ARM Bootloader 的实现-------C 和 ASM 混合编程

Gavin Li ver 0.1 Tuesday, June 03, 2003

Cirrus Logic 的 clps7111~Ep9312 系列 ARM core 的 CPU 内置 128 字节的 boot 程序。这个 boot 程序为把操作系统下载到裸机提供了极大的方便。这样再焊接电路板之前不用把操作系统预先写入 Flash，而且日后升级操作系统也非常方便。
这个 boot 程序的功能是：

1. 设置串行口的参数为：9600， 8N1，No FlowControl。
2. 然后送出一个 < 字符
3. 开始接收 2K 字节程序（Bootloader）
4. 送出一个 > 字符
5. 跳转去执行这 2K 的程序。

烧写操作系统的过程是:

1. 连接 ARM target 的产性口和 PC 的串行口

   ```
   ARM                 PC
   RX  ----------------  TX
   TX  ----------------  RX
   GND ----------------  GND
   ```

2. 从 BOOT 程序引导 ARM target
3. 在 Windows NT4.0 的 console 中, 设置串行口的参数 9600 8N1

   ```
   C:>mode COM2: baud=9600 data=8 parity=n stop=1
   ```

4. 在 console 中把 bootloader 送到串行口。/b 表示以二进制方式

   ```
   C:>copy /b bootldr.bin COM1:
   ```

5. 在 console 中, 根据 bootloader 的设置来调整串行口的参数 115200 8N1

   ```
   C:>mode COM2: baud=115200 data=8 parity=n stop=1
   ```

6. 在 console 中把 vxworks image 送到串行口。/b 表示以二进制方式

   ```
   C:>copy /b vxworks COM1:
   ```

7. Power off ARM target，设置其从 Flash 启动。
8. reboot，进入 VxWorks

这 2K 字节的程序就是我们说的 ARM Bootloader，它的任务一般是：

1. 必要的硬件初始化
2. 从串行口接收 VxWorks 的二进制文件，并写入 Flash
3. 在这过程中，显示一些提示信息。

像 Bootloader 这样底层的程序一般认为是要用纯汇编来写的。但是用汇编写的程序可读性肯定没有用 C 写的程序好。汇编程序不宜维护，没办法向其它类型的 CPU 去移植。这些方面 C 的程序是没有问题的！☺

那么 Bootloader 能否用纯 C 语言去写呢？不可能。因为有些操作特殊寄存器的指令也是特殊指令，用 C 是实现不了的。有些功能用 C 也是不能实现的。用 C 不能作的有：

1. 操作 CP15 寄存器的指令
2. 中断使能
3. 堆栈地址的设定

所以，只要知道这几条汇编指令可以了，不必学习所有的汇编指令。是不是上手很快呀。下面来看看我们在 Bootloader 中所用到的汇编部分：

```
  asm ("_my_start:
        mov     r14,  #0x70
        mcr     p15, 0, r14, c1, c0, 0   /* MMU disabled, 32 Bit mode, Little endian */
        mrs     r14, cpsr
        bic     r14, r14, #0x1f         /* Mask */
        orr     r14, r14, #0xc0 + 0x13  /* Diasble IRQ and FIQ, SVC32 Mode */
        msr     cpsr, r14
        ldr     r13, =0xc0020000        /* Setup Stack */
  ");
```

简单吧，比看几十 K 的汇编程序感觉好得多吧。
也许你会问：硬件的初始化怎么办？那是要操作寄存器的。
我说：看看这段 C 的代码：

```
*((DWORD*)(dwHardwareBase + HW1_SYSCON1)) = SYSCON1_VALUE;
```

明白了吧，ARM 中把寄存器映射在内存中了，就跟读写内存没有区别。
现在编写程序的问题已经全部解决了，但是否就没有问题了呢？不是。你的程序应该写成什么样呢？怎么编译生成二进制文件呢？
让我们先写一个程序试一下吧：

```
#define DWORD    unsigned int
int main(void)
{

  register DWORD  dwHardwareBase;
  asm ("_my_start:
        mov     r14,  #0x70
        mcr     p15, 0, r14, c1, c0, 0   /* MMU disabled, 32 Bit mode, Little endian */
        mrs     r14, cpsr
        bic     r14, r14, #0x1f         /* Mask */
        orr     r14, r14, #0xc0 + 0x13  /* Diasble IRQ and FIQ, SVC32 Mode */
        msr     cpsr, r14
        ldr     r13, =0xc0020000        /* Setup Stack */
  ");
  dwHardwareBase = (DWORD)0x80000000;
  return 0;
}
```

编译一下：

```
C:>ccarm -c -O2 -mcpu=arm710 -mlittle-endian -nostdlib -fvolatile -nostdinc –static
sam1.c
C:>ldarm -o sam1.out -Ttext 10000000  sam1.o
ldarm: warning: cannot find entry symbol _start; defaulting to 10000000
sam1.o(.text+0xc):fake: undefined reference to `__gccmain'
sam1.o(.text+0xc):fake: relocation truncated to fit: ARM_26 __gccmain
```

我们发现应该把 main 定义成 _start

```
#define DWORD    unsigned int
void start(void)  //gcc 需要把它定义成_start，vxworks 的 egcs 要把它定义成 start。
{
   register DWORD  dwHardwareBase;
   asm ("_my_start:
          mov     r14,  #0x70
          mcr     p15, 0, r14, c1, c0, 0   /* MMU disabled, 32 Bit mode, Little endian */
          mrs     r14, cpsr
          bic     r14, r14, #0x1f        /* Mask */
          orr     r14, r14, #0xc0 + 0x13  /* Diasble IRQ and FIQ, SVC32 Mode */
          msr     cpsr, r14
          ldr     r13, =0xc0020000       /* Setup Stack */
   ");
   dwHardwareBase = (DWORD)0x80000000;
}
```

