Flash_OTP 代码说明

一、这次发布的版本为 flash 和 OTP 启动自动判断并加载功能，具体功能如下：

1. 如果是 flash 里有有效的数据，那么我们回去自动的从 flash 加载到 ram 中运行代码，有效数据的判断依据是 flash 0地址处前两个字节不为默认数据0xff；在 flash_otp 版本中，flash 总是有优先加载的权利

2. 如果 flash 里没有有效数据，那么默认的从 OTP 的0地址处加载 code，而0地址中 code 会把加载代码指向 otp 的 A 地址处(A 地址为 otp 从最后向前判断 otp 的前两个字节是不是有效字节，判断方式类似 flash，从 OTP 的0xc000地址向前判断，每次递减段的大小为0x4000)，因此我们可以复用 OTP 地址0x4000，0x8000，0xc000，启动顺序从后往前判断，遇有效地址则前面的 otp 数据都 没有用了，因此如果要复用则建议先从0x4000处写入，如果找不到有效数据，代码设置会默认从 OTP 的0x8000处启动

注：整体的 code 是支持两种模式启动的，code 中的对段代码用了一个 if else 结构 `if (first_flash_byte != 0xff && second_flash_byte != 0xff)` 这个判断 flash 启动，如果只是使用 otp 启动，则我们可以注释前面的 if 段的 code，直接把开始的那个`#if 1` 改为 `#if 0`即可，或者是用 otp_single_function 这个版本也可以

二、改动的文件有三个：

Main.c 、cstartup.S 、boot.link

具体改动的地方：

1.cstartup.S 中

去除 FLL_STACK 初始化，

去除 RAM 中 cache 空间的设置，和初始化(ZERO_TAG)

在 SETIC 后，加入 rotp 函数，在改函数中检测并读取 flash/otp 中相应的位置到 RAM 中， 覆盖0x808000到0x80ac00的内存位置。

将对 bss 段的初始化放到 rotp 完成之后， 这样 data 和 bss 段初始化都是在 RAM 被重新覆盖后（即_start_bss_ _end_bss_ _dstored_ _start_data_ _end_data_ 值覆盖为新 image 中的值）

2.Boot.link 中

增加 textotp 段在 vectors 段之后，保证初始启动时 rotp 一定是被加载到 RAM 中的

将数据段 data bss 等整体往后偏移，data 段起始地址为0x80b400,原先为0x808f00。这样前面 11KBRAM 空间(0x808000=0x80abff)放代码段，后面2KRAM 空间(0x80ac00-0x80b3ff)可给 firmware 开发者自定义代码段，再接着是3KRAM 空间(0x80b400-0x80bfff)放 data 段 bss 段和 stack 段。

3.Main.c 中

增加了 rotp 函数的实现，并定义在 textotp 段中;在 main 中我们主要做的事情是去判断 image 从什么地方加载到 ram 中去，现在是两种方式加载，flash 和 otp，其中 flash 的加载回去判断前两个字节，flash 初始化时为默认值0xff，如果读 flash 前两个字节的数据都不是0xff（不管是不是正确的 image 数据），则认为 image 需要从 flash 中加载进去（除去人为操作从 otp 启动），flash 的加载具有优先性（在 flash 和 otp 都有数据的时候从 flash 加载到 ram）；对于 otp 操作，有效数据判断仍然是2个字节，从后往前，因此如果多次利用 otp，则需要从前面开始烧录，每个 image 占的大小不超过16k，64k 的 otp 可以烧写4次；对于 otp 启动我们需要注意的是在芯片最后面写入启动命令，具体为在 otp 的0xfff4处写入060298bf ，在0xfff8处写入0003f83f。

