# 信息物理系统—技术与系统
# Cyber Physical System – Technique and System

第3章 软件平台

刘松波

# 第3章 CPS计算平台实现
# --软件技术

**3.1、无操作系统**

3.2、嵌入式操作系统

3.3、嵌入式操作系统中的基本概念

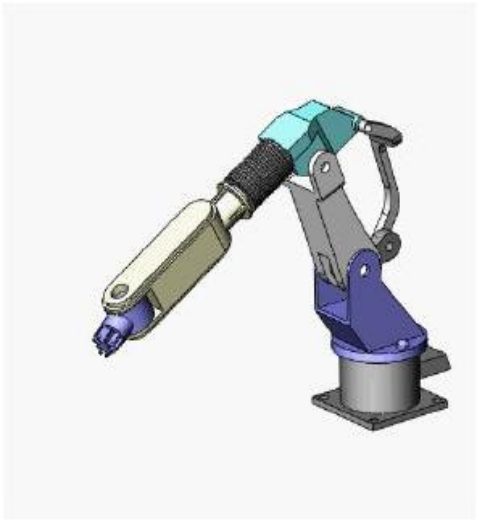
3.4、典型嵌入式操作系统

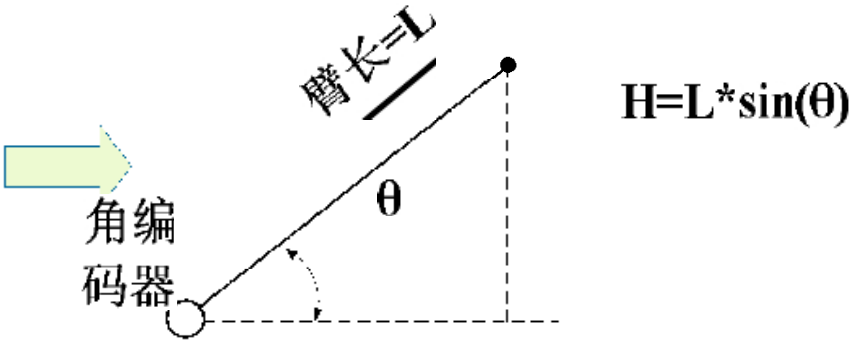
# 3.1、无操作系统

- **3.1.1 嵌入式系统软件高级语言开发**
- 3.1.2 嵌入式系统C与标准C几点区别
- 3.1.3 嵌入式系统中的可移植性
- 3.1.4 嵌入式系统软件调度方法

# 3.1.1 嵌入式系统软件高级语言开发

**为什么要用高级语言？　理由1：大幅减少编程劳动量**






**C语言：1行**

```
#include<math.h>
Float H,L,Cta;
H=L*sin(Cta);
```

**汇编语言：2000行**

**(1)泰勒展开**

**(2)多项式运算**

**(3)浮点计算**

# 3.1.1 嵌入式系统软件高级语言开发

**为什么要用C语言？　　理由2：代码的通用性**

```
main()
{
  …
}
```

C语言源文件

VC 编译器

ICC430 编译器

Keil-C51 编译器

PC机(8086)器码

430微处理器机器码

51微处理器机器码

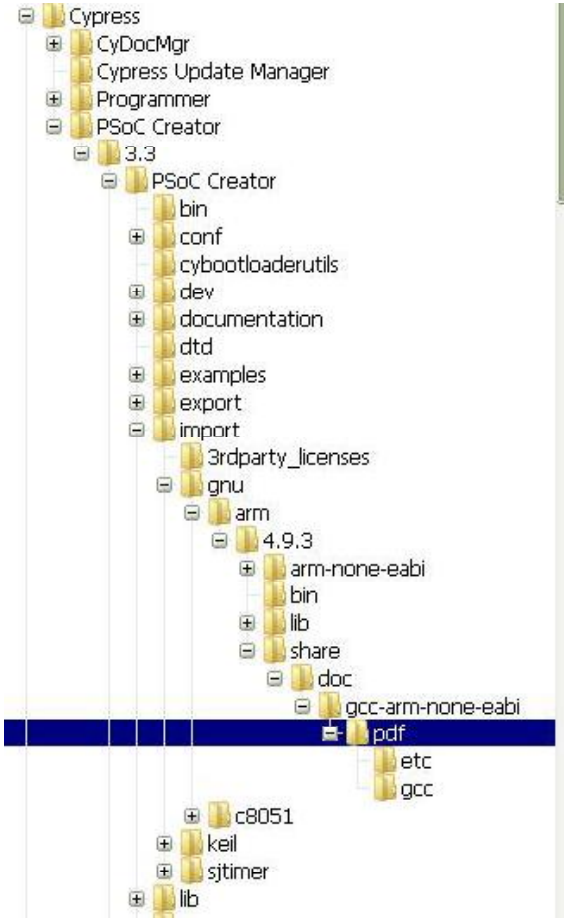
- C语言可以为世界上任何一款处理器编程 (包括所有单片机)
- 在任何一款单片机上开发的程序，都可以移植到别的处理器上。
- 代码的模块化、可复用性 => 减少重复劳动、加快开发速度。

