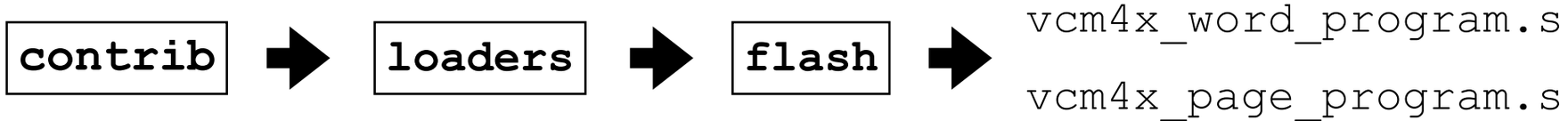
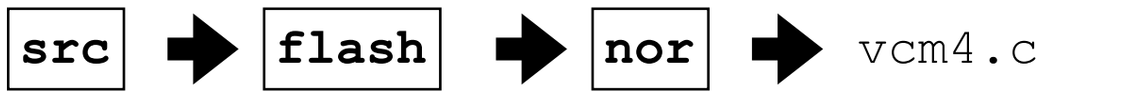
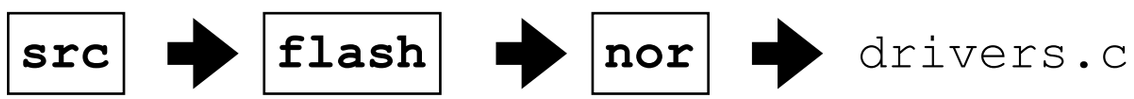
OPENOCD Porting Guide

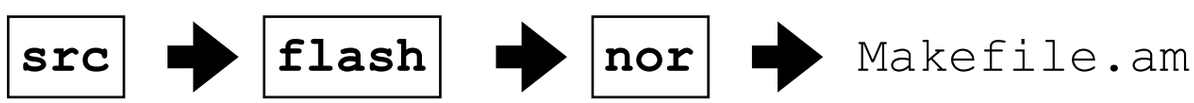
DARKO PANCEV

Important Porting Files









How to build OpenOCD image?

- Move to root directory of OpenOCD
- Run ./configure
- After finish run ‘make clean && make’
- OpenOCD image will be in ‘src’ directory as **openocd**
- To test the openocd image whether it build successfully or not, run ‘./src/openocd -v’ to see the openocd version like this:

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
```

How to flash new image with openocd using SWD interface?

- Copy your binary file to ‘tcl’ directory.
- Move to ‘tcl’ directory.
- From ‘tcl’ director run:
- ../src/openocd -f board/vertexcom-cm4.cfg -c “program your_image.bin reset exit 0”

tcl/board/vertexcom-vcm4.cfg

```
source [find interface/jlink.cfg]

transport select swd

source [find target/vcm4.cfg]
```

tcl/target/vcm4.cfg

```
source [find target/swj-dp.tcl]

if { [info exist CHIPNAME] } {
   set _CHIPNAME $CHIPNAME
} else {
   set _CHIPNAME vcm4
}

if { [info exists ENDIAN] } {
   set _ENDIAN $ENDIAN
} else {
   set _ENDIAN little
}

# Work-area is a space in RAM used for flash programming
# By default use 16kB
if { [info exists WORKAREASIZE] } {
   set _WORKAREASIZE $WORKAREASIZE
} else {
   set _WORKAREASIZE 0x10000
}

if { [info exists CPUTAPID] } {
   set _CPUTAPID $CPUTAPID
} else {
   set _CPUTAPID 0x2ba01477
}

adapter_khz 1000

swj_newdap $_CHIPNAME cpu -expected-id $_CPUTAPID

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME cortex_m -endian $_ENDIAN -chain-position $_TARGETNAME

