Task(Cloud\_DoOTA\_Tasklet,NULL,0);  break;  case CLOUDM\_EV\_LOG\_ON\_CMD:  Log\_Config\_S \*remote\_log = (Log\_Config\_S \*)msg->data;  Kami\_Info("Remote log enabled\n");  if(cloudm\_context.log\_run == 1)  {  if(cloudm\_context.log\_on.log\_enable != remote\_log->log\_enable || cloudm\_context.log\_on.log\_freq != remote\_log->log\_freq)  {  cloudm\_context.log\_on.log\_enable = remote\_log->log\_enable;  cloudm\_context.log\_on.log\_freq = remote\_log->log\_freq;  if(cloudm\_context.log\_on.log\_enable )  KSAL\_Start\_Timer(cloudm\_context.uploadTimer, remote\_log->log\_freq \* 1000 \* 5, true);  else  KSAL\_Stop\_Timer(cloudm\_context.uploadTimer);  }  }  break;  case CLOUDM\_EV\_LOG\_TIMER\_CMD:  Kami\_Info("Remote log upload\n");  if(cloudm\_context.log\_uploading == 0)  {  cloudm\_context.log\_uploading = 1;  KSAL\_Schedule\_Task(Cloud\_logUpload\_Tasklet,NULL,0);  }  break;  default:  break;  }  }    return(thread\_alive);  } |

#### 1.2.2 iot\_mng

物联网管理线程（iot\_mng）主要负责web端与移动端的通信处理，通过MQTT信息通信，处理以下事件：

1）IOTM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）IOTM\_EV\_ITEM\_REPORT\_CMD SD卡状态报告；

3）CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY 云设备上线通知，负责进行通信；

|  |
| --- |
| static bool\_t Iot\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case IOTM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" IOTM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case IOTM\_EV\_ITEM\_REPORT\_CMD:  IotMng\_ItemInfoReport\_S \*item = (IotMng\_ItemInfoReport\_S\*)msg->data;  if(iotm\_context.is\_running == 1)  reportFWCurrentSdStatusItem(item->item\_str);  break;  case CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY:  Kami\_Info("CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY recieve\n");  Cloud\_Online\_Status\_S \*online\_info = (Cloud\_Online\_Status\_S \*)(msg->data);  if(online\_info->bindStatus == 0 && 0 == iotm\_context.is\_running)  {  Iot\_Core\_Run(online\_info->orgid\_cloud);  iotm\_context.is\_running = 1; // 解决wifi断开重连后启动多次Iot\_Core\_Run问题  }  break;  case CLOUDM\_EV\_IOT\_INFO\_NOTIFY:  default:  break;  }  }    return(thread\_alive);  } |

#### 1.2.3 webrtc\_mng

实时通讯管理线程（webrtc \_mng）主要负责camera端视频上传和web端查看CVR视频，主要处理以下事件：

1）WEBRTCM\_EV\_SHUTDOWN\_CMD 终止自身线程，通过跳出while(1)结束；

2）CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY 云设备上线通知，

3）WEBRTCM\_EV\_VIDEO\_UPLOAD\_CMD 视频上传命令

4）WEBRTCM\_EV\_VIDEO\_UPLOAD\_STOP\_CMD 视频上传停止命令

5）WEBRTCM\_EV\_CVR\_VIEW\_START\_CMD 开始看CVR视频命令

6）WEBRTCM\_EV\_CVR\_VIEW\_STOP\_CMD 结束看CVR命令

|  |
| --- |
| static bool\_t WebRtc\_Mng\_Process\_Event (const KSAL\_Message\_T \* msg)  {  bool\_t thread\_alive = true;  if(msg != NULL)  {  switch (msg->event\_id)  {  case WEBRTCM\_EV\_SHUTDOWN\_CMD:  Kami\_Info(" MEDIAM\_EV\_SHUTDOWN\_CMD received\n");  thread\_alive = false;  break;  case CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY:  Cloud\_Online\_Status\_S \*online\_info = (Cloud\_Online\_Status\_S\*)msg->data;  Kami\_Info(" CLOUDM\_EV\_DEVICE\_ONLINE\_NOTIFY received\n");  if(online\_info->bindStatus == 0 && 0 == webrtc\_contex.is\_running)  {  webrtc\_contex.is\_running = 1;  KSAL\_Create\_Task(Webrtc\_UploadVideo\_Task, NULL, NULL);  }  break;  case WEBRTCM\_EV\_VIDEO\_UPLOAD\_CMD:  if(webrtc\_contex.is\_running == 1)  {  Webrtc\_UploadVideo\_Cmd\_S \*upload\_cmd = (Webrtc\_UploadVideo\_Cmd\_S \*)msg->data;  upload\_cmd->sender\_appid = msg->sender\_app\_id;  upload\_cmd->sender\_tid = msg->sender\_thread\_id;  int id = WebRtc\_GetandSet\_VideoCmdFreeId(upload\_cmd);  Kami\_Info("--id %d--------\n",id);  if(id >= MAX\_EV\_UPLOAD\_CMD)  {  Kami\_Warn("too manny upload video task\n");  Webrtc\_UploadVideoResult\_S result\_ntfy = {0};  result\_ntfy.ev\_id = upload\_cmd->ev\_id;  result\_ntfy.upload\_result = UPLOAD\_VIDEO\_RESULT\_UPLOAD\_FAILED;  result\_ntfy.uploade\_start\_time = 0;  result\_ntfy.video\_time = 0;  result\_ntfy.utc\_start\_time = 0;  KSAL\_Send(upload\_cmd->sender\_appid, upload\_cmd->sender\_tid,  WEBRTCM\_EV\_VIDEO\_UPLOAD\_ACK,&result\_ntfy,sizeof(Webrtc\_UploadVideoResult\_S));  }  else if (upload\_cmd->upload\_type == 0)  {  if(id < MAX\_EV\_UPLOAD\_CMD)  Kvs\_ProducerVideoUpload\_NewEventPengding();  }  }  break;    case WEBRTCM\_EV\_VIDEO\_UPLOAD\_STOP\_CMD:  Kvs\_ProducerVideoUpload\_Stop();  break;  case WEBRTCM\_EV\_CVR\_VIEW\_START\_CMD:  Webrtc\_CvrView\_Cmd\_S \*viewCmd =(Webrtc\_CvrView\_Cmd\_S \*)msg->data;  webrtc\_run(viewCmd);  break;  case WEBRTCM\_EV\_CVR\_VIEW\_STOP\_CMD:  Webrtc\_CvrView\_Cmd\_S \*viewCmd =(Webrtc\_CvrView\_Cmd\_S \*)msg->data;  webrtc\_exit((char\*)(viewCmd->signal\_channel));  break;  default:  break;  }  }  return(thread\_alive);  } |