周立功 LPC2200 开发板引导代码分析

---by 铁匠,2005.07.18

1. 周立功公司的 LPC 系列开发板，开发环境：
   编译软件：ADS 1.2 ，调试仿真：AXD+EASY JTAG。

2. 周立功公司为了方便 ADS 的使用，建立了工程模板：
   ARM Executable Image for lpc2200 // ARM 指令模板
   Thumb Executable Image for lpc2200 // Thumb 指令模板
   ARM Executable Image for UCOSII(for lpc2200) // ARM 指令模板(UCOSII 用)
   Thumb Executable Image for UCOSII(for lpc2200) // Thumb 指令模板(UCOSII 用)
   ASM for lpc2200 // 汇编模板
   Thumb ARM Interworking Image for lpc2200 // ARM 和 Thumb 交叉工作模板
   6 种模式，工作方式大同小异。差别在于用到 UCOSII 的进行了系统移植。而指令集不同，模板也略有不同。本文只分析 ARM 指令模板。

3. 文件功能模块划分(以老版本模板进行分析)：
   Config.h 定义了数据基本类型、系统时钟设置。
   Target.h 定义了软复位函数和目标板初始化函数。
   Lpc2294.h 定义了芯片的内部寄存器地址映射、固件函数。
   Target.c 目标板代码，包括异常处理程序和目标板初始化程序,用户根据程序的需要修改本文件。
   Heap.s 堆空间初始化。
   IRQ.s 中断处理。
   Stack.s 栈空间初始化。 //最新模板中没有
   Startup.s 启动引导代码。 //最新模板中没有
   Mem_a.scf
   Mem_b.scf
   Mem_c.scf
   Main.c

4. Lpc2294.h 说明：主要定义了寄存器的地址映射和固件函数。(固件函数含义不知)
估计是出厂时,这些地址写入了固化函数，在这里声明。

```
/* Define firmware Functions */
/* 定义固件函数 */
#define rm_init_entry()              ((void (*)())(0x7fffff91))()
#define rm_undef_handler()           ((void (*)())(0x7fffffa0))()
#define rm_prefetchabort_handler()   ((void (*)())(0x7fffffb0))()
#define rm_dataabort_handler()       ((void (*)())(0x7fffffc0))()
#define rm_irqhandler()              ((void (*)())(0x7fffffd0))()
#define rm_irqhandler2()             ((void (*)())(0x7fffffe0))()
#define iap_entry(a, b)              ((void (*)())(0x7ffffff1))(a, b)
```

5. Config.h 说明：
   只有下面 4 个选项需要设置：
   /* 系统设置, Fosc、Fcclk、Fcco、Fpclk 必须定义*/

```
#define Fosc        11059200          //晶振频率,10MHz~25MHz，应当与实际一至
#define Fcclk       (Fosc * 4)        //系统频率，必须为 Fosc 的整数倍(1~32)，且<=60MHZ
#define Fcco        (Fcclk＊4)        //CCO 频率,必须为 Fcclk 的 2、4、8、16 倍,范围 156MHz~320MHz
#define Fpclk       (Fcclk / 4) * 1 //VPB 时钟频率，只能为(Fcclk / 4)的 1 ~ 4 倍
```

通常只需要设置 Fosc 即可。 需要更改 Fcclk 和 Fcco 时，参照注释。

系统时钟在 TargetResetInit 函数中实现(Target.c),具体内容请参考源码。

倍频和分频的计算计算顺序：

1) 先根据系统需要的频率选择 Fosc,必须 10-25MHZ.(假定 Fosc=10MHZ)
2) 根据需要的系统频率 Fcclk,选择倍频系数 M,M = Fcclk/Fosc(假定 Fcclk=40MHZ,则 M=4).
3) M 的值为 PLLCFG bits 4:0.(PLLCFG 的低 5 位为 二进制 00100)
4) 根据 F cclk 的频率选择 Fcco. Fcco = cclk * 2 * P,则进行下列计算

   Fcco 最小/Fcclk=156/40=3.9,因此 2*P 的值必须大于 3.9,最接近的值为 4.

   此时 P=4/2=2。由于 2 的 PLLCFG bits 6:5 次方等于 P,因此 PLLCFG bits 6:5 为二进制 00.