$_TARGETNAME configure -work-area-phys 0x20000000 -work-area-size $_WORKAREASIZE -work-area-backup 0

set _FLASHNAME $_CHIPNAME.flash
flash bank $_FLASHNAME vcm4 0x00000000 0 0 0 $_TARGETNAME

reset_config none separate
```

tcl/target/vcm4.cfg

```
proc vcm4_default_examine_end {} {
   # disable watchdog timer
   mww 0x40048040 0xaa5555aa
   mww 0x40048044 0x00000000
}

proc vcm4_default_reset_start {} {
   # after reset the clock run at 6 MHz
   #adapter_khz 1000
}

proc vcm4_default_reset_init {} {
   # configure the clock to run at 150 MHz
   #mww 0x4004b004 0x0000ba52
   #mww 0x4004b008 0x05800000
   #sleep 10
   #mww 0x40047004 0x00000002
   #sleep 10

   # boost JTAG frequency
   # adapter_khz 6000
}

# default hooks
$_TARGETNAME configure -event examine-end { vcm4_default_examine_end }
$_TARGETNAME configure -event reset-start { vcm4_default_reset_start }
$_TARGETNAME configure -event reset-init { vcm4_default_reset_init }
```

src/flash/nor/vcm4.c

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "imp.h"
#include <target/algorithm.h>
#include <target/armv7m.h>
#include <helper/types.h>

#define VCM4_VERSION_ID 0x4004803C

/* vcm4 flash csr register */
enum vcm4_fcsr_registers {
    FCSR_BASE = 0x40020000,

#define FCSR_REG(offset) (FCSR_BASE + offset)

    FCSR_FLASH_CMD         = FCSR_REG(0x000),  // flash controller command register
    FCSR_FLASH_ADDR        = FCSR_REG(0x004),  // flash controller address regsiter
    FCSR_FLASH_CFG         = FCSR_REG(0x008),  // flash controller configuration register
    FCSR_FLASH_CACHE       = FCSR_REG(0x00C),  // flash controller cache configuration register
    FCSR_FLASH_SR          = FCSR_REG(0x010),  // flash controller SPI flash status register
    FCSR_FLASH_ID          = FCSR_REG(0x014),  // flash controller SPI flash ID register
    FCSR_FLASH_CACHEHIT    = FCSR_REG(0x018),  // cache hit rate counting register
    FCSR_FLASH_INVADDR_S   = FCSR_REG(0x020),  // invalid/flush cache start address
    FCSR_FLASH_INVADDR_E   = FCSR_REG(0x024),  // invalid/flush cache end address
    FCSR_FLASH_CACHE_INV   = FCSR_REG(0x028),  // invalid cache control register
    FCSR_FLASH_CACHE_FLUSH = FCSR_REG(0x02C),  // flush cache control register
    FCSR_FLASH_BUF0        = FCSR_REG(0x100),  // flash controller read/write buffer 0
    FCSR_FLASH_BUF63       = FCSR_REG(0x1FC),  // flash controller read/write buffer 63
};

/* vcm4 flash bit-fields */
#define FLASH_CMD_ACT_Pos (31)
#define FLASH_CMD_ACT_Msk (0x1 << FLASH_CMD_ACT_Pos)

#define FLASH_CMD_POLL_Pos (24)
#define FLASH_CMD_POLL_Msk (0x1 << FLASH_CMD_POLL_Pos)

#define FLASH_CMD_LENGTH_Pos (16)
#define FLASH_CMD_LENGTH_Msk (0xFF << FLASH_CMD_LENGTH_Pos)

#define FLASH_CMD_CMDMODE_Pos (12)
#define FLASH_CMD_CMDMODE_Msk (0x7 << FLASH_CMD_CMDMODE_Pos)

#define FLASH_CMD_CMDADDR4_Pos (11)
#define FLASH_CMD_CMDADDR4_Msk (0x1 << FLASH_CMD_CMDADDR4_Pos)

#define FLASH_CMD_CMDADDR_Pos (10)
#define FLASH_CMD_CMDADDR_Msk (0x1 << FLASH_CMD_CMDADDR_Pos)

#define FLASH_CMD_CMDWR_Pos (9)
#define FLASH_CMD_CMDWR_Msk (0x1 << FLASH_CMD_CMDWR_Pos)

#define FLASH_CMD_CMDDATA_Pos (8)
#define FLASH_CMD_CMDDATA_Msk (0x1 << FLASH_CMD_CMDDATA_Pos)

#define FLASH_CMD_CMDID_Pos (0)
#define FLASH_CMD_CMDID_Msk (0xFF << FLASH_CMD_CMDID_Pos)
```