编译一下：

```
C:>ccarm -c -O2 -mcpu=arm710 -mlittle-endian -nostdlib -fvolatile -nostdinc –static
sam1.c
C:>ldarm -o sam1.out -Ttext 10000000  sam1.o
C:>objdumparm -d sam1.out > sam1.asm
```

现在来看看汇编的源代码：

```
sam1.out:     file format coff-arm-little

Disassembly of section .text:

10000000 <_start>:
10000000:      e1a0c00d        mov     ip, sp
10000004:      e92dd800        stmdb   sp!, {fp, ip, lr, pc}
10000008:      e24cb004        sub     fp, ip, #4

1000000c <_my_start>:
1000000c:      e3a0e070        mov     lr, #70
10000010:      ee01ef10        mcr     15, 0, lr, cr1, cr0, {0}
10000014:      e10fe000        mrs     lr, cpsr
10000018:      e3cee01f        bic     lr, lr, #1f
1000001c:      e38ee0d3        orr     lr, lr, #d3
```

```
10000020:      e129f00e        msr     cpsr, lr
10000024:      e59fd000        ldr     sp, 1000002c <$$lit_   1>
10000028:      e91ba800        ldmdb   fp, {fp, sp, pc}

1000002c <$$lit_   1>:
1000002c:      c0020000        andgt   r0, r2, r0

10000030 <__CTOR_LIST__>:
10000030:      ffffffff swinv  0x00ffffff
10000034:      00000000        andeq   r0, r0, r0

10000038 <__DTOR_LIST__>:
10000038:      ffffffff swinv  0x00ffffff
1000003c:      00000000        andeq   r0, r0, r0
```

哈哈！在<_start>和<_my_start>之间的代码在干什么？是在保存现场吧，还用到了堆栈。而这时堆栈还没初始化，向堆栈里写东西那不乱套了！应该屏蔽这段代码。那就在<_start>之前放一个跳转指令跳到<_my_start>吧。

```
#define DWORD    unsigned int
asm ("
.text
.global _start
_start:
   bl      _my_start /* Omit the entry code for function c_start() */
");
void c_start(void)
{
   register DWORD  dwHardwareBase;
   asm ("_my_start:
           mov     r14,  #0x70
           mcr     p15, 0, r14, c1, c0, 0   /* MMU disabled, 32 Bit mode, Little endian */
           mrs     r14, cpsr
           bic     r14, r14, #0x1f         /* Mask */
           orr     r14, r14, #0xc0 + 0x13  /* Diasble IRQ and FIQ, SVC32 Mode */
           msr     cpsr, r14
           ldr     r13, =0xc0020000        /* Setup Stack */
   ");
   dwHardwareBase = (DWORD)0x80000000;
}
```

再编译看看汇编代码：

```
sam1.out:     file format coff-arm-little

Disassembly of section .text:

10000000 <_start>:
```

```
10000000:     eb000002      bl      10000010 <_my_start>

10000004 <_c_start>:
10000004:     e1a0c00d      mov     ip, sp
10000008:     e92dd800      stmdb   sp!, {fp, ip, lr, pc}
1000000c:     e24cb004      sub     fp, ip, #4

10000010 <_my_start>:
10000010:     e3a0e070      mov     lr, #70
10000014:     ee01ef10      mcr     15, 0, lr, cr1, cr0, {0}
10000018:     e10fe000      mrs     lr, cpsr
1000001c:     e3cee01f      bic     lr, lr, #1f
10000020:     e38ee0d3      orr     lr, lr, #d3
10000024:     e129f00e      msr     cpsr, lr
10000028:     e59fd000      ldr     sp, 10000030 <$$lit_   1>
1000002c:     e91ba800      ldmdb   fp, {fp, sp, pc}

10000030 <$$lit_   1>:
10000030:     c0020000      andgt   r0, r2, r0

10000034 <__CTOR_LIST__>:
10000034:     ffffffff swinv  0x00ffffff
10000038:     00000000      andeq   r0, r0, r0

1000003c <__DTOR_LIST__>:
1000003c:     ffffffff swinv  0x00ffffff
10000040:     00000000      andeq   r0, r0, r0
```

成功了！！！！
剩下的就是写 Flash 的程序，我就不多写了，这里给出源代码。需要说明的是：
下面的这段指令是生成 2K 的二进制文件

```
debug bootldr.bin
rcx
800
w
q
```

所有的全局变量都定义成 const，因为在 linux 中用 gcc，ld，objcopy 处理过程序后如果不是定义的是 const，生成的二进制文件很大，不知道为什么。谁知道原因请来信告知。(gavinux@yahoo.com)
在汇编代码的尾部有：<__CTOR_LIST__> 和<__DTOR_LIST__>，象是 C++的构造函数和析构函数的列表，不只是否？我现在还不知道什么编译连接的选项能把它去掉。☹

