[ucgui 原创]关于 UCGUI 的配制文件的说明

UCGUI 与大家熟悉的 UCOS 一样，也提供了大量的可配制功能选项，采用的方式也是在头文件中进行预定义，这与 UCOS 是一样的

. 预定义将决定可以使用的 UCGUI 图形功能以及 UCGUI 运行所须要的功能，下面我们将讨论一下 UCGUI 的配制请情，并探讨一下其

最小资源占用是多少，即最小内存占用量.

从 GUI 源码当中，可以看到有这样一个文件间---Config，这个文件夹下面包含了以下胡三个头文件:

1.GUICONF.h----------基本的 GUI 预定义控制.
2.GUITouchConf.h----------关于触屏的控制预定义.
3.LCDConf.h----------有关 LCD 夜晶显示的参数控制.

一. 关于 GUICONF.h

这个文件当中，包含了最基本的一些 GUI 图形预定义控制.首先看其内容如下:

```
#ifndef GUICONF_H
#define GUICONF_H

#define GUI_OS                  (1) /* Compile with multitasking support */
#define GUI_SUPPORT_TOUCH       (1) /* Support a touch screen (req. win-manag
er) */
#define GUI_SUPPORT_UNICODE     (1) /* Support mixed ASCII/UNICODE strings
*/

#define GUI_DEFAULT_FONT        &GUI_Font6x8
#define GUI_ALLOC_SIZE          12500 /* Size of dynamic memory ... For WM and
 memory devices*/

/*********************************************************************
*
*       Configuration of available packages
*/

#define GUI_WINSUPPORT          1 /* Window manager package available */
#define GUI_SUPPORT_MEMDEV      1 /* Memory devices available */
#define GUI_SUPPORT_AA          1 /* Anti aliasing available */

#endif /* Avoid multiple inclusion */
```

可以看到，这里面的配制并不是很多，只有大约十项左右.

1.GUI_OS----------是否须要多任务支持.

可以看到. 在 GUI_Protected.h 文件当中有如下的代码:

```
.....
#if !GUI_OS
  #define GUI_LOCK()
  #define GUI_UNLOCK()
  #define GUITASK_INIT()
#else
  #define GUI_LOCK() GUI_Lock()
  #define GUI_UNLOCK() GUI_Unlock()
  #define GUITASK_INIT() GUITASK_Init()
#endif
......
```

并且在 Core\guitask.C 文件中，如果 GUI_OS 未定义，则会:

```
void GUI_Unlock(void) {}
void GUI_Lock(void) {}
void GUITASK_Init(void) {}
void GUITASK_StoreDefaultContext(void) {}
```

将上面的几个函数都定义为空的.那什么都不做.所以 GUI_OS 的主要作用是在进行多任时支持时的互锁功能.当有了多任务支持,

才有必要提供锁的功能. 这种锁功能对于资源的访问可以做到独占性而避免出现死锁或使用过程中被修改的情况,在模拟器当中

，这几个函数的实现都是通过 WINDOWS 的信息量功能来实现的..

GUI_OS 未定义时,将被定义为 0.

```
#ifndef GUI_OS
  #define GUI_OS          0
#endif
```

2.关于 GUI_SUPPORT_TOUCH

这个是关于触摸屏的支持，首先必须是非功过硬件有触摸的支持，详情参看源码.

3.关于 GUI_SUPPORT_UNICODE

是否支持 UNICODE 码的预定义，如果进行了预定义,那么显示字符前会先转化为 UNICODE 码. 主要在 GUIUC0.C 文件中提供.

4.GUI_DEFAULT_FONT

缺省字体，在 GUI\CORE\FONT 文件夹下，提供了十几种字体的支持，这里的预定义将决定哪种字体将被支持，要 UCGUI 中字体是

点阵形式处理的，而且是与 GUI 编译在一起，使用哪种字体就会将哪种字体编译进去.在这里. 如果要支持多种字体，而又采用

字体与 GUI 代码编译在一起的做法，将使 GUI 的资源占用非常大.缺省字体占用关不多 8K 左右.