src/flash/nor/vcm4.c

```
#define WINBOND_CMDID_WRSR1             0x01    // write status register 1
#define WINBOND_CMDID_WRSR2             0x31    // write status register 2
#define WINBOND_CMDID_RDSR1             0x05    // read status register 1
#define WINBOND_CMDID_RDSR2             0x35    // read status register 2
#define WINBOND_CMDID_PAGE_PROG         0x02    // page program
#define WINBOND_CMDID_READ_DATA         0x03    // read data
#define WINBOND_CMDID_WRITE_DISABLE     0x04    // write disable
#define WINBOND_CMDID_WRITE_ENABLE      0x06    // write enable
#define WINBOND_CMDID_FAST_READ         0x0B    // fast read
#define WINBOND_CMDID_RDCR              0x15    // read configuration register
#define WINBOND_CMDID_SECTOR_ERASE      0x20    // sector erase
#define WINBOND_CMDID_QUADPAGE_PROG     0x32    // quad page program
#define WINBOND_CMDID_BLOCK_ERASE_32K   0x52    // 32K block erase
#define WINBOND_CMDID_BLOCK_ERASE_64K   0xD8    // 64K block erase
#define WINBOND_CMDID_CHIP_ERASE        0xC7    // chip erase

#define WINBOND_MF  0xEF

struct vcm4_info {
    uint32_t code_page_size;
    bool probed;
    struct target *target;
};

struct vcm4_device_spec {
    uint32_t version_id;
    const char *variant;
    uint8_t sector_size_kb;
    unsigned int flash_size_kb;
};

static const struct vcm4_device_spec vcm4_known_device_table[] = {
    {
        .version_id = 0x19061001,
        .variant = "phoenix",
        .sector_size_kb = 4,
        .flash_size_kb = 2048,
    },
};
```

src/flash/nor/vcm4.c

```
static int vcm4_flash_write_enable(struct vcm4_info *chip)
{
    int res = ERROR_OK;
    uint32_t temp = 0;

    temp |= (1 << FLASH_CMD_ACT_Pos);
    temp |= (WINBOND_CMDID_WRITE_ENABLE << FLASH_CMD_CMDID_Pos);

    res = target_write_u32(chip->target, FCSR_FLASH_CMD, temp);

    res = vcm4_flash_wait_for_action_done(chip, 100);

    return res;
}


static int vcm4_flash_program_word(struct vcm4_info *chip, uint32_t addr, uint32_t data)
{
    int res = ERROR_OK;
    uint32_t temp = 0;

    res = vcm4_flash_write_enable(chip);
    if (res != ERROR_OK) {
        return res;
    }

    temp |= (1 << FLASH_CMD_ACT_Pos);
    temp |= (1 << FLASH_CMD_CMDADDR_Pos);
    temp |= (1 << FLASH_CMD_CMDWR_Pos);
    temp |= (1 << FLASH_CMD_CMDDATA_Pos);
    temp |= (3 << FLASH_CMD_LENGTH_Pos); // 1 word (4 bytes) - 1
    temp |= (WINBOND_CMDID_PAGE_PROG << FLASH_CMD_CMDID_Pos);

    res = target_write_u32(chip->target, FCSR_FLASH_BUF0, data);
    if (res != ERROR_OK) {
        return res;
    }

    res = target_write_u32(chip->target, FCSR_FLASH_ADDR, addr);
    if (res != ERROR_OK) {
        return res;
    }

    res = target_write_u32(chip->target, FCSR_FLASH_CMD, temp);
    if (res != ERROR_OK) {
        return res;
    }

    res = vcm4_flash_wait_for_action_done(chip, 100);

    return res;
}
```