最后问大家一个问题：向 bootloader 这样对文件大小很敏感的程序（2K size）我们都能够用以上的方法实现，那么是不是 VxWorks BSP 里的所有程序都能用这个办法实现呢?

```
/*********************************************************************
;  bootloader for Cirrus Logic clps7111 ARM processor
;  Author : Gavin Li
;  web: http://embedded.homeunix.org
;  rev 0
;
;  INTEL Flash        F160C3, F320C3
;  -- For downloading vxWorks images.
;  -- The size of the .bin file has to be limited to 2K and this program
;     is to be downloaded by the on-chip boot program first.
;  -- Runs in the on-chip 2K SRAM
;  -- Works together with FTS (on the PC side)
;  -- F160C3 has 2 M memory with 8 segments and F320C3 has 8 M memory with
;     2 segments. F320C3's memory space covers F160C3 and vxWork's image is
;     normally smaller than 1M, therefor this program works for both 320
;     and 160 though it is originally for 160.
;  -- Erase 0.5M or 1M flash according to the image size
;
;  This program is free software; you can redistribute it and/or modify
;  it under the terms of the GNU General Public License as published by
;  the Free Software Foundation; either version 2 of the License, or
;  (at your option) any later version.
;
;  This program is distributed in the hope that it will be useful,
;  but WITHOUT ANY WARRANTY; without even the implied warranty of
;  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;  GNU General Public License for more details.
;
;*********************************************************************/

/*  For VxWorks
   rem asarm -EL -o start.o start.s
   ccarm -c -O2 -mcpu=arm710 -mlittle-endian -nostdlib -fvolatile -nostdinc -static
bootldr.c
   ldarm -o bootldr.out -Ttext 10000000  bootldr.o
   objdumparm -d bootldr.out > bootldr.asm
   coffarmtobin  < bootldr.out > bootldr.bin
   debug bootldr.bin
   rcx
   800
   w
   q
```

```
*/
/*  For GNU gcc
   PATH=/home/gavin/armtools/bin:$PATH
   arm-linux-gcc -O2 -mcpu=arm710 -mlittle-endian -nostdlib -fvolatile -nostdinc -static
-e _start -Ttext 10000000 -o bootldr.out bootldr.c
   arm-linux-objcopy -O binary -S bootldr.out bootldr.bin
   arm-linux-objdump -d bootldr.out bootldr.asm

   remeber to make bootldr.bin to 2K size for clps7111/clps7211 system board
You will get these warnings, Those are correct, just forget them.  :-)
   bootldr.c: In function `RecvData':
   bootldr.c:252: warning: assignment of read-only variable `g_dwSaveAddr'
   bootldr.c:253: warning: assignment of read-only variable `g_dwSaveCount'
   bootldr.c:268: warning: assignment of read-only variable `g_dwCheckSum'
*/
/*  -B/home/gavin/armtools  */

#include "clps7111.h"

/* flash command defines */
#define FLASH_COMMAND_READ_ID           0x90
#define FLASH_COMMAND_READ              0xFF
#define FLASH_COMMAND_ERASE             0x20
#define FLASH_COMMAND_CONFIRM           0xD0
#define FLASH_COMMAND_CLEAR             0x50
#define FLASH_COMMAND_WRITE             0x40
#define FLASH_COMMAND_STATUS            0x70
#define FLASH_COMMAND_SUSPEND           0xB0
#define FLASH_COMMAND_RESUME            0xD0
#define FLASH_COMMAND_CONFIG_SETUP      0x60
#define FLASH_COMMAND_LOCK_BLOCK        0x01
#define FLASH_COMMAND_UNLOCK_BLOCK      0xD0

#define FLASH_STATUS_READY              0x80
#define FLASH_STATUS_ERASE_SUSPENDED    0x40
#define FLASH_STATUS_WRITE_SUSPENDED    0x04
#define FLASH_STATUS_ERROR          0x3E

#define FLASH_STATUS_ERASE_ERROR        0x3A
#define FLASH_STATUS_PROGRAMING_ERROR   0x1A

#define FLASH_START_ADDRESS             0x70000000
#define FLASH_CHECK_EMPTY_END_ADDRESS   0x70100000
#define FLASH_ERASE_8K_BLOCKS           8
#define FLASH_ERASE_64K_BLOCKS          31
#define DRAM_BLOCK_SIZE                 0x40000     /* 256KB */
```

```
#define SYSCON1_VALUE         0x00840100   /* EXCKEN, BZMOD=TOG,
UART1EN */
#define SYSCON2_VALUE         0x00000006   /* 16bit DRAM, KBD6 */
#define MEMCFIG1_VALUE        0x03010100   /* CS3: 8bitC, CS2,1: 32bit, CS0:
16bit */
#define DRAM_FRESH_RATE       0x83
/*#define UART1_CONFIG          0x74005 */     /*  38400  8N1  Enable FIFO */
/*#define UART1_CONFIG          0x60003 */     /*  57600  8N1  Enable FIFO */
#define UART1_CONFIG          0x74001      /* 115200  8N1  Enable FIFO */
#define SYSCON2_VALUE_BEEP_ON  0x00004006   /* BuzzerFreq, 16bit DRAM,
KBD6 */
#define SYSCON2_VALUE_BEEP_OFF 0x00000006   /* 16bit DRAM, KBD6 */

#define HbDdrAData            0x00    /* all inputs */
#define HbDdrBData            0x14    /* bit 2,4 output */
#define HbDdrDData            0x00    /* all output  (direction def reversed) */
#define HbDdrEData            0x0F    /* all output */
#define HbDrBData             0x00    /* HbDrBData & HbDrDData disable printer heating
*/
#define HbDrDData             0x00
#define HbDrEData             0x02    /* UCHARge battery, disable step motor */

