1269C GFX

Microchip图形库及其应用

课程目标

完成本课程后，您将能够：

- 应用技巧来协助编写用于图形库的底层驱动程序
- 编写在**LCD**面板上显示图像、字体和原语的程序
- 编写在**LCD**面板上显示和控制控件的程序
- 创建能充分利用**Microchip**图形库的**GUI**应用程序代码

日程安排

- 图形库概述
- **LCD**系统硬件
- 编写底层驱动程序的技巧和诀窍
- 原语层
- 使用字体和图像
- 绘制控件
- 与用户交互（消息回调）
- 高级功能（绘图回调）
- 汇总（多屏幕）

Microchip图形库概述

Microchip图形库特性

- 适用于16位和32位PIC MCU
- 免费授权的库
  - 包含源代码
  - 支持多个LCD控制器
- 支持多种输入方法
- 显示大小和分辨率独立
- 低成本全功能的开发工具
  - Explorer 16（129.99美元）
  - 图形PICtail Plus子板（135.00美元）
  - 免费的图像和字体转换工具
- 下载网址：www.microchip.com/graphics

图形PICtail™ Plus演示板

- 适用于**16**位和**32**位**PIM**
- 跳线选择：
  - 已安装的LCD模块，内置控制器（LGDP4531）

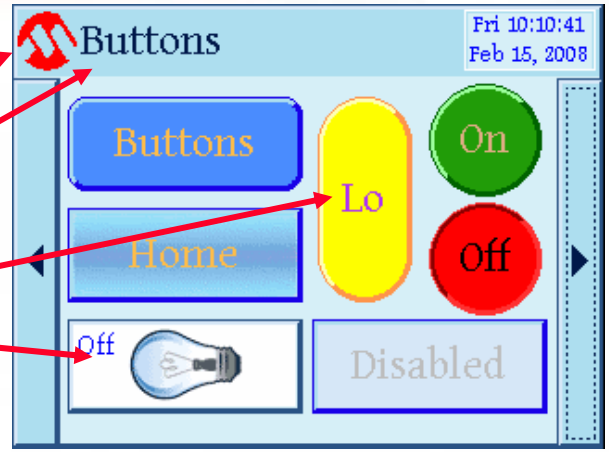
  或者
  - RGB式样连接器和SSD1906控制器

图形库

图像

静态文本

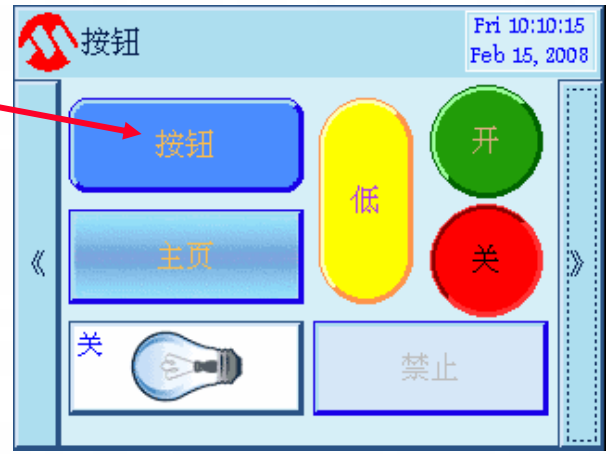
按钮



免费

以及更多....