src/flash/nor/vcm4.c

```
static int vcm4_flash_program_page(struct vcm4_info *chip, uint32_t addr, uint32_t bytes, uint8_t *buf)
{
    int res = ERROR_OK;

    uint32_t i = 0;
    uint32_t temp = 0;
    uint32_t data;

    res = vcm4_flash_write_enable(chip);
    if (res != ERROR_OK) {
        return res;
    }

    for (i = 0; i < bytes; i += 4) {
        memcpy(&data, &buf[i], sizeof(uint32_t));
        res = target_write_u32(chip->target, FCSR_FLASH_BUF0 + i, data);
        if (res != ERROR_OK) {
            return res;
        }
    }

    temp |= (1 << FLASH_CMD_ACT_Pos);
    temp |= (1 << FLASH_CMD_CMDADDR_Pos);
    temp |= (1 << FLASH_CMD_CMDWR_Pos);
    temp |= (1 << FLASH_CMD_CMDDATA_Pos);
    temp |= ((bytes - 1) << FLASH_CMD_LENGTH_Pos);
    temp |= (WINBOND_CMDID_PAGE_PROG << FLASH_CMD_CMDID_Pos);

    res = target_write_u32(chip->target, FCSR_FLASH_ADDR, addr);
    if (res != ERROR_OK) {
        return res;
    }

    res = target_write_u32(chip->target, FCSR_FLASH_CMD, temp);
    if (res != ERROR_OK) {
        return res;
    }

    res = vcm4_flash_wait_for_action_done(chip, 1000);

    return res;
}
```

src/flash/nor/vcm4.c

```
static int vcm4_flash_write(struct vcm4_info *chip, uint32_t offset, const uint8_t *buffer, uint32_t bytes)
{
    struct target *target = chip->target;
    uint32_t buffer_size = 16384;
    struct working_area *write_algorithm;
    struct working_area *source;
    uint32_t address = offset;
    struct reg_param reg_params[5];
    struct armv7m_algorithm armv7m_info;
    int res = ERROR_OK;

    LOG_INFO("writing buffer to flash offset=0x%"PRIx32" bytes=0x%"PRIx32, offset, bytes);

    assert(bytes % 4 == 0);

    /* allocate working area with flash programming code */
    if (target_alloc_working_area(target, sizeof(vcm4_flash_write_code), &write_algorithm) != ERROR_OK) {
        LOG_INFO("can't allocate working area for write algorithm use slow mode!");
        for (uint32_t i = 0; i < bytes; i += 256) {
            res = vcm4_flash_program_page(chip, offset, 256, (uint8_t *)buffer);
            offset += 256;
            buffer += 256;
        }
        return ERROR_OK;
    }

    res = target_write_buffer(target, write_algorithm->address,
                              sizeof(vcm4_flash_write_code),
                              vcm4_flash_write_code);

if (res != ERROR_OK) {
    return res;
}

/* memory buffer */
while (target_alloc_working_area(target, buffer_size, &source) != ERROR_OK) {
    buffer_size /= 2;
    if (buffer_size <= 256) {
        /* free working area, write algorithm already allocated */
        target_free_working_area(target, write_algorithm);
        LOG_WARNING("No large enough working area available, can't do block memory writes");
        return ERROR_TARGET_RESOURCE_NOT_AVAILABLE;
    }
}

armv7m_info.common_magic = ARMV7M_COMMON_MAGIC;
armv7m_info.core_mode = ARM_MODE_THREAD;

init_reg_param(&reg_params[0], "r0", 32, PARAM_IN_OUT);     /* buffer start, status (out) */
init_reg_param(&reg_params[1], "r1", 32, PARAM_OUT);        /* buffer end */
init_reg_param(&reg_params[2], "r2", 32, PARAM_OUT);        /* flash target address */
init_reg_param(&reg_params[3], "r3", 32, PARAM_OUT);        /* bytes  */
init_reg_param(&reg_params[4], "r4", 32, PARAM_OUT);        /* flash base */

buf_set_u32(reg_params[0].value, 0, 32, source->address);
buf_set_u32(reg_params[1].value, 0, 32, source->address + source->size);
buf_set_u32(reg_params[2].value, 0, 32, address);
buf_set_u32(reg_params[3].value, 0, 32, bytes);
buf_set_u32(reg_params[4].value, 0, 32, FCSR_BASE);
```