PLL 初始化过程：

1) 写 PLLCFG 和 PLLCON:启动 PLL,但是不连接 PLL 到系统内核。

   ```
   PLLCON = 0x01 //PLLE = 1，PLLC = 0
   PLLCFG = 0x04 //假定 Fosc=10MHZ,Fcclk=40MHZ 计算
   ```

2) 保存中断寄存器，关中断
3) 连续写 PLLSEED(必须连续，不能被中断)

   ```
   PLLSEED = 0xAA
   PLLSEED = 0x55
   ```

4) 查询 PLLSTAT 一直到 PLOCK=1

   ```
   While((PLLSTART & 0x40) == 0)
   {
       Wait; //等待，一直到 PLOCK=1
   }
   ```

5) 写 PLLCFG 和 PLLCON:启动 PLL,连接 PLL 到系统内核。

   ```
   PLLCON = 0x03 //PLLE = 1，PLLC = 1
   PLLCFG = 0x04 //假定 Fosc=10MHZ,Fcclk=40MHZ 计算
   ```

6) 连续写 PLLSEED(必须连续，不能被中断)

   ```
   PLLSEED = 0xAA
   PLLSEED = 0x55
   ```

7) 查询 PLLSTAT,等待 PLOCK=1,PLLE=1,PLLC=1,MSEL4:1 和 PSEL1:0(PLLSTAT 的位 6-5)

   看是否完全符合，如果符合执行下一步，不符合报错。

8) 恢复原来的中断寄存器状态
9) PLL 附录：

Address	Name	Description	Access
0xE01FC080	PLLCON	PLL Control Register. Holding register for updating PLL control bits. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W
0xE01FC084	PLLCFG	PLL Configuration Register. Holding register for updating PLL configuration values. Values written to this register do not take effect until a valid PLL feed sequence has taken place.	R/W
0xE01FC088	PLLSTAT	PLL Status Register. Read-back register for PLL control and configuration information. If PLLCON or PLLCFG have been written to, but a PLL feed sequence has not yet occurred, they will not reflect the current PLL state. Reading this register provides the actual values controlling the PLL, as well as the status of the PLL.	RO
0xE01FC08C	PLLFEED	PLL Feed Register. This register enables loading of the PLL control and configuration information from the PLLCON and PLLCFG registers into the shadow registers that actually affect PLL operation.	WO

Table 22: PLL Control Register (PLLCON - 0xE01FC080)

PLLCON	Function	Description	Reset Value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the PLL and allow it to lock to the requested frequency. See PLLSTAT register, Table 24.	0
1	PLLC	PLL Connect. When PLLC and PLLE are both set to one, and after a valid PLL feed, connects the PLL as the clock source for the LPC2114/2124/2212/2214. Otherwise, the oscillator clock is used directly by the LPC2114/2124/2212/2214. See PLLSTAT register, Table 24.	0
7:2	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 23: PLL Configuration Register (PLLCFG - 0xE01FC084)

PLLCFG	Function	Description	Reset Value
4:0	MSEL4:0	PLL Multiplier value. Supplies the value "M" in the PLL frequency calculations. **Note:** For details on selecting the right value for MSEL4:0 see section "PLL Frequency Calculation" on page 64.	0
6:5	PSEL1:0	PLL Divider value. Supplies the value "P" in the PLL frequency calculations. **Note:** For details on selecting the right value for PSEL1:0 see section "PLL Frequency Calculation" on page 64.	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 24: PLL Status Register (PLLSTAT - 0xE01FC088)

PLLSTAT	Function	Description	Reset Value
4:0	MSEL4:0	Read-back for the PLL Multiplier value. This is the value currently used by the PLL.	0
6:5	PSEL1:0	Read-back for the PLL Divider value. This is the value currently used by the PLL.	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	PLLE	Read-back for the PLL Enable bit. When one, the PLL is currently activated. When zero, the PLL is turned off. This bit is automatically cleared when Power Down mode is activated.	0
9	PLLC	Read-back for the PLL Connect bit. When PLLC and PLLE are both one, the PLL is connected as the clock source for the LPC2114/2124/2212/2214. When either PLLC or PLLE is zero, the PLL is bypassed and the oscillator clock is used directly by the LPC2114/2124/2212/2214. This bit is automatically cleared when Power Down mode is activated.	0
10	PLOCK	Reflects the PLL Lock status. When zero, the PLL is not locked. When one, the PLL is locked onto the requested frequency.	0
15:11	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Table 25: PLL Control Bit Combinations

PLLC	PLLE	PLL Function
0	0	PLL is turned off and disconnected. The system runs from the unmodified clock input.
0	1	The PLL is active, but not yet connected. The PLL can be connected after PLOCK is asserted.
1	0	Same as 0 0 combination. This prevents the possibility of the PLL being connected without also being enabled.
1	1	The PLL is active and has been connected as the system clock source.

