第一章　嵌入式系统的基本概念

嵌入式系统是**嵌入到**对象体系中的以嵌入式计算机为核心的专用计算机系统。

基本要素：嵌入、专用性、计算机

嵌入式系统软件结构一般包含四个层面：板级支持包（BSP）层、实时操作系统（RTOS）层、应用程序接口（API）层、应用程序层。

操作系统不是必须的，但都必须运行 BootLoader

嵌入式系统的硬件组成

硬件组成：处理器、存储器、I/O 设备，通信接口等

嵌入式系统中常用的通信接口包括哪些？

嵌入式系统的软件组成

板级支持包、实时操作系统、应用程序接口、应用程序

嵌入式微处理器(Microprocessor)就是和通用计算机的微处理器对应的 CPU。
嵌入式微控制器(Microcontroller Unit，MCU)又称为单片机，
嵌入式 DSP (Digital Signal Processor)处理器有两个发展方向：
嵌入式 SOC

SOPC 是 SOC 的一种，同时具备软硬件在系统可编程的功能(可裁剪、可扩充、可升级)

存储器类型

易失性：RAM,SRAM,DRAM

非易失性：ROM,EPROM,EEPROM,FLASH

嵌入式操作系统 uC/OS，VxWorks，WinCE, Linux(uClinux)

高精尖技术及实时性要求极高的领域中的是 VxWorks

Linux 必须运行在有 MMU（内存管理单元）的处理器上。

uClinux 可以运行在没有 MMU 的处理器上。

什么是软硬件协同设计？

什么是交叉开发？

第二章　ARM 技术概述

ARM 采用 RISC 体系结构

CISC 指令集设计的主要趋势是增加指令集的复杂度。而复杂指令集的高性能是以宝贵、有限的芯片面积为代价的。

RISC 的中心思想是精简指令集的复杂度、简化指令实现的硬件设计，硬件只执行很有限的最常用的那部分指令，大部分复杂的操作则由简单指令合成。RISC 思想大幅度提高了计算机性能价格比。

技术贡献

流水线——实现指令并行操作

高时钟频率和单周期执行

缺点

与 CISC 相比，通常 RISC 的代码密度低

RISC 不能执行 x86 代码

RISC 给优化编译程序带来困难

- Load/store 体系结构
- 固定的 32 位指令
- 3 地址指令格式

ARM7TDMI 是第一个支持 Thumb 的核

完成相同的操作，Thumb 指令集通常需要更多的指令，因此在对系统运行时间要求苛刻的应用场合 ARM 指令集更为适合

Thumb 指令集没有包含进行异常处理时需要的一些指令，因此在异常中断时，还是需要使用 ARM 指令，这种限制决定了 Thumb 指令需要和 ARM 指令配合使用。

ARM7TDMI

- ARM7　　32 位 ARM 体系结构 4T 版本
- T　　Thumb 16 位压缩指令集
- D　　支持片上 Debug
- M　　增强型 Multiplier
- I　　Embedded ICE 硬件，支持片上断点和观察点

- 进入 Thumb 状态
- 当操作数寄存器 Rm 的状态位 bit［0］为 1 时，执行 BX Rm 指令进入 Thumb 状态。如果处理器在 Thumb 状态进入异常，则当异常处理返回时，自动切换到 Thumb 状态。
- 进入 ARM 状态
- 当操作数寄存器 Rm 的状态位 bit［0］为 0 时，执行 BX Rm 指令进入 ARM 状态。如果处理器进行异常处理，在此情况下，把 PC 放入异常模式链接寄存器 LR 中，从异常向量地址开始执行也可以进入 ARM 状态

ARM 寄存器组成

R13 用作堆栈指针 SP

R14 用作子程序链接寄存器 LR

R15 用作程序计数器 PC

R15 使用的注意事项？

CPSR,SPSR 的格式，SPSR 的作用

共 7 种处理器模式

CPSR（当前程序状态寄存器）的低 5 位用于定义当前工作模式

- 除用户模式外的其他 6 种模式称为特权模式
- 特权模式中除系统模式以外的 5 种模式又称为异常模式，即
  - 快速中断 FIQ（Fast Interrupt Request）
  - 普通中断 IRQ（Interrupt Request）
  - 管理 SVC（Supervisor）
  - 中止 ABT（Abort）

