

目录

1 中断点亮小灯	3
1.1 原理分析	3
1.2 调试过程	4
1.2.1 中断初始化程序分析	4
1.2.2 中断寄存器分析	5
1.2.3 中断寄存器配置	6
1.2.4 中断服务子程序	8
1.3 结果	9
1.4 问题与总结	9
2 双机异步通信查询实现	10
2.1 原理分析	10
2.2 调试过程	11
2.2.1 uart 初始化程序	11
2.2.2 uart clock 配置	11
2.2.3 uart 循环发送程序	13
2.3 结果	14
2.4 问题与总结	14
3 异步通信中断发送实现	15
3.1 原理分析	15
3.2 调试过程	15
3.2.1 初始化程序	15
3.2.2 中断服务子程序	16
3.3 结果	17
3.4 问题与总结	17
4 双机异步通信中断发送，查询接收实现	18
4.1 原理分析	18
4.2 调试过程	18
4.2.1 主程序	18
4.3 结果	19
4.4 问题与总结	19

5 异步通信中断接收实现	21
5.1 原理分析	21
5.2 调试过程	21
5.2.1 UART 寄存器分析	21
5.2.2 UART 寄存器配置	22
5.3 结果	24
5.4 问题与总结	24

1 中断点亮小灯

示例程序 int 实现功能为中断控制控制 LED 灯亮，此处分析 int 程序

1.1 原理分析

利用 S3C2440 手册，分析解读中断初始化程序。S3C2440A 中的中断控制器接受来自 60 个中断源的请求。提供这些中断源的是内部外设，如 DMA 控制器、UART、IIC 等等。在这些中断源中，UARTn、AC97 和 EINTn 中断对于中断控制器而言是“或”关系。当从内部外设和外部中断请求引脚收到多个中断请求时，中断控制器在仲裁步骤后请求 ARM920T 内核的 FIQ 或 IRQ。仲裁步骤由硬件优先级逻辑决定并且写入结果到帮助用户通告是各种中断源中的哪个中断发生了的中断挂起寄存器中。

控制流程图如下：

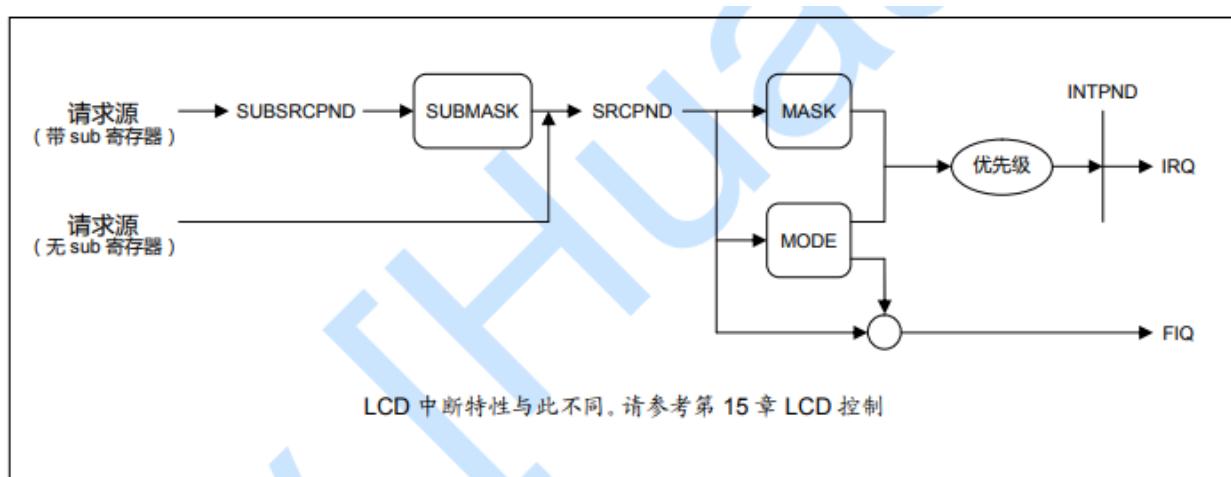


图 1：中断控制流程图

1. 先收集到 SRCPND
2. 用 INTMOD 判断是否为 FIQ
3. 判断完后进入屏蔽环节 INTMSK 进行筛选
4. PRIORITY 进行优先级划分
5. 划分结果存储在 INTOFFSET

原理图如下：

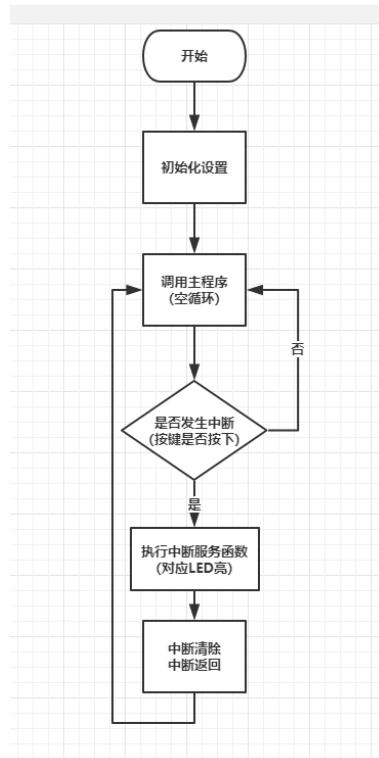


图 2：中断点亮小灯原理图

1.2 调试过程

1.2.1 中断初始化程序分析

head.s 程序：

功能：初始化，设置中断模式、管理模式的栈，设置好中断处理函数。

```
1 Reset:
2   ldr sp, =4096
3   bl disable_watch_dog
4   msr cpsr_c, #0xd2      @ 进入中断模式
5   ldr sp, =3072          @ 设置中断模式栈指针
6
7   msr cpsr_c, #0xd3      @ 进入管理模式
```

```

8
9   bl  init_led          @ 初始化LED的GPIO管脚
10  bl  init_irq          @ 调用中断初始化函数，在init.c中
11  msr cpsr_c, #0x5f    @ 设置I-bit=0，开IRQ中断
12
13  ldr lr, =halt_loop    @ 设置返回地址
14  ldr pc, =main          @ 调用main函数
15 halt_loop:
16   b   halt_loop