5.GUI_ALLOC_SIZE

GUI_ALLOC_SIZE 指定 UCGUI 可自己动态使用的内存量，默认的是 12500,

差不多是 12K 左右.UCGUI 中使用动态内存管理是类似 NEW,DELETE 的,原理是一样的，但比较简单一点. 使用的是双向链表，在整

个内存(12500)没有使用之前,链表中只有一个节点，当申请过一次后，就会变成两个节点，一个记载申请的结点的内存信息,另

外一个则是剩余内存信息.

GUIAlloc.c 文件当中提供了内存管理的功能.

```
typedef struct {
  tALLOCINT Off;       /* Offset of memory area */
  tALLOCINT Size;      /* usable size of allocated block */
  HANDLE Next;      /* next handle in linked list */
  HANDLE Prev;
} tBlock;

/*********************************************************************
*            Static data
*/

GUI_HEAP GUI_Heap;       /* Public for debugging only */
static tBlock aBlock[GUI_MAXBLOCKS];

struct {
  int       NumUsedBlocks, NumFreeBlocks, NumFreeBlocksMin;       /* For statistical p
urposes only */
  tALLOCINT NumUsedBytes, NumFreeBytes, NumFreeBytesMin;
} GUI_ALLOC;

#define HMEM2PTR(hMem) (void*)&GUI_Heap.abHeap[aBlock[hMem].Off]
```

GUIHEAP 在 GUI.h 中提供的.

```
typedef union {
  int aintHeap[GUI_ALLOC_SIZE/4];      /* required for proper alignement */
  U8  abHeap[GUI_ALLOC_SIZE];
} GUI_HEAP;

extern GUI_HEAP GUI_Heap;      /* Public for debugging only */
```

以上几点是内存管理的关键数据结构.

GUI_Heap 是一个联合体结构, 其内数据可以以 INT 整型, 一次以四个字节来访问; 也可以以字节来访问,一次访问一个字节, 这

块内存是以数组的形式来提供的.

GUI_HEAP GUI_Heap; 这个全局变量即是 UCGUI 中可以动态分配使用的全部内存, 在分配使用时, 是按照块来使用的.tatic

tBlock aBlock[GUI_MAXBLOCKS];这个全局变量定义了可以用来管理动态内存的节点信息的块数组.

GUI_MAXBLOCKS 的大小如下:

```
#ifndef GUI_MAXBLOCKS
  #define GUI_MAXBLOCKS (2+GUI_ALLOC_SIZE/32)
#endif
```

如果是总共内存为 12500 个字节的内存, 那么则要 48 个这样的块结构.其占用内存大小为 48*sizeof(tBlock)=48*16=768 字节,这

是最大的占用, 小的时候为一半, 结点大小为 8 个字节.这 768 个字节是为了动态内存分配负出的代价.

当第一次申请一块内存时. 整个动态内存的最初状态是:

```
  aBlock[0].Size = (1<<GUI_BLOCK_ALIGN); /* occupy minimum for a block */
  aBlock[0].Off  = 0;
  aBlock[0].Next = 0;
```

那么申请动态内存之前, 第一步就是先从块结构数组(即记录动态内存分配状况的数组)中找一个空闲的元素. 来作为记录将要

分配内存的节点. 第二步,所有的块结构数组当中查找出可供使用的内存, 并将此内存的偏移地址, 这个偏移是相对于 GUI_Heap

这个整个动态内存的起始点的偏移.

举一个明显的例子如下:
当初始化时:

```
aBlock[0].Size  =  4;
aBlock[0].Off  =  0;
aBlock[0].Next  =  0;
aBlock[0].Pre  =  0;
```

当成功分配两块大小为 500 的内存时;

```
aBlock[0].Size  =  4;
aBlock[0].Off  =  0;
aBlock[0].Next  =  &aBlock[1];
aBlock[0].Pre  =  &aBlock[2];

aBlock[1].Size  =  500;
aBlock[1].Off  =  3;
aBlock[1].Next  =  &aBlock[2];
aBlock[1].Pre  =  &aBlock[0];


aBlock[2].Size  =  500;
aBlock[2].Off  =  503;
aBlock[2].Next  =  &aBlock[0];
aBlock[2].Pre  =  &aBlock[1];
```

此时 GUI_Heap 的内存使用状态如下:
从 GUI_Heap.abHeap[0]到 GUI_Heap.abHeap[3]为一块，使用状态记录于 aBlock[0];
从 GUI_Heap.abHeap[4]到 GUI_Heap.abHeap[503]为一块，使用状态记录于 aBlock[1];
从 GUI_Heap.abHeap[504]到 GUI_Heap.abHeap[1007]为一块，使用状态记录于 aBlock[2];
此时剩余的是 GUI_Heap.abHeap[1008]---GUI_Heap.abHeap[12499]这一段.