未定义 UND（Undefined）

ARM 处理器总共有 37 个寄存器，可以分为以下两类寄存器

- 31 个通用寄存器
  - R0-R15；

- R13_svc、R14_svc;
- R13_abt、R14_abt;
- R13_und、R14_und;
- R13_irq、R14_irq;
- R8_fiq-R14_fiq。

- 6 个状态寄存器
  - CPSR；SPSR_svc、SPSR_abt、SPSR_und、SPSR_irq 和 SPSR_fiq

- 子程序返回：MOV PC,LR
- BX LR

ARM7 的 3 级流水线，哪 3 级？（冯.诺依曼结构）

ARM9 的 5 级流水线，哪 5 级？（哈佛结构）

ARM 的存储器层次

寄存器组——存储器层次的顶层，访问时间几个 ns （访问速度最快）

片上 RAM——具有和片上寄存器组同级的读/写速度

片上 Cache——10ns，2 级片外 Cache（几十 ns）

主存储器——访问时间 50ns

硬盘——几十 ms

- 什么是 ARM 异常中断向量表？它有何作用？存储在什么地方？
- 支持大端、小端存储 格式（S3C44B0X 如何选择大、小端存储）
- 可支持的数据格式（字节、半字、字）

第三章 ARM 指令集

1．以下立即数是否合法？

0x0103C000 ✗

0x12800000 √

可以由0x4A循环右移10位得到

2．请列举2个合法立即数？

0x4000003B（0xED循环右移2位）

0x0016C000（0x5B循环右移18位）

- 前变址模式：

LDR R0,[R1, # 4] ;R0←[R1 + 4]

- 自动变址模式：

LDR R0,[R1, # 4]!;R0←[R1 + 4],R1←R1 + 4

- 后变址模式：

LDR R0,[R1], # 4 ;R0←[R1],R1←R1 + 4

- **堆栈类型**
- 满递增：指令 LDMFA、STMFA；
- 空递增：指令 LDMEA、STMEA；
- 满递减：指令 LDMFD、STMFD；
- 空递减：指令 LDMED、STMED。

递增递减都是针对入栈操作而言，满堆栈和空堆栈的操作在入栈和出栈时也是不同的

使用 MRS 和 MSR 指令，通过修改 CPSR 寄存器，实现打开/关闭 IRQ 中断的两个子程序，注意不能影响其它位？

```
;子程序：使能IRQ中断
Enable_IRQ
    MRS   R0, CPSR
    BIC   R0, R0,#0x80
    MSR   CPSR_c,R0
    MOV   PC,LR
```

```
;子程序：禁止IRQ中断
Disable_IRQ
    MRS   R0, CPSR
    ORR   R0, R0,#0x80
    MSR   CPSR_c,R0
    MOV   PC,LR
```

- ARM 汇编语言的伪操作、宏指令与伪指令
- 与 ARM 指令集相比，Thumb 指令集具有哪些局限？
- ARM 处理器如何进入和退出 Thumb 指令模式？
- ATPCS 定义寄存器组中的**{R0~R3}**作为参数传递和结果返回寄存器。如果参数数目超过四个，则使用堆栈进行传递。入栈的顺序与参数顺序相反，即最后一个数据先入栈。

- 汇编语言访问 C 语言定义的全局变量
- C 语言调用汇编语言（主程序用 C 语言，子程序用汇编语言）
- 汇编程序要使用 **EXPORT** 关键字来作声明：本程序可被其他程序调用；
- C 语言程序使用 **extern** 关键字来声明所调用的汇编程序。
- 汇编程序调用 C 语言程序（主程序用汇编语言，子程序用 C 语言）

第四章 基于 S3C44B0X 嵌入式系统应用开发实例

S3C44B0X 是一款 SoC，S3C44B0X 的核是 ARM7TDMI
S3C44B0X 主要管脚的作用：
nGCS[7:0](片选)，OM[1:0](bank0 的总线宽度)
ENDIAN(大小端选择)，EINT[7:0](外部中断请求)
nRESET(复位)

基于 ARM 的嵌入式最小系统

启动代码 Bootloader 的执行过程：

1. 建立异常向量表
2. 禁止看门狗，关闭所有中断
3.初始化存储系统
4.初始化各模式下的堆栈指针
5.初始化应用程序执行环境
6. 切换处理器模式
7.跳转到 main()函数