src/flash/nor/vcm4.c

```
        res = target_run_flash_async_algorithm(target, buffer, bytes/4, 4,
                                               0, NULL,
                                               5, reg_params,
                                               source->address, source->size,
                                               write_algorithm->address, 0,
                                               &armv7m_info);

        if (res == ERROR_FLASH_OPERATION_FAILED) {
            LOG_ERROR("error executing vcm3 flash write algorithm");
            res = ERROR_FAIL;
        }

        target_free_working_area(target, source);
        target_free_working_area(target, write_algorithm);

        destroy_reg_param(&reg_params[0]);
        destroy_reg_param(&reg_params[1]);
        destroy_reg_param(&reg_params[2]);
        destroy_reg_param(&reg_params[3]);
        destroy_reg_param(&reg_params[4]);

        return res;
    }
```

contrib/loaders/flash/vcm4x_word_program.s

```
/*
 * Params :
 * r0 = workarea start
 * r1 = workarea end
 * r2 = flash address
 * r3 = byte count
 * r4 = flash base [0x40020000]
 *
 * r6 - temp
 * r7 - rp
 * r8 - wp, tmp
 */

#define FCSR_FLASH_CMD_OFFSET   0x000
#define FCSR_FLASH_ADDR_OFFSET  0x004
#define FCSR_FLASH_BUF0_OFFSET  0x100

wait_fifo:
    ldr     r8, [r0, #0]                        /* read wp */
    cmp     r8, #0                              /* abort if wp == 0 */
    beq     exit
    ldr     r7, [r0, #4]                        /* read rp */
    cmp     r7, r8                              /* wait until rp != wp */
    beq     wait_fifo
    ldr     r6, FLASH_WRITE_ENABLE              /* write enable */
    str     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
wait_write_enable:
    ldr     r6, [r4, #FCSR_FLASH_CMD_OFFSET]    /* load CMD register */
    tst     r6, #0x80000000                     /* ACT (bit31) == 1 => operation in progress */
    bne     wait_write_enable                   /* wait more ... */
    ldr     r6, [r7], #0x4                      /* read one word from src, increment ptr */
    str     r6, [r4, #FCSR_FLASH_BUF0_OFFSET]
    mov     r6, r2
    adds    r2, #0x4                            /* increment the addres */
    str     r6, [r4, #FCSR_FLASH_ADDR_OFFSET]
    ldr     r6, FLASH_WORD_PROGRAM
    str     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
wait_write_program:
    ldr     r6, [r4, #FCSR_FLASH_CMD_OFFSET]    /* load CMD register */
    tst     r6, #0x80000000                     /* ACT (bit31) == 1 => operation in progress */
    bne     wait_write_program                  /* wait more ... */
    cmp     r7, r1                              /* wrap rp at end of buffer */
    bcc     no_wrap
    mov     r7, r0
    adds    r7, #8                              /* skip loader args */
no_wrap:
    str     r7, [r0, #4]                        /* store rp */
    subs    r3, #4                              /* decrement byte count */
    bne     wait_fifo
exit:
    bkpt    #0x00

FLASH_WRITE_ENABLE: .word 0x81000006
FLASH_WORD_PROGRAM: .word 0x81030702
```