三、文件更改的地方解释（粉红色标示改动）：

cstartup.S：

```
    .code   16

@/*****************************************************************************
**********************************

@                                       MACROS AND DEFINIITIONS

@*****************************************************************************
**********************************

                @ Mode, correspords to bits 0-5 in CPSR

    .equ MODE_BITS,       0x1F    @ Bit mask for mode bits in CPSR

    .equ IRQ_MODE,        0x12    @ Interrupt Request mode

    .equ SVC_MODE,        0x13    @ Supervisor mode

    .equ IRQ_STK_SIZE,    0x100



@*****************************************************************************
**********************************

@                                       TC32 EXCEPTION VECTORS

@*****************************************************************************
**********************************

    .section    .vectors,"ax"

    .global       __reset

    .global        __irq

    .global     __start

__start:                @ MUST,  referenced by boot.link
```

```
    .extern irq_handler
    .org 0x0
    tj    __reset
    .org 0x8
    .word    (0x544c4e4b)
    .word    (0x00880100)
    .org 0x10
    tj       __irq
    .org 0x18
@*********************************************************************
***********************************
@                               LOW-LEVEL INITIALIZATION
@*********************************************************************
***********************************
    .extern  main
@    .extern rotp  /*read data/code from otp----disable for optimize*/
    .org 0x20
__reset:
@    下面屏蔽的是对 FLL_STACK 的更改，去掉了 FLL_STK 的初始化
@    tloadr    r0, FLL_D
@    tloadr    r1, FLL_D+4
@    tloadr    r2, FLL_D+8


@FLL_STK:
```

```
@    tcmp    r1, r2
@    tjge    FLL_STK_END
@    tstorer r0, [r1, #0]
@    tadd    r1, #4
@    tj       FLL_STK
@FLL_STK_END:
    tloadr    r0, DAT0
    tmcsr    r0
    tloadr    r0, DAT0 + 8
    tmov    r13, r0
    tloadr    r0, DAT0 + 4
    tmcsr    r0
    tloadr    r0, DAT0 + 12
    tmov    r13, r0
@这个下面是去掉了 ram 的 cache 以及对 cache 的初始化
@    tloadr    r1, DAT0 + 28
@    tloadr    r2, DAT0 + 32
@
@ZERO_TAG:
@    tcmp    r1, r2
@    tjge    ZERO_TAG_END
@    tstorer    r0, [r1, #0]
@    tadd    r1, #4
```

```
@    tj       ZERO_TAG

@ZERO_TAG_END:



SETIC:

    tloadr     r1, DAT0 + 24

    tmov    r0, #64;          /* LDW modify */

    tstorerb    r0, [r1, #0]

    tmov    r0, #64;          /* LDW modify */

    tstorerb    r0, [r1, #1]

    @tmov       r0, #0;

    @tstorerb    r0, [r1, #2]

@    tjl    rotp      /* LDW add--- disable for optimize 在这里添加了我们自己的函数*/

    tjl ROTP    /*这个是把 rotp 函数固化在了 cstartup.S 文件里面，用汇编代码，因此我们的这个函数可以屏蔽了*/

    tmov    r0, #0

    tloadr    r1, DAT0 + 16

    tloadr    r2, DAT0 + 20

ZERO:

    tcmp    r1, r2

    tjge    ZERO_END

    tstorer    r0, [r1, #0]

    tadd    r1, #4

    tj       ZERO
```

```
ZERO_END:
    tloadr       r1, DATA_I
    tloadr       r2, DATA_I+4
    tloadr       r3, DATA_I+8
COPY_DATA:
    tcmp       r2, r3
    tjge       COPY_DATA_END
    tloadr       r0, [r1, #0]
    tstorer     r0, [r2, #0]
    tadd       r1, #4
    tadd       r2, #4
    tj           COPY_DATA
COPY_DATA_END:
@    tjl    rotp    /*LDW added*/
    tjl    main
END:    tj    END
    .balign    4
DAT0:
    .word    0x12              @IRQ    @0
    .word    0x13              @SVC    @4
    .word    (irq_stk + IRQ_STK_SIZE)
    .word    (0x80c000)          @12  stack end   /* LDW modify */
    .word    (_start_bss_)            @16
```

```
    .word    (_end_bss_)              @20
    .word   (0x80060c)               @24
    .word   (0x808500)               @28
    .word   (0x808600)               @32
DATA_I:
    .word    _dstored_          /* LDW data store position */
    .word    _start_data_
    .word    _end_data_
FLL_D:
    .word    0xffffffff
    .word    (_start_data_)
    .word    (0x80be00)                      /* LDW modify */
    .align 4
__irq:
    tpush       {r14}
    tpush       {r0, r1, r2, r3}
    tmrss       r0
    tpush       {r0}
    tpush       {r4, r5, r6, r7}
    tmov       r7, r9
    tmov       r6, r10
    tmov       r5, r11
    tmov       r4, r12
```

```
    tpush       {r4, r5, r6, r7}
    @ tmov       r0, ip
    @ tpush      {r0}
    tjl         irq_handler
    tpop       {r4, r5, r6, r7}
    tmov       r9,r7
    tmov       r10,r6
    tmov       r11,r5
    tmov       r12,r4
    @tpop        {r0}
    @tmov       ip, r0
    tpop       {r4, r5, r6, r7}
    tpop        {r0}
    tmssr       r0
    tpop        {r0, r1, r2, r3}
    treti       {r15}
    .org 256
@@@@@add by QS and i use the Os level and then make use O2
ROTP:
tpush    {r4, r5, r6, r7, lr}
    tloadr    r3, .L51
    tmov    r2, #196
    tstorerb    r2, [r3]
```

```
    tloadr    r3, .L51+4
    tmov    r2, #1
    tstorer    r2, [r3]
    tloadr    r0, .L51+8
    tmov    r3, #1
    tneg    r3, r3
    tstorerh    r3, [r0]
    tadd    r1, r3, #0
    tloadr    r3, .L51+12
    tstorerb    r1, [r3]
    tsub    r3, r3, #89
    tstorerb    r2, [r3]
    tloadr    r3, .L51+16
    tloadr    r2, [r3]
    tsub    sp, sp, #12
.L3:
```