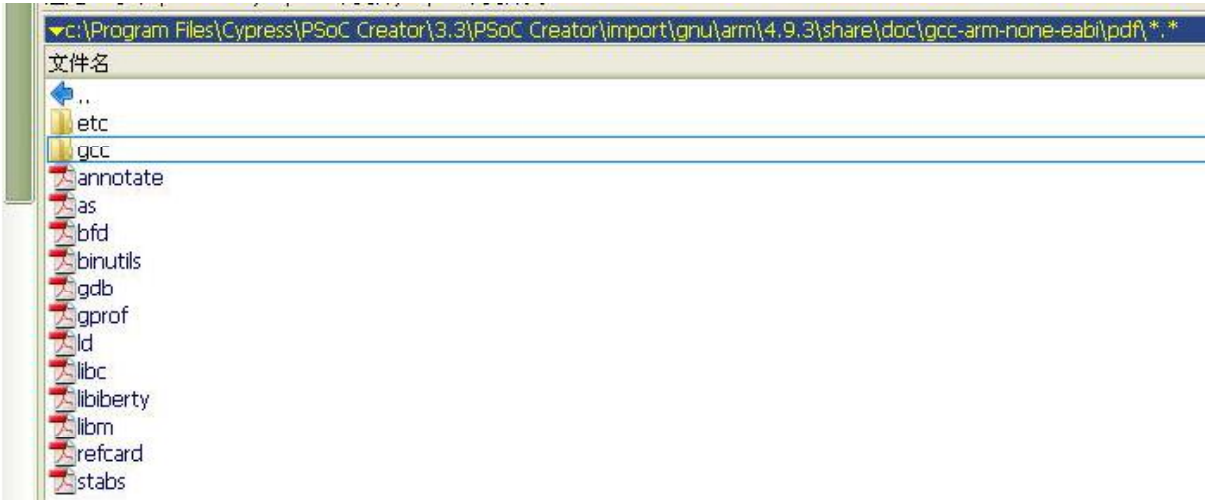
# 3.1.1 嵌入式系统软件高级语言开发

**单片机的C语言怎么学？**

| TC/ VC++ /C#... | C-430 | Keil-C |
| --- | --- | --- |
| 与PC机有关的特殊部分 | 与MSP430单片机有关的特殊部分 | 与8051单片机有关的特殊部分 |
| 标准C（ANSI-C）（大一已经学过） | 标准C（ANSI-C） | 标准C（ANSI-C） |

**结论：共性部分同学们在大一就已经学过**
**只需要学习与特定处理器相关部分（很少）**




**PSoC Creator集成开发环境所配备的PSoC 5LP编译器为GCC-ARM（GNU Compiler Collection for ARM Embedded Processors），是一款免费的C/C++编译器。其文档位于图中gcc目录内。**

**C预处理器、链接器、汇编器、调试器、函数库等相关软件工具的文档位于图中的pdf目录及其子目录内。**

# 3.1、无操作系统

- 3.1.1 嵌入式系统软件高级语言开发
- **3.1.2 嵌入式系统C与标准C几点区别**
- 3.1.3 嵌入式系统中的可移植性
- 3.1.4 嵌入式系统软件调度方法

# 3.1.2 嵌入式系统C与标准C几点区别

## 之一：变量定义

**MSP430的C语言**

| 变量类型 | 比特数 |
|---|---|
| char | 8 |
| unsigned char | |
| int | 16 |
| unsigned int | |
| long | 32 |
| unsigned long | |
| long long | 64 |
| unsigned long long | |
| float | 32 |
| double | 32或64 |

**冯-诺依曼架构**

**8051的C语言**

| 变量类型 | 比特数 |
|---|---|
| bit | 1 |
| sbit | 1 |
| char | 8 |
| unsigned char | |
| int | 16 |
| unsigned int | |
| long | 32 |
| unsigned long | |
| float | 32 |
| double | 32 |

| 定位符 | 变量位置 |
|---|---|
| bdata | 20-2F |
| data | 内部128 |
| idata | 内部256 |
| pdata | 外部256 |
| xdata | 外部64k |
| code | ROM64k |

**哈佛架构**

**处理器、编译器类型的不同，在变量定义上与VC略有不同。**

## 几个特殊关键词的含义:

**const**：定义常量。const关键字定义的常量被放在ROM中，常用于定义如系数表、显示段码表等。

**static**：相当于本地全局变量，在函数内使用，可以避免全局变量使用混乱。

**volatile**：定义“挥发性”变量。编译器将认为该变量的值会随时改变，对该变量的任何操作都不会被优化掉。

```
XBYTE[2]=0x55;
XBYTE[2]=0x56;
XBYTE[2]=0x57;
XBYTE[2]=0x58;
```

# 3.1.2 嵌入式系统C与标准C几点区别

## 之二：特殊寄存器操作

### MSP430的C语言

定义特殊寄存器：
__no_init volatile unsigned
char 寄存器名 @ 寄存器地址;
【例】：
__no_init volatile unsigned
char P1OUT @ 0x0021; //定义P1
或：#include <msp430x42x.h>

操作特殊寄存器：

```
P1OUT=0x01;   //写特殊寄存器
a=P1OUT;      //读特殊寄存器
P1OUT^=0x01; //读-改-写操作
```

### 8051的C语言

定义特殊寄存器：
sfr 寄存器名=寄存器地址;

【例】：
sfr P1=0x90; //定义P1端口

或：#include <reg51.h>

操作特殊寄存器：

```
P1=0x01;      //写特殊寄存器
a=P1;         //读特殊寄存器
P1^=0x01;     //读-改-写操作
```