contrib/loaders/flash/vcm4x_word_program.s

```
arm-none-eabi-gcc -c vcm4x_word_program.s

arm-none-eabi-objdump -d vcm4x_word_program.o


vcm4x_word_program.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <wait_fifo>:
   0:    f8d0 8000        ldr.w    r8, [r0]
   4:    f1b8 0f00        cmp.w    r8, #0
   8:    d01d             beq.n    46 <exit>
   a:    6847             ldr      r7, [r0, #4]
   c:    4547             cmp      r7, r8
   e:    d0f7             beq.n    0 <wait_fifo>
  10:    f8df 6034        ldr.w    r6, [pc, #52]    ; 48 <FLASH_WRITE_ENABLE>
  14:    6026             str      r6, [r4, #0]

00000016 <wait_write_enable>:
  16:    6826             ldr      r6, [r4, #0]
  18:    f016 4f00        tst.w    r6, #2147483648  ; 0x80000000
  1c:    d1fb             bne.n    16 <wait_write_enable>
  1e:    f857 6b04        ldr.w    r6, [r7], #4
  22:    f8c4 6100        str.w    r6, [r4, #256]   ; 0x100
  26:    4616             mov      r6, r2
  28:    3204             adds     r2, #4
  2a:    6066             str      r6, [r4, #4]
  2c:    4e07             ldr      r6, [pc, #28]    ; (4c <FLASH_WORD_PROGRAM>)
  2e:    6026             str      r6, [r4, #0]

00000030 <wait_write_program>:
  30:    6826             ldr      r6, [r4, #0]
  32:    f016 4f00        tst.w    r6, #2147483648  ; 0x80000000
  36:    d1fb             bne.n    30 <wait_write_program>
  38:    428f             cmp      r7, r1
  3a:    d301             bcc.n    40 <no_wrap>
  3c:    4607             mov      r7, r0
  3e:    3708             adds     r7, #8

00000040 <no_wrap>:
  40:    6047             str      r7, [r0, #4]
  42:    3b04             subs     r3, #4
  44:    d1dc             bne.n    0 <wait_fifo>

00000046 <exit>:
  46:    be00             bkpt     0x0000

00000048 <FLASH_WRITE_ENABLE>:
  48:    81000006         .word    0x81000006

0000004c <FLASH_WORD_PROGRAM>:
  4c:    81030702         .word    0x81030702
```

```
static const uint8_t vcm4_flash_write_code[] = {
    /* see contrib/loaders/flash/vcm4x_word_program.S */
    0xd0, 0xf8, 0x00, 0x80,
    0xb8, 0xf1, 0x00, 0x0f,
    0x1c, 0xd0,
    0x47, 0x68,
    0x47, 0x45,
    0xf7, 0xd0,
    0xdf, 0xf8, 0x34, 0x60,
    0x26, 0x60,

    0x26, 0x68,
    0x16, 0xf0, 0x00, 0x4f,
    0xfb, 0xd1,
    0x57, 0xf8, 0x04, 0x6b,
    0xc4, 0xf8, 0x00, 0x61,
    0x16, 0x46,
    0x04, 0x32,
    0x66, 0x60,
    0x07, 0x4e,
    0x26, 0x60,

    0x26, 0x68,
    0x16, 0xf0, 0x00, 0x4f,
    0xfb, 0xd1,
    0x8f, 0x42,
    0x01, 0xd3,
    0x07, 0x46,
    0x08, 0x37,

    0x47, 0x60,
    0x04, 0x3b,
    0xdc, 0xd1,

    0x00, 0xbe,
    0x06, 0x00, 0x00, 0x81,
    0x02, 0x07, 0x03, 0x81,
};
```