```

中断初始化程序

此段代码为中断初始化的核心，设置了中断开启的模式，并调用了各个初始化函数，如 init_led 和 init_irq

```

1 HandleIRQ:
2   sub lr, lr, #4          @ 计算返回地址
3   stmdb sp!, { r0-r12,lr } @ 保存使用到的寄存器
4                               @ 注意，此时的sp是中断模式的sp
5                               @ 初始值是上面设置的3072
6
7   ldr lr, =int_return     @ 设置调用ISR即EINT_Handle函数后的返回地址
8   ldr pc, =EINT_Handle    @ 调用中断服务函数，在interrupt.c中

```

中断跳转程序

此处为中断跳转程序，当出现特定中断时会跳转到这里，然后按照这里的指示跳转到中断服务函数，这里是跳转到 EINT_Handle 内。

1.2.2 中断寄存器分析

打开 s3c24xx.h，这里使用到的 interrupt registes，我们可以看到

```

1 #define SRCPND      (*(volatile unsigned long *)0x4A000000)
2 #define INTMOD       (*(volatile unsigned long *)0x4A000004)
3 #define INTMSK        (*(volatile unsigned long *)0x4A000008)
4 #define PRIORITY      (*(volatile unsigned long *)0x4A00000c)
5 #define INTPND        (*(volatile unsigned long *)0x4A000010)
6 #define INTOFFSET     (*(volatile unsigned long *)0x4A000014)
7 #define SUBSRCPND    (*(volatile unsigned long *)0x4A000018)
8 #define INTSUBMSK    (*(volatile unsigned long *)0x4A00001c)

```

中断寄存器

这里使用了大量中断相关寄存器。其中 **SRCPND**：当一个中断发生后，那么相应的位会被置 1，表示一个或一类中断发生了。

INTMOD: 当 INTMOD 中某位被设置为 1 时，它对应的中断被设为 FIQ，CPU 将进入快速中断模式。

INTMSK: 32 位，用来屏蔽 SRCPND 寄存器所标识的中断。但只能屏蔽 IRQ 中断，不能屏蔽 FIQ 中断。

PRIORITY: 用于设置 IRQ 中断的优先级。

INTPND: 中断优先级仲裁器选出优先级最高中断后，这个中断在 INTPND 寄存器中的相应位被置 1，随后，CPU 进入中断模式处理它。同一时间内，此寄存器只有一位被置 1。

INTOFFSET: 用来表示 INTPND 寄存器中哪位被置 1 了，即记录 INTPND 中位 [x] 为 1 的位 x 的值。清除 INTPND、SRCPND 时自动清除。

SUBSRCPND: 次级源挂起寄存器，当一个中断发生后，那么相应的位会被置 1，表示一个中断发生了。

INTSUBMSK: 中断次级屏蔽寄存器，此寄存器有 11 位，其每一位都与一个中断源相联系。如果某个指定位被设置为 1，则相应中断源的中断请求不会被 CPU 所服务（请注意即使在这种情况下，SRCPND 寄存器的相应位也设置为 1）。如果屏蔽位为 0，则可以服务中断请求。

1.2.3 中断寄存器配置

打开 init.c，这里对中断寄存器配置，我们可以看到

```
1 void init_irq( )
2 {
3     // S2,S3对应的2根引脚设为中断引脚 EINT0,ENT2
4     GPFCON &= ~(GPF0_msk | GPF2_msk);
5     GPFCON |= GPF0_eint | GPF2_eint;
6
7     // S4对应的引脚设为中断引脚EINT11
8     GPGCON &= ~GPG3_msk;
9     GPGCON |= GPG3_eint;
10
11    // 对于EINT11，需要在EINTMASK寄存器中使能它
12    EINTMASK &= ~(1<<11);
13
14    /*
15     * 设定优先级：
16     * ARB_SEL0 = 00b, ARB_MODE0 = 0: REQ1 > REQ3, 即EINT0 > EINT2
17     * 仲裁器1、6无需设置
18     * 最终：
19     * EINT0 > EINT2 > EINT11即K2 > K3 > K4
20     */
21
22    PRIORITY = (PRIORITY & ((~0x01) | (0x3<<7))) | (0x0 << 7);
```

```

22
23 // EINT0、EINT2、EINT8_23使能
24 INTMSK &= (~(1<<0)) & (~(1<<2)) & (~(1<<5));
25 }

```

中断寄存器配置

2440 的外部中断引脚 EINT 与通用 IO 引脚 F 和 G 复用，要想使用中断功能，就要把相应的引脚配置成中断模式，如我们想把端口 F0 设置成外部中断，而其他引脚功能不变，则 $GPFCON=(GPFCON \& \sim 0x3) | 0x2$ 。