结论：(1)特殊寄存器经过定义之后，可以像变量一样操作。
(2)包含单片机头文件，就已经完成了变量定义工作。

# 3.1.2 嵌入式系统C与标准C几点区别

## 之三：位操作

### MSP430的C语言

```
P2OUT |=  0x01;   //P2.0置高
P2OUT &= ~0x02;//P2.1置低
P2OUT ^=  0x04;  //P2.2取反

P2OUT |=  BIT0;   //P2.0置高
P2OUT &= ~BIT1;//P2.1置低
P2OUT ^=  BIT2;  //P2.2取反

P1OUT |=    BIT1+BIT2+BIT3;
            //P1.1/2/3全置高
P1OUT&=~(BIT1+BIT2+BIT3);
            //P1.1/2/3全置低
```

### 8051的C语言

```
sbit P20=P2^0;
sbit P21=P2^1;
sbit P22=P2^2;

P20=1;   //P2.0置高
P21=0;   //P2.0置高
P22^=1; //P2.0置高
```

**结论：若处理器不支持单个位的操作，则要用左边的方式实现。**

# 3.1.2 嵌入式系统C与标准C几点区别

## 之四：中断

### MSP430的C语言

```
#pragma vector = 中断向量
__interrupt void 函数名(void)
{中断服务程序}
```

中断向量：

0xFFFE=RESET ；0xFFFC=NMI中断
0xFFF8=ADC；　0xFFF4=WDT看门狗
0xFFF2=串口接收; 0xFFF0=串口发送

… …

【范例】

```
#pragma vector = WDT_VECTOR
__interrupt void WDT_ISR (void)
{
  ...在这里写看门狗中断服务程序
}
```

### 8051的C语言

```
void 函数名(void) interrupt X
using Y
{中断服务程序}
```

X: 0=外中断0；1=定时器0；
　　2=外中断1；3=定时器1；
　　4=串口中断；

Y：0~3中断服务程序内用的寄存器组

【范例】

```
void UART_ISR(void) interrupt 4
{
  ...在这里写串口中断服务程序
}
```

**结论：(1)C语言将中断作为特殊函数 (2)该函数由中断机制调用**

# 3.1.2 嵌入式系统C与标准C几点区别

## 之五：内部函数(intrinsic Function)

### MSP430的C语言

使用前要 #include <intrinsics.h>

__low_power_mode_0(); //进入低功耗模式0
__low_power_mode_1(); //进入低功耗模式1
__low_power_mode_2(); //进入低功耗模式2
__low_power_mode_3(); //进入低功耗模式3
__delay_cycles(long int cycles); //延迟
__enable_interrupt(); //打开总中断开关
__disable_interrupt(); //关闭总中断开关
__no_operation(); //空操作
__swap_bytes(x); //高低字节交换，返回整型值
…
…

### 8051的C语言

使用前要 #include <intrins.h>

_crol_(); 字符循环左移
_cror_ ();字符循环右移
_irol_(); 整数循环左移
_iror_(); 整数循环右移
_lrol_(); 长整数循环左移
_lror_(); 长整数循环右移
_nop_(); 空操作(8051 NOP 指令)
_testbit_(); 测试并清零位(8051 JBC 指令)
…

**结论：(1)内部函数是将该款单片机的一些特殊指令做成函数形式**
**(2)在使用前，要包含内部函数头文件。**

# 3.1.2 嵌入式系统C与标准C几点区别

## 之六：函数的可重入性

**思考题：主程序中1+2+3计算会出错吗？**

```
===主程序===
while(1)
{
   …
   result=sum(1,2,3);
   if (result!=6) LED=ON;
   …
}
```

```
===中断程序===
void ISR() interrupt X
{
   …
   res=sum(4,5,6);
   …
}
```

```
===sum程序===
int sum(int x,y,z)
{
   int s;
   s=x+y+z;
   return(s);
}
```

**MSP430的C语言**

函数默认采用栈传递参数，允许重入和递归调用，不出错。

**8051的C语言**

函数默认采用静态传递参数，不允许重入和递归调用，会出错。

*用 reentrant关键字指定栈传递

**结论：(1)低端单片机因资源少，函数用静态传递，不可重入。**
**(2)中高端单片机编译器，函数用栈传递，允许重入。**

# 3.1.2 嵌入式系统C与标准C几点区别

## 讨论与总结：

1、即使是初学者，也完全可以在不深入了解汇编指令系统的情况下直接开始C语言开发。

2、任何一款处理器的C语言，都是在标准C语言（ANSI-C）上增加对相应处理器的特殊操作而构成。

3、C编译器的表现通常非常优秀。如果没有几年的手工汇编经验，很难写出比C编译器更高效的代码。

4、如果在编程过程中将硬件差异集中到某个很小的局部，整个程序通过很简单的修改就能编译成另一CPU的机器码。这就是所谓的“代码移植”，是嵌入式软件设计最重要的思想之一。

5、在一开始就树立可移植性、模块化的概念，养成好的编程习惯，对减少重复劳动、加快开发速度有很大帮助——这正是下一部分的主要目的。

# 3.1、无操作系统

- 3.1.1 嵌入式系统软件高级语言开发
- 3.1.2 嵌入式系统C与标准C几点区别
- **3.1.3 嵌入式系统中的可移植性**
- 3.1.4 嵌入式系统软件调度方法