contrib/loaders/flash/vcm4x_page_program.s

```
/*
 * Params :
 * r0 = workarea start
 * r1 = workarea end
 * r2 = flash address
 * r3 = byte count
 * r4 = flash base [0x40020000]
 *
 * r5 - flash buffer, r6 - temp, r7 - rp, r8 - wp, tmp
 */

#define FCSR_FLASH_CMD_OFFSET   0x000
#define FCSR_FLASH_ADDR_OFFSET  0x004

    ldr     r5, FLASH_BUF_START_ADDR
wait_fifo:
    ldr     r8, [r0, #0]                        /* read wp */
    cmp     r8, #0                              /* abort if wp == 0 */
    beq     exit
    ldr     r7, [r0, #4]                        /* read rp */
    cmp     r7, r8                              /* wait until rp != wp */
    beq     wait_fifo
    ldr     r8, FLASH_BUF_END_ADDR
    cmp     r5, r8
    blt     fill_flash_buf                      /* jump to fill_flash_buf in case r8 is bigger */
    ldr     r6, FLASH_WRITE_ENABLE
    str     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
wait_write_enable:
    ldr     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
    tst     r6, #0x80000000
    bne     wait_write_enable
    ldr     r6, [r7], #0x4                      /* read one word from src, increment ptr */
    str     r6, [r5], #0x4                      /* store one word to flash buffer, and increment flash buf */
    mov     r6, r2
    adds    r2, #0x100
    str     r6, [r4, #FCSR_FLASH_ADDR_OFFSET]
    ldr     r6, FLASH_PAGE_PROGRAM
    str     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
wait_write_program:
    ldr     r6, [r4, #FCSR_FLASH_CMD_OFFSET]
    tst     r6, #0x80000000
    bne     wait_write_program
    ldr     r5, FLASH_BUF_START_ADDR
    b       wrap_rp_at_end_of_buffer
fill_flash_buf:
    ldr     r6, [r7], #0x4                      /* read one word from src, increment ptr */
    str     r6, [r5], #0x4                      /* store one word to flash buffer, and increment flash buf */
wrap_rp_at_end_of_buffer:
    cmp     r7, r1                              /* wrap rp at end of buffer */
    bcc     no_wrap
    mov     r7, r0
    adds    r7, #8                              /* skip loader args */
no_wrap:
    str     r7, [r0, #4]                        /* store rp */
    subs    r3, #4                              /* decrement byte count */
    bne     wait_fifo
exit:
    bkpt    #0x00

FLASH_BUF_START_ADDR: .word 0x40020100
FLASH_BUF_END_ADDR: .word 0x400201fc
FLASH_WRITE_ENABLE: .word 0x81000006
FLASH_PAGE_PROGRAM: .word 0x81ff0702
```

