INGENIC®
T31 System API Reference

Date: 2022-04
Viewer: Jason Xu

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

INGENIC®

T31 System API Reference

Release history

Date	Revision	Change
2022-04	1.0	First release

Disclaimer

Ingenic Semiconductor Co. Ltd

Add: Junzheng R&D Center, Phase II, Zhongguancun Software Park, Dongbeiwangxi Road, Haidian District, Beijing, China

Tel: 86-10-56345000

Fax: 86-10-56345001

Http: //www.ingenic.com

Content

1 Related concepts

The main Function of system control is to connect the modules and define the data flow. Here are some important concepts：

1.1 Device

Device is a collection of Functions. For example, framesource completes the output of video source data, and encoder completes the Function of video encoding or image encoding. Here, framesource and encoder are the concepts of device. Device is just a collection concept, not a specific data flow node.

1.2 Group

Group is the smallest unit of data input. A device can have multiple groups, and each group can only receive one channel of data input.

Group can have multiple outputs（1.3 Output）.

Group is also a container for specific "Functions", as explained in 1.5 Channel.

1.3 Output

Output is the smallest unit of a group's one-way data output. A group can have multiple outputs, and each output can only output one channel of data.

1.4 Cell

Cell is a collection of Device, Group and Output information. Presented by impcell data structure.

Cell is mainly used for bind. According to the definition of Device, Group and Output, Output is the node of data Output, and Group is the node of data input.

During the binding process, the cell index of data Output node and data input node are referred as Output and input’s Group respectively (therefore, as cell of data input, output is a meaningless value).

1.5 Channel

Channel is usually a single Function unit, and specific Functions will be set to the Channel while creating it (instantiating)

For example：

- ✧ For encoder, a channel can perform H264 encoding or JPEG encoding. The specific encoding Functions (types and parameters) are specified when the channel is created.
- ✧ For IVS, a channel performs the Function of a specific algorithm, and the specific algorithm (type and parameters are specified when the channel is created).
- ✧ For OSD, there is a concept “region” similar to channel. A region is a specific overlay area, which can be a picture (image), cover (occlusion), etc…
- ✧ For framesource, a channel outputs the original frames, and the channel of framesource is actually a group.

As a 【Function】al unit, Channel usually needs to register in the Group (except Framesource) to receive data. After the Channel is registered in the Group, it will get the data entered by the Group.

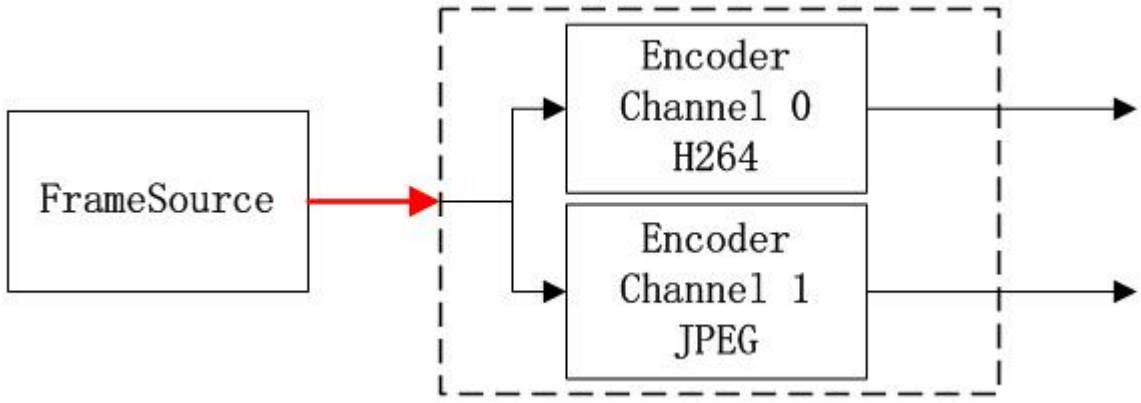
The number of Channels that can be Registered to Groups of different Devices is also different.