#define UART1_TX_FIFO_FULL     (1<<23)
#define UART1_TX_BUSY          (1<<11)
#define UART1_RX_FIFO_EMPTY    (1<<22)
#define BUZZER_TOGGLE          (1<<9)

#define UCHAR    unsigned char
#define DWORD    unsigned int

/* Function prototypes */
/*
void   SetCp15(DWORD dwCp15Val);
DWORD  ReadCp15();
*/
void   PrintMsg(UCHAR * pszMsg);
void   RecvData(void);
DWORD  RecvDword(void);
void   SendChar(register UCHAR ch);
UCHAR  ReceiveChar(void);
void   PrintDword(DWORD dwValue);
void   ToAscii(UCHAR ch, UCHAR * pcAscii);
void   Beep(void);
UCHAR* FlashBlockErase(UCHAR * pucBlockAddr, DWORD dwBlocks, DWORD
dwBlockSize);
```

```
void   FlashChipErase(void);
void   FlashCheckEmpty(void);
void   FlashWrite(void);
void   FlashChipUnlock(void);
void   FlashErrorHandler(UCHAR * pszMsg, UCHAR * pcAddr2Dump);

/* Globle Variable */
const DWORD  g_dwDramBase[] = {0xc0080000, 0xc0200000, 0xc0280000,
0xc0800000, 0xc0880000, 0xc0a00000, 0xc0a80000}; /*DRAM_BLOCK_SIZE each*/
const DWORD  g_dwSaveAddr;  /* Will changed by RecvData() */
const DWORD  g_dwSaveCount;   /* Will changed by RecvData() */
const DWORD  g_dwCheckSum;
const UCHAR  g_AsciiTable[]         = "0123456789abcdef";
const UCHAR  g_szByteMsg[]          = "0xXX ";
const UCHAR  g_szWordMsg[]          = "0xXXXXYYYY\n";
const UCHAR  g_szBootLdrStart[]     = "+++START LOADER\r\n";
const UCHAR  g_szReceiving[]        = "Receiving data ...\r\n";
const UCHAR  g_szReceived[]         = "Receiving finished!\n";
const UCHAR  g_szIds[]              = "Device code/Manufacturer code: ";
const UCHAR  g_szUnlockingFlash[]   = "Unlocking Flash ...\n";
const UCHAR  g_szErasingFlash[]     = "Erasing flash ...\n";
const UCHAR  g_szProgrammingFlash[] = "Programming Flash ...\n";
const UCHAR  g_szDone[]             = "Data has been programmed into flash\n";
const UCHAR  g_szBootLdrEnd[]       = "+++BOOTLOADER END\n";
const UCHAR  g_szEraseFlashError[]  = "Error erasing flash\n";
const UCHAR  g_szFlashNotEmpty[]    = "Error flash not empty\n";
const UCHAR  g_szProgramFlashError[] = "Error programming flash\n";
const UCHAR  g_szDramError[]        = "DRAM Error!\n";
/* Implementation */
asm ("
.text
.global _start
_start:
   bl      _my_start /* Omit the entry code for function c_start() */
");
void c_start(void)
{

   register DWORD* pdwFlashStartAddr;
   register DWORD* pdwFlashEndAddr;
   register DWORD* pdwDramStartAddr;
   register DWORD  dwHardwareBase;
   asm ("_my_start:
          mov     r14,  #0x70
          mcr     p15, 0, r14, c1, c0, 0   /* MMU disabled, 32 Bit mode, Little endian */
          mrs     r14, cpsr
```

```
            bic     r14, r14, #0x1f         /* Mask */
            orr     r14, r14, #0xc0 + 0x13  /* Diasble IRQ and FIQ, SVC32 Mode */
            msr     cpsr, r14
            ldr     r13, =0xc0020000        /* Setup Stack */
    ");
    dwHardwareBase = (DWORD)HW1base;
    *((DWORD*)(dwHardwareBase + HW1_INTMR1)) = 0; /* disable Interrupt */
    *((DWORD*)(dwHardwareBase + (0x1000+HW2_INTMR2))) = 0; /* disable
Interrupt */
    *((DWORD*)(dwHardwareBase + HW1_SYSCON1)) = SYSCON1_VALUE;
    *((DWORD*)(dwHardwareBase + (0x1000+HW2_SYSCON2))) =
SYSCON2_VALUE;
    *((DWORD*)(dwHardwareBase + HW1_MEMCFG1)) = MEMCFIG1_VALUE;
    *((DWORD*)(dwHardwareBase + HW1_DRFPR))   = DRAM_FRESH_RATE;
    /* **** We can call functions from here! **** */
    PrintMsg((UCHAR*)&g_szReceiving);
    PrintMsg((UCHAR*)&g_szBootLdrStart);
    Beep();
    *((DWORD*)(dwHardwareBase + HW1_UBRLCR1)) = UART1_CONFIG; /*
115200 8n1 */
/*
    make sure that before reinitializing UART there is enough
    time to send out last message
    So far Beep gives enough time
*/

    pdwFlashStartAddr = (DWORD *)FLASH_START_ADDRESS;

    RecvData();  /* receive data */
    PrintMsg((UCHAR*)&g_szReceived);

    PrintMsg((UCHAR*)&g_szIds);  /* display Flash Id message */
    *((UCHAR *)pdwFlashStartAddr) = FLASH_COMMAND_READ_ID;  /* read ID */
    PrintDword(*pdwFlashStartAddr);

    PrintMsg((UCHAR*)&g_szUnlockingFlash);
    FlashChipUnlock();
    PrintMsg((UCHAR*)&g_szErasingFlash);
    FlashChipErase();
    FlashCheckEmpty();
    PrintMsg((UCHAR*)&g_szProgrammingFlash);
    FlashWrite();
    PrintMsg((UCHAR*)&g_szDone);
    PrintMsg((UCHAR*)&g_szBootLdrEnd);
    Beep();
    while(1);
```

```
}
void PrintMsg(UCHAR * pszMsg)
{
   UCHAR ch;
   for (;;)
   {
        ch = *pszMsg++;
        if (ch) SendChar(ch);
        else break;
   }
}
void PrintDword(DWORD dwValue)
{
   register DWORD dw;
   for ( dw = 0; dw < 4; dw++)
   {
        ToAscii((dwValue>>(dw*8)), (UCHAR*)&g_szWordMsg + 8 - 2*dw);
   }
   PrintMsg((UCHAR*)&g_szWordMsg);
}
void ToAscii(UCHAR ch, UCHAR * pcAscii)
{
   pcAscii[0] = g_AsciiTable[(ch>>4) & 0x0F];