# 3.1.3 嵌入式系统中的可移植性

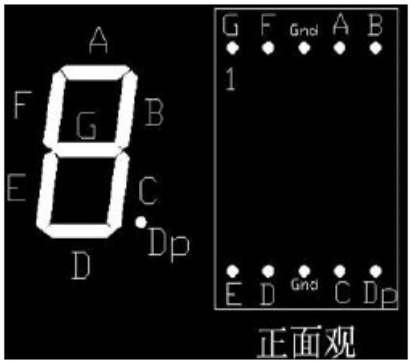
- 1）考虑微处理器差异（消除CPU差异）
- 若希望自己写的程序在不同处理器上运行，首先需要了解这些处理器的不同之处。其中包括硬件的不同以及特殊语法的差异。例如MSP430微控制器没有位操作指令，8051微控制器有，若两者之间程序需相互移植，首先需要考虑这个差异。

```
#include "reg51.h"                  /*51单片机*/
sbit LED=P2^0;                      /*定义I/O口，(51I/O口低电平驱动)*/
#define LED_ON      LED=0           /*LED亮*/
#define LED_OFF     LED=1           /*LED灭*/
```

```
#include "MSP430X42X.h"                         /*MSP430单片机 */
#define LED_ON      P2OUT |= BIT0               /*LED亮*/
#define LED_OFF     P2OUT &=~BIT0               /*LED灭*/
```

# 3.1.3 嵌入式系统中的可移植性 续1

- 2）**硬件差异**




更改数码管硬件布线，
无需重写段码表

```
//-----------------------------------------------------
// 宏定义，数码管a-g各段对应的比特，更换硬件只用改动以下8行
//-----------------------------------------------------
#define d       0x01        //  AAAAA
#define g       0x02        // F     B
#define b       0x04        // F     B
#define a       0x08        //  GGGGG
#define e       0x10        // E     C
#define f       0x20        // E     C
#define c       0x40        //  DDDDD

//-----------------------------------------------------
//  用宏定义自动生成段码表
//-----------------------------------------------------
  unsigned char code LED_Tab[] =  //LED段码表，放在ROM中
{
    a + b + c + d + e + f,          // Displays "0"
    b + c,                          // Displays "1"
    a + b + d + e + g,              // Displays "2"
    a + b + c + d + g,              // Displays "3"
    b + c + f + g,                  // Displays "4"
    a + c + d + f + g,              // Displays "5"
    a + c + d + e + f + g,          // Displays "6"
    a + b + c,                      // Displays "7"
    a + b + c + d + e + f + g,      // Displays "8"
    a + b + c + d + f + g,          // Displays "9"
};
```

# 3.1.3 嵌入式系统中的可移植性 续2

- 3）封装
  - 封装是指将软/硬件对象的属性与行为绑定在一起，并放置在一个模块单元内。该单元负责将所描述的属性隐藏起来，外界对其内部属性的所有访问只能通过提供的应用程序接口实现。

```
* 名        称：UART_Init()
* 功        能：初始化串口。设置其工作模式及波特率。
* 入口参数：Baud          波特率      (300~115200)
            Parity        奇偶校验位 ('n'=无校验    'p'=偶校验    'o'=奇校验)
            DatsBits      数据位位数  (7或8)
            StopBits      停止位位数  (1或2)
* 出口参数：返回值为1时表示初化成功，返回0表示参数出错，设置失败
* 范        例: UART_Init(9600,'n',8,1) //设成9600bps，无校验，8位数据，1位停止位
              UART_Init(2400,'p',7,2) //设成2400bps，偶校验，7位数据，2位停止位
*********************************************************/
char UART_Init(long int Baud,char Parity,char DataBits,char StopBits)
{
   时钟选择、波特率自动计算、不适应的波特率自动剔除、校验位、寄存器设置…
   代码很长很复杂，但是全部封装隐藏在函数体内，对外接口非常简洁
}
```

**Good！**

**接口简单**

**程序复杂但永久免维护**

- 学者常见问题

```
TMOD=0x20;    //定时器T1自装载模式，产生波特率
TH1=0xfd;
TL1=0xfd;       // 9600bps @ 11.0592MHz
TR1=1;
SCON=0x50;    //串口模式
EA=1;ES=1;    //中断允许
```

