Ubooti启动文件解析

何良斌

# start.S

#include <config.h>

#include <version.h>

#if defined(CONFIG\_OMAP1610)

#include <./configs/omap1510.h>

#elif defined(CONFIG\_OMAP730)

#include <./configs/omap730.h>

#elif defined(CONFIG\_HISILICON)

#include <asm/arch/platform.h>

#endif

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Jump vector table as in table 3.1 in [1]

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*\*\*\*\*\*\*.globl声明一个符号可以被外部使用，相当于声明一个全局变量\*\*\*\*\*\*/

.globl \_start

\_start:

/\*\*\*\*\*\*\*该部分为处理器的异常向量表，地址0x0000 0000 ~ 0x0000 0020\*\*\*\*\*\*/

b reset

ldr pc, \_undefined\_instruction

ldr pc, \_software\_interrupt

ldr pc, \_prefetch\_abort

ldr pc, \_data\_abort

ldr pc, \_not\_used

ldr pc, \_irq

ldr pc, \_fiq

/\*\*\*\*\*\*\*.word伪操作是分配第一段子内存单元，并用expr初始化，此处即为存放中断函数地址\*\*\*\*\*\*/

\_undefined\_instruction:

.word undefined\_instruction

\_software\_interrupt:

.word software\_interrupt

\_prefetch\_abort:

.word prefetch\_abort

\_data\_abort:

.word data\_abort

\_not\_used:

.word not\_used

\_irq:

.word irq

\_fiq:

.word fiq

/\*\*\*\*\*\*\*.align等同于balign\*\*\*\*\*\*/

.balignl 16,0xdeadbeef

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Startup Code (reset vector)

\*

\* do important init only if we don't start from memory!

\* setup Memory and board specific bits prior to relocation.

\* relocate armboot to ram

\* setup stack

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*\*\*\*\*\*定义在/board/hi3515v100/config.mk中\*\*\*\*\*\*\*/

\_TEXT\_BASE:

.word TEXT\_BASE

.globl \_armboot\_start

\_armboot\_start:

.word \_start

.globl \_img\_end

\_img\_end:

.word \_\_img\_end

/\*

\* These are defined in the board-specific linker script.

\*/

.globl \_bss\_start

\_bss\_start:

.word \_\_bss\_start

.globl \_bss\_end

\_bss\_end:

.word \_end

#ifdef CONFIG\_USE\_IRQ

/\* IRQ stack memory (calculated at run-time) \*/

.globl IRQ\_STACK\_START

IRQ\_STACK\_START:

.word 0x0badc0de

/\* IRQ stack memory (calculated at run-time) \*/

.globl FIQ\_STACK\_START

FIQ\_STACK\_START:

.word 0x0badc0de

#endif

#ifdef CONFIG\_HISILICON

\_clr\_remap\_rom\_entry:

.word ROM\_TEXT\_ADRS + do\_clr\_remap - TEXT\_BASE

\_clr\_remap\_nand\_entry:

.word NAND\_TEXT\_ADRS + do\_clr\_remap - TEXT\_BASE

#endif

/\*

\* the actual reset code

\*/

/\*\*\*\*\*\*\*实际的处理代码从这里开始\*\*\*\*\*\*/

reset:

/\*

\* set the cpu to SVC32 mode

\*/

mrs r0,cpsr /\*\*\*\*\*\*将CPRS寄存器中的值传送给R0\*\*\*\*\*\*\*/

bic r0,r0,#0x1f /\*\*\*\*\*\*清楚指定位r0 = r0&(!0x1f)\*\*\*\*\*\*\*/

orr r0,r0,#0xd3 /\*\*\*\*\*\*或操作r0 = r0|0xd3\*\*\*\*\*\*\*/

msr cpsr,r0 /\*\*\*\*\*\*将R0的值送回CPRS寄存器中\*\*\*\*\*\*\*/

/\*

\* we do sys-critical inits only at reboot,

\* not when booting from ram!