配置完引脚后，还需要配置具体的中断功能。我们要打开某一中断的屏蔽，这样才能响应该中断，相对应的寄存器为 INTMSK；还要设置外部中断的触发方式，如低电平、高电平、上升沿、下降沿等，相对应的寄存器为 EXTINTn。另外由于 EINT4 到 EINT7 共用一个中断向量，EINT8 到 EINT23 也共用一个中断向量，而 INTMSK 只负责总的中断向量的屏蔽，要具体打开某一具体的中断屏蔽，还需要设置 EINTMASK。

还有一些其他的配置，如当需要用到快速中断时，要使用 INTMOD，当需要配置中断优先级时，要使用 PRIORITY 等。优先级如下：

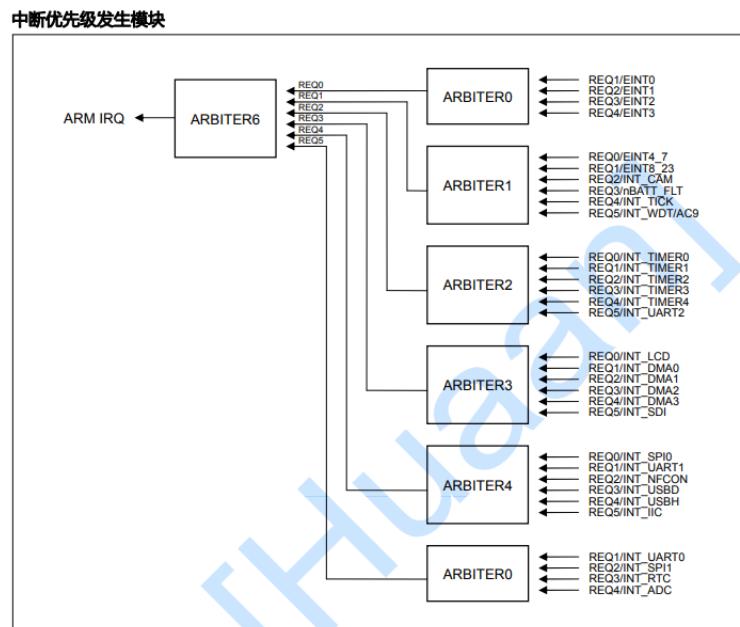


图 3: 中断优先级

1.2.4 中断服务子程序

打开 interrupt.c，这里是中断服务子程序，我们可以看到

```
1 void EINT_Handle()
2 {
3     unsigned long oft = INTOFFSET;
4     unsigned long val;
5     switch( oft )
6     {
7         // S2被按下
8         case 0:
9         {
10             GPFDAT |= (0x7<<4);    // 所有LED熄灭
11             GPFDAT &= ~(1<<4);    // LED1点亮
12             break;
13         }
14
15         // S3被按下
16         case 2:
17         {
18             GPFDAT |= (0x7<<4);    // 所有LED熄灭
19             GPFDAT &= ~(1<<5);    // LED2点亮
20             break;
21         }
22         // K4被按下
23         case 5:
24         {
25             GPFDAT |= (0x7<<4);    // 所有LED熄灭
26             GPFDAT &= ~(1<<6);    // LED4点亮
27             break;
28         }
29         default:
30             break;
31     }
32     // 清中断
33     if( oft == 5 )
34         EINTPEND = (1<<11);    // EINT8_23合用IRQ5
35     SRCPND = 1<<oft;
36     INTPND = 1<<oft;
37     //SRCPND = SRCPND;
38 }
```

中断服务子程序

这里用到 INTOFFSET 寄存器，通过读取此寄存器，获得最优先级的中断请求。

除此以外还需要把 SRCPND 和 INTPND 中的相应的位清零（通过置 1 来清零），因为当中断发生时，2440 会自动把这两个寄存器中相对应的位置 1，以表示某一中断发生，如果不在中断处理函数内把它们清零，系统会一直执行该中断函数。另外还是由于前面介绍过的，有一些中断是共用一个中断向量的，而一个中断向量只能有一个中断执行函数，因此具体是哪个外部中断，还需要 EINTPEND 来判断，并同样还要通过置 1 的方式把相应的位清零。

1.3 结果

中断点亮小灯

1.4 问题与总结

此程序缺乏防抖，容易造成误触，需要进一步开发。

2 双机异步通信查询实现

2.1 原理分析

S3C2440A 的通用异步收发器 (UART) 配有 3 个独立异步串行 I/O (SIO) 端口，每个都可以是基于中断或基于 DMA 模式操作。即 UART 可以通过产生中断或 DMA 请求来进行 CPU 和 UART 之间的数据传输。

S3C2440A 的 UART 包括了可编程波特率，红外 (IR) 发送/接收，插入 1 个或 2 个停止位，5 位、6 位、7 位或 8 位的数据宽度以及奇偶校验。

每个 UART 包含一个波特率发生器、发送器、接收器和一个控制单元，波特率发生器可以由 PCLK、FCLK/n 或 UEXTCLK (外部输入时钟) 时钟驱动。

原理图如下：

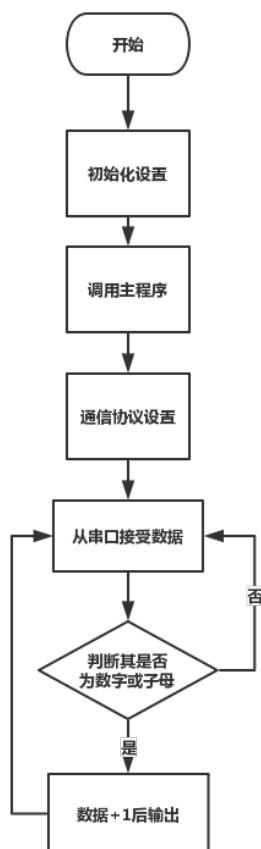


图 4：双机异步通信查询实现原理图

2.2 调试过程

2.2.1 uart 初始化程序

head.s 程序：

功能：初始化，设置 clock 和把代码搬到 sram 快速运行。

```
1     bl  clock_init          @ 设置MPLL，改变FCLK、HCLK、PCLK
2     bl  memsetup            @ 设置存储控制器以使用SDRAM
3     bl  copy_steppingstone_to_sram    @ 复制代码到SDRAM中
```