如果在此基础上，要现分一块大小为为 1002 字节的，则会首先讨调整大小(1002+1<<GUI_BLOCK_ALIGN-1) & 0x3，将其大小调整

为 4 的倍数对齐.

其后，则会通过查找，找到 aBlock[3].Size 记录为 0，那么这个块节点可以用来记录一个内存配信息.

再次，从 aBlock[0]查起，发觉 0---3 已用,再醒 aBlock[1]，得知 4-503 已用,同此原理,找到 0---1007 为已用，之后都没有使用,

那么都是可以用的，那么此次分配将从 1008 开始...直到 1008----1008+1004.

```
aBlock[0].Size = 4;
aBlock[0].Off = 0;
aBlock[0].Next = &aBlock[1];
aBlock[0].Pre = &aBlock[3];

aBlock[1].Size = 500;
aBlock[1].Off = 3;
aBlock[1].Next = &aBlock[2];
aBlock[1].Pre = &aBlock[0];


aBlock[2].Size = 500;
aBlock[2].Off = 503;
aBlock[2].Next = &aBlock[3];
aBlock[2].Pre = &aBlock[1];

aBlock[3].Size = 1004;
aBlock[3].Off = 2012;
aBlock[3].Next = &aBlock[0];
aBlock[3].Pre = &aBlock[2];
```

那么这种情况是最理想的状态下，如果说内存经过多次分配与释放，那么情况将复杂一些..

比如说，如果刚才所说的那个第一次申请的 500 个字节内存已经用完了，被释放掉了...那么如下所示..

```
aBlock[0].Size = 4;
aBlock[0].Off = 0;
aBlock[0].Next = &aBlock[2];    //不再指向 1,直接指向 0.
aBlock[0].Pre = &aBlock[3];

aBlock[1].Size = 0;
aBlock[1].Off = 0;
aBlock[1].Next = 0;
aBlock[1].Pre = 0;


aBlock[2].Size = 500;
aBlock[2].Off = 503;
aBlock[2].Next = &aBlock[3];
aBlock[2].Pre = &aBlock[0];  //不再指向 1，直接指向 0.
```

```
aBlock[3].Size  =  1004;
aBlock[3].Off  =  2012;
aBlock[3].Next  =  &aBlock[0];
aBlock[3].Pre  =  &aBlock[2];
```

那么在这种情况下，如果申请一块 200 字节的内存，则会将 aBlock[1]来记录此次内存分配.
那么这样直接将会导致知一个结果

就是在 aBlock[1]与 aBlock[2]这间产生一个 300 字节的 HOLE.即洞，那么如果再来一次 200 蒙阴的内存分配. 那么 aBlock[4]将会

用于记载此次内存分配.

所以总结一下...
1. 申请了 500 字节,
2. 再申请 500 字节,
3. 再申请一次 1002 字节
   然后，进行释放,将第一次的 500 字节释放掉
4. 接着再申请一次 200,
5. 再申请一次 100,
6. 现申请一次 400.

那么此时的内存格局将会是什么样子呢???

如下所示:

```
aBlock[0].Size  =  4;
aBlock[0].Off  =  0;
aBlock[0].Next  =  &aBlock[2];    //不再指向 1,直接指向 0.
aBlock[0].Pre  =  &aBlock[3];

aBlock[1].Size  =  200;  //第 1 次申请了 500,被 free,后被用作第 4 次配 200.
aBlock[1].Off  =  4;
aBlock[1].Next  =  &aBlock[2];
aBlock[1].Pre  =  &aBlock[1];


aBlock[2].Size  =  500;
aBlock[2].Off  =  503;
aBlock[2].Next  =  &aBlock[3];
aBlock[2].Pre  =  &aBlock[0];  //不再指向 1，直接指向 0.
```

```
aBlock[3].Size  =  1004;
aBlock[3].Off  =  2012;
aBlock[3].Next  =  &aBlock[4];
aBlock[3].Pre  =  &aBlock[2];

aBlock[4].Size  =  100;  //第 5 次分配 100. 还是在原来的第 1 次配 500 的空间内.
aBlock[4].Off  =  203;
aBlock[4].Next  =  &aBlock[5];
aBlock[4].Pre  =  &aBlock[3];

aBlock[5].Size  =  400;  //第 6 次分配 400, 第 1 次申请的 500  free 后已经用 300,剩 200 不够,只能从后面中找空间
aBlock[5].Off  =  2013+400;
aBlock[5].Next  =  &aBlock[0];
aBlock[5].Pre  =  &aBlock[4];
```

