WL83资源差分升级说明文档

本文档是WL83资源差分升级说明文档

版本

版本	时间	备注
v0.1	2025/8/19	

需求背景

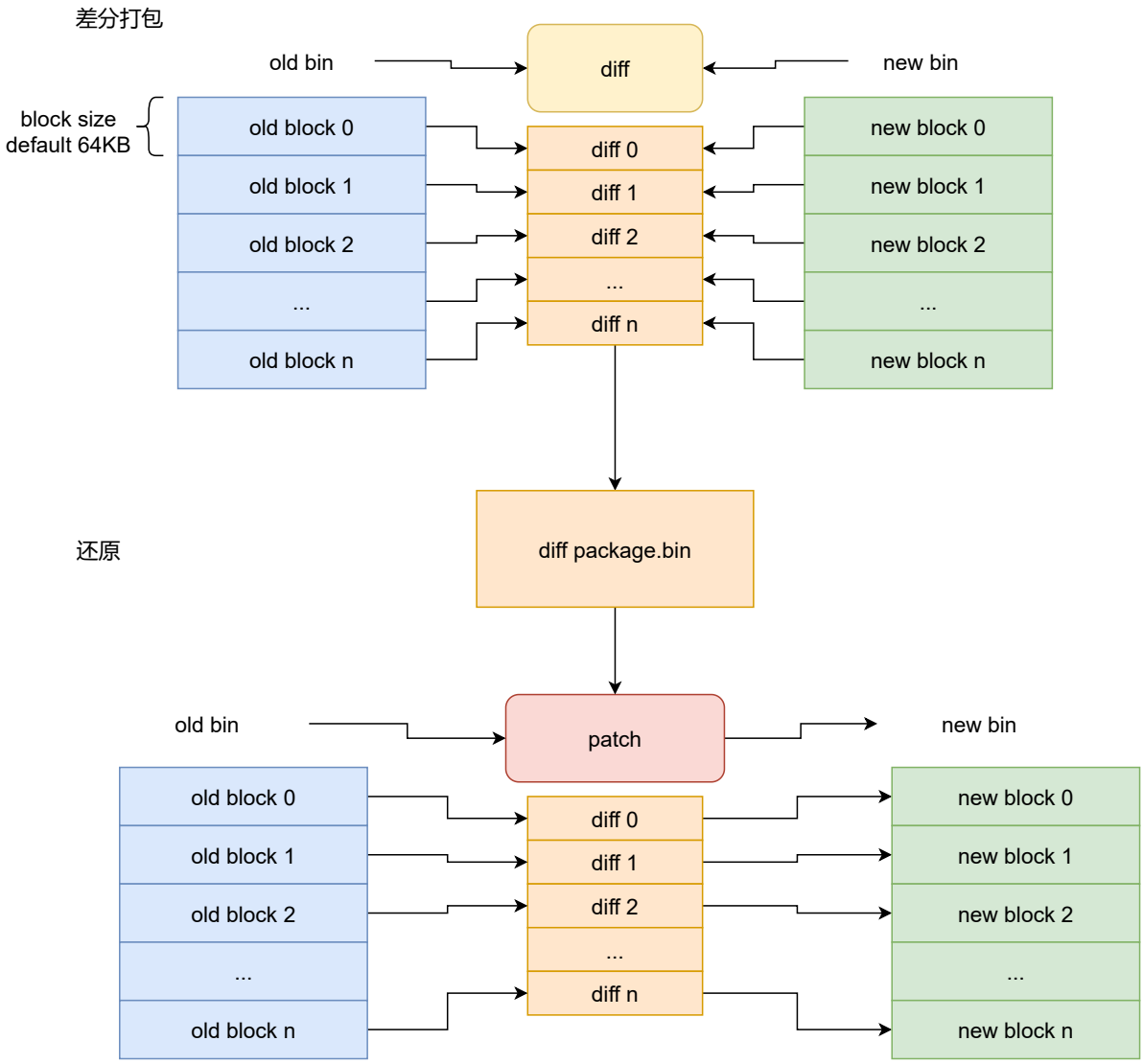
- WL83客户案需求，OTA包用全量包过大，改用差分升级方式，减小OTA包大小

注意事项

- 差分升级，在新旧固件差异变化不大时，可以有效减小OTA包的大小
- 新旧固件差异越大，差分包越大
  - 最极端情况下新旧固件完全不同，且新固件无法压缩，则差分包大小等于新固件大小
- 用户一定要管理好固件，在差分还原时，旧固件必须完全一致！不一致则无法还原

差分原理

- 新旧固件差分，即新固件与旧固件进行比较，将不相同的信息记录
- 在嵌入式平台中，由于ram资源有限，对固件进行差分不能直接整块区域进行差分
  - 例如在某些芯片平台，固件大小是2MB，但可用ram只有512KB甚至更小
  - 需要进行固件进行分块处理，例如将2MB大小的固件，拆开每块64KB进行差分