初始化程序

此段代码为初始化的核心，调用了各个初始化函数，如 clock_init。

2.2.2 uart clock 配置

打开 init.c，这里对 clock 进行配置，我们可以看到

```
1 #define S3C2410_MPLL_200MHZ      ((0x5c<<12)|(0x04<<4)|(0x00))
2 #define S3C2440_MPLL_200MHZ      ((0x5c<<12)|(0x01<<4)|(0x02))
3 void clock_init(void)
4 {
5     // LOCKTIME = 0x00ffff； // 使用默认值即可
6     CLKDIVN = 0x03；           // FCLK:HCLK:PCLK=1:2:4， HDIVN=1,PDIVN=1
7
8     /* 如果HDIVN非0，CPU的总线模式应该从“fast bus mode”变为“asynchronous bus
9      mode” */
10    __asm__(

11        "mrc    p15, 0, r1, c1, c0, 0\n"           /* 读出控制寄存器 */
12        "orr    r1, r1, #0xc0000000\n"             /* 设置为“asynchronous bus mode”
13        /*
14        "mcr    p15, 0, r1, c1, c0, 0\n"           /* 写入控制寄存器 */
15    );
16
17    /* 判断是S3C2410还是S3C2440 */
18    if ((GSTATUS1 == 0x32410000) || (GSTATUS1 == 0x32410002))
19    {
20        PLLCON = S3C2410_MPLL_200MHZ; /* 现在，FCLK=200MHz,HCLK=100MHz,PCLK
21                                     =50MHz */
```

```

19     }
20     else
21     {
22         MPLLCON = S3C2440_MPLL_200MHZ; /* 现在, FCLK=200MHz,HCLK=100MHz,PCLK
23             =50MHz */
24     }

```

clock 配置

对于 MPLLCON 寄存器, [19:12] 为 MDIV, [9:4] 为 PDIV, [1:0] 为 SDIV 有如下计算公式:

$$\text{MPLL}(\text{FCLK}) = (2 * \text{m} * \text{Fin}) / (\text{p} * 2^{\text{s}})$$

其中: m = MDIV + 8, p = PDIV + 2, s = SDIV

对于本开发板, Fin = 12MHz, 设置 CLKDIVN, 令分频比为: FCLK:HCLK:PCLK=1:2:4,

FCLK=200MHz,HCLK=100MHz,PCLK=50MHz

同时, 打开 interrupt .c, 我们可以看到

```

25     ULC0N0 &= 0xFFFFFFFF00;
26     ULC0N0 |= 0X03; // 8N1(8个数据位, 无校验, 1个停止位)
27     /**
28     UCON0 = 0x05; // 查询方式, UART时钟源为PCLK
29     /**
30     UFCON0 = 0x00; // 不使用FIFO
31     UMC0N0 = 0x00; // 不使用流控
32     UBRDIV0 = UART_BRD; // 波特率为115200

```

clock 配置

这里把 UART 时钟源设置为 PCLK, 并且不使用 FIFO, 不使用流控, 设置波特率为 115200。

S3C2440A 中的时钟控制逻辑可以产生必须的时钟信号, 包括 CPU 的 FCLK, AHB 总线外设的 HCLK 以及 APB 总线外设的 PCLK。S3C2440A 包含两个锁相环 (PLL): 一个提供给 FCLK、HCLK 和 PCLK, 另一个专用于 USB 模块 (48MHz)。时钟控制逻辑可以不使用 PLL 来减慢时钟, 并且可以由软件连接或断开各外设模块的时钟, 以降低功耗。

其中 FCLK, HCLK 和 PCLK 的分析如下:

FCLK 是提供给 ARM920T 的时钟。

HCLK 是提供给用于 ARM920T, 存储器控制器, 中断控制器, LCD 控制器, DMA 和 USB 主机模块的 AHB 总线的时钟。

PCLK 是提供给用于外设如 WDT, IIS, I2C, PWM 定时器, MMC/SD 接口, ADC, UART, GPIO, RTC 和 SPI 的 APB 总线的时钟。

S3C2440A 还支持对 FCLK、HCLK 和 PCLK 之间分频比例的选择。该比例由 CLKDIVN 控制寄存器中的 HDIVN 和 PDIVN 所决定。

分频比如下：

HDIVN	PDIVN	HCLK3_HALF/ HCLK4_HALF	FCLK	HCLK	PCLK	分频比例
0	0	-	FCLK	FCLK	FCLK	1:1:1 (默认)
0	1	-	FCLK	FCLK	FCLK/2	1:1:2
1	0	-	FCLK	FCLK/2	FCLK/2	1:2:2
1	1	-	FCLK	FCLK/2	FCLK/4	1:2:4
3	0	0/0	FCLK	FCLK/3	FCLK/3	1:3:3
3	1	0/0	FCLK	FCLK/3	FCLK/6	1:3:6
3	0	1/0	FCLK	FCLK/6	FCLK/6	1:6:6
3	1	1/0	FCLK	FCLK/6	FCLK/12	1:6:12
2	0	0/0	FCLK	FCLK/4	FCLK/4	1:4:4
2	1	0/0	FCLK	FCLK/4	FCLK/8	1:4:8
2	0	0/1	FCLK	FCLK/8	FCLK/8	1:8:8
2	1	0/1	FCLK	FCLK/8	FCLK/16	1:8:16