那么此时内存空间图如下:

0-----3--(第 4 次分配占用)---203--(第 5 次分配占用)---303--(free)---503----(第二次)-----1003----(第三次)---2012----

--(第六次)----2413

这样, 这个内存分配的过程与原理就比较清晰的展示出来了....

5.关于 GUI_WINSUPPORT

GUI_WINSUPPORT 是用于是否须要窗口支持的, 在 UCGUI 当中提供了 Frame winodws,edit, button, progress 等基本的图形控件,

与 WINDOWS 上的类似, 但目前的功能则还不尽人意.图形效果与功能都不强..

6.关于 GUI_SUPPORT_MEMDEV

GUI_SUPPORT_MEMDEV 是指显示时, 是直接写一个一个象素到显示设备, 还是在内存当中当画好所有要画一屏幕点后再全部一次

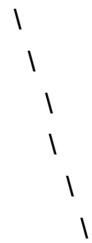
写到显示设置.
这两点的主要区别是, 如果一个一个点写, 则会出现闪的现象, 因为一个一个点的画, 如果速度慢, 可以扦到画一个复杂的图

形时,是分几步一点点的画出来的...
但是如果先画到内存当中, 再一次性 COPY 所有象素点到显示设备就不会有这种现象, 是一个连续的过程..画点时是连续的画,没

有间隔.

7.关于 GUI_SUPPORT_AA

GUI_SUPPORT_AA 是指是否须要对边界进行模糊填充的功能，比如说，画一条斜线，可以看到是一段一段的:



这样从效果上看，那么不是特别美观，如果可以在线的偏离的周围进行线的颜色的淡化处理,即在线的周围填充一点*近线的颜

色的象素点..那么看上去将会偏离得没那么明显，比较模糊一点.

二. GUITouch.h 的配制.

```
#ifndef GUITOUCH_CONF_H
#define GUITOUCH_CONF_H


#define GUI_TOUCH_AD_LEFT  20
#define GUI_TOUCH_AD_RIGHT  240
#define GUI_TOUCH_SWAP_XY    1
#define GUI_TOUCH_MIRROR_X    0
#define GUI_TOUCH_MIRROR_Y    1

#endif /* GUITOUCH_CONF_H */
```

由以上可以看出.GUITouch.h 可预定义的选项更少了，只是一些数值上的定义，基本不会影响到 UCGUI 编译后的代码大小.

三.关于 LCDConf.h

```
#define LCD_XSIZE      (320)   /* X-resolution of LCD, Logical coor. */
#define LCD_YSIZE      (240)   /* Y-resolution of LCD, Logical coor. */

#define LCD_BITSPERPIXEL (16)
```

```
#define LCD_CONTROLLER 1375

//Add by houhh 20050420
/*********************************************************************
*
*               List of physical colors
*
**********************************************************************
*/

/*********************************************************************
*
*               Full bus configuration
*
**********************************************************************
*/

#define LCD_READ_MEM(Off)          *((U16*)        (0xc00000+(((U32)(Off))<<1)))
#define LCD_WRITE_MEM(Off,data)     *((U16*)        (0xc00000+(((U32)(Off))<<1)))
=data
#define LCD_READ_REG(Off)          *((volatile U16*)(0xc1ffe0+(((U16)(Off))<<1)))
#define LCD_WRITE_REG(Off,data)     *((volatile U16*)(0xc1ffe0+(((U16)(Off))<<1)))
=data

   .
   .
   .
   .

#endif /* LCDCONF_H */
```

1. LCD_XSIZE/LCD_YSIZE

这是指定 LCD 显示屏的宽高的.

2.LCD_BITSPERPIXEL

是指定屏幕上一个象素由几位来表示. 位数越多, 能够表示的颜色数就越多. 一屏所占用的内存就越多.

3.