Table 26: PLL Feed Register (PLLFEED - 0xE01FC08C)

PLLFEED	Function	Description	Reset Value
7:0	PLLFEED	The PLL feed sequence must be written to this register in order for PLL configuration and control register changes to take effect.	undefined

Table 27: PLL Divider Values

PSEL Bits (PLLCFG bits 6:5)	Value of P
00	1
01	2
10	4
11	8

Table 28: PLL Multiplier Values

MSEL Bits (PLLCFG bits 4:0)	Value of M
00000	1
00001	2
00010	3
00011	4
...	...
11110	31
11111	32

PLL Example

System design asks for F_{osc}= 10 MHz and requires cclk = 60 MHz.

Based on these specifications, M = cclk / F_{osc} = 60 MHz / 10 MHz = 6. Consequenty, M-1 = 5 will be written as PLLCFG 4:0.

Value for P can be derived from P = F_{cco} / (cclk * 2), using condition that F_{cco} must be in range of 156 MHz to 320 MHz. Assuming the lowest allowed frequency for F_{cco} = 156 MHz, P = 156 MHz / (2*60 MHz) = 1.3. The highest F_{cco} frequency criteria produces P = 2.67. The only solution for P that satisfies both of these requirements and is listed in Table 27 is P = 2. Therefore, PLLCFG 6:5 = 1 will be used.

PLL Frequency Calculation

The PLL equations use the following parameters:

F_{OSC}	the frequency from the crystal oscillator
F_{CCO}	the frequency of the PLL current controlled oscillator
cclk	the PLL output frequency (also the processor clock frequency)
M	PLL Multiplier value from the MSEL bits in the PLLCFG register
P	PLL Divider value from the PSEL bits in the PLLCFG register

The PLL output frequency (when the PLL is both active and connected) is given by:

$$cclk = M * F_{osc} \quad \text{or} \quad cclk = \frac{F_{cco}}{2 * P}$$

The CCO frequency can be computed as:

$$F_{cco} = cclk * 2 * P \quad \text{or} \quad F_{cco} = F_{osc} * M * 2 * P$$

The PLL inputs and settings must meet the following:

- F_{osc} is in the range of 10 MHz to 25 MHz.
- cclk is in the range of 10 MHz to F_{max} (the maximum allowed frequency for the LPC2114/2124/2212/2214).
- F_{cco} is in the range of 156 MHz to 320 MHz.

ATTENTION：在 KEIL ARM 里面，configuration wizard 里面设置 PLL Setup 的值 MSEL = M
PSLE = P(上面公式中的)

6. Target.h 分析:

```
extern void Reset(void);            //软复位函数
extern void TargetInit(void);       //目标初始化
```

只实现了 2 个函数的声明。

用 IN_TARGET 宏,防止 2 个函数在 Target.c 中声明为外部函数。

7. Target.c 分析：

```
void __irq IRQ_Exception(void) //中断意外处理,自己修改
void FIQ_Exception(void)    //快速中断意外处理,自己修改
void TargetInit(void)       //目标初始化，需要自己添加
void TargetResetInit(void) //调用 main 函数前执行的初始化函数,自己修改
```

功能：内存映射设定、实现 PLL、存取器加速、设置 VIC

8. Heap.s 分析：

声明了一个 0 初始化的数据段 Heap,定义了一个全局的符号 bottom_of_heap,分配空间 1 字节。

```
        AREA    Heap, DATA, NOINIT
        EXPORT bottom_of_heap
bottom_of_heap      SPACE   1
        END
```

9. Stack.s 分析：

定义了一个数据段 Stacks,定义了一个全局符号 StackUsr，分配空间 1 字节。

```
    AREA    Stacks, DATA, NOINIT
```

```
    EXPORT StackUsr
StackUsr    SPACE   1
    END
```