**poor！**

**没有进行封装**

**带来大量重复劳动**

# 3.1.3 嵌入式系统中的可移植性 续3

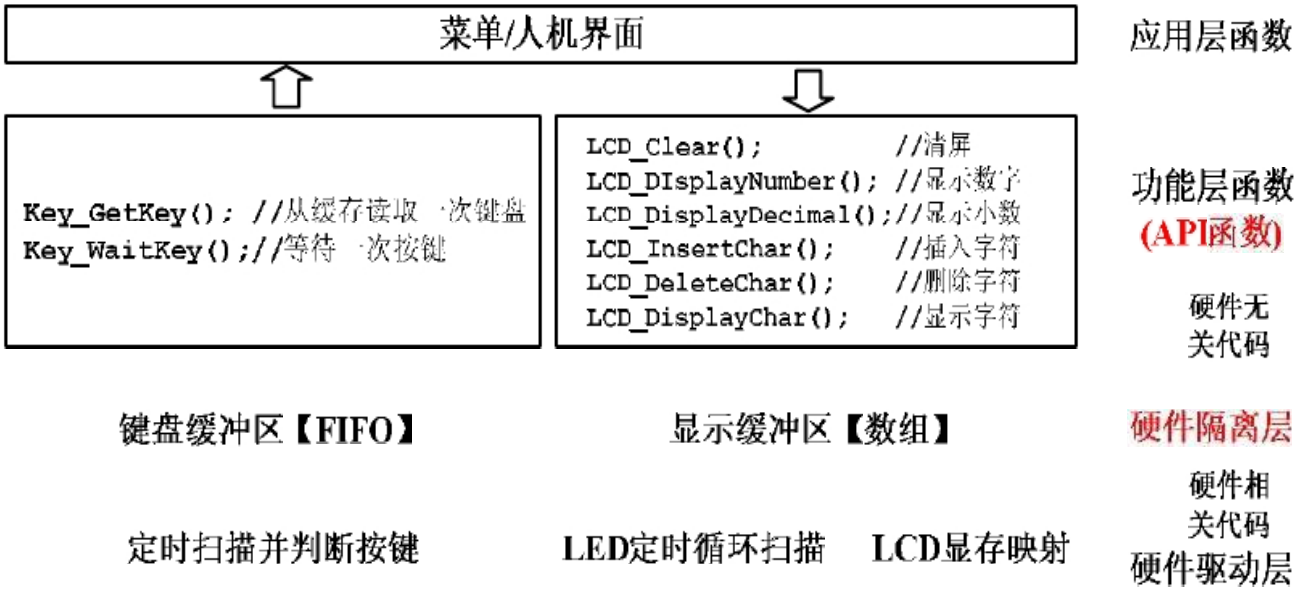
- 4）应用程序接口(API)
  - 接口是对软/硬件对象的抽象，对外提供的服务和访问功能，即：对象封装后对外呈现成什么样？原则：（1）使用方便（2）功能丰富（3）但不要过多过滥
  - 【例】为液晶显示模块规划应用程序接口：


```
void    LCD_Init();                 //初始化LCD
void    LCD_On();                   //开启LCD
void    LCD_Off();                  //关闭LCD
void    LCD_DisplayDecimal(); //显示小数
void    LCD_DisplayNumber ();//显示整数
void    LCD_DisplayChar();      //在指定位置显示字符
void    LCD_InsertChar();       //插入一个字符
void    LCD_DeleteChar();       //删除一个字符
void    LCD_Clear();            //清屏
```

# 3.1.3 嵌入式系统中的可移植性 续4

- 5）软件层次

**例：菜单程序**




**（1）杜绝跨层调用　　（2）功能模块可以任意拼接、组合**

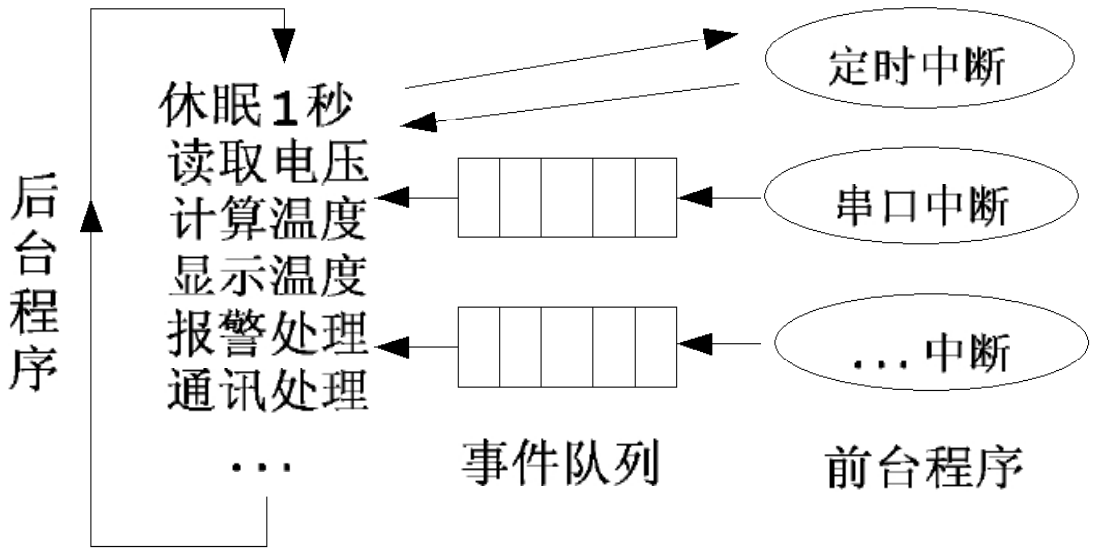
**（3）要有硬件隔离层(HAL)　（4）更换任意层代码，上下层建筑不变**

# 3.1、无操作系统

- 3.1.1 嵌入式系统软件高级语言开发
- 3.1.2 嵌入式系统C与标准C几点区别
- 3.1.3 嵌入式系统中的可移植性
- **3.1.4 嵌入式系统软件调度方法**