图 5：分频比例

2.2.3 uart 循环发送程序

我们为了能够直观看到结果，将 main.c 的代码修改成发送机子和接收机子。

发送机子主程序：

功能：每次复位即向 uart0 发送 a。

```
1 int main()
2 {
3     uart0_init();
4     unsigned char c='a';
5     putc(c);
6     return 0;
7 }
```

发送机子 main.c 程序

接收机子主程序：

功能：查询 uart0。

```
1 int main()
2 {
3     unsigned char c;
4     uart0_init();
5     while(1)
6     {
```

```

7     c =getc();
8     if (isDigit(c) || isLetter(c))
9         putc(c+1);
10    }
11    return 0;
12 }
```

接收机子 main.c 程序

2.3 结果

每次按下发送机子的复位键，接收机子即可接收到“a”。

```

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

图 6: 双机异步通信查询实现

2.4 问题与总结

1. 原程序无法成功发送，需要自己看懂 putc 并发送。
2. 需要看原理图来查看哪一个管脚是发送，哪一个是接收。

3 异步通信中断发送实现

把中断点亮小灯和双机异步通信查询实现结合，实现异步通信中断发送实现。

3.1 原理分析

在 int 的基础上，加入 clock_init 等初始化函数配置 uart 即可

原理图如下：

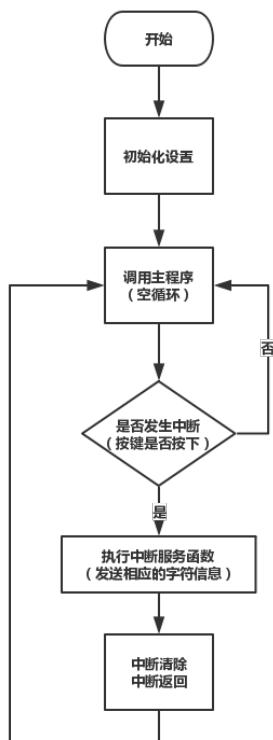


图 7：异步通信中断发送实现原理图

3.2 调试过程

3.2.1 初始化程序

功能：初始化，设置 clock 和把代码搬到 sram 快速运行，并且设置中断模式、管理模式的栈，设置好中断处理函数。

```

4   bl  clock_init          @ 设置MPLL，改变FCLK、HCLK、PCLK
5   bl  memsetup            @ 设置存储控制器以使用SDRAM
6   bl  copy_steppingstone_to_sdram    @ 复制代码到SDRAM中
7   bl  init_led             @ 初始化LED的GPIO管脚
8   bl  init_irq              @ 调用中断初始化函数，在init.c中

```

初始化程序

3.2.2 中断服务子程序

打开 interrupt .c，这里是中断服务子程序，我们可以看到

```

39 void EINT_Handle()
40 {
41     unsigned long oft = INTOFFSET;
42     unsigned long val;
43     unsigned char a='a';
44     unsigned char b='b';
45     unsigned char c='c';
46     uart0_init(); // 波特率115200，8N1(8个数据位，无校验位，1个停止位)
47     switch( oft )
48     {
49         // S2被按下
50         case 0:
51         {
52             putc(a);
53             break;
54         }
55
56         // S3被按下
57         case 2:
58         {
59             putc(b);
60             break;
61         }
62
63         // K4被按下
64         case 5:
65         {
66             putc(c);
67             break;
68         }
69         default:

```

```

70         break;
71     }
72     if( oft == 5 )
73         EINTPEND = (1<<11); // EINT8_23 合用 IRQ5
74     SRCPND = 1<<oft;
75     INTPND = 1<<oft;
76 }

```

中断服务子程序

当按下对应按键的时候，uart0 会发送出 "a", "b", "c" 等字母

3.3 结果

当按下对应按键的时候，屏幕上会打印出 "a", "b", "c" 等字母

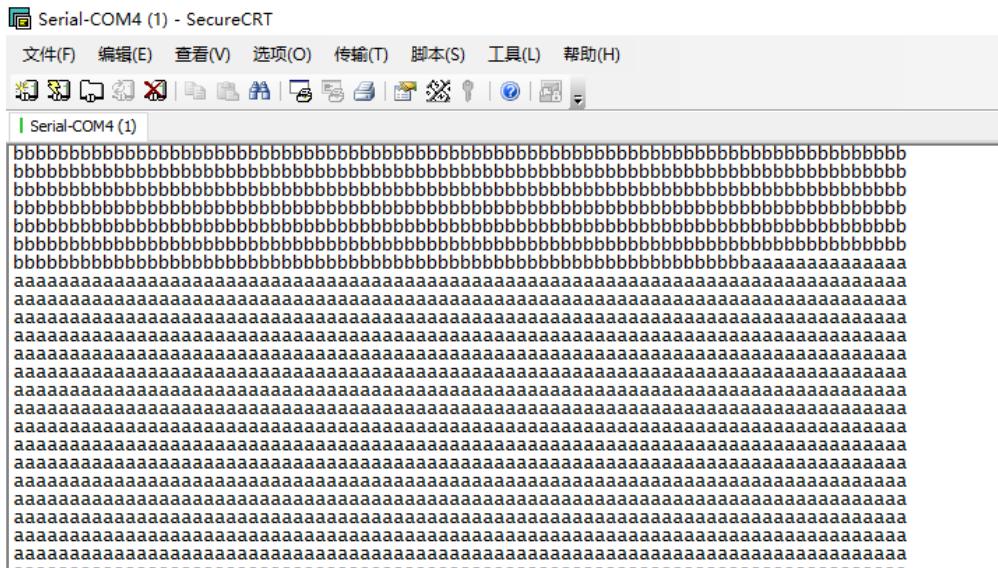


图 8：分频比例

3.4 问题与总结

中断程序与 uart 程序结合的时候，可以把一样性质的函数放在一起，不改变函数名，这样 makefile 便无需修改。

一步一步修改，不可一次改完直接运行，这样的话难以 debug。

4 双机异步通信中断发送，查询接收实现

双机通过中断发送，查询接收实现实现互相通信，并增加小灯以作展示。

4.1 原理分析

在异步通信中断发送实现的基础上，加入双机异步通信查询接收

原理图如下：

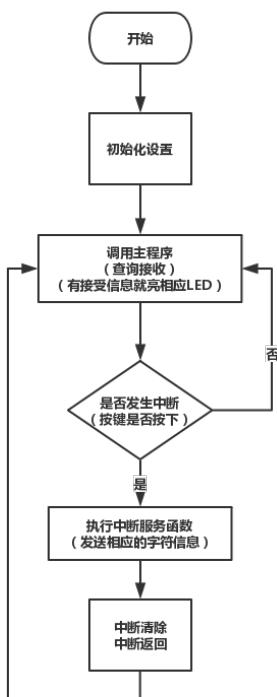


图 9：双机异步通信中断发送，查询接收实现原理图

4.2 调试过程

4.2.1 主程序