\*/

#ifndef CONFIG\_SKIP\_LOWLEVEL\_INIT

/\*

\* flush v4 I/D caches

\*/

mov r0, #0

mcr p15, 0, r0, c7, c7, 0 /\* flush v3/v4 cache \*/

mcr p15, 0, r0, c8, c7, 0 /\* flush v4 TLB <input type="text" >\*/

/\*

\* disable MMU stuff and caches

\*/

mrc p15, 0, r0, c1, c0, 0

bic r0, r0, #0x00002300 /\* clear bits 13, 9:8 (--V- --RS) \*/

bic r0, r0, #0x00000087 /\* clear bits 7, 2:0 (B--- -CAM) \*/

orr r0, r0, #0x00000002 /\* set bit 2 (A) Align \*/

mcr p15, 0, r0, c1, c0, 0

ldr r0, =REG\_BASE\_SCTL

ldr r1, [r0, #0x8c]

and r1, r1, #0x60

lsr r4, r1, #5

@Check if I need jump to rom

@movs r0, pc, lsr#24 /\* Z flag if r0 == 0 then 1 else 0 \*/

@bne do\_clr\_remap

mov r0, pc, lsr#24

cmp r0, #0x0

bne do\_clr\_remap

cmp r4, #2 /\* boot from nand flash\*/

ldreq pc, \_clr\_remap\_nand\_entry

cmp r4, #0 /\* boot from nor flash \*/

ldreq pc, \_clr\_remap\_rom\_entry

do\_clr\_remap:

ldr r4, =REG\_BASE\_SCTL

@ldr r0, =REG\_VALUE\_SC\_NOLOCK

@str r0, [r4, #REG\_VALUE\_SC\_LOCKED]

ldr r0, [r4, #REG\_SC\_CTRL]

@Set clear remap bit.

orr r0, #(1<<8)

str r0, [r4, #REG\_SC\_CTRL]

@Setup ITCM (ENABLED, 2KB)

ldr r0, =( 1 | (MEM\_CONF\_ITCM\_SIZE<<2) | MEM\_BASE\_ITCM)

mcr p15, 0, r0, c9, c1, 1

@enable I-Cache now

mrc p15, 0, r0, c1, c0, 0

orr r0, r0, #0x00001000 /\* set bit 12 (I) I-Cache \*/

mcr p15, 0, r0, c1, c0, 0

@Setup lowlevel sp

ldr sp, =(MEM\_BASE\_ITCM + MEM\_SIZE\_ITCM)

@Check if I'm running in static mem bank

mov r0, pc, lsr#28

cmp r0, #(TEXT\_BASE>>28)

/\*

\* Go setup Memory and board specific bits prior to relocation.

\*/

beq relocate

/\*\*\*\*\*\*跳转到lowlevel\_init,该函数在board/hi3515v100/lowlevel\_init.c\*\*\*\*\*\*\*/

bl lowlevel\_init /\* go setup pll,mux,memory \*/

#endif

#ifndef CONFIG\_SKIP\_RELOCATE\_UBOOT

/\*\*\*\*\*\*重定向代码，也就是uboot将自己从flash复制到RAM中\*\*\*\*\*\*\*/

relocate: /\* relocate U-Boot to RAM \*/

ldr r0, =REG\_BASE\_SCTL

ldr r6, [r0, #0x8c]

and r6, #0x60

lsr r4, r6, #5

adr r0, \_start /\* r0 <- current position of code \*/

/\*\*\*\*\*\*获取\_TEXT\_BASE地址,定义在/board/hi3515v100/config.mk中\*\*\*\*\*\*\*/

ldr r1, \_TEXT\_BASE /\* test if we run from flash or RAM \*/

cmp r0, r1 /\* don't reloc during debug \*/

beq stack\_setup

ldr r2, \_armboot\_start

/\*\*\*\*\*\*\_img\_end定义在/board/hi3515v100/u-boot.lds中\*\*\*\*\*\*\*/

ldr r3, \_img\_end

sub r2, r3, r2 /\* r2 <- size of armboot \*/

cmp r4, #2

ldreq r2, =(CFG\_NAND\_U\_BOOT\_ONE\_PART)

add r2, r0, r2 /\* r2 <- source end address \*/

copy\_loop:

ldmia r0!, {r3-r10} /\* copy from source address [r0] \*/

stmia r1!, {r3-r10} /\* copy to target address [r1] \*/

cmp r0, r2 /\* until source end addreee [r2] \*/

ble copy\_loop

#endif /\* CONFIG\_SKIP\_RELOCATE\_UBOOT \*/

/\* Set up the stack \*/

//初始化堆栈

stack\_setup:

ldr r0, \_TEXT\_BASE /\* upper 128 KiB: relocated uboot \*/

sub r0, r0, #CFG\_MALLOC\_LEN /\* malloc area \*/

sub r0, r0, #CFG\_GBL\_DATA\_SIZE /\* bdinfo \*/

#ifdef CONFIG\_USE\_IRQ

sub r0, r0, #(CONFIG\_STACKSIZE\_IRQ+CONFIG\_STACKSIZE\_FIQ)