   pcAscii[1] = g_AsciiTable[ch & 0x0F];
}
/*************************************************************/
void RecvData()
{
   register DWORD dwBlockCounter, dwInBlockCounter;
   register DWORD dwCharCounter;
   register UCHAR * pucBuf;
   register DWORD dwCheckSum;
   register UCHAR uc;

   g_dwSaveAddr  = RecvDword();    /* buffer address */
   g_dwSaveCount = dwCharCounter = RecvDword();    /* count */
   dwCheckSum = dwBlockCounter = 0;
   while (1)
   {
        pucBuf = (UCHAR*)(g_dwDramBase[dwBlockCounter++]);
        for (dwInBlockCounter=0;
dwInBlockCounter<(DWORD)DRAM_BLOCK_SIZE; dwInBlockCounter++)
        {
           uc = ReceiveChar();
           *pucBuf = uc;
           if (uc != *pucBuf) {PrintMsg((UCHAR*)g_szDramError); while(1);}
```

```
            pucBuf++;
            dwCheckSum += uc;
            --dwCharCounter;
            if ((dwCharCounter == 0) || (dwCharCounter & 0x80000000))  /*
(int)dwCharCounter <= 0 */
            {
                  g_dwCheckSum = dwCheckSum;
                  return;
            }
         }
   }
}
DWORD RecvDword(void)
{
   register DWORD  i;
   register DWORD dw;
   for (i=dw=0; i<4; i++)
   {
         dw |= ReceiveChar()<<(8*i);
   }
   return dw;
}
/*************************************************************************
***
;
;      UART driver
;
;*************************************************************************
***/
void SendChar(register UCHAR ch)
{
   register DWORD dwReg;
   dwReg = HW1base;
   while (*((DWORD*)(dwReg+HW1_SYSFLG1)) & (UART1_TX_FIFO_FULL));
   while (*((DWORD*)(dwReg+HW1_SYSFLG1)) & (UART1_TX_BUSY));
   *((UCHAR*)(dwReg + HW1_UARTDR1)) = ch;
}
UCHAR ReceiveChar(void)
{
   register DWORD dwReg;
   dwReg = HW1base;
   while (*((DWORD*)(dwReg + HW1_SYSFLG1)) & (UART1_RX_FIFO_EMPTY));
   return *((UCHAR*)(dwReg + HW1_UARTDR1));
}
void Beep(void)
{
```

```
   register DWORD  i;
   register DWORD dwReg;
   dwReg = (DWORD)(HW1base + 0x1000 + HW2_SYSCON2);
   *((DWORD*)dwReg) = SYSCON2_VALUE_BEEP_ON;
   for (i=0; i<0x30000; i++);
   *((DWORD*)dwReg) = SYSCON2_VALUE_BEEP_OFF;
/*
   dwReg = (DWORD)(HW1base + HW1_SYSCON1);
   for (j=0x600; j!=0; j--)
   {
         *pdwReg |= BUZZER_TOGGLE;
         for (i=0x100; i!=0; i--);
         *pdwReg &= ~BUZZER_TOGGLE;
         for (i=0x100; i!=0; i--);
   }
*/
}
/*---------------------------------------------------------------------------*/
void FlashChipUnlock(void)
{
   register DWORD dw;
   register UCHAR* pucFlashAddr;
   register UCHAR  ucCmdSetup  = FLASH_COMMAND_CONFIG_SETUP ;
   register UCHAR  ucCmdUnlock = FLASH_COMMAND_UNLOCK_BLOCK;
   pucFlashAddr = (UCHAR*)FLASH_START_ADDRESS;
   for (dw=0; dw<FLASH_ERASE_8K_BLOCKS; dw++)
   {
         *pucFlashAddr = ucCmdSetup;
         *pucFlashAddr = ucCmdUnlock;
         pucFlashAddr += 0x2000;
   }
   for (dw=0; dw<FLASH_ERASE_64K_BLOCKS; dw++)
   {
         *pucFlashAddr = ucCmdSetup;
         *pucFlashAddr = ucCmdUnlock;
         pucFlashAddr += 0x10000;
   }
   *((UCHAR*)FLASH_START_ADDRESS) =
(UCHAR)FLASH_COMMAND_READ;
}
UCHAR* FlashBlockErase(UCHAR * pucBlockAddr, DWORD dwBlocks, DWORD
dwBlockSize)
{
   register UCHAR ucStatus;
   register DWORD dw;
   for (dw=0; dw<dwBlocks; dw++)
```

```
   {
        *((DWORD*)pucBlockAddr) = (FLASH_COMMAND_CONFIRM<<16) |
FLASH_COMMAND_ERASE;
        *pucBlockAddr = FLASH_COMMAND_STATUS;
        do {
           ucStatus = *pucBlockAddr;
        } while (!(ucStatus & FLASH_STATUS_READY));
        if (ucStatus & FLASH_STATUS_ERROR)
FlashErrorHandler((UCHAR*)&g_szEraseFlashError, pucBlockAddr);
        pucBlockAddr += dwBlockSize;
   }
   return pucBlockAddr;
}
void FlashChipErase(void)
{
   FlashBlockErase(FlashBlockErase((UCHAR*)FLASH_START_ADDRESS,
FLASH_ERASE_8K_BLOCKS, 0x2000),
                   FLASH_ERASE_64K_BLOCKS,
                   0x10000);
   *((UCHAR*)FLASH_START_ADDRESS) =
(UCHAR)FLASH_COMMAND_READ;
}
void FlashCheckEmpty()
{
   register DWORD dwStartAddr;
   dwStartAddr = (DWORD)FLASH_START_ADDRESS;
   do
   {
        if (*((DWORD*)dwStartAddr) != (DWORD)(0xFFFFFFFF))
           FlashErrorHandler((UCHAR*)&g_szFlashNotEmpty, (UCHAR
*)dwStartAddr);
        dwStartAddr += 4;
   }
   while (dwStartAddr < (DWORD)FLASH_CHECK_EMPTY_END_ADDRESS);
}
void FlashWrite()
{
   register DWORD dwValue;
   register DWORD dwBlockCounter, dwInBlockCounter;
   register DWORD dwCheckSum;
   DWORD * pdwFlashAddr;
   DWORD * pdwBufAddr;
   int     iLength;
   pdwFlashAddr = (DWORD*)FLASH_START_ADDRESS;
   iLength = g_dwSaveCount;
   dwBlockCounter = 0;
```

```
   while (1)
   {
         pdwBufAddr = (DWORD*)(g_dwDramBase[dwBlockCounter++]);
         for (dwInBlockCounter=0;
dwInBlockCounter<((DWORD)DRAM_BLOCK_SIZE/4); dwInBlockCounter++)
         {
            dwValue = *pdwBufAddr++;