```
1 int main()
2 {
3     unsigned char c;
4     uart0_init();
```

```

5     GPFCON &= ~(GPF4_msk | GPF5_msk | GPF6_msk);
6     GPFCON |= GPF4_out | GPF5_out | GPF6_out;
7
8     while(1)
9     {
10        c =getc();
11        if (c == 'a'){
12            GPFDAT &= ~(1<<4);
13            GPFDAT |= (1<<5);
14            GPFDAT |= (1<<6);
15        }
16        else if (c == 'b'){
17            GPFDAT |= (1<<4);
18            GPFDAT &= ~(1<<5);
19            GPFDAT |= (1<<6);
20        }
21        else if (c == 'c'){
22            GPFDAT |= (1<<4);
23            GPFDAT |= (1<<5);
24            GPFDAT &= ~(1<<6);
25        }
26    }
27    return 0;
28 }
```

主程序

没有按下按键时，其一直在查询 uart0 寄存器的数值，若有接收，则点亮对应的灯。当按下对应按键的时候，uart0 会发送出" a ", " b ", " c " 等字母。

4.3 结果

当按下对应按键的时候，对方机子会亮对应的灯，反之亦是。

4.4 问题与总结

使用的是查询接收，效率不高，需进一步改进。

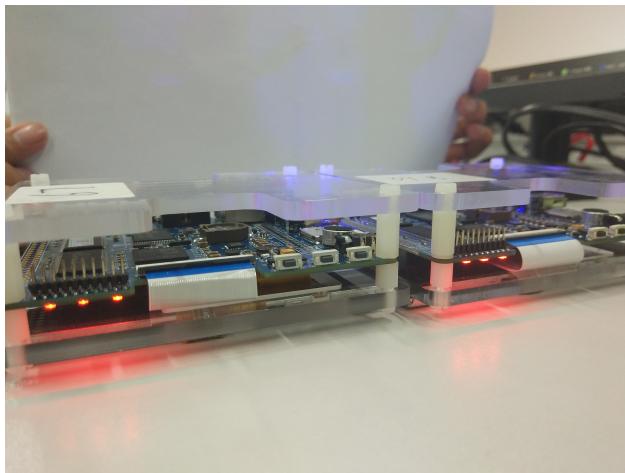


图 10: 原始状态

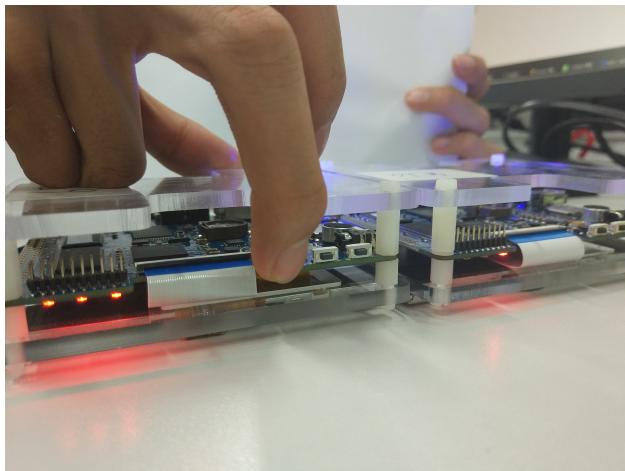


图 11: 按下 1 号机子

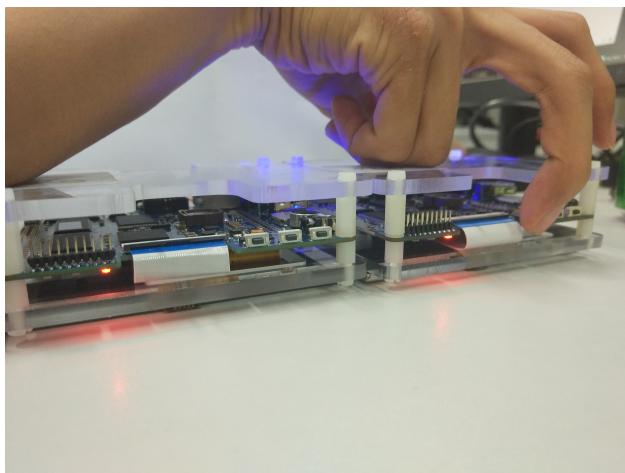


图 12: 按下 2 号机子

5 异步通信中断接收实现

一台机子使用中断发送，一台机子使用中断接收。

5.1 原理分析

在异步通信中断发送实现的基础上，打开 uart0 的中断，以用于中断接收，并用 led 来展示。

原理图如下：

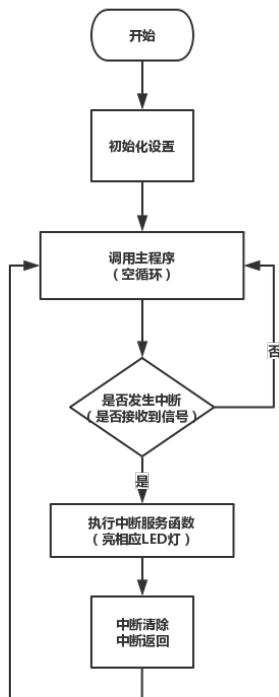


图 13: 异步通信中断接收实现原理图

5.2 调试过程

5.2.1 UART 寄存器分析

打开 s3c24xx.h，这里使用到的 interrupt registes，我们可以看到