#endif

sub sp, r0, #12 /\* leave 3 words for abort-stack \*/

clear\_bss:

ldr r0, \_bss\_start /\* find start of bss segment \*/

ldr r1, \_bss\_end /\* stop here \*/

mov r2, #0x00000000 /\* clear \*/

clbss\_l:str r2, [r0] /\* clear loop... \*/

add r0, r0, #4

cmp r0, r1

ble clbss\_l

/\*\*\*\*\*\*跳转到start\_armboot函数，该函数在/lib\_arm/board.c\*\*\*\*\*\*\*/

ldr pc, \_start\_armboot

\_start\_armboot:

.word start\_armboot

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Interrupt handling

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

@

@ IRQ stack frame.

@

#define S\_FRAME\_SIZE 72

#define S\_OLD\_R0 68

#define S\_PSR 64

#define S\_PC 60

#define S\_LR 56

#define S\_SP 52

#define S\_IP 48

#define S\_FP 44

#define S\_R10 40

#define S\_R9 36

#define S\_R8 32

#define S\_R7 28

#define S\_R6 24

#define S\_R5 20

#define S\_R4 16

#define S\_R3 12

#define S\_R2 8

#define S\_R1 4

#define S\_R0 0

#define MODE\_SVC 0x13

#define I\_BIT 0x80

/\*

\* use bad\_save\_user\_regs for abort/prefetch/undef/swi ...

\* use irq\_save\_user\_regs / irq\_restore\_user\_regs for IRQ/FIQ handling

\*/

.macro bad\_save\_user\_regs

@ carve out a frame on current user stack

sub sp, sp, #S\_FRAME\_SIZE

stmia sp, {r0 - r12} @ Save user registers (now in svc mode) r0-r12

ldr r2, \_armboot\_start

sub r2, r2, #(CONFIG\_STACKSIZE+CFG\_MALLOC\_LEN)

sub r2, r2, #(CFG\_GBL\_DATA\_SIZE+8) @ set base 2 words into abort stack

@ get values for "aborted" pc and cpsr (into parm regs)

ldmia r2, {r2 - r3}

add r0, sp, #S\_FRAME\_SIZE @ grab pointer to old stack

add r5, sp, #S\_SP

mov r1, lr

stmia r5, {r0 - r3} @ save sp\_SVC, lr\_SVC, pc, cpsr

mov r0, sp @ save current stack into r0 (param register)

.endm

.macro irq\_save\_user\_regs

sub sp, sp, #S\_FRAME\_SIZE

stmia sp, {r0 - r12} @ Calling r0-r12

@ !!!! R8 NEEDS to be saved !!!! a reserved stack spot would be good.

add r8, sp, #S\_PC

stmdb r8, {sp, lr}^ @ Calling SP, LR

str lr, [r8, #0] @ Save calling PC

mrs r6, spsr

str r6, [r8, #4] @ Save CPSR

str r0, [r8, #8] @ Save OLD\_R0

mov r0, sp

.endm

.macro irq\_restore\_user\_regs

ldmia sp, {r0 - lr}^ @ Calling r0 - lr

mov r0, r0

ldr lr, [sp, #S\_PC] @ Get PC

add sp, sp, #S\_FRAME\_SIZE

subs pc, lr, #4 @ return & move spsr\_svc into cpsr

.endm

.macro get\_bad\_stack

ldr r13, \_armboot\_start @ setup our mode stack

sub r13, r13, #(CONFIG\_STACKSIZE+CFG\_MALLOC\_LEN)

sub r13, r13, #(CFG\_GBL\_DATA\_SIZE+8) @ reserved a couple spots in abort stack

str lr, [r13] @ save caller lr in position 0 of saved stack

mrs lr, spsr @ get the spsr

str lr, [r13, #4] @ save spsr in position 1 of saved stack

mov r13, #MODE\_SVC @ prepare SVC-Mode

@ msr spsr\_c, r13

msr spsr, r13 @ switch modes, make sure moves will execute

mov lr, pc @ capture return pc

movs pc, lr @ jump to next instruction & switch modes.