2 Module binding（Bind）

After the two Groups are connected by Bind, the data of the source Group will be automatically sent to the destination Group.

Because Group is the smallest unit of data input，Output is the smallest unit of data output，So IMP_ System_ Bind (IMPCell *srcCell, IMPCell * dstCell), the deviceid, groupid and outputid of srcCell are valid, while dstcell only has deviceid and groupid valid (outputid is meaningless because it is considered as data input).

The following is an example of a simple bind.



In the figure above, the output of Framesource is bound to a Group of Encoder. Two Channels are Registered in the Encoder Group, so the Encoder Group has two output channels: H264 and JPEG.
Reference code：

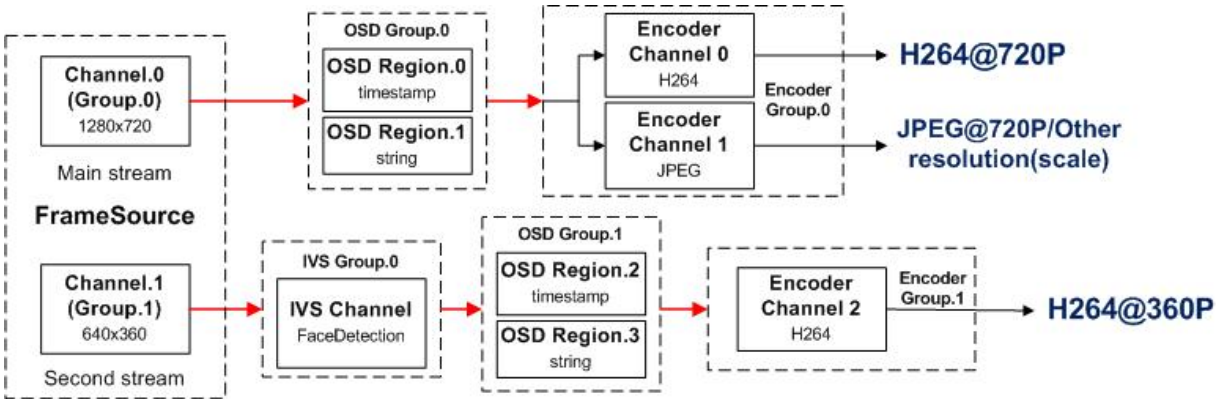
```
IMPCell fs_chn0 = {DEV_ID_FS, 0, 0};
//FrameSource deviceID:DEV_ID_FS groupID:0 outputID:0

IMPCell enc_grp0 = {DEV_ID_ENC, 0, 0};
//ENC deviceID:DEV_ID_ENC groupID:0 outputID:0,    The third parameter enc_grp0 here is meaningless.
int ret = IMP_System_Bind(&fs_chn0, &enc_grp0);
```

```
if (ret < 0)
    printf("Bind FrameSource Channel0 and Encoder Group0 failed\n");
```

Bind concatenates the data streams of the system. According to the【Function】AI requirements of different products, bind's strategies may be different.

The following is a schematic diagram of a typical dual channel bitstream product applying Bind：



In the figure above, Framesource has two outputs， They are Channel 0, the main stream (1280 x 720) and Channel 1, the second stream (640 x 360).

- Main stream：FrameSource's Channel0 Bind OSD Group.0，OSD Group.0 Bind Encoder Group.0：

  - ✧ OSD Group. 0 registers two regions to display time stamp and string information respectively；
  - ✧ Encoder group.0 has registered two channels for H264 encoding and JPEG encoding respectively. If the image size of JPEG encoding channel is not equal to the input setting (Channel 0 of Framesource), then it will be scaled (Software at T10) to achieve the purpose of arbitrary resolution capture.