```
1 /*UART registers*/
2 #define ULCON0      (*(volatile unsigned long *)0x50000000)
```

```

3 #define UCON0          (*(volatile unsigned long *)0x50000004)
4 #define UFCON0         (*(volatile unsigned long *)0x50000008)
5 #define UMCON0          (*(volatile unsigned long *)0x5000000c)
6 #define UTRSTAT0        (*(volatile unsigned long *)0x50000010)
7 #define UTXH0           (*(volatile unsigned char *)0x50000020)
8 #define URXH0           (*(volatile unsigned char *)0x50000024)
9 #define UBRDIV0         (*(volatile unsigned long *)0x50000028)

```

UART 寄存器分析

这里使用了大量 UART 相关寄存器，后面会详细分析。

5.2.2 UART 寄存器配置

打开 init.c，这里对中断寄存器配置，我们可以看到

```

26 void init_irq( )
27 {
28     SUBSRCPND |= 0x3;
29     SRCPND |= 0x1<<28;
30     INTPND |= 0x1<<28;
31
32     INTSUBMSK &= ~(0x1);
33     INTSUBMSK |= (0x1<<1);
34     INTMSK &= ~(0x1<<28);
35 }

```

中断寄存器配置

```

36 void uart0_init(void)
37 {
38     GPHCON&=0x3c0000;
39     GPHCON|=0x2faaa;
40     GPBCON = 0x1dd7fc; //GPB5,6,8,10 设置为输出
41     GPBDAT|=0x560; //4个LED全灭
42     GPHUP=0x1ff; //H口上拉禁止
43     GPFCON &=~((3<<0)|(3<<4)|(3<<6)|(3<<8)) ;
44     GPFCON |= ((2<<0)|(2<<4)|(2<<6)|(2<<8)); //GPF0, GPF2, GPF3, GPF4工作在第二功
45     能状态，即中断
46
47     EINTPEND=(1<<4);
48
49     ULCON0 &=0xFFFFFFFF00;
50     ULCON0 |=0X03;      // 8N1(8个数据位，无校验，1个停止位)
      UCON0    = 0x09;      // 中断请求

```