下面省略.....

Boot.link 的改动：

/* to tell the linker the program begin from __start label in cstartup.s, thus do not treat it as a unused symbol */

ENTRY(__start)

```
SECTIONS

{

. = 0x0;

        .vectors :

        {

        *(.vectors)

        *(.vectors.*) /* MUST as follows, when compile with -ffunction-sections -fdata-sections, session name may changed */

        }

/*. = 0x100;*/

        .textotp :          /* add by LDW for m-otp 这里是给我们的函数分配固定的 ram 空间 */

        {

        *(.textotp)

        *(.textotp.*)
```

```
        }
/*. = 0x500;*/
        .ram_code :
        {
        *(.ram_code)
        *(.ram_code.*)
        }
        .text :
        {
        *(.text)
        *(.text.*)
        }
        .rodata :
        {
        *(.rodata)
```

```
        *(.rodata.*)

        }

PROVIDE(_dstored_ = .);

PROVIDE(_code_size_ = .);



. = 0x80b400; /* 0x100 alighned, must greater than or equal to:0x808000 + ram_code_size + irq_vector(0x100) + IC_tag(0x100) + IC_cache(0x800) == 0x808a00 + ram_code_size ，在这里腾出 data 空间13kB*/

        .data :

          AT (  _dstored_ )

          {

          PROVIDE(_start_data_ = . );

          *(.data);

          *(.data.*);

          PROVIDE(_end_data_ = . );

          }
```

```
        .bss :

        {

PROVIDE(_start_bss_ = .);

        *(.sbss)

        *(.sbss.*)

        *(.bss)

        *(.bss.*)

        }

PROVIDE(_end_bss_ = .);



}
```

Main.c 文件的改变：

只是多添加了一个函数 rotp 的实现，具体看 main.c 文件