# 3.1.4 嵌入式系统软件调度方法

## （一）前后台程序的基本概念



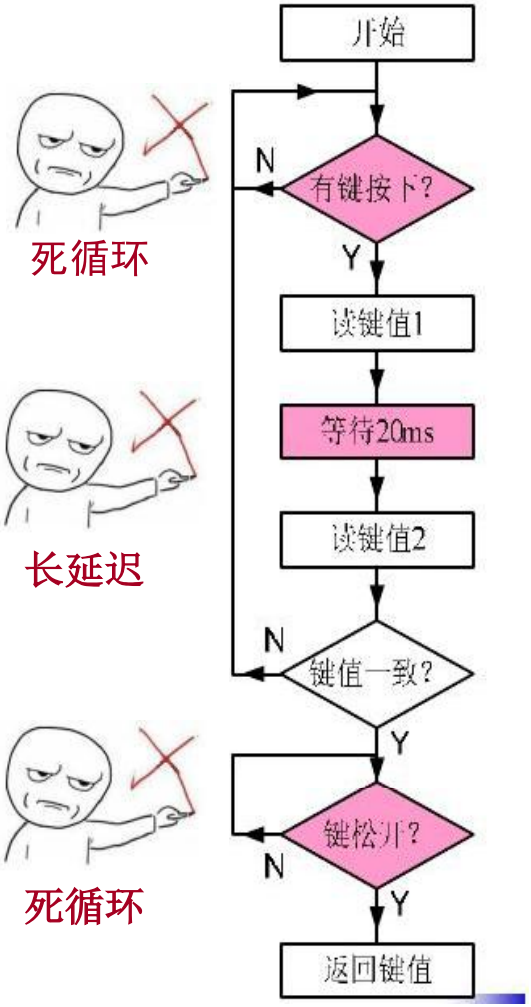

**任务(Task)：指完成某一单一功能的程序**
**后台程序：对时间要求不严格的任务，通常在主循环内执行**
**前台程序：要求快速响应或者时间严格的任务，通常中断内**
**队列/缓冲：用于暂存后台来不及处理的前台事件**

# （二）前后台程序的编写基本原则

## （1）任何一个任务都不能阻塞CPU。

每个任务都应主动结束，让出CPU。不能有等待、死循环、长延时等环节（对初学者来说是难点！）

```
#define KEY_IO (P1IN&(BIT5+BIT6+BIT7)) //P1.567
#define NOKEY (BIT5+BIT6+BIT7)   //低电平有效
char KEY_GetKey()          //读键函数
{ unsigned char TempKey1,TempKey2;
 WAITKEY:
    while(KEY_IO==NOKEY); // 等待键按下
    TempKey1=KEY_IO;      // 读一次键值
    Delay_ms(20);         // 延迟 20ms
    TempKey2=KEY_IO;      // 再读一次键值
    if(TempKey1!=TempKey2) // 如果两次不相等
      {
        goto WAITKEY;     // 认为是抖动，重新读取
      }
    while(KEY_IO!=NOKEY); // 等待按键释放
    return(TempKey1);     // 返回键值
}
```




一般经典键盘程序，实际上是病态代码！

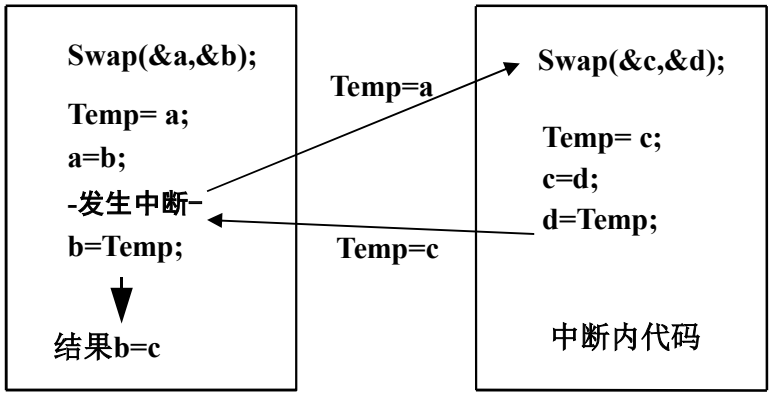
# 3.1.4 嵌入式系统软件调度方法

## （2）关注函数重入问题

**可重入（Reentrancy）：函数在自己调用自己的时候，不必担心数据被破坏。**
从软件工程角度对函数可重入的作用可以解释为：具有可重入性的函数能够被多个任务同时调用。在前后台程序中，任务都是顺序执行的，不存在多个任务同时调用一个函数的情况。但可能出现前台中断程序与后台任务同时调用某个函数。因此对于前后台公用函数，必须是可重入的。

【例】：

```
int Temp;
void Swap(int *x,int *y )
{/*交换两个整型变量*/
        Temp= *x;
        *x=*y;
        *y=Temp;
}
```



# 3.1.4 嵌入式系统软件调度方法

**（3）临界代码保护（Critical Code Protection）**

临界代码：运行时不可分割的代码，这部分代码不允许被中断打断。

**1） 依靠软件产生时间严格时序的程序段。**

```
/*******************************************************
 * 名        称：Pulse_10us()
 * 功        能：产生10us脉冲
 * 入口参数：无
 *******************************************************/
void Pulse_10us()
{
    _DINT();        // -------以下是临界代码区，不允许被中断-----
    IO=1;                       // 置高
    __delay_cycles(8); // 高电平持续10us (IO赋值本身耗2us)
    IO=0;                       // 置低
    _EINT();        // -------以上是临界代码区----------------
}
```