.endm

.macro get\_irq\_stack @ setup IRQ stack

ldr sp, IRQ\_STACK\_START

.endm

.macro get\_fiq\_stack @ setup FIQ stack

ldr sp, FIQ\_STACK\_START

.endm

/\*

\* exception handlers

\*/

.align 5

undefined\_instruction:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_undefined\_instruction

.align 5

software\_interrupt:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_software\_interrupt

.align 5

prefetch\_abort:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_prefetch\_abort

.align 5

data\_abort:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_data\_abort

.align 5

not\_used:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_not\_used

#ifdef CONFIG\_USE\_IRQ

.align 5

irq:

get\_irq\_stack

irq\_save\_user\_regs

bl do\_irq

irq\_restore\_user\_regs

.align 5

fiq:

get\_fiq\_stack

/\* someone ought to write a more effiction fiq\_save\_user\_regs \*/

irq\_save\_user\_regs

bl do\_fiq

irq\_restore\_user\_regs

#else

.align 5

irq:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_irq

.align 5

fiq:

get\_bad\_stack

bad\_save\_user\_regs

bl do\_fiq

#endif

# main.c

void mian\_loop(void)

{

#ifndef CFG\_HUSH\_PARSER

static char lastcommand[CFG\_CBSIZE] = { 0, };

int len;

int rc = 1;

int flag;

#endif

#if defined(CONFIG\_BOOTDELAY) && (CONFIG\_BOOTDELAY >= 0)

char \*s;

int bootdelay;

#endif

#ifdef CONFIG\_PREBOOT

char \*p;

#endif

#ifdef CONFIG\_BOOTCOUNT\_LIMIT

unsigned long bootcount = 0;

unsigned long bootlimit = 0;

char \*bcs;

char bcs\_set[16];

#endif /\* CONFIG\_BOOTCOUNT\_LIMIT \*/

#if defined(CONFIG\_VFD) && defined(VFD\_TEST\_LOGO)

ulong bmp = 0; /\* default bitmap \*/

extern int trab\_vfd (ulong bitmap);

#ifdef CONFIG\_MODEM\_SUPPORT

if (do\_mdm\_init)

bmp = 1; /\* alternate bitmap \*/

#endif

trab\_vfd (bmp);

#endif /\* CONFIG\_VFD && VFD\_TEST\_LOGO \*/

#ifdef CONFIG\_BOOTCOUNT\_LIMIT

bootcount = bootcount\_load();

bootcount++;

bootcount\_store (bootcount);

sprintf (bcs\_set, "%lu", bootcount);

setenv ("bootcount", bcs\_set);

bcs = getenv ("bootlimit");

bootlimit = bcs ? simple\_strtoul (bcs, NULL, 10) : 0;

#endif /\* CONFIG\_BOOTCOUNT\_LIMIT \*/

#ifdef CONFIG\_MODEM\_SUPPORT

debug ("DEBUG: main\_loop: do\_mdm\_init=%d\n", do\_mdm\_init);

if (do\_mdm\_init) {

char \*str = strdup(getenv("mdm\_cmd"));

setenv ("preboot", str); /\* set or delete definition \*/

if (str != NULL)

free (str);

mdm\_init(); /\* wait for modem connection \*/

}

#endif /\* CONFIG\_MODEM\_SUPPORT \*/

#ifdef CONFIG\_VERSION\_VARIABLE

{

extern char version\_string[];

setenv ("ver", version\_string); /\* set version variable \*/

}

#endif /\* CONFIG\_VERSION\_VARIABLE \*/

#ifdef CFG\_HUSH\_PARSER

u\_boot\_hush\_start ();

#endif

#ifdef CONFIG\_AUTO\_COMPLETE

install\_auto\_complete();

#endif

#ifdef CONFIG\_PREBOOT

if ((p = getenv ("preboot")) != NULL) {

# ifdef CONFIG\_AUTOBOOT\_KEYED

int prev = disable\_ctrlc(1); /\* disable Control C checking \*/

# endif

# ifndef CFG\_HUSH\_PARSER

run\_command (p, 0);

# else

parse\_string\_outer(p, FLAG\_PARSE\_SEMICOLON |

FLAG\_EXIT\_FROM\_LOOP);

# endif

# ifdef CONFIG\_AUTOBOOT\_KEYED

disable\_ctrlc(prev); /\* restore Control C checking \*/

# endif

}

#endif /\* CONFIG\_PREBOOT \*/

#if defined(CONFIG\_BOOTDELAY) && (CONFIG\_BOOTDELAY >= 0)

s = getenv ("bootdelay");

bootdelay = s ? (int)simple\_strtol(s, NULL, 10) : CONFIG\_BOOTDELAY;

debug ("### main\_loop entered: bootdelay=%d\n\n", bootdelay);