- 由此有一下结论
  - 差分块越大，差分比较的范围越大，制作出的差分包可能会更小
  - 新旧固件差异点越集中在固件的后面，差分包越小
- 在WL83平台中，可用内存较大，故可以根据实际情况来定义分块大小

使用方法

打包工具

利用isd_download.exe工具，可将两个bin文件进行差分，输出差分包

```
isd_download.exe -make-diff -old-bin <file> -new-bin <file> -output <file> [-
diff-block-size <size>] [-compress-block-size <size>]

#-old-bin：旧固件
#-new-bin：新固件
#-output：输出差分包名
#-diff-block-size：差分块大小，可选，默认64KB，压缩块越大，差分包越小
#-compress-block-size：压缩块大小，可选，默认8KB，压缩块越大，差分包越小
```

PS：压缩块大小，一般不需要配置，若平台的ram资源充足，可增大

SDK

差分还原API头文件`bspatch_user.h`

```
#ifndef __BSPATCH_USER_H__
#define __BSPATCH_USER_H__

#include <stdint.h>
#include "cpu.h"

//差分换用用户接口，用于JL差分打包工具打包后固件做差分还原

//错误码
#define BSPATCH_RET_NONE                                                0

#define BSPATCH_RET_USER_BASE
(BSPATCH_RET_NONE + 200)
#define BSPATCH_RET_USER_NO_INIT
(BSPATCH_RET_USER_BASE + 1) //未初始化
#define BSPATCH_RET_USER_READ_HEAD_ERR
(BSPATCH_RET_USER_BASE + 2) //读取差分包头失败
#define BSPATCH_RET_USER_CHECK_HEAD_ERR
(BSPATCH_RET_USER_BASE + 3) //差分包头校验失败
#define BSPATCH_RET_USER_INIT_COMF_ERR
(BSPATCH_RET_USER_BASE + 4) //初始化解压缩失败
#define BSPATCH_RET_USER_DIFF_SRC_ERR
(BSPATCH_RET_USER_BASE + 5) //差分包校验错误
#define BSPATCH_RET_USER_DEINIT_COMF_ERR
(BSPATCH_RET_USER_BASE + 6) //注销解压缩失败
#define BSPATCH_RET_USER_INIT_BSPATCH_ERR
(BSPATCH_RET_USER_BASE + 7) //注销解差分还原失败
#define BSPATCH_RET_USER_OLD_NOT_MATCH
(BSPATCH_RET_USER_BASE + 8) //旧固件不匹配
#define BSPATCH_RET_USER_INVAL_PARAM
(BSPATCH_RET_USER_BASE + 9) //无效参数
#define BSPATCH_RET_USER_REINIT
(BSPATCH_RET_USER_BASE + 10) ///重复初始化
#define BSPATCH_RET_USER_ALLOC_FAIL
(BSPATCH_RET_USER_BASE + 11) //内存申请失败
#define BSPATCH_RET_USER_GET_DIFF_INFO_ERR
(BSPATCH_RET_USER_BASE + 13) //获取差分包信息失败

/* 差分包的关键信息 */
struct bdiff_info {
    u32 u32BlockCount;  // 分块个数
    u32 u32BlockSize;   // 块大小
    u32 u32OldCrc;      // 旧版本固件数据校验码
    u32 u32OldLength;   // 旧版本数据长度
    u32 u32NewCrc;      // 新版本固件数据校验码
    u32 u32NewLength;   // 新版本数据长度
};

//差分流程句柄，用户获取并保留
struct bsuser_handle {
    struct bdiff_info info;
    void *priv; //可以保存用户的私有数据
};
```

```c
/* 用户层差分还原文件操作集合 */
struct bspatch_user_ops {

    /**
    * @brief 读取差分包源数据回调函数，用户实现
    *
    * @param[in] handle 差分还原句柄
    * @param[in] buff   底层读取数据缓存
    * @param[in] offest 读取数据偏移量
    * @param[in] len    读取的长度
    * @retval           读取到数据长度
    */
    u32 (*diff_src_read)(struct bsuser_handle *handle, u8 *buff, u32 off, u32
len);

    /**
    * @brief 获取旧固件回调函数，由用户申请提供读取buff，用户实现
    *
    * @param[in] handle 差分还原句柄
    * @param[out] buff  二级指针，应用申请获取旧固件缓存并传递给底层
    * @param[in] offest 读取数据偏移量
    * @param[in] len    读取的长度
    * @retval           读取到数据长度
    */
    u32 (*old_src_request)(struct bsuser_handle *handle, u8 **buff, u32 off, u32
len);

    /**
    * @brief 与old_src_request配套旧固件读取完毕释放回调函数，用户实现
    *
    * @param[in] handle 差分还原句柄
    * @param[in] buff   返回在old_src_request中提供的buff基地址，底层用完告知应用释放
    * @retval           0代表成功，其他代表失败
    */
    int (*old_src_release)(struct bsuser_handle *handle, u8 *buff);
};

/**
* @brief 差分还原，初始化函数
*
* @param[in] handle     二级指针，初始化成功返回本次差分还原句柄，失败返回NULL
* @param[in] user_ops   用户层差分还原文件操作集合
* @retval               成功返回0，其他则失败
*/
int bspatch_user_init(struct bsuser_handle **handle, const struct
bspatch_user_ops *user_ops, void *priv);

/**
* @brief 差分还原，注销接口，注销本次差分还原流程的资源
*
* @param[in] handle     初始化时获取到的差分还原句柄
* @retval               成功返回0，其他则失败
*/
int bspatch_user_deinit(struct bsuser_handle *handle);

/**
* @brief 差分还原出新固件，新固件读取接口，任意读取
```

```
*
* @param[in] handle      初始化时获取到的差分还原句柄
* @param[in] buff        应用读取缓存
* @param[in] offest      读取数据偏移量
* @param[in] len         读取的长度
* @retval                成功返回读取长度，其他则失败
*/
int bspatch_user_new_read(struct bsuser_handle *handle, u8 *buff, u32 off, u32
length);

/**
* @brief 差分还原出新固件，新固件读取接口，任意读取，但读取缓存是底层提供，应用层不需要提供缓存
*
* @param[in] handle      初始化时获取到的差分还原句柄
* @param[out] buff       二级指针，底层新固件缓存，底层传递buff基地址给应用
* @param[in] offest      读取数据偏移量
* @param[in] len         读取的长度
* @retval                成功返回读取长度，其他则失败
*/
int bspatch_user_request_newblock(struct bsuser_handle *handle, u8 **buff, u32
off, u32 lenght);

/**
* @brief 与bspatch_user_request_newblock配套，用完了底层提供的新固件缓存后，释放掉
*
* @param[in] handle 差分还原句柄
* @param[in] buff    返回在bspatch_user_request_newblock中提供的新固件buff基地址，应用
使用完及时释放
* @retval            0代表成功，其他代表失败
*/
int bspatch_user_release_newblock(struct bsuser_handle *handle, u8 *buff);

/**
* @brief 校验差分包的正确性，可选使用该函数
*
* @param[in] handle 差分还原句柄
* @retval            0代表成功，其他差分包校验失败
*/
int bspatch_user_diff_package_check(struct bsuser_handle *handle);

/**
* @brief 差分的crc校验算法，与打包工具crc算法一致
*
* @param[in] ptr 计算crc数据
* @param[in] len 数据长度
* @param[in] init 初始crc值
* @retval        0代表成功，其他差分包校验失败
*/
u32 bspatch_user_calcual_crc(void *ptr, u32 len, u32 init);

#endif
```