- Second stream：FrameSource's Channel1 Bind IVS Group.0，IVS Group.0 Bind OSD Group.1，OSD Group.1 Bind Encoder Group.1：

  - ✧ IVS group. 0 has registered a Channel for mobile detection；
  - ✧ OSD group.1 registers two regions to display time stamp and string information respectively；
  - ✧ Encoder Group. 1 registers a Channel to encode H264；
  - ✧ It is worth noting that IVS Bind is prior to OSD because the time stamp of OSD may cause misjudgment of IVS mobile detection.

Reference code：

Main stream data stream Bind：

```
IMPCell fs_chn0 = {DEV_ID_FS, 0, 0};
IMPCell osd_grp0 = {DEV_ID_OSD, 0, 0};
IMPCell enc_grp0 = {DEV_ID_ENC, 0, 0};
int ret = IMP_System_Bind(&fs_chn0, &osd_grp0);
if (ret < 0)
    printf("Bind FrameSource Channel0 and OSD Group0 failed\n");

int ret = IMP_System_Bind(&osd_grp0, &enc_grp0);
if (ret < 0)
    printf("Bind OSD Group0 and Encoder Group0 failed\n");
```

Second stream data stream Bind：

```
IMPCell fs_chn1_output0 = {DEV_ID_FS, 1, 0};
IMPCell ivs_grp0 = {DEV_ID_IVS, 0, 0};
IMPCell osd_grp1 = {DEV_ID_OSD, 1, 0};
IMPCell enc_grp1 = {DEV_ID_ENC, 1, 0};

int ret = IMP_System_Bind(&fs_chn1_output0, &ivs_grp0);
if (ret < 0)
    printf("Bind FrameSource Channel1 and IVS Group0 failed\n");

int ret = IMP_System_Bind(&ivs_grp0, &osd_grp1);
if (ret < 0)
    printf("Bind IVS Group0 and OSD Group1 failed\n");

int ret = IMP_System_Bind(&osd_grp1, &enc_grp1);
if (ret < 0)
    printf("Bind OSD Group1 and Encoder Group1 failed\n");
```

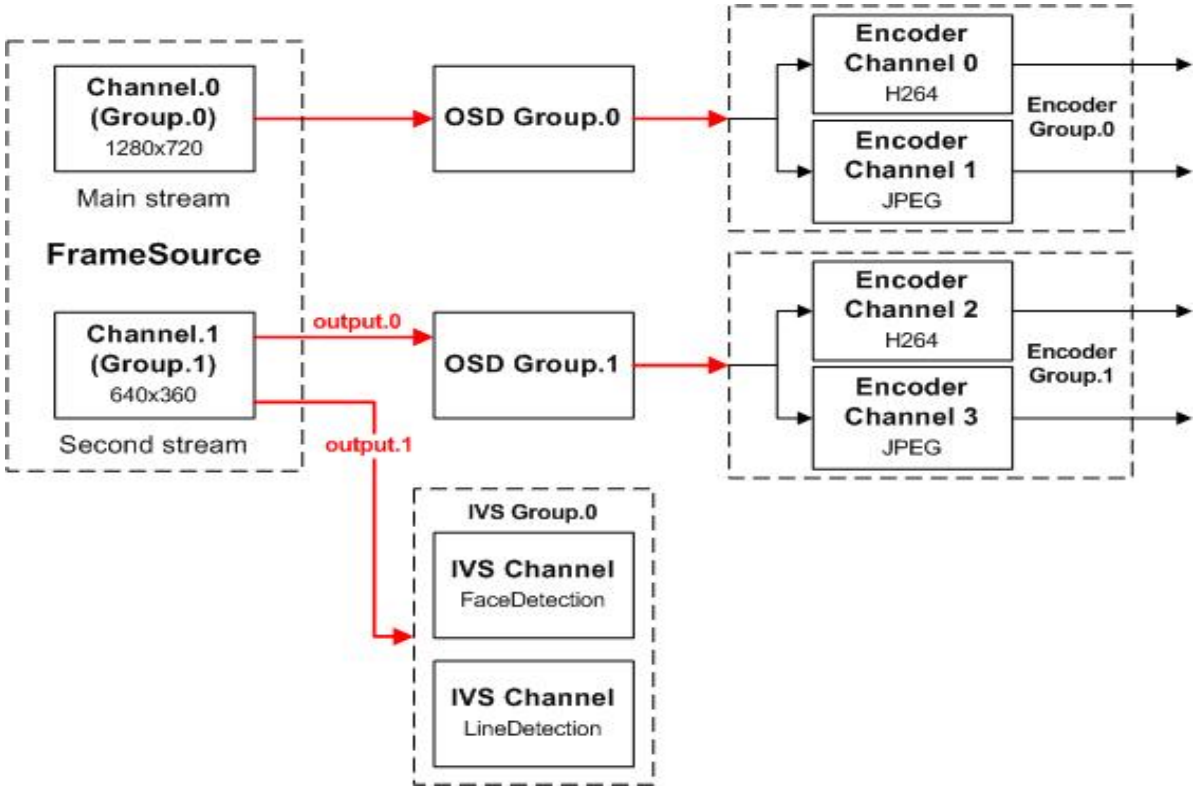
【NB】:

It is recommended that all Bind operations to be performed during system initialization.

After Framesource is enabled, Bind and UnBind operations cannot be called dynamically. UnBind can only be performed after disabling framesource

DestroyGroup can't be called until after UnBind.

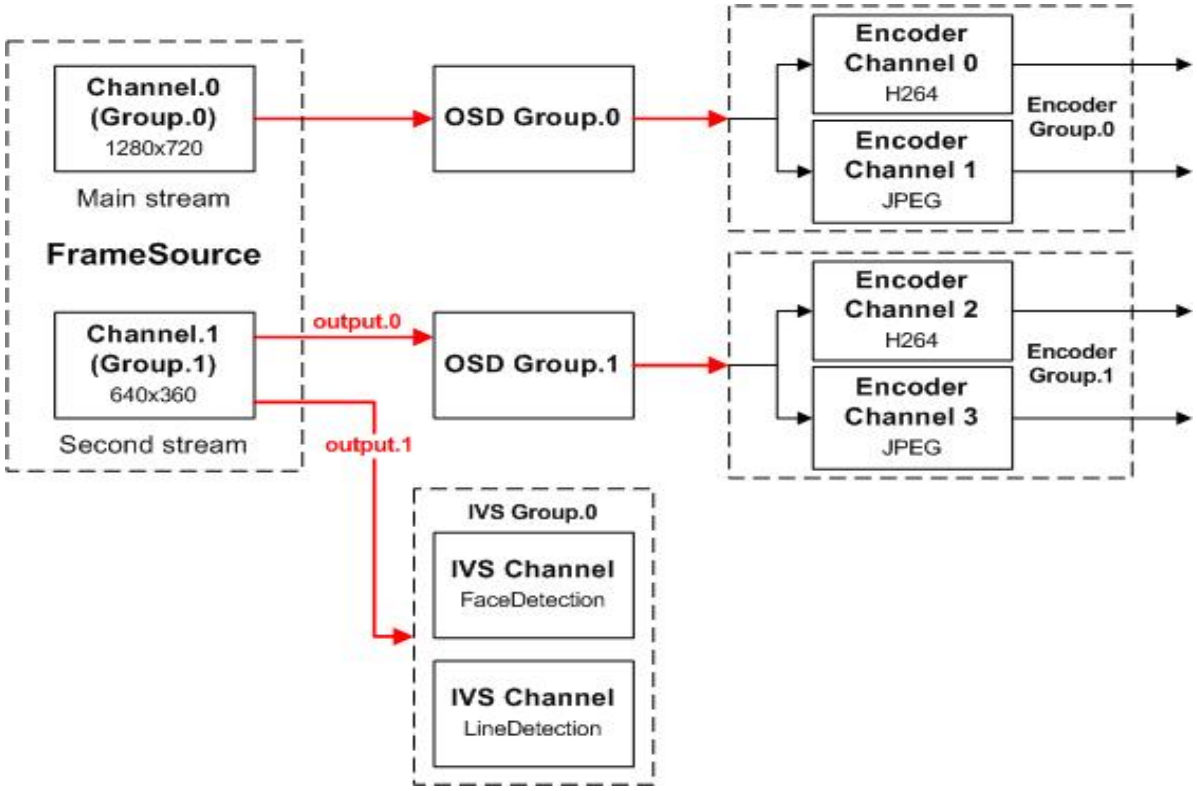
Bind can be a tree structure. The following figure is an example：



