Dual-core Communication User Guide Version 1.0

Revision history

Version/Date	Modification
V1.0: 2023/12/12 Zengjie	- The first version
	-

Contents

1 Introduction

1.1 Purpose and scope

This document introduces the mechanism and usage of dual core data transmission and resource mutual exclusion.

MUTEX

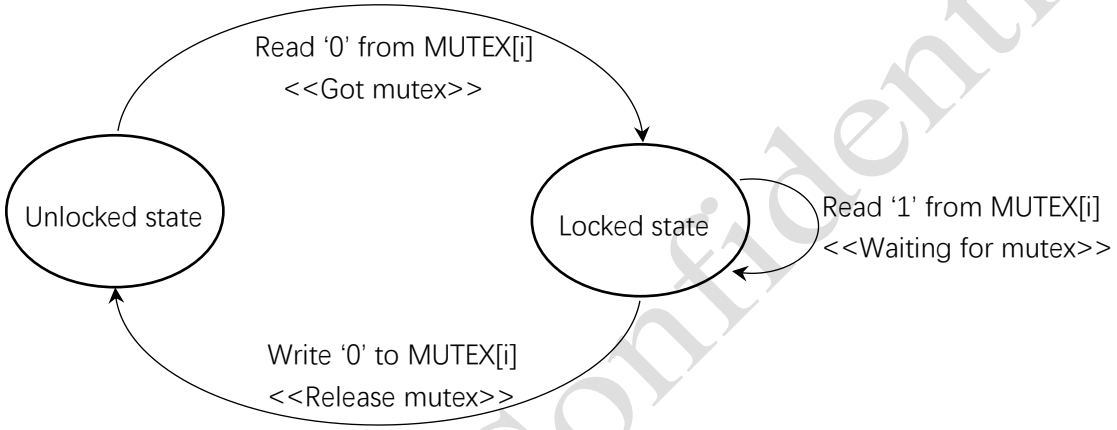

Figure 1-1 Mutex State Transition

MUTEX is used for mutually exclusive access to shared resources between dual cores, and a total of 32 MUTEX registers are used by users. When a CPU sends a read request to mutex[i], if it reads 0, the corresponding mutex[i] will enter the locked state and be automatically set to 1 (without software participation), which means that the CPU has successfully acquired the lock. If it reads 1, it means that the mutex has been acquired by another CPU. Writing 0 to the corresponding mutex[i] will unlock the locked state.

Note ：

1. the same CPU can successfully acquire a lock multiple times, and therefore the lock must be released the same number of times before it can actually be released and acquired by another CPU.
2. release the lock when you're done using the resource.
3. a lock cannot be released until it has been acquired.