# ifdef CONFIG\_BOOT\_RETRY\_TIME

init\_cmd\_timeout ();

# endif /\* CONFIG\_BOOT\_RETRY\_TIME \*/

#ifdef CONFIG\_POST

if (gd->flags & GD\_FLG\_POSTFAIL) {

s = getenv("failbootcmd");

}

else

#endif /\* CONFIG\_POST \*/

#ifdef CONFIG\_BOOTCOUNT\_LIMIT

if (bootlimit && (bootcount > bootlimit)) {

printf ("Warning: Bootlimit (%u) exceeded. Using altbootcmd.\n",

(unsigned)bootlimit);

s = getenv ("altbootcmd");

}

else

#endif /\* CONFIG\_BOOTCOUNT\_LIMIT \*/

s = getenv ("bootcmd");

debug ("### main\_loop: bootcmd=\"%s\"\n", s ? s : "<UNDEFINED>");

/\* abortboot (bootdelay)返回1则不启动boot，否则启动boot \*/

if (bootdelay >= 0 && s && !abortboot (bootdelay)) {

/\*llz added to pass mac address from uboot to kernel by merge the value of ethaddr to bootargs.\*/

// extern void merge\_mac\_to\_bootargs(void);

// merge\_mac\_to\_bootargs();

# ifdef CONFIG\_AUTOBOOT\_KEYED

int prev = disable\_ctrlc(1); /\* disable Control C checking \*/

# endif

# ifndef CFG\_HUSH\_PARSER

run\_command (s, 0);

# else

parse\_string\_outer(s, FLAG\_PARSE\_SEMICOLON |

FLAG\_EXIT\_FROM\_LOOP);

# endif

# ifdef CONFIG\_AUTOBOOT\_KEYED

disable\_ctrlc(prev); /\* restore Control C checking \*/

# endif

}

/\* # ifdef CONFIG\_MENUKEY该部分代码不会调用 \*/

# ifdef CONFIG\_MENUKEY

if (menukey == CONFIG\_MENUKEY) {

/\* add by heliangbin 2012-7-21 \*/

printf("main\_loop #################trace2!\n");

s = getenv("menucmd");

if (s) {

# ifndef CFG\_HUSH\_PARSER

run\_command (s, 0);

/\* add by heliangbin 2012-7-21 \*/

printf("main\_loop #################trace3!\n");

# else

parse\_string\_outer(s, FLAG\_PARSE\_SEMICOLON |

FLAG\_EXIT\_FROM\_LOOP);

# endif

}

}

#endif /\* CONFIG\_MENUKEY \*/

#endif /\* CONFIG\_BOOTDELAY \*/

#ifdef CONFIG\_AMIGAONEG3SE

{

extern void video\_banner(void);

video\_banner();

}

#endif

/\*

\* Main Loop for Monitor Command Processing

\*/

#ifdef CFG\_HUSH\_PARSER

parse\_file\_outer();

/\* This point is never reached \*/

for (;;);

#else

for (;;) {

#ifdef CONFIG\_BOOT\_RETRY\_TIME

if (rc >= 0) {

/\* Saw enough of a valid command to

\* restart the timeout.

\*/

reset\_cmd\_timeout();

}

#endif

len = readline (CFG\_PROMPT);

flag = 0; /\* assume no special flags for now \*/

if (len > 0)

strcpy (lastcommand, console\_buffer);

else if (len == 0)

flag |= CMD\_FLAG\_REPEAT;

#ifdef CONFIG\_BOOT\_RETRY\_TIME

else if (len == -2) {

/\* -2 means timed out, retry autoboot

\*/

puts ("\nTimed out waiting for command\n");

# ifdef CONFIG\_RESET\_TO\_RETRY

/\* Reinit board to run initialization code again \*/

do\_reset (NULL, 0, 0, NULL);

# else

return; /\* retry autoboot \*/

# endif

}

#endif

if (len == -1)

puts ("<INTERRUPT>\n");

else

rc = run\_command (lastcommand, flag);

/\* add by heliangbin 2012-7-21 \*/

printf("main\_loop lastcommand = %s!\n", lastcommand);

//if (rc <= 0) {

/\* invalid command or not repeatable, forget it \*/

lastcommand[0] = 0;

//}

}

#endif /\*CFG\_HUSH\_PARSER\*/

}

注：涉及重要文件有：board.c main.c env\_common.c hi3515v100\_230M.h