10. IRQ.s 分析（对 ARM 理解不够深刻，不是完全明白）：

定义了一个宏，实现中断意外的处理。功能为切换到系统模式，执行用户编写的 C 语言中断处理，然后切换回中断模式。过程中有压栈和出栈操作，对现场状态进行保护和恢复。

```
NoInt       EQU 0x80
USR32Mode   EQU 0x10
SVC32Mode   EQU 0x13
SYS32Mode   EQU 0x1f
IRQ32Mode   EQU 0x12
FIQ32Mode   EQU 0x11

    CODE32
    AREA    IRQ,CODE,READONLY
    MACRO
$IRQ_Label HANDLER $IRQ_Exception_Function
        EXPORT  $IRQ_Label                      ; 输出的标号
        IMPORT  $IRQ_Exception_Function         ; 引用的外部标号
$IRQ_Label
        SUB     LR, LR, #4                      ; 计算返回地址
        STMFD   SP!, {R0-R3, R12, LR}           ; 保存任务环境
        MRS     R3, SPSR                        ; 保存状态
        STMFD   SP!, {R3}
        STMFD   SP, {LR}^
; 保存用户状态的 SP,注意不能回写
; 如果回写的是用户的 SP，所以后面要调整 SP
        SUB     SP, SP, #4
        MSR     CPSR_c, #(NoInt | SYS32Mode)    ; 切换到系统模式
        BL      $IRQ_Exception_Function         ; 调用 c 语言的中断处理程序
        MSR     CPSR_c, #(NoInt | IRQ32Mode)    ; 切换回 irq 模式
        LDMFD   SP, {LR}^
; 恢复用户状态的 SP,注意不能回写
; 如果回写的是用户的 SP，所以后面要调整 SP
        ADD     SP, SP, #4                      ;
        LDMFD   SP!, {R3}
        MSR     SPSR_cxsf, R3
        LDMFD   SP!, {R0-R3, R12, PC}^          ;
    MEND
;/* 以下添加中断句柄，用户根据实际情况改变 */

;Timer0_Handler  HANDLER Timer0
    END
```

10. Startup.s 分析：

向量区域 vectors,建立复位中断向量表。

未定义指令、软中断、取指令中止、取数据中止，四种中断采用死循环处理等待 WDT 溢出系统复位。

FIQ 中断调用外部函数 FIQ_Exception 处理。

对于系统复位，调用 ResetInit 函数处理。

__user_initial_stackheap 定义用户初始化堆栈函数,调用库函数初始化堆和栈。（不知道何用）

__rt_div0 定义整数除法除数为 0 错误处理函数，替代原始的__rt_div0 减少目标代码大小。

定义了各个堆栈的大小：

```
USR_STACK_LEGTH         EQU         256
SVC_STACK_LEGTH         EQU         0
FIQ_STACK_LEGTH         EQU         0
IRQ_STACK_LEGTH         EQU         256
ABT_STACK_LEGTH         EQU         0
UND_STACK_LEGTH         EQU         0
```

MyStacks 分别给 5 种模式分配了堆栈。

```
AREA    MyStacks, DATA, NOINIT, ALIGN=2
SvcStackSpace       SPACE   SVC_STACK_LEGTH * 4  ;管理模式堆栈空间
IrqStackSpace       SPACE   IRQ_STACK_LEGTH * 4  ;中断模式堆栈空间
FiqStackSpace       SPACE   FIQ_STACK_LEGTH * 4  ;快速中断模式堆栈空间
AbtStackSpace       SPACE   ABT_STACK_LEGTH * 4  ;中止义模式堆栈空间
UndtStackSpace      SPACE   UND_STACK_LEGTH * 4  ;未定义模式堆栈
```

*StackSpace 使用 SPACE 指令分配了堆栈空间的大小。

在代码区：

```
StackSvc            DCD     SvcStackSpace + (SVC_STACK_LEGTH - 1)* 4
StackIrq            DCD     IrqStackSpace + (IRQ_STACK_LEGTH - 1)* 4
StackFiq            DCD     FiqStackSpace + (FIQ_STACK_LEGTH - 1)* 4
StackAbt            DCD     AbtStackSpace + (ABT_STACK_LEGTH - 1)* 4
StackUnd            DCD     UndtStackSpace + (UND_STACK_LEGTH - 1)* 4
```

Stack* 指明了堆栈最初的指针位置。

注意：由于 InitStack 分别将上面的 Stack*初始化到各个模式的堆栈，因此如果堆栈长度设置为 0，则堆栈的起始指针 Stack*越界，因此周立功公司提供的模板有错误。

InitStack 函数，分别设置不同模式下面的堆栈工作模式和 SP 值。

注：StackUsr 是 Stack.s 中定义的，具体分配空间大小由 Stack.s 决定。周立功提供的模板有问题，因为 StackUsr 是栈的最低地址，但是 InitStack 函数调用时，应该把 SP 设为初始指针（最高地址）。