Share Memory

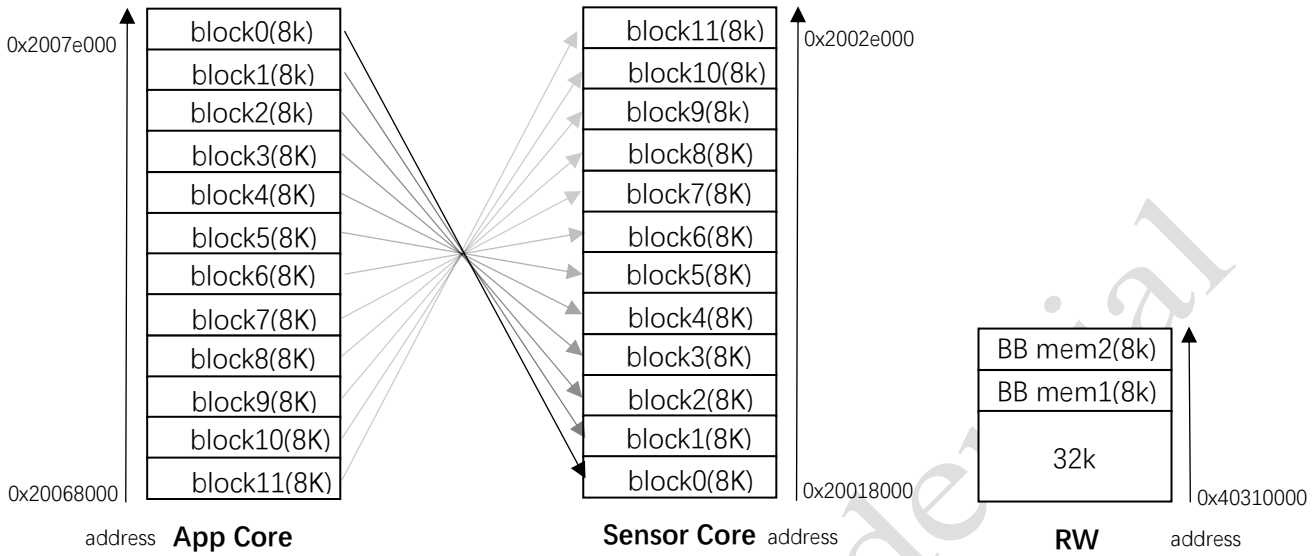

Figure 1-2 Share Memory Mappings

The share memory contains a total of 12 blocks, each occupying 8k space, each block can be mapped to four locations: App Core, Sensor Core, RW ram1, RW ram2, The same block mapped to different regions has different access addresses. When more than one block is mapped to the same region, the one with the smallest block id takes effect. Accesses to the share memory that are not connected to the sub system will get a DEADBEEF.

IPC

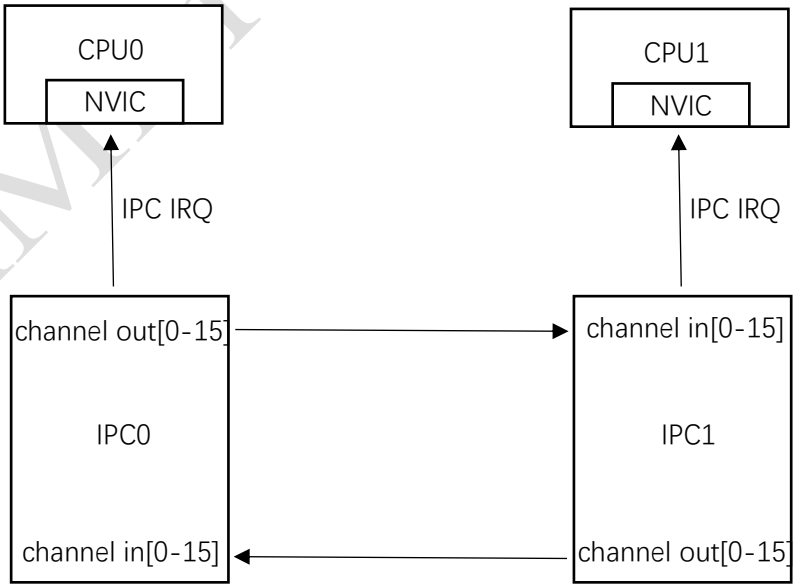

Figure 1-3 IPC Channel Connection

IPC(Inter-Process Communication) is used to transfer information between cpus. Each CPU notifies other cpus through IPC interrupts to realize communication. IPC communicates through channels, 16 in channels, 16 out channels in total. The out channel of CPU0 is connected to the in channel of CPU1, and the out channel of CPU1 is connected to the in channel of CPU0. When the IPC module receives an IPC signal from another CPU, it will generate an interrupt to notify the current CPU.

IPM

IPM is a combination of IPC and Share Memory. When core0 wants to send data to core1, it first acquires control of a block of share ram and fills it with data. Then it sends an IPC signal to core1. After receiving the signal, core1 obtains the control of the corresponding block of share ram and accesses the data in it. In this way, the data transfer from core0 to core1 is realized, and vice versa.