无操作系统下的驱动程序的设计规则：

每个硬件设备的驱动程序都会被单独定义为一个软件模块，它包含硬件功能实现的.c 文件和函数声明的.h 文件。

I/O 接口的作用

S3C44B0X I/O 接口的编址方式

掌握 I/O 端口的应用编程

如：在 S3C44B0X 的 I/O 端口外接 2 盏 LED 灯，请用 C 程序实现 LED1、LED2 灯依次点亮，接着 LED1、LED2 灯依次熄灭。（设 LED1、LED2 依次接在端口 C 的第 0、1 管脚,置 0 为点亮，置 1 为熄灭）

```
Void Port_init(void)
{
rPDATC=0x0003;
rPCONC=0x00000005;
rPUPC=0x0;
}
```

```
Void led1_on(void)
{
    rPDATC &=0xFFFE;
}

Void led1_off(void)
{
rPDATC |=1;
}
Void led2_on(void)
{
    rPDATC & =0xFFFD;
}

Void led2_off(void)
{
rPDATC |=(1<<1);
}
```

```
Void main (void)
{
    Port_init();
    led1_on();
    delay(1000);
    led2_on();
    delay(1000);
    led1_off();
    delay(1000);
    led2_off();
}
```

S3C44B0X 中断控制器应用编程

中断系统应用

1. 中断控制器初始化;
2. 编写中断服务程序。

中断服务程序

在中断服务程序中要完成以下几个部分的内容:

屏蔽相同中断源的中断请求

中断功能实现

清除中断挂起位

重新使能屏蔽掉的中断设计

```
中断控制器初始化
void init_Ext(void)
{          /*使能中断*/
           rINTMOD=0x0;            /*IRQ 中断模式处理*/
           rINTCON=0x01;           /*允许 IRQ 中断，向量中断模式*/
           /*配置外部 4567 的中断服务程序*/
           rINTMSK&=~(BIT_GLOBAL|BIT_EINT4567);
           pISR_EINT4567=(int)Eint_lsr;
           /*对端口 G 的配置*/
           rPCONG=0xFFFF;       /*EINT 模式*/
           rPUPG=0x0;                  /*上拉电阻使能*/
           rEXTINT=rEXTINT|0x22220000;   /*EINT4567 下降沿*/
           rI_ISPC=BIT_EINT4567;              /*清除挂起位*/
           rEXTINTPND=0xF;     /*清除 EXTINTPND 寄存器*/
}
```

```
中断服务程序
void    Eint_lsr (void)
{
   unsigned char which_int;
   rINTMSK = rINTMSK | BIT_EINT4567; //禁止  EINT4567
  Which_int=rEXTINTPND;
  rEXTINTPND=0xF;
   rI_Ispc=BIT_EINT4567;
//  中断服务程序主体开始
。。。
//中断服务程序主体结束
rINTMSK &= (~(BIT_GLOBAL|BIT_EINT4567));   //重新使能 EINT4567

}
```

例子 INT2 和 INT0

```
中断初始化
void init_Ext(void)
{        /*使能中断*/
         rINTMOD=0x0;          /*IRQ 中断模式处理*/
         rINTCON=0x01;         /*允许 IRQ 中断，向量中断模式*/
         /*设置中断服务程序的入口地址*/
         pISR_EINT2=INT2_int;
         pISR_EINT0=INT0_int;
         /*对端口 G 的配置*/
         rPCONG=0xFFFF;      /*EINT 模式*/
         rPDATG=0x00;
         rPUPG=0x0;                  /*上拉电阻使能*/
         rEXTINT=0x22222222;   /*所有外部硬件中断为下降沿模式*/
         rI_ISPC=(BIT_INT2|BIT_INT0);              /*清除挂起位*/
        /*使能外部中断 BIT_INT0 和 BIT_INT2*/
         rINTMSK&=(~(BIT_GLOBAL|BIT_INT2|BIT_INT0));
}
```

```
void   INT0_int (void)
{
  rINTMSK = rINTMSK | BIT_EINT0; //禁止  EINT0
  rI_ISPC = BIT_EINT0;       //清除挂起位
  LED_off( );                        //熄灭 LED
  rINTMSK &= (~(BIT_GLOBAL|BIT_EINT0)); //重新使能 EINT0
}
void   INT2_int (void)
{
  rINTMSK = rINTMSK | BIT_EINT2; //禁止  EINT2
  rI_ISPC = BIT_EINT2;       //清除挂起位
  LED_on( );                         //点亮 LED
  rINTMSK &= (~(BIT_GLOBAL|BIT_EINT2)); //重新使能 EINT2
}
```