```

51     UFCON0  = 0x00;      // 不使用 FIFO
52     UMCN0   = 0x00;      // 不使用流控
53     UBRDIV0 = UART_BRD; // 波特率为 115200
54 }

```

中断寄存器配置

配置分析：UART 线路控制寄存器：

我们选取 8 个数据位，无校验，1 个停止位

UART 控制寄存器：

ULCONn	位	描述	初始状态
保留	[7]	-	0
红外模式	[6]	决定是否使用红外模式 0 = 普通模式操作 1 = 红外 Tx/Rx 模式	0
奇偶校验模式	[5:3]	指定在 UART 发送和接收操作期间奇偶校验产生和检查的类型 0xx = 无奇偶校验 100 = 奇校验 110 = 固定/检查奇偶校验为 1 101 = 偶校验 111 = 固定/检查奇偶校验为 0	000
停止位数	[2]	指定用于结束帧信号的停止位的个数 0 = 每帧 1 个停止位 1 = 每帧 2 个停止位	0
字长度	[1:0]	指出每帧用于发送或接收的数据位的个数 00 = 5 位 01 = 6 位 10 = 7 位 11 = 8 位	00

图 14: ULCON0

选取 1 = 电平（当非 FIFO 模式中 Tx 缓冲器为空或 FIFO 模式中达到 Tx FIFO 触发深度时请求中断），并且 UFCON0 寄存器中，不使用 FIFO。UMCN0 中，不使用流控。

Tx 中断类型	[9]	中断请求类型。 0 = 脉冲(非 FIFO 模式中 Tx 缓冲器一变为空或 FIFO 模式中达到 Tx FIFO 触发深度就请求中断) 1 = 电平(当非 FIFO 模式中 Tx 缓冲器为空或 FIFO 模式中达到 Tx FIFO 触发深度时请求中断)	0
Rx 中断类型	[8]	中断请求类型。 0 = 脉冲(非 FIFO 模式中 Rx 缓冲器接收到数据或 FIFO 模式中达到 Rx FIFO 触发深度则立刻请求中断) 1 = 电平(当非 FIFO 模式中 Rx 缓冲器正在接收数据或 FIFO 模式中达到 Rx FIFO 触发深度请求中断)	0
Rx 超时使能	[7]	当使能了 UART FIFO 使能/禁止 Rx 超时中断。该中断是一个接收中断 0 = 禁止 1 = 使能	0
Rx 错误状态中断使能	[6]	异常时允许 UART 产生中断，如接收操作期间的断点、帧错误、奇偶错误或溢出错误。 0 = 不产生接收错误状态中断 1 = 产生接收错误状态中断	0
环回模式	[5]	设置环回模式为 1 使得 UART 进入环回模式。此模式只用于测试。 0 = 正常操作 1 = 环回模式	0
发出断点信号	[4]	设置此位使得 UART 在单帧期间发出一个断点信号。此位在发出断点信号后将自动清零。 0 = 正常传输 1 = 发出断点信号	0

图 15: UCON0

我们还要打开对应的中断允许寄存器，INTMSK &= ~(0x1<<28)；

同时，由于我们还要定义是接收还是发送，也要打开 INTSUBMSK，

中断屏蔽 (INTMSK) 寄存器

此寄存器由 32 位组成，其每一位都涉及一个中断源。如果某个指定为被设置为 1，则 CPU 不会去服务来自相应中断源（请注意即使在这种情况下，SRCPND 寄存器的相应位也设置为 1）。如果屏蔽位为 0，则可以服务中断请求。

寄存器	地址	R/W	描述	复位值
INTMSK	0X4A000008	R/W	决定屏蔽哪个中断源。被屏蔽的中断源将不会服务 0 = 中断服务可用 1 = 屏蔽中断服务	0xFFFFFFFF

INTMSK	位	描述	初始状态
INT_ADC	[31]	0 = 可服务 1 = 屏蔽	1
INT_RTC	[30]	0 = 可服务 1 = 屏蔽	1
INT_SPI1	[29]	0 = 可服务 1 = 屏蔽	1
INT_UART0	[28]	0 = 可服务 1 = 屏蔽	1
INT_IIC	[27]	0 = 可服务 1 = 屏蔽	1
INT_USBH	[26]	0 = 可服务 1 = 屏蔽	1
INT_USBD	[25]	0 = 可服务 1 = 屏蔽	1

图 16: INTMSK

INTSUBMSK &= ~(0x1);INTSUBMSK |= (0x1<<1);

中断次级屏蔽 (INTSUBMSK) 寄存器

此寄存器有 11 位，其每一位都与一个中断源相联系。如果某个指定位被设置为 1，则相应中断源的中断请求不会被 CPU 所服务（请注意即使在这种情况下，SRCPND 寄存器的相应位也设置为 1）。如果屏蔽位为 0，则可以服务中断请求。

寄存器	地址	R/W	描述	复位值
INTSUBMSK	0X4A00001C	R/W	决定屏蔽哪个中断源。被屏蔽的中断源将不会服务 0 = 中断服务可用 1 = 屏蔽中断服务	0xFFFF

INTSUBMSK	位	描述	初始状态
保留	[31:15]	未使用	0
INT_AC97	[14]	0 = 可服务 1 = 屏蔽	1
INT_WDT	[13]	0 = 可服务 1 = 屏蔽	1
INT_CAM_P	[12]	0 = 可服务 1 = 屏蔽	1
INT_CAM_C	[11]	0 = 可服务 1 = 屏蔽	1
INT_ADC_S	[10]	0 = 可服务 1 = 屏蔽	1
INT_TC	[9]	0 = 可服务 1 = 屏蔽	1
INT_ERR2	[8]	0 = 可服务 1 = 屏蔽	1
INT_TXD2	[7]	0 = 可服务 1 = 屏蔽	1
INT_RXD2	[6]	0 = 可服务 1 = 屏蔽	1
INT_ERR1	[5]	0 = 可服务 1 = 屏蔽	1
INT_TXD1	[4]	0 = 可服务 1 = 屏蔽	1
INT_RXD1	[3]	0 = 可服务 1 = 屏蔽	1
INT_ERR0	[2]	0 = 可服务 1 = 屏蔽	1
INT_TXD0	[1]	0 = 可服务 1 = 屏蔽	1
INT_RXD0	[0]	0 = 可服务 1 = 屏蔽	1

图 17: INTSUBMSK

5.3 结果

当按下对应按键的时候，对方机子会亮对应的灯。

5.4 问题与总结

仍然没能把两个中断写在同一个程序内，需要进一步改进。

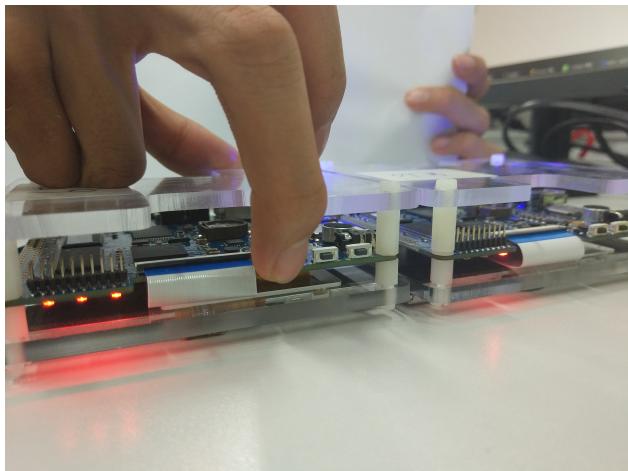


图 18: 按下 1 号机子