IPM Sync

IPM Sync is a component based on IPM driver. A simple communication protocol is used to implement the function of command and data transfer between different cores.

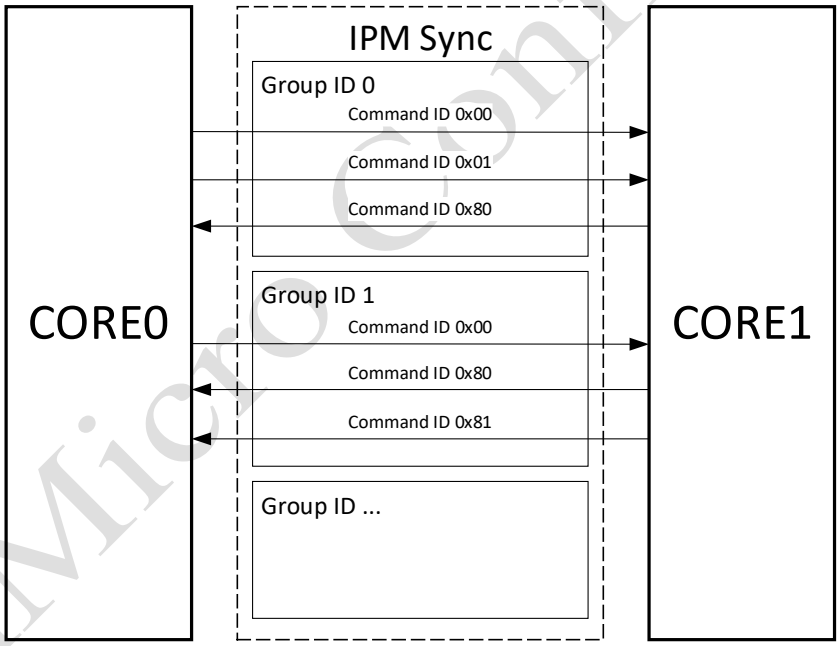

Figure 1-4 IPM Sync Component Structure

Commands are divided into groups, which could be addressed by defined Group ID. Groups could be defined in different modules or libraries. Group ID and Command ID are in size of 1 byte. Group ID could be defined as any value, while Command ID must be defined from 0x00 to 0x7f for commands from CORE0 to CORE1, or 0x80 to 0xFF for commands from CORE1 to CORE0. The lower 7bits of Command ID corresponding to the command callbacks, the MSb indicated the direction of commands. The IPM Sync Command format is shown as followed.

Group ID (1 Byte)	Command ID (1 Byte)	Data Length (2 Bytes)	Data (0~65536 Bytes)

Figure 1-5 IPM Sync Command format

1.2 Acronyms and abbreviations

The table below defines the acronyms and abbreviations used in this document.

Acronym	Definition
MUTEX	Mutual Exclusive
IPC	Inter-Process Communication

Table 1-1 Definitions and acronyms

1.3 References

2 Interface

2.1 MUTEX

Enum and Define

Mutex id defination

MUTEX_HW_ID_HW	Mutex id for HW
MUTEX_HW_ID_IPM	Mutex id for IPM
MUTEX_HW_ID_PMU_DAIF	Mutex id for PMU and DAIF
MUTEX_HW_ID_GPADC	Mutex id for GPADC
MUTEX_HW_ID_USER_FIRST	Mutex id for user first id
MUTEX_HW_ID_NUM	Number of mutex id