In the figure above, the Channel 1 (Group. 1) back end of Framesource Binds two Groups to Output data from output. 0 and output. 1 respectively. The advantage of such a Bind in this example is that IVS Group can work in parallel with OSD Group. 1.

【NB】:

The Bind method in this example may affect the general mobile detection, so it is not recommended for general mobile detection.



3 Module API

3.1 System Module API

API Name	Function
IMP_System_Init	IMP System initialization.
IMP_System_Exit	IMP System deinitialization.
IMP_System_GetTimeStamp	Obtain the time stamp of IMP system in microseconds.
IMP_System_RebaseTimeStamp	Set the time stamp of IMP system in microseconds.
IMP_System_ReadReg32	Read the value of the 32-bit register.
IMP_System_WriteReg32	Write value to 32-bit register.
IMP_System_GetVersion	Get imp system's version number.
IMP_System_GetCPUInfo	Get CPU model information.
IMP_System_Bind	Bind source Cell and destination Cell.
IMP_System_UnBind	Unbind source Cell and destination Cell.
IMP_System_GetBindbyDest	Gets the source cell information bound to the destination cell.
IMP_System_MemPoolRequest	Apply for mempool on rmem.
IMP_System_MemPoolFree	Free up the mempool rmem of memory

Tab 3-1 System API table

3.1.1 IMP_System_Init

【Function】

IMP System initialization.

【Grammar】

int IMP_System_Init(void);

【Formal parameter】

Nothing.

【Return value】

0: success；others: 0 failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

After calling this API, the underlying data structure will be initialized, but the hardware unit will not be initialized.

Before any operation of IMP, this interface must be called first.

3.1.2 IMP_System_Exit

【Function】

IMP System deinitialization.

【Grammar】

int IMP_System_Exit(void);

【Formal parameter】

Nothing.

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

After this 【Function】 is called, all memory and handle of IMP are released and hardware unit is closed.

After calling this API, if you want to use imp again, you need to initialize the system again.

3.1.3 IMP_System_GetTimeStamp

【Function】

Obtain the time stamp of IMP system in microseconds.

【Grammar】

int64_t IMP_System_GetTimeStamp(void);

【Formal parameter】

Nothing.

【Return value】

time(usec).

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

The time stamp is automatically initialized after the system is initialized, and the time stamp is invalid after the system is deinitialized.

3.1.4 IMP_System_RebaseTimeStamp

【Function】

Set the time stamp of IMP system in microseconds.

【Grammar】

uint32_t IMP_System_ReadReg32(uint32_t u32Addr);

【Formal parameter】

Parameter name	Describe	Input / output
basets	Base time	**Input**

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

This API is compatible with IMP_ AI_ Enable. They are used together, and imp_ AI_ Disable must be executed before the enters sleep mode.

3.1.5 IMP_System_ReadReg32

【Function】

Read the value of the 32-bit register.

【Grammar】

uint32_t IMP_System_ReadReg32(uint32_t regAddr);

【Formal parameter】

Parameter name	Describe	Input / output
regAddr	The physical address of the register	**Input**

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

nothing.

3.1.6 IMP_System_WriteReg32

【Function】

Write value to 32-bit register.

【Grammar】

void IMP_System_WriteReg32(uint32_t regAddr, uint32_t value);

【Formal parameter】

Parameter name	Describe	Input / output
regAddr	The physical address of the register	**Input**
value	The value to write to	**Input**

【Return value】

Nothing.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

In case you don't know the meaning of register, please call this API carefully. Otherwise it may cause system error.