            *((UCHAR*)pdwFlashAddr) = FLASH_COMMAND_WRITE;
            *pdwFlashAddr = ((FLASH_COMMAND_STATUS<<16) |
(dwValue&0x0000FFFF));
            while (!(*pdwFlashAddr & FLASH_STATUS_READY));

            *pdwFlashAddr = (dwValue&0xFFFF0000) | FLASH_COMMAND_WRITE;
            *((UCHAR*)pdwFlashAddr) = FLASH_COMMAND_STATUS;
            while (!(*pdwFlashAddr & FLASH_STATUS_READY));
            pdwFlashAddr++;
            if ((iLength-=4) <= 0)
            {
                  *((UCHAR*)FLASH_START_ADDRESS) =
(UCHAR)FLASH_COMMAND_READ;
/*
                  dwCheckSum = 0;
                  pdwFlashAddr = (DWORD*)FLASH_START_ADDRESS;
                  for (iLength=(int)g_dwSaveCount; iLength!=0; iLength--)
                  {
                     dwCheckSum += *((UCHAR*)pdwFlashAddr);
                     (UCHAR*)pdwFlashAddr++;
                  }
                  if (dwCheckSum != g_dwCheckSum)
PrintMsg((UCHAR*)g_szProgramFlashError);
*/
                  return;
            }
         }
   }
}
void FlashErrorHandler(UCHAR * pszMsg, UCHAR * pcAddr2Dump)
{
   register DWORD dw, dw1;
   *((UCHAR*)FLASH_START_ADDRESS) =
(UCHAR)FLASH_COMMAND_READ;
   PrintMsg(pszMsg);
   PrintDword((DWORD)pcAddr2Dump);
   for (dw=0; dw<16; dw++)
   {
         for (dw1=0; dw1<16; dw1++)
```

```
        {
          ToAscii(*pcAddr2Dump++, (UCHAR*)&g_szByteMsg + 2);
          PrintMsg((UCHAR*)&g_szByteMsg);
        }
        PrintMsg("\r\n");
    }
    Beep();
    PrintMsg((UCHAR*)&g_szBootLdrEnd);
    while(1);
}
/**** end ****/
/*****************************************************************
void SetCp15(DWORD uCp15Val)
{
    asm ("mcr p15, 0, %0, c1, c0, 0;" : : "r"(uCp15Val) );
}
DWORD ReadCp15()
{
    register DWORD x;
    asm ("mrc p15, 0, %0, c1, c0, 0;" : "=r"(x) : );
    return x;
}
*****************************************************************/



/* ***************************************************************
clps7111.h
Cirrus CLPS7111 (ARM710A core) CPU registers defination
***************************************************************/
#ifndef __CLPS7111_H__
#define __CLPS7111_H__

#define         HW1base      0x80000000

//ports A-D all 8 bits wide
#define         HW1_PADR     0x00    // Port A data register
#define         HW1_PBDR     0x01    // Port B data register
#define         HW1_PDDR     0x03    // Port D data register
#define         HW1_PADDR    0x40    // Port A data direction register
#define         HW1_PBDDR    0x41    // Port B data direction register
#define         HW1_PDDDR    0x43    // Port D data direction register

//port E 3 bits wide
#define         HW1_PEDR     0x80    // Port E data register
#define         HW1_PEDDR    0xc0    // Port E data direction register
```

```
#define         HW1_SYSCON1   0x100   // System control register [32]
#define         HW1_SYSFLG1   0x140   // System status flags [RO,32]
#define         HW1_MEMCFG1   0x180   // Exp/ROM mem cfg register 1 [32]
#define         HW1_MEMCFG2   0x1c0   // Exp/ROM mem cfg register 2 [32]
#define         HW1_DRFPR     0x200   // DRAM refresh period reg. [8]
#define         HW1_INTSR1    0x240   // Interrupt status register [RO,16]
#define         HW1_INTMR1    0x280   // Interrupt mask register [16]
#define         HW1_LCDCON    0x2c0   // LCD control register [32]
#define         HW1_TC1D      0x300   // data to/from TC1 [16]
#define         HW1_TC2D      0x340   // data to/from TC2 [16]
#define         HW1_RTCDR     0x380   // real time clock data reg. [32]
#define         HW1_RTCMR     0x3c0   // real time clock match reg. [32]
#define         HW1_PMPCON    0x400   // DC to DC pump control reg. [12]
#define         HW1_CODR      0x440   // CODEC data I/O reg. [8]
#define         HW1_UARTDR1   0x480   // UART1 FIFO data register [8]
#define         HW1_UBRLCR1   0x4c0   // UART1 bit rate and line ctrl reg. [32]
#define         HW1_SYNCIO    0x500   // SSI data reg. for master only [16]
#define         HW1_PALLSW    0x540   // LSW of LCD palette register [32]
#define         HW1_PALMSW    0x580   // MSW of LCD palette register [32]
#define         HW1_STFCLR    0x5c0   // clear startup reason flags [WO,-]
#define         HW1_BLEOI     0x600   // clear batt. low interrupt [WO,-]
#define         HW1_MCEOI     0x640   // clear media change interrupt [WO,-]
#define         HW1_TEOI      0x680   // clear tick/watchdog interrupt [WO, -]
#define         HW1_TC1EOI    0x6c0   // clear TC1 interrupt [WO,-]
#define         HW1_TC2EOI    0x700   // clear TC2 interrupt [WO,-]
#define         HW1_RTCEOI    0x740   // clear RTC match interrupt [WO,-]
#define         HW1_UMSEOI    0x780   // clear UART modem stat changed [WO,-]
#define         HW1_COEOI     0x7c0   // clear CODEC sound interrupt [WO,-]