Initialize hardware mutex

mutex_hw_init()	Initialize hardware mutex

Sturctures

Function Interface

mutex_hw_lock

Function: __STATIC_FORCEINLINE bool mutex_hw_lock(uint8_t mutex);
Description: Lock the specified mutex, if the specified mutex is already locked, its state remains unchanged
Parameter:

mutex	id of the mutex to be locked

Return:

true	mutex is successfully locked
false	mutex was already locked

mutex_hw_unlock

Function: __STATIC_FORCEINLINE void mutex_hw_unlock(uint8_t mutex);
Description: Unlock the specified mutex, if the specified mutex is already unlocked, its state remains unchanged
Parameter:

mutex	id of the mutex to be unlocked

2.2 SHARE MEMORY

Enum and Define

SHMEM_START_ADDRESS	Start address of share memory
SHMEM_BLOCK_OTO_INDEX	Block to its index or index to block
SHMEM_OWNER_POSITION	Share memory's position of current core
SHMEM_BLOCK_SIZE	Size of block
SHMEM_TOTAL_SIZE	Share memory's total size
SHMEM_ADDRESS_TO_BLOCK	Address to block conversion
SHMEM_BLOCK_TO_ADDRESS	Block to address conversion
SHMEM_NOT_OBTAIN_FLAG	Not obtain flag

shmem_position_t

share memory position type defination

SHMEM_IN_APP_CORE	In APP core
SHMEM_IN_SENSOR_HUB_CORE	In Sensor Hub core
SHMEM_IN_BB_MEM_2	In BB memory 2
SHMEM_IN_BB_MEM_1	In BB memory 1

shmem_config_t

share memory block type defination

SHMEM_BLOCK_0	Block0
SHMEM_BLOCK_1	Block1
SHMEM_BLOCK_2	Block2
SHMEM_BLOCK_3	Block3
SHMEM_BLOCK_4	Block4
SHMEM_BLOCK_5	Block5
SHMEM_BLOCK_6	Block6

SHMEM_BLOCK_7	Block7
SHMEM_BLOCK_8	Block8
SHMEM_BLOCK_9	Block9
SHMEM_BLOCK_10	Block10
SHMEM_BLOCK_11	Block11
SHMEM_BLOCK_NUM	Number of block

Sturctures

shmem_config_t

share memory config

```
1.  typedef struct {
2.      /// Block
3.      shmem_block_t block;
4.      /// Position
5.      shmem_position_t position;
6.  } shmem_config_t;
```

Function Interface

shmem_position_set

Function: __STATIC_FORCEINLINE void shmem_position_set(shmem_block_t block, shmem_position_t position);
Description: set shmem block position
Parameter:

block	block
position	block’s position

shmem_position_get

Function: __STATIC_FORCEINLINE shmem_position_t shmem_position_get(shmem_block_t block);
Description: get shmem block position
Parameter:

block	block

shmem_init

Function: __STATIC_FORCEINLINE void shmem_init(const shmem_config_t *config, int config_num);
Description: Initialize share memory
Parameter:

config	configuration
config_num	number of configuration

shmem_obtain_by_block

Function: __STATIC_FORCEINLINE void shmem_obtain_by_block(shmem_block_t block);
Description: Obtain share memory by block
Parameter:

block	block

shmem_obtain_by_address

Function: __STATIC_FORCEINLINE void shmem_obtain_by_address(void *address);
Description: Obtain share memory by address
Parameter:

address	address

2.3 IPM

Enum and Define

ipm_id_t

ipm channel id

IPM_ID_BT_H2C	System reserve for Bluetooth
IPM_ID_BT_C2H	System reserve for Bluetooth
IPM_ID_USER_FIRST	For user
IPM_ID_NUM	Number of ipm id

ipm_event_callback_t

Callback for incoming IPM messages

```
typedef void (*ipm_event_callback_t) (ipm_id_t id, om_fifo_t *fifo);
```

Sturctures

ipm_config_t

IPM config

```
typedef struct {
    /// ID
    ipm_id_t id;
    /// exchange memory address
    void *exchange_mem_address;
    /// exchange memory length
    unsigned int exchange_mem_length;
} ipm_config_t;
```