11. mem_a.scf、mem_b.scf、mem_c.scf 分析：

参数	mem_a.scf	mem_b.scf	mem_c.scf
ROM_LOAD	0x80000000	0x80000000	0x00000000
ROM_EXEC	0x80000000	0x80000000	0x00000000
IRAM	0x40000000	0x40000000	0x40000000
STACKS	0x40004000 UNINIT	0x40004000 UNINIT	0x40004000 UNINIT
ERAM	0x81000000	0x80040000	0x8000,0000
HEAP	+0 UNINIT	+0 UNINIT	+0 UNINIT
模式	RelOutChip	DebugInExram	DebugInChipFlash RelInChip
Flash 用 Bank	Bank 0	Bank 1 或者未用	未用
SRAM 用 Bank	Bank 1	Bank 0	Bank 0

定义了各个段的加载位置。内部 RAM 为 0x4000,0000-0x4000,3FFF(16K bytes 静态 RAM)，
片外 RAM ： 0x8000,0000-0x8007,FFFF(BANK0)， 0x8100,0000-0x8107,FFFF(BANK1)
片外 FLASH：0x8000,0000-0x801F,FFFF(BANK0)， 0x8100,0000-0x811F,FFFF(BANK1)。
片外 SRAM: 512K bytes 片外 FLASH: 2M bytes
个人认为：mem_b.scf ERAM 应该为 0x8000,0000

Mem_a.scf 如下：

```
ROM_LOAD 0x80000000
{
    ROM_EXEC 0x80000000
    {
        Startup.o (vectors, +First)
        * (+RO)
    }

    IRAM 0x40000000
    {
        Startup.o (+RW,+ZI)
    }

    STACKS 0x40004000 UNINIT
    {
        stack.o (+ZI)
    }

    ERAM 0x80040000
    {
        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        heap.o (+ZI)
```

```
    }
}
```

12. 新版本模版与老版本的不同：

1) config.h 增加

```
#ifndef __CONFIG_H
#define __CONFIG_H
……
#endif
```

防止多次包含 config.h。

2) IRQ.s 保护现场时，增加了对 R3 的保护。

3) heap.s 和 stack.s 去掉。

4) target.h 增加

```
#ifndef __TARGET_H
#define __TARGET_H
    #ifdef __cplusplus
    extern "C" {
    #endif
……

    #ifdef __cplusplus
    }
    #endif
#endif
```

增加了对 C++的语法支持。

5) target.c 实现了部分库函数的功能。

6) Startup.s 修改内容比较多，新定义了一些区域。

```
        AREA    Heap, DATA, NOINIT
bottom_of_heap    SPACE   1
        AREA    StackBottom, DATA, NOINIT
bottom_of_Stacks    SPACE   1
        AREA    HeapTop, DATA, NOINIT
top_of_heap
        AREA    Stacks, DATA, NOINIT
StackUsr
```

定义了 4 个符号含义分别为：

bottom_of_heap 堆底

bottom_of_Stacks 栈底（最低指针）

top_of_heap 堆顶（最初指针）

StackUsr 用户堆栈

(注意：堆向上生长、栈向下生长，最初的位置为顶。因此栈顶地址>栈底，堆顶<堆底。)

4 个符号用于其它源文件，而用 AREA 定义的区域名，则在*.scf 中被应用。

使用*.scf 来确定这些区域的位置,如 Startup.o (MyStacks)。

典型的 mem_a.scf：

```
ROM_LOAD 0x80000000
```

```
{
    ROM_EXEC 0x80000000
    {
        Startup.o (vectors, +First)
        * (+RO)
    }

    IRAM 0x40000000
    {
        Startup.o (MyStacks)
    }

    STACKS_BOTTOM +0 UNINIT
    {
        Startup.o (StackBottom)
    }

    STACKS 0x40004000 UNINIT
    {
        Startup.o (Stacks)
    }

    ERAM 0x81000000
    {
        * (+RW,+ZI)
    }

    HEAP +0 UNINIT
    {
        Startup.o (Heap)
    }

    HEAP_BOTTOM 0x81080000 UNINIT
    {
        Startup.o (HeapTop)
    }
}
```

13. 附件（用于阅读源码进行分析）：


LPC2200 Project module_new.rar


LPC2200 Project module.rar

联系方式： benfu@126.com QQ: 58862973
个人网站： http://www.eestudio.net QQ 群：4055086 (只欢迎热心助人和真心学习的人)