3.1.7 IMP_System_GetVersion

【Function】

Get imp system version number.

【Grammar】

int IMP_System_GetVersion(IMPVersion *pstVersion);

【Formal parameter】

Parameter name	Describe	Input / output
pstVersion	IMP system version number structure pointer	Output

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

Nothing.

3.1.8 IMP_System_GetCPUInfo

【Function】

Get CPU model information.

【Grammar】

const char* IMP_System_GetCPUInfo(void);

【Formal parameter】

Nothing.

【Return value】

CPU model string.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

The 【Return value】 value is a string of CPU model type. For example, for T10, there are "T10" and "T10 Lite".

3.1.9 IMP_System_Bind

【Function】

Binding source Cell and destination Cell.

【Grammar】

int IMP_System_Bind(IMPCell *srcCell, IMPCell *dstCell);

【Formal parameter】

Parameter name	Describe	Input/output
srcCell	source Cell points.	Input
dstCell	destination Cell points.	Input

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

According to the concepts of Device, Group and Output, each Device may have multiple Groups, and each Group may have multiple Outputs. The Group acts as the input interface of the Device, while the Output acts as the output interface of the Device. Therefore, Binding is actually connecting an Output of the output device to a Group of the input device.

Once successfully bound, the data generated by the source Cell (Output) will be automatically transferred to the destination Cell (Group).

3.1.10 IMP_System_UnBind

【Function】

Unbind source Cell and destination Cell.

【Grammar】

int IMP_System_UnBind(IMPCell *srcCell, IMPCell *dstCell);

【Formal parameter】

Parameter name	Describe	Input/output
srcCell	source Cell points.	Input
dstCell	destination Cell points.	Input

【Return value】

0: success； others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

Nothing.

3.1.11 IMP_System_GetBindbyDest

【Function】

Gets the source cell information bound to the destination cell.

【Grammar】

int IMP_System_GetBindbyDest(IMPCell *dstCell, IMPCell *srcCell);

【Formal parameter】

Parameter	Describe	Input/output

name		
dstCell	source Cell points.	Input
srcCell	destination Cell points.	Output

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

3.1.12 IMP_System_MemPoolRequest

【Function】

Apply for mempool on rmem.

【Grammar】

```
int IMP_System_MemPoolRequest(int poolId,size_t size,const char *name);
```

【Formal parameter】

Parameter name	Describe	Input/output
poolId	Application poolID	Input
size	Application size	Input
name	Pool name	Input

【Return value】

0: success；others: failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【NB】

Video memory pool mainly provides large physical continuous memory management 【Function】 for media processing, which is responsible for memory allocation and recovery, and gives full play to the role of memory buffer pool, so that physical memory resources can be used reasonably in each media processing module.

Video memory pool is built on the basis of the original reserved memory rmem (no page table created by the kernel) for block memory management. Every memory pool applied is physically continuous, and the memory applied on the memory pool is also physically continuous.

If memory pool is used, the size of memory pool must be configured before system initialization. According to different tasks, the size and number of memory pool applications are different.

3.1.13 IMP_System_MemPoolFree

【Function】

Free up the mempool rmem of memory。

【Grammar】

int IMP_System_MemPoolFree(int poolId);

【Formal parameter】

None.

【Return value】 value】

0: success；<0 failure.

【Dependence】

Head file：imp_system.h

Lib file：libimp.a / libimp.so

【Example】

During the bitstream running, the memory pool cannot be released. Only after all resources exit, the memory pool can be released. In addition, related memory requests in the SDK do not use the memory pool.

4.1 System Data Type

Name	Definition
IMPVersion	IMP System version number definition.

4.1.1 IMPVersion

【Explanation】

IMP System version number definition.

【Defunition】

```
typedef struct {
        char aVersion[64];
} IMPVersion;
```

【Member】

Parameter name	Describe
aVersion[64]	IMP System version number

【NB】

None.