Function Interface

ipm_init

Function: void ipm_init(const ipm_config_t *config, int config_num);
Description: Initialize ipm
Parameter:

config	configuration
config_num	number of configuration

ipm_send_signal

Function: __STATIC_FORCEINLINE void ipm_send_signal(ipm_id_t id);
Description: send ipm signal
Parameter:

id	ipm id

ipm_send_signal_local

Function: __STATIC_FORCEINLINE void ipm_send_signal_local(ipm_id_t id);
Description: send local ipm signal
Parameter:

id	ipm id

ipm_send_to_fifo

Function: unsigned int ipm_send_to_fifo(ipm_id_t id, const void *pdata, unsigned int length);
Description: send ipm data to fifo, not signalling
Parameter:

id	ipm id
pdata	pointer to the data to be sent
length	length of the data

Return:

length	sent length

ipm_fifo_avail_length

Function: unsigned int ipm_fifo_avail_length(ipm_id_t id);
Description: get ipm fifo available length
Parameter:

id	ipm id

Return:

length	fifo available length

ipm_send

Function: unsigned int ipm_send(ipm_id_t id, const void *pdata, unsigned int length);
Description: send ipm data to fifo and signalling
Parameter:

id	ipm id
pdata	pointer to data to be sent
legth	length of data

Return:

lenth	sent length

ipm_register_event_callback

Function: void ipm_register_event_callback(ipm_id_t id, ipm_event_callback_t event_cb);
Description: register ipm event callback
Parameter:

id	ipm id
event_cb	receive event callback

2.4 IPM Sync

Enum and Define

ipm_sync_reveive_callback

Callback for incoming IPM Sync Commands

```
1.  typedef void (*ipm_sync_receive_callback)(uint8_t *data, uint16_t data_len);
```

Sturctures

ipm_sync_group_t

IPM Sync group

```
1.  typedef struct {
2.      ipm_sync_receive_callback   *cb;
3.      uint8_t                     cmd_num;
4.      uint8_t                     group_id;
5.  } ipm_sync_group_t;
```

Function Interface

ipm_sync_init

Function: void ipm_sync_init(ipm_id_t transmit_id,
ipm_id_t receive_id,
ipm_sync_group_t **group_table,
uint8_t group_num);

Description: Initialize IPM Sync

Parameter:

transmit_id	the IPM id for transmit
receive_id	the IPM id for receive
group_table	the table of defined group pointers
group_num	the number of defined group pointers

ipm_sync_send_cmd

Function: void ipm_sync_send_cmd(uint8_t group_id,
uint8_t cmd_id,
uint8_t *data,
uint16_t data_len);

Description: send IPM Sync command to the other core immediately and notify the other core.

Parameter:

group_id	the IPM Sync Group id
cmd_id	the IPM Sync Command id
data	the pointer of data to be send
data_len	the length of data to be send

ipm_sync_send_cmd_to_fifo

Function: void ipm_sync_send_cmd(uint8_t group_id,
uint8_t cmd_id,
uint8_t *data,
uint16_t data_len);

Description: send IPM Sync command to the FIFO and do not notify the other core temporally.

Parameter:

group_id	the IPM Sync Group id
cmd_id	the IPM Sync Command id
data	the pointer of data to be send
data_len	the length of data to be send