S3C44BOX UART 应用编程

```
URAT 初始化：主要是根据需要对寄存器进行相应的配置。
以中断模式为例
Void   uart_Init (int mclk,int baud,char port) //中断模式下初始化设置
{    int i;
```

```
    if (mclk == 0)
    {   mclk = MCLK; }
    i = (int)((mclk/16.)/baud + 0.5) -1;
    if (port == 0)                    //UART0 配置各控制寄存器
    {    rUFCON0 = 0x0;                 // 禁用 FIFO
         rUMCON0 = 0x0;                //禁用 AFC
         rULCON0=0x3;                  //正常模式，无奇偶校验，一个停止位，8 个数据位
         rUCON0=0x345;                 //RX 电平触发，TX 电平触发，禁用延时中断，使用 RX
错误中断，正常操作模式，中断请求或轮询模式
         rUBRDIV0 = i;                  //( (int)(mclk/16/baud + 0.5) -1 );
       }
    if ((rINTPND&BIT_UTXD0))      /检测 uart 发送中断是否已经挂起
      {
        rI_ISPC = BIT_UTXD0; //清除未决中断
      }
    if ((rINTPND&BIT_URXD0))      //检测 uart 接收中断是否已经挂起
      {
        rI_ISPC = BIT_URXD0; //清除未决中断
      }
rINTMOD=0x0;
rINTCON=0x01;
rINTMSK&=~(BIT_GLOBAL|BIT_UTXD0|BIT_URXD0);
pISR_URXD0=(unsigned)Uart0_RxInt;       // 将串口接收 ISR 函数的入口地址放入到相应
的中断向量跳转地址中
pISR_UTXD0=(unsigned)Uart0_TxInt;       // 将串口发送 ISR 函数的入口地址放入到相应
的中断向量跳转地址中
}
```

UART 通信过程中的两个基本函数——字符的收与发

```
接收字符的实现函数：
char Uart_Getch(void)
{
    if(whichUart==0)
    {
        while(!(rUTRSTAT0 & 0x1)); //准备好接收数据
        return RdURXH0();
    }
    else
    {
```

```
        while(!(rUTRSTAT1 & 0x1)); /准备好接收数据
        return    RdURXH1();
    }
}
```

```
发送字符的实现函数：
void Uart_SendByte(int data)
{
    if(whichUart==0)
    {
        if(data=='\n')
      {
          while(!(rUTRSTAT0 & 0x2));
          WrUTXH0('\r');   //发送回车字符
      }
      while(!(rUTRSTAT0 & 0x2)); //等待，直到 THR 为空.
      WrUTXH0(data);
    }
else
    {
        if(data=='\n')
        {
            while(!(rUTRSTAT1 & 0x2));
            WrUTXH1('\r');
        }
        while(!(rUTRSTAT1 & 0x2));   //等待，直到 THR 为空
        WrUTXH1(data);
    }
}
```

S3C44B0X 看门狗定时器应用编程

```
什么是看门狗？看门狗功能？什么是喂狗？
void initWDTimer(void)
{
  rINTMOD=0x0;          /*IRQ 中断模式处理*/
   rINTCON=0x01;        /*允许 IRQ 中断，向量中断模式*/
  rINTMSK &= ~(BIT_GLOBAL | BIT_WDT);    //使能中断
   rI_ISPC=BIT_WDT;
   pISR_WDT=WDT_int;
    rWTCON = (255<<8) | (3<<3) |(1<<2);
                                          //时钟周期= 1/256/128，中断使能
    rWTDAT = 2000;
```

```
    rWTCNT = 2000;
    rWTCON = rWTCON | (1<<5);   // 使能看门狗
}
喂狗程序：
Void   feeddog (void)
{
        rWTCNT=rWTDAT;
}
```

看门狗中断的编程

重新调整看门狗定时器的预分频值和分频器的分频值，让看门狗定时器每两秒发生一次中断，并在五秒后复位。

改变 rWTDAT，rWTCNT 和 f_ucSecondNo