MCU 的分时软件框架

相对于PC的 Windows 系统而言，单片机嵌入式系统的软件复杂程度要低一些。

但无论是手机应用还是相对简单小家电应用，与 Windows 一样，其软件系统也同样存在快速开发、功能扩展、软件维护等需求，一个好的软件框架是设计师所追求的。

从应用目标来看，事件驱动的结构化框架是一种好的选择，下面的时间片分时框架将作为一个例子，对应用于家电控制等中小型规模的嵌入式控制系统而言是较理想的。

首先对下面的几个概念加以定义：

时基　系统在运行时，将系统运行的时间轨迹进行等分后的时间片段。也称为时间片基准或时基。时基长度因不同的系统应用而不同。如下的图形可以更好地帮助理解时基的概念。

任务　它是软件内部使用的概念，是将应用系统的各种软件行为进行结构化分割后得到的系统行为的较小部分。这些较小的软件行为可以在MCU的运行速度下，在每一个时基长度中都可以得到完成。

正因为这种结构化的分割，使得系统的开发、扩展和维护易于进行。从某种意义说，软件系统的设计更多的关注点和资源将放在这种结构化的分割中。更好、更合理的分割，将造就更好的分时软件框架。

硬件底层驱动　任务的一种，用于与硬件相关的操作。例如红外遥控器数据、按键信号的读取，LED的点亮等。

事件驱动　任务的一种，处理由硬件底层驱动传递进来的或将要传递给硬件底层驱动的外部事件信息。

以C语言来描述的分时软件框架例子如下，时间基准为0.5ms，其中使用了四个时基循环，系统的一次任务循环为2.0ms。当然，时基的长度和系统的时基循环长度因应用不同而不同，分时软件框架可以很好地进行伸缩。

```
//=========================================================
//
//          MCU的分时软件框架
//
//
// MCU:        Atmega168@4MHZ
//
// DATE:     V31    @ 09012005MONPM
//
//=========================================================

#define  ICD_SHOWER      0                                  // 0/1 : ICD_SHOWER OFF/ON

#define  uchar  unsigned char
#define uint    unsigned int
#define ulong unsigned long

#define  schar  signed char
#define sint    signed int
#define slong signed long

#include   <avr/io.h>
#include   <avr/interrupt.h>
#include   <avr/wdt.h>
#include   <avr/pgmspace.h>

#include   <M168.h>


//----------          I/O PIN define          ------------------


//----------          RAM & BIT define        -----------------


//----------          const        define       ------------------


//--------------      const code /  表    -------------


//------------------      option_reload_pro /  配置重载       ----------------

void option_reload_pro( void )
  {
  }
```

```
//------------------      initial_pro /  初始化    ----------------

void initial_pro ( void )
   {
        cli();
        wdt_reset();

        option_reload_pro();
   }

//---------------    time_base_pro /  时间基准     ----------------

void time_base_pro( void )
   {

   }


//---------------    key_pro /  读键     ----------------

void key_pro( void )
   {

   }

//---------------    key_process_pro /  键处理     ----------------

void key_process_pro( void )
   {

   }

//---------------    remoter_pro /  读遥控器     ----------------

void remoter_pro( void )
   {

   }

//---------------    remoter_process_pro /  遥控器数据处理     ----------------

void remoter_process_pro( void )
   {

   }

//---------------    MAIN    process     -----------------

int main( void )
   {
      initial_pro();

      while( 1 )
         {
            option_reload_pro();

            if( !bit_is_clear( TIFR0,OCF0A) )                                  // time_base: 0.5ms
               {
                  TIFR0 |= (1 << OCF0A );

                  T_BASE0_R.BYTE ++;
                  if( T_BASE0_R.BYTE == 0 )
                     T_BASE1_R.BYTE ++;

                 remoter_pro();

                  BUFF_0_R = T_BASE0_R.BYTE & 0B00000011;
                  switch ( BUFF_0_R )
                        {
                              case 0:
                                          key_pro();
                                          break;
                              case 1:
                                          key_process_pro();
                                          break;
                              case 2:
                                          break;
                              case 3:
                                          remoter_process_pro();
                                          time_base_pro();
                                          break;
                              default:
                                          break;
```

```
                    }
                }
            }
        }
```

时基的产生可以有多种方法。以 PIC 或 AVR MCU 为例，它们都有硬件定时器，确定好时基长度后，用硬件定时器产生相应的时间间隔。在例子中没有使用中断方式，因为时基长度是大于任务执行所需的时间长度的，采用查询方式足矣。

initial_pro（）为初始化程序，option_reload_pro（）为配置重载程序，它将有关于MCU的不变的系统配置，例如定时器长度、I/O口方向设置等重复地装入。从另一个角度看，也增加了系统的可靠性。值得注意的是，不同的MCU中有些参数的重载写入时会改变系统的运行状态。

remoter_pro（）为红外遥控器的输入的底层驱动，数据编码以NEC码为例。依香侬采样定理，其实时要求相对较高，以0.5ms为采样间隔；remoter_process_pro()为事件驱动程序，处理由有效红外遥控器按键操作触发的系统动作；key_pro()为按键的底层驱动；key_process_pro()为事件驱动程序，处理由有效按键操作触发的系统动作。