OTA 双区，顾名思义即原始代码区（A 区）和升级备份区（B 区），在一次升级传输之后不再执行擦除并搬移的流程，直接在重启后跳转到新的文件中执行。

```
beken_ota = 1
GLOBAL_DEFINES += CONFIG_OTA_DUAL_SWITCH
```

功能开关如图，在 light.mk 中定义 CONFIG_OTA_DUAL_SWITCH

分区地址划分定义在 board.h 中

```
#define PARTITION_STACK_CPU_ADDR    (0x1F00)
#define PARTITION_APP_CPU_ADDR      (0x16A00)
#define PARTITION_OTA_BAC_CPU_ADDR  (0x45B00)
```

PARTITION_APP_CPU_ADDR 是基础代码区，PARTITION_OTA_BAC_CPU_ADDR 是升级备份区。默认初次烧写时从代码区执行，不可更换。

编译时通过修改 bk3633.ld 的地址值来生成不同分区的升级文件

第一次填写 PARTITION_APP_CPU_ADDR 地址，工程编译出可烧录文件，并将 bk3633_alios\platform\mcu\bk3633\bin 路径下 bk3633_ble_app_oad.bin 文件名修改为 a.bin

```
_vector_start = 0x16A00;

SECTIONS
{
/* vectors go to vectors region */
    .vectors 0x16A00 : /*Do not change the section order, for image*/
    {
      KEEP(*(sys.vectors))      Yulong, a year ago • Add the bk3633 AliO
      KEEP(*(sys.handlers))
    } > flash
```

然后修改 PARTITION_OTA_BAC_CPU_ADDR 地址，生成的 bk3633_ble_app_oad.bin 文件名修改为 b.bin，并注意切勿烧录此时生成的工程文件

```
/* PARTITION_APP_CPU_ADDR */
_vector_start = 0x45B00;

SECTIONS
{
/* vectors go to vectors region */
    .vectors 0x45B00 : /*Do not change the section order, for image*/
    {
      KEEP(*(sys.vectors))
      KEEP(*(sys.handlers))
    } > flash
```

同文件夹下执行 python 文件 bind.py。其中输入的 a.bin、b.bin，输出 bindAB.bin，拼接偏移量 Bsec_addr，填充数据 tap 都可通过修改相应参数来自定义

```
out = open("bindAB.bin", "wb")
print "新建输出文件: ", out.name

in1 = open("a.bin", "rb")
print "读取文件: ", in1.name

in2 = open("b.bin", "rb")
print "读取文件: ", in2.name

Bsec_addr = 0x40000
tap = 0xff
```

执行完成后同文件夹下生成的 bindAB.bin 即为拼接完成的文件，可存放于服务器升级

在 hilink_ota_service 下新建 ota_get_board_sec 函数供 service 获悉当前执行区域，返回值为 HAL_PARTITION_APPLICATION 或 HAL_PARTITION_OTA_TEMP

```
static int ota_get_board_sec()
```