#define         HW1_HALT      0x800   // enter idle state [WO,-]
#define         HW1_STDBY     0x840   // enter standby state [WO,-]

#define         HW2base       (HW1base|0x1000)
#define         HW2_FRBADDR   0x000   // LCD frame buffer start address [24]
#define         HW2_SYSCON2   0x100   // System control register 2
#define         HW2_SYSFLG2   0x140   // System status register 2
#define         HW2_INTSR2    0x240   // Interrupt status register [RO]
#define         HW2_INTMR2    0x280   // Interrupt mask register
#define         HW2_UARTDR2   0x480   // UART2 data register [8/11]
#define         HW2_UBRLCR2   0x4c0   // UART2 control register [32]
#define         HW2_SRXEOF    0x600   // Write to clear Rx FIFO overflow flag
#define         HW2_KBDEOI    0x700   // Write to clear keyboard interrupt [WO,-]

//these registers have the same offset in each bank and share some bitfields
//it's useful to give them a generic name.
#define         HW_SYSCON     HW1_SYSCON1
#define         HW_SYSFLG     HW1_SYSFLG1
```

```
#define         HW_UBRLCR     HW1_UBRLCR1
#define         HW_UARTDR     HW1_UARTDR1

//SYSCON1/SYSCON2 bitfield
//bits 0-3: keyboard scan
#define         SYSCON1_KBS_MASK     15
#define         SYSCON1_KBS_ALLHIGH   0
#define         SYSCON1_KBS_ALLLOW    1
#define         SYSCON1_KBS_ALLHIZ    2
#define         SYSCON1_KBS_COL0      8
#define         SYSCON1_KBS_COL1      9
#define         SYSCON1_KBS_COL2     10
#define         SYSCON1_KBS_COL3     11
#define         SYSCON1_KBS_COL4     12
#define         SYSCON1_KBS_COL5     13
#define         SYSCON1_KBS_COL6     14
#define         SYSCON1_KBS_COL7     15

#define         SYSCON1_TC1M        (1<<4)  // TC1 mode (set=prescale)
#define         SYSCON1_TC1S        (1<<5)  // TC1 source (set=512KHz)
#define         SYSCON1_TC2M        (1<<6)  // TC2 mode
#define         SYSCON1_TC2S        (1<<7)  // TC2 source
#define         SYSCON1_UART1EN      (1<<8)  // enable UART1
#define         SYSCON1_BZTOG       (1<<9)  // drive buzzer directly
#define         SYSCON1_BZMOD       (1<<10) // 0: buzzer uses BZTOG
#define         SYSCON1_DBGEN       (1<<11) // debug mode
#define         SYSCON1_LCDEN       (1<<12) // enable LCD controller
#define         SYSCON1_CDENTX      (1<<13) // CODEC i/f enable Tx
#define         SYSCON1_CDENRX      (1<<14) // CODEC i/f enable Rx
#define         SYSCON1_SIREN       (1<<15) // HP SIR encoding enable
#define         SYSCON1_ADCKSEL_MASK (3<<16) // ADC clock select
#define         SYSCON1_ADCKSEL_SHIFT     16
#define         SYSCON1_EXCKEN      (1<<18) // external expansion clock enable
#define         SYSCON1_WAKEDIS     (1<<19) // disable wakeup switchon
#define         SYSCON1_IRTXM       (1<<20) // IrDA Tx mode strategy

//SYSCON2
//bit 0: serial interface select
#define         SYSCON2_SERSEL      1
#define         SYSCON2_CODEC       1       //  CODEC Enable
#define         SYSCON2_KBD6        (1<<1)  // if high only pins 0 to 5 of pA are kbd
#define         SYSCON2_DRAMWID     (1<<2)  // 1=16-bit DRAM, 0-32-bit DRAM
#define         SYSCON2_KBWDIS      (1<<3)  // enforce IRQ mask register for
wakeup
#define         SYSCON2_PCMCIA1     (1<<5)  // enable PCMCIA interface 1 (cs4)
#define         SYSCON2_PCMCIA2     (1<<6)  // enable PCMCIA interface 2 (cs5)
```

```
#define         SYSCON2_UART2EN     (1<<8)  // enable UART2
#define         SYSCON2_OSTB        (1<<12) // twiddle clocks somehow
#define         SYSCON2_CLKENSL     (1<<13) // high/low: output run/clken on
run/clken pin

//SYSFLG/SYSFLG2 bitfield
#define         SYSFLG1_MCDR        (1<<0)  // media changed direct read
#define         SYSFLG1_DCDET       (1<<1)  // 1: mains power
#define         SYSFLG1_WUDR        (1<<2)  // wakeup direct read
#define         SYSFLG1_WUON        (1<<3)  // started by wakeup

//bits 4-7: display ID nibble
#define         SYSFLG1_DID_mask    (15<<4)

#define         SYSFLG1_CTS         (1<<8)  // UART1 CTS
#define         SYSFLG1_DSR         (1<<9)  // UART1 DSR
#define         SYSFLG1_DCD         (1<<10) // UART1 DCD
#define         SYSFLG1_UBUSY1      (1<<11) // UART1 busy

#define         SYSFLG1_NBFLG       (1<<12) // new battery (clear w/ STFCLR)
#define         SYSFLG1_RSTFLG      (1<<13) // reset pressed (clear w/ STFCLR)
#define         SYSFLG1_PFFLG       (1<<14) // power fail (clear w/ STFCLR)
#define         SYSFLG1_CLDFLG      (1<<15) // power on reset (clear w/ STFCLR)

//bits 16-21: number of 64Hz ticks since last RTC increment
#define         SYSFLG1_RTCDIV_mask  (63<<16)

#define         SYSFLG1_URXFE1      (1<<22) // UART rx FIFO empty
#define         SYSFLG1_UTXFF1      (1<<23) // UART tx FIFO full