contrib/loaders/flash/vcm4x_page_program.s

```
vcm4x_page_program.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <wait_fifo-0x2>:
   0:     4d17                ldr       r5, [pc, #92]      ; (60 <FLASH_BUF_START_ADDR>)

00000002 <wait_fifo>:
   2:     f8d0 8000           ldr.w     r8, [r0]
   6:     f1b8 0f00           cmp.w     r8, #0
   a:     d028                beq.n     5e <exit>
   c:     6847                ldr       r7, [r0, #4]
   e:     4547                cmp       r7, r8
  10:     d0f7                beq.n     2 <wait_fifo>
  12:     f8df 8050           ldr.w     r8, [pc, #80]      ; 64 <FLASH_BUF_END_ADDR>
  16:     4545                cmp       r5, r8
  18:     db16                blt.n     48 <fill_flash_buf>
  1a:     4e13                ldr       r6, [pc, #76]      ; (68 <FLASH_WRITE_ENABLE>)
  1c:     6026                str       r6, [r4, #0]

0000001e <wait_write_enable>:
  1e:     6826                ldr       r6, [r4, #0]
  20:     f016 4f00           tst.w     r6, #2147483648    ; 0x80000000
  24:     d1fb                bne.n     1e <wait_write_enable>
  26:     f857 6b04           ldr.w     r6, [r7], #4
  2a:     f845 6b04           str.w     r6, [r5], #4
  2e:     4616                mov       r6, r2
  30:     f512 7280           adds.w    r2, r2, #256       ; 0x100
  34:     6066                str       r6, [r4, #4]
  36:     4e0d                ldr       r6, [pc, #52]      ; (6c <FLASH_PAGE_PROGRAM>)
  38:     6026                str       r6, [r4, #0]

0000003a <wait_write_program>:
  3a:     6826                ldr       r6, [r4, #0]
  3c:     f016 4f00           tst.w     r6, #2147483648    ; 0x80000000
  40:     d1fb                bne.n     3a <wait_write_program>
  42:     f8df 501c           ldr.w     r5, [pc, #28]      ; 60 <FLASH_BUF_START_ADDR>
  46:     e003                b.n       50 <wrap_rp_at_end_of_buffer>

00000048 <fill_flash_buf>:
  48:     f857 6b04           ldr.w     r6, [r7], #4
  4c:     f845 6b04           str.w     r6, [r5], #4

00000050 <wrap_rp_at_end_of_buffer>:
  50:     428f                cmp       r7, r1
  52:     d301                bcc.n     58 <no_wrap>
  54:     4607                mov       r7, r0
  56:     3708                adds      r7, #8

00000058 <no_wrap>:
  58:     6047                str       r7, [r0, #4]
  5a:     3b04                subs      r3, #4
  5c:     d1d1                bne.n     2 <wait_fifo>

0000005e <exit>:
  5e:     be00                bkpt      0x0000

00000060 <FLASH_BUF_START_ADDR>:
  60:     40020100  .word     0x40020100

00000064 <FLASH_BUF_END_ADDR>:
  64:     400201fc  .word     0x400201fc

00000068 <FLASH_WRITE_ENABLE>:
  68:     81000006  .word     0x81000006

0000006c <FLASH_PAGE_PROGRAM>:
  6c:     81ff0702  .word     0x81ff0702
```

```
static const uint8_t vcm4_flash_write_code[] = {
    /* see contrib/loaders/flash/vcm4x_page_program.S */
    0x17, 0x4d,

    0xd0, 0xf8, 0x00, 0x80,
    0xb8, 0xf1, 0x00, 0x0f,
    0x28, 0xd0,
    0x47, 0x68,
    0x47, 0x45,
    0xf7, 0xd0,
    0xdf, 0xf8, 0x50, 0x80,
    0x45, 0x45,
    0x16, 0xdb,
    0x13, 0x4e,
    0x26, 0x60,

    0x26, 0x68,
    0x16, 0xf0, 0x00, 0x4f,
    0xfb, 0xd1,
    0x57, 0xf8, 0x04, 0x6b,
    0x45, 0xf8, 0x04, 0x6b,
    0x16, 0x46,
    0x12, 0xf5, 0x80, 0x72,
    0x66, 0x60,
    0x0d, 0x4e,
    0x26, 0x60,

    0x26, 0x68,
    0x16, 0xf0, 0x00, 0x4f,
    0xfb, 0xd1,
    0xdf, 0xf8, 0x1c, 0x50,
    0x03, 0xe0,

    0x57, 0xf8, 0x04, 0x6b,
    0x45, 0xf8, 0x04, 0x6b,

    0x8f, 0x42,
    0x01, 0xd3,
    0x07, 0x46,
    0x08, 0x37,

    0x47, 0x60,
    0x04, 0x3b,
    0xd1, 0xd1,

    0x00, 0xbe,

    0x00, 0x01, 0x02, 0x40,
    0xfc, 0x01, 0x02, 0x40,
    0x06, 0x00, 0x00, 0x81,
    0x02, 0x07, 0xff, 0x81,
};
```

Phoenix and Sirius Performance Comparison

	SIRIUS (VC7300)	**PHOENIX (VC6320)**
MCU Clock Speed	150 MHz	39.3216 MHz
Adapter KHz (Interface Speed)	1000 KHz	1000 KHz
Test Image Size	~250 KB (WISUN)	~280 KB (G3PLC)
FLASH IP	Embedded Flash	SPI Flash
FastMode programming block size allocated in RAM	16 KB	16 KB
SlowMode WORD PROGRAM	~10 Seconds	~130 Seconds
SlowMode PAGE PROGRAM	-	~38 Seconds
FastMode WORD PROGRAM	~3 Seconds	~11 Seconds
FastMode PAGE PROGRAM	-	~11 Seconds