# 3.1.4 嵌入式系统软件调度方法

## 2）共享资源互斥性造成的临界代码

程序中任何可被占用的实体都称为“资源”。资源可以是硬件设备，如定时器、串口、打印机、键盘、LCD显示器等，也可以是变量、数组、队列、等数据。

可以被一个以上任务所占用的资源叫做“共享资源”。共享资源若不能同时被多任务占用，则具有“互斥性”（Mutual Exclusive）。

【例】：前后台同时调用液晶显示函数造成花屏

```
==后台程序==
void main()
{
   …
  LCD_Display(123);
   …
}
```

```
==前台程序==
void ISR() interrupt X
{
   …
   LCD_Display(456);
   …
}
```

# 3.1.4 嵌入式系统软件调度方法

**3）函数重入造成的临界代码（见函数重入，略）**

**4） CPU字长造成的临界代码**

【例】：ADC定时采样程序，若处理器是8位的，你能找出临界代码区吗？

==后台程序==

```
void main()
{ int Temp;
    …
    Temp=ADC_Val; 赋值
    …
}
```

==前台程序==

```
void ISR() interrupt X
{
    …
    ADC_Val=ADC_GetVal(); ADC采样16位
    …
}
```

# 3.1.4 嵌入式系统软件调度方法

## （4）临界代码的保护方法

### 1）关中断、开中断

隐患：如果在A函数的临界代码区调用了另一个函数B，B函数内也有临界代码区，从B函数返回时，中断被打开了，这将造成A函数后续代码失去临界保护。所以，使用该方法时，不能在临界代码区调用任何具其他有临界代码的函数！

### 2） 利用硬件栈保存中断使能状态

优点：不影响函数调用

缺点：很多单片机的C语言不支持操作硬件栈

### 3） 利用变量保存中断使能状态

缺点：每一段临界代码要消耗2-4字节内存

### 4） 利用模拟栈保存中断使能状态

缺点：速度慢

# 3.1.4 嵌入式系统软件调度方法

- 总结
  - 后台中任务顺序执行。每个后台任务中的内存（局部变量）在任务结束后可以全部释放，让给下一个任务使用。即使在RAM很少的处理器上也能同时执行众多任务。
  - 后台中任务顺序执行。天然避免了后台任务资源互斥问题，但仍需考虑前后台之间的资源互斥问题 。
  - 前后台程序的结构灵活，实现形式与实现手段多样，是使用最广的程序结构，但缺乏架构标准，维护、升级、排错都很困难。
  - 必须要程序员自己来判断和处理临界代码的隐患。
  - 程序多任务的执行依靠每个任务的非阻塞性来保证，是编程最大的难点，下一节将介绍的FSM将是解决这一问题的利器！

# 第3章 CPS计算平台实现 --软件技术

3.1、无操作系统

**3.2、嵌入式操作系统**

3.3、嵌入式操作系统中的基本概念

3.4、典型嵌入式操作系统

# 3.2、嵌入式操作系统

- **3.2.1 使用嵌入式操作系统的必要性**
- 3.2.2 嵌入式操作系统的特点
- 3.2.3 嵌入式操作系统的分类

# 3.2.1 使用嵌入式操作系统的必要性

- 1）嵌入式操作系统的应用提高了系统的可靠性
  - 在前后台系统模式下，常常采用串行编程的方法设计软件，程序模块之间的耦合度往往较高，一旦系统遇到强干扰，就会使运行的程序产生异常、出错、跑飞，甚至死循环，致使系统崩溃。
  - 在嵌入式操作系统管理的系统中，任务与任务之间的通信和控制都通过内核来实现，相互之间很少有直接的联系，在一般情况下，系统受到干扰后可能只是引起若干任务中的一个被破坏，而这个被破坏的任务可以通过专门的系统监控任务对其进行修复，如把有问题的任务删除掉等。

# 3.2.1 使用嵌入式操作系统的必要性 续1

- 2）提高了开发效率，缩短了开发周期
  - 在嵌入式实时操作系统环境下，开发一个复杂的应用程序，通常可以按照软件工程中的解耦原则将整个程序分解为多个任务模块。每个任务模块的调试、修改几乎不影响其他模块。商业软件一般都提供了良好的多任务调试环境。

# 3.2.1 使用嵌入式操作系统的必要性 续2

- 3）提高系统性能
  - 嵌入式实时操作系统为并发执行程序提供了可能，充分挖掘了CPU特别是高性能CPU的潜能；
  - 嵌入式操作系统本来就是为运行多用户、多任务操作系统而设计的，特别适于运行多任务实时系统。
  - 一方面多任务并发执行可以极大地提高系统运行效率，降低整体成本；另一方面，多任务设计有利于提高系统的可靠性和稳定性，使其更容易做到不崩溃。

# 3.2.1 使用嵌入式操作系统的必要性 续3

- 从某种意义上说，没有操作系统的计算机(裸机)是没有用的。在嵌入式应用中，只有把CPU嵌入到系统中，同时又把操作系统嵌入进去，才是真正的计算机嵌入式应用。开发人员一旦使用操作系统，就会对操作系统产生很大的依赖性。
- 使用嵌入式实时操作系统最大的缺点是增加了额外的ROM/RAM 开销、2% ~ 5%的CPU额外负荷以及内核的费用