- 给底层提供操作旧固件和差分包的接口，即可获取到本次差分还原的句柄
- 可以任意还原读取到新固件的任意偏移地址数据
- `bsuser_handle`使用完毕后，务必调用`bspatch_user_deinit`进行注销，否则会出现内存泄漏问题！！

- 库中使用到的缓存，消耗内存资源总大小是：

  ```
  bspatch_ram = compress-block-size * 2 + diff-block-size
  ```

  即默认参数消耗bspatch_ram =8KB * 2 + 64KB = 80KB

  在内存较大的平台可适当调整该参数

示例

SDK

SDK的示例代码在`apps/common/example/update/diff_upgrade/diff_upgrade_resource.c`

示例中展示了差分还原预留区域`res1`过程，差分包通过http服务进行下载传输

`isd_config.ini`新增预留区域

```
[RESERVED_EXPAND_CONFIG]
USER_ADR=AUTO; [固定预留给客户，避免客户量产后，想通过升级新增重要信息的保存却没有预先预留空间]
USER_LEN=0x1000;
USER_OPT=1;

RES1_ADR=AUTO;
RES1_LEN=0x100000;
RES1_OPT=1;
RES1_FILE=res1.bin;

BAK_ADR=AUTO;
BAK_LEN=0x100000;
BAK_OPT=1;
```

差分打包

```
@echo off

@echo
********************************************************************************
@echo           SDK WL83
@echo
********************************************************************************
@echo %date%

cd %~dp0


.\isd_download.exe -make-diff -old-bin .\res1.bin -new-bin .\res1-new.bin -diff-block-size 0x100000 -compress-block-size 0x10000 -output update-res1-diff.bin

pause
```

示例中，差分块大小1MB，压缩块大小64KB

生成`update-res1-diff.bin`即可用于差分升级

结论

-

参考

-