3 Example

3.1 Initializing MUTEX

```
1.  mutex_hw_init();
```

3.2 Locking and unlocking between dual cores

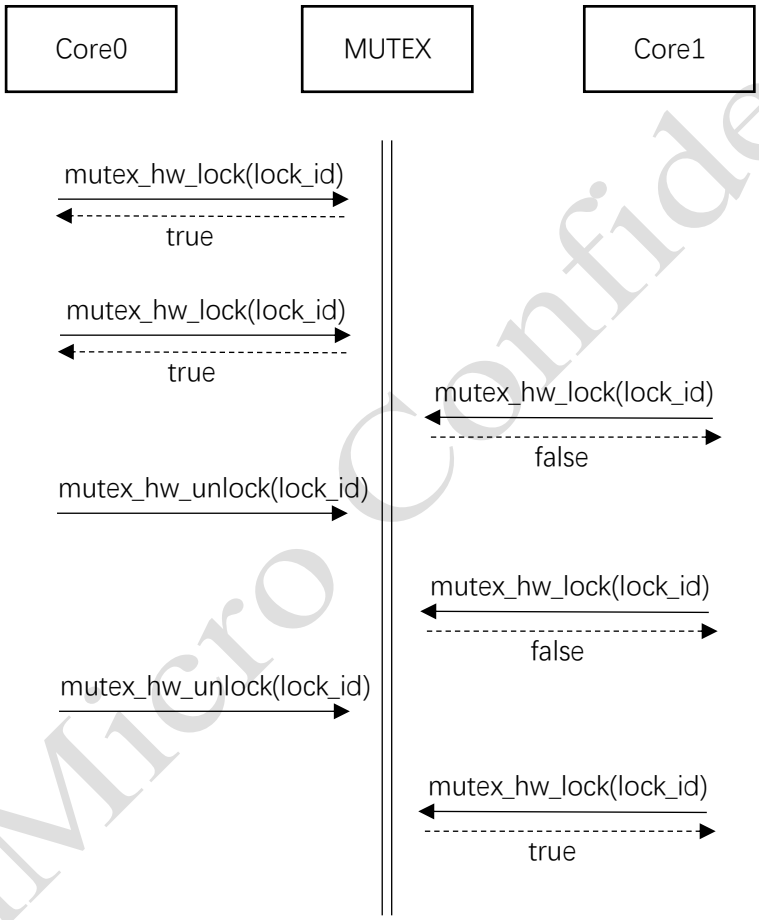

Figure 3-1 MUTEX lock and unlock process

3.3 Get specified memory block by current core

```
1.  while (!mutex_hw_lock(lock_id));
2.  shmem_obtain_by_block(block_id);
3.  /// process the block...
4.  mutex_hw_unlock(lock_id);
```

3.4 Set block to specified memory-mapped region

```
shmem_config_t config;
while (!mutex_hw_lock(lock_id));
config.block = block_id;
/// position: IN_APP_CORE, IN_SENSOR_CORE, IN_BB_MEM_1, IN_BB_MEM_2
config.position = position;
shmem_init(&config, 1);
/// process the block...
mutex_hw_unlock(lock_id);
```

3.5 Initializing IPM Partition

```
static ipm_config_t ipm_config_table[] = {
    /// partition 1
    {
        (ipm_id_t)(IPM_ID_USER_FIRST + 0),
        (void*)((uint32_t)SHMEM_BLOCK_TO_ADDRESS(SHMEM_BLOCK_0) + 4352),
        1024
    },
     /// partition 2
    {
        (ipm_id_t)(IPM_ID_USER_FIRST + 1),
        (void*)((uint32_t)SHMEM_BLOCK_TO_ADDRESS(SHMEM_BLOCK_1) + 4352),
        1024
    },
};

ipm_init(ipm_config_table, sizeof(ipm_config_table)/sizeof(ipm_config_table[0]));
```

This code configures the partition of the ipm and binds the ipm id to the specified share memory address. Users should configure this in both core0 and core1.

3.6 IPM send data to another core

```
1. ipm_send(ipm_id, data, length);
```

Data is sent via the specified ipm id, and upon successful delivery, the other core is notified by an interrupt.

3.7 IPM receive data from another core

```
1. /// the callback is triggered when an ipm signal from another core is received
2. void ipm_user_rec_callback(ipm_id_t id, om_fifo_t *fifo)
3. {
4.     /// process received data in fifo...
5. }
6. /// register callback
7. ipm_register_event_callback(ipm_id, ipm_user_rec_callback);
```

After the user callback function is registered, if there is a data reception for the corresponding ipm id, the corresponding callback function will be triggered, and the user can implement the data processing logic in the callback function.