#define         SYSFLG1_CRXFE       (1<<24) // CODEC rx FIFO empty
#define         SYSFLG1_CTXFF       (1<<25) // CODEC tx FIFO full
#define         SYSFLG1_SSIBUSY     (1<<26) // 1: SSI is shifting data

//0: data clear to read

#define         SYSFLG1_BOOTBIT0    (1<<27)
#define         SYSFLG1_BOOTBIT1    (1<<28)

//      bootbit0 bootbit1 boot option
//      0       0   32-bit
//      0       1   8-bit
//      1       0   16-bit
//      1       1   reserved

#define         SYSFLG1_RESERVED    (1<<29)
```

```
//bits 30-31: version ID
#define         SYSFLG1_VERID_mask    (3<<30)


#define         SYSFLG2_CHKMODE       (1<<6)   // 18/13 mode flag
#define         SYSFLG2_UBUSY2        (1<<11)  // UART2 busy
#define         SYSFLG2_URXFE2        (1<<22)  // UART2 rx FIFO empty
#define         SYSFLG2_UTXFF2        (1<<23)  // UART2 tx FIFO full

//        ;; MEMCFG1
//        ;; Expansion and ROM selects
//        ;; byte0 : Ncs0
//        ;; byte1 : Ncs1
//        ;; byte2 : Ncs2
//        ;; byte3 : Ncs3
//        ;; MEMCFG2
//        ;; Expansion and ROM selects
//        ;; byte0 : cs4
//        ;; byte1 : cs5
//        ;; byte2 : RESERVED (local SRAM)
//        ;; byte3 : Boot ROM (cs7)
//        ;; chip select bytes (cs7, cs4-5 in non-PCMCIA mode, Ncs0-3)
//        ;; b0-1: bus width
//        ;; maps onto expansion transfer mode according to value here
//        ;; and BOOTBIT{0,1} (=BOOTWID[1:0])
//        ;; b2-3: random access wait state
//        ;;  value      #states (nS) speed
//        ;;  0          4      250
//        ;;  1          3      200
//        ;;  2          2      150
//        ;;  3          1      100
//        ;; b4-5: sequential access wait state
//        ;;  value      #states (nS) speed
//        ;;  0          3      150
//        ;;  1          2      120
//        ;;  2          1      80
//        ;;  3          0      40
//        ;; b6: SQAEN
//        ;; b7: CLKENB

#define         MEMCFG_WIDTH32              (0<<0)
#define         MEMCFG_WIDTH16              (1<<0)
#define         MEMCFG_WIDTH8               (2<<0)
#define         MEMCFG_WIDTHRESERVED        (3<<0)
#define         MEMCFG_RA_1WAIT             (3<<2)
#define         MEMCFG_RA_2WAITS            (2<<2)
```

```
#define         MEMCFG_RA_3WAITS            (1<<2)
#define         MEMCFG_RA_4WAITS            (0<<2)
#define         MEMCFG_SA_0WAITS            (3<<4)
#define         MEMCFG_SA_1WAIT             (2<<4)
#define         MEMCFG_SA_2WAITS            (1<<4)
#define         MEMCFG_SA_3WAITS            (0<<4)
#define         MEMCFG_SQAEN                (1<<6)
#define         MEMCFG_CLKENB               (1<<7)

//DRAM refresh period register
#define         DRFPR_RFSHEN                (1<<7)
#define         DRFPR_RFDIV_MASK            127

//interrupt bits for INTSR[12] and INTMR[12]
#define         INT1_EXTFIQ     (1<<0)      // external FIQ
#define         INT1_BLINT      (1<<1)      // battery low FIQ
#define         INT1_WEINT      (1<<2)      // watchdog expired FIQ
#define         INT1_MCINT      (1<<3)      // media changed FIQ
#define         INT1_CSINT      (1<<4)      // CODEC sound
#define         INT1_EINT1      (1<<5)      // external IRQ 1
#define         INT1_EINT2      (1<<6)      // external IRQ 2
#define         INT1_EINT3      (1<<7)      // external IRQ 3
#define         INT1_TC1OI      (1<<8)      // TC1 underflow
#define         INT1_TC2OI      (1<<9)      // TC2 underflow
#define         INT1_RTCMI      (1<<10)     // RTC compare match
#define         INT1_TINT       (1<<11)     // 64Hz tick
#define         INT1_UTXINT1    (1<<12)     // UART1 tx FIFO half empty

//          or no data in Tx hold. reg.
#define         INT1_URXINT1    (1<<13)     // UART1 rx FIFO half full

//   or valid data in Rx hold. reg.
#define         INT1_UMSINT     (1<<14)     // UART1 modem status changed
#define         INT1_SSEOTI     (1<<15)     // SSI end of transfer
#define         INT2_KBDINT     (1<<0)      // keyboard interrupt
#define         INT2_UTXINT2    (1<<12)     // UART2 tx FIFO half empty
#define         INT2_URXINT2    (1<<13)     // UART2 rx FIFO half full

//PSR bits
#define         PSR_I                       (1<<7)          // IRQ disable bit

#define         PSR_F                       (1<<6)          // IRQ disable bit

#define         PSR_LOCK                    (PSR_I+PSR_F)
#define         PSR_MODEMASK                    0x1f            // Mode mask
```

```
#define     PSR_MODEUSER                0x10
#define     PSR_MODEFIQ                 0x11
#define     PSR_MODEIRQ                 0x12
#define     PSR_MODESVC                 0x13
#define     PSR_MODEABORT               0x17
#define     PSR_MODEUNDEF               0x1b

#endif /*#ifndef __CLPS7111_H__*/
```