# 3.2、嵌入式操作系统

- 3.2.1 使用嵌入式操作系统的必要性
- **3.2.2 嵌入式操作系统的特点**
- 3.2.3 嵌入式操作系统的分类

# 3.2.2 嵌入式操作系统的特点

- 1）体积小。
  - 嵌入式系统有别于一般的计算机处理系统，它不具备大容量的存储介质，而大多使用闪存和RAM作为存储介质。这就要求嵌入式操作系统只能存储和运行在有限的空间中，不能使用虚拟内存，中断的使用也受到限制。因此，嵌入式操作系统必须结构紧凑，体积微小；
- 2）实时性强。
  - 大多数嵌入式操作系统工作在实时性要求很高的场合。实时性强是嵌入式系统最大的优点，在嵌入式软件中最核心的莫过于嵌入式实时操作系统；

# 3.2.2 嵌入式操作系统的特点 续1

- 3）可裁剪
  - 从硬件环境来看，通用计算机系统具有标准化的CPU存储和I/O架构，而嵌入式系统的硬件环境只有标准化的CPU，没有标准的存储、I/O和显示器单元；
  - 从应用环境来看，通用操作系统面向复杂多变的应用，而嵌入式操作系统面向单一设备的固定的应用；
  - 从开发界面来看，通用操作系统给开发者提供一个“黑箱”，让开发者通过一系列标准的系统调用来使用操作系统的功能，而嵌入式操作系统试图为开发者提供一个“白箱”，让开发者可以自主控制系统的所有资源。
  - 对于一个嵌入式设备，它的功能是确定的，因此只要从原有操作系统中把这个特定应用所需的功能拿来即可。

# 3.2.2 嵌入式操作系统的特点 续2

- 4）可靠性高。
  - 在大多数情况下，嵌入式系统一旦开始运行就不需要人的过多干预，这就要求负责系统管理的嵌入式操作系统具有较高的稳定性和可靠性，而通用操作系统则无需具备这种特点。这导致桌面操作环境与嵌入式环境在设计思路上有重大的不同。
- 5）特殊的开发调试环境
  - 一个完整的嵌入式系统的集成开发环境一般需要提供的工具是编译/连接器、内核调试/跟踪器和集成图形界面开发平台。其中的集成图形界面开发平台包括编辑器、调试器、软件仿真器和监视器等。

# 3.2、嵌入式操作系统

- 3.2.1 使用嵌入式操作系统的必要性
- 3.2.2 嵌入式操作系统的特点
- **3.2.3 嵌入式操作系统的分类**

# 3.2.3 嵌入式操作系统的分类

- 目前，嵌入式操作系统总数超过150个。国外嵌入式操作系统已经从简单走向成熟。国内嵌入式操作系统的研究开发有两种类型，一类是基于国外操作系统二次开发完成的，如海信集团的基于Windows CE的机顶盒系统；另一类是中国自主开发的嵌入式操作系统，如凯思集团公司自主开发的嵌入式操作系统Hopen OS(女娲计划)、北京科银京成技术有限公司开发的DeltaOS(道系统)等

# 3.2.3 嵌入式操作系统的分类 续1

- **从应用面来分类**
  - 嵌入式操作系统可分为面向低端设备的嵌入式操作系统和面向高端设备的嵌入式操作系统两类。低端设备如各种工业控制系统、计算机外设、民用消费品的微波炉、洗衣机、冰箱等，其典型的操作系统是μC/OS-Ⅱ。高端设备如信息化家电、掌上电脑、机顶盒、WAP手机、路由器，常用的操作系统有Windows CE、Linux等。

# 3.2.3 嵌入式操作系统的分类 续2

- **从专业化程度来分类**。
  - 嵌入式操作系统可分为通用型嵌入式操作系统和专用型嵌入式操作系统两类。常见的通用型嵌入式操作系统有Linux、VxWorks、Windows CE、μC/OS-Ⅱ等。常用的专用型嵌入式操作系统有Smart Phone、Pocket PC、Symbian等

# 3.2.3 嵌入式操作系统的分类 续3

- **按实时性要求来分类。**
  - 嵌入式操作系统可分为强(硬)实时性嵌入式操作系统和弱(软)实时性嵌入式操作系统两类。强实时性嵌入式操作系统主要面向控制、通信等领域，如WindRiver公司的VxWorks、ISI的pSOS、QNX系统软件公司的QNX、ATI的Nucleus等。弱实时性嵌入式操作系统主要面向消费类电子产品(包括PDA、移动电话、机顶盒、电子书、WebPhone等)，如微软面向手机应用的Smart Phone操作系统。

# 3.2.3 嵌入式操作系统的分类 续4

- **按复杂程度分类**。
  - 简单操作系统、实时操作系统、面向Internet的操作系统。

# 第3章 CPS计算平台实现
# --软件技术

3.1、无操作系统

3.2、嵌入式操作系统

**3.3、嵌入式操作系统中的基本概念**

3.4、典型嵌入式操作系统

# 3.3、嵌入式操作系统中的基本概念

- *第0303章--CPS平台实现--软件技术--嵌入式操作系统中的基本概念.pptx*

# 第3章 CPS计算平台实现 --软件技术

3.1、无操作系统

3.2、嵌入式操作系统

3.3、嵌入式操作系统中的基本概念

**3.4、典型嵌入式操作系统**

# 3.4、典型嵌入式操作系统

- *第0304章--CPS平台实现--软件技术--典型嵌入式操作系统.pptx*

- 谢谢！