3.8 Add IPM Sync Command or Group

A default System IPM Sync Group is defined in projects. In order to add new IPM Sync Command, we need to add Command ID definition in ipm_sync_system_cmd_id_t. For receive commands, the corresponding callback must be defined in ipm_sync_system_cmd_cbs. If callback is unnecessary, it should be set as NULL.

```
1. typedef enum {
2.     // core0 commands
3.     IPM_SYNC_SYSTEM_CMD_ID_BT_CONTROLLER_INIT       = 0x00,
4.     IPM_SYNC_SYSTEM_CMD_ID_SLEEP_CTRL               = 0x01,
5.
6.     // core1 commands
7.     IPM_SYNC_SYSTEM_CMD_ID_BT_CONTROLLER_INIT_DONE  = 0x80,
8. } ipm_sync_system_cmd_id_t;
9.
10. static ipm_sync_receive_callback ipm_sync_system_cmd_cbs[] = {
11.     ipm_sync_cmd_bt_controller_done_cb,
12. };
```

In order to add a new group, we need to define a new callback tables, and add new group id to ipm_sync_group_id_t.

```
1. typedef enum {
```

```
    IPM_SYNC_GROUP_ID_FT        = 0x00,
    IPM_SYNC_GROUP_ID_SYSTEM    = 0x01,
} ipm_sync_group_id_t;

ipm_sync_group_t system_ipm_sync_group_core0 = {
    .cb = ipm_sync_system_cmd_cbs,
    .cmd_num = sizeof(ipm_sync_system_cmd_cbs) / sizeof(ipm_sync_receive_callback),
    .group_id = IPM_SYNC_GROUP_ID_SYSTEM,
};
```

Then the new IPM Sync Group should be added to IPM Sync Group Table.

```
#if (OM6681A_CORE0)
extern ipm_sync_group_t ft_ipm_sync_group_core0;
extern ipm_sync_group_t system_ipm_sync_group_core0;
ipm_sync_group_t *ipm_sync_group[] = {
    &ft_ipm_sync_group_core0,
    &system_ipm_sync_group_core0,
};
#else
extern ipm_sync_group_t ft_ipm_sync_group_core1;
extern ipm_sync_group_t system_ipm_sync_group_core1;
ipm_sync_group_t *ipm_sync_group[] = {
    &ft_ipm_sync_group_core1,
    &system_ipm_sync_group_core1,
};
#endif  /* (OM6681A_CORE0) */
```

3.9 Initializing IPM Sync

Then initialize the IPM Sync with transmit IPM ID, receive IPM ID and IPM Sync Group Table.

```
void ipm_sync_group_init(void)
{
    #if (OM6681A_CORE0)
    ipm_sync_init(IPM_ID_SYSTEM_APP2SEN, IPM_ID_SYSTEM_SEN2APP, ipm_sync_group, sizeof(ipm_
sync_group) / sizeof(ipm_sync_group_t *));
    #else
    ipm_sync_init(IPM_ID_SYSTEM_SEN2APP, IPM_ID_SYSTEM_APP2SEN, ipm_sync_group, sizeof(ipm_
sync_group) / sizeof(ipm_sync_group_t *));
    #endif
}
```

3.10 IPM Sync send command

The specified command addressed by Group ID and Command ID will be sent directly to the other core. The other core will be notified immediately, if the core is in sleep mode, it will be waked up.

```
1. ipm_sync_send_cmd(IPM_SYNC_GROUP_ID_SYSTEM, IPM_SYNC_SYSTEM_CMD_ID_SLEEP_CTRL, &sleep_enabl
   e, 1);
```

The specified command addressed by Group ID and Command ID will be sent to the FIFO. When the other core is notified by calling ipm_sync_send_cmd, the commands in the FIFO will be handled sequentially.

```
2. ipm_sync_send_cmd_to_fifo(IPM_SYNC_GROUP_ID_SYSTEM, IPM_SYNC_SYSTEM_CMD_ID_SLEEP_CTRL, &sle
   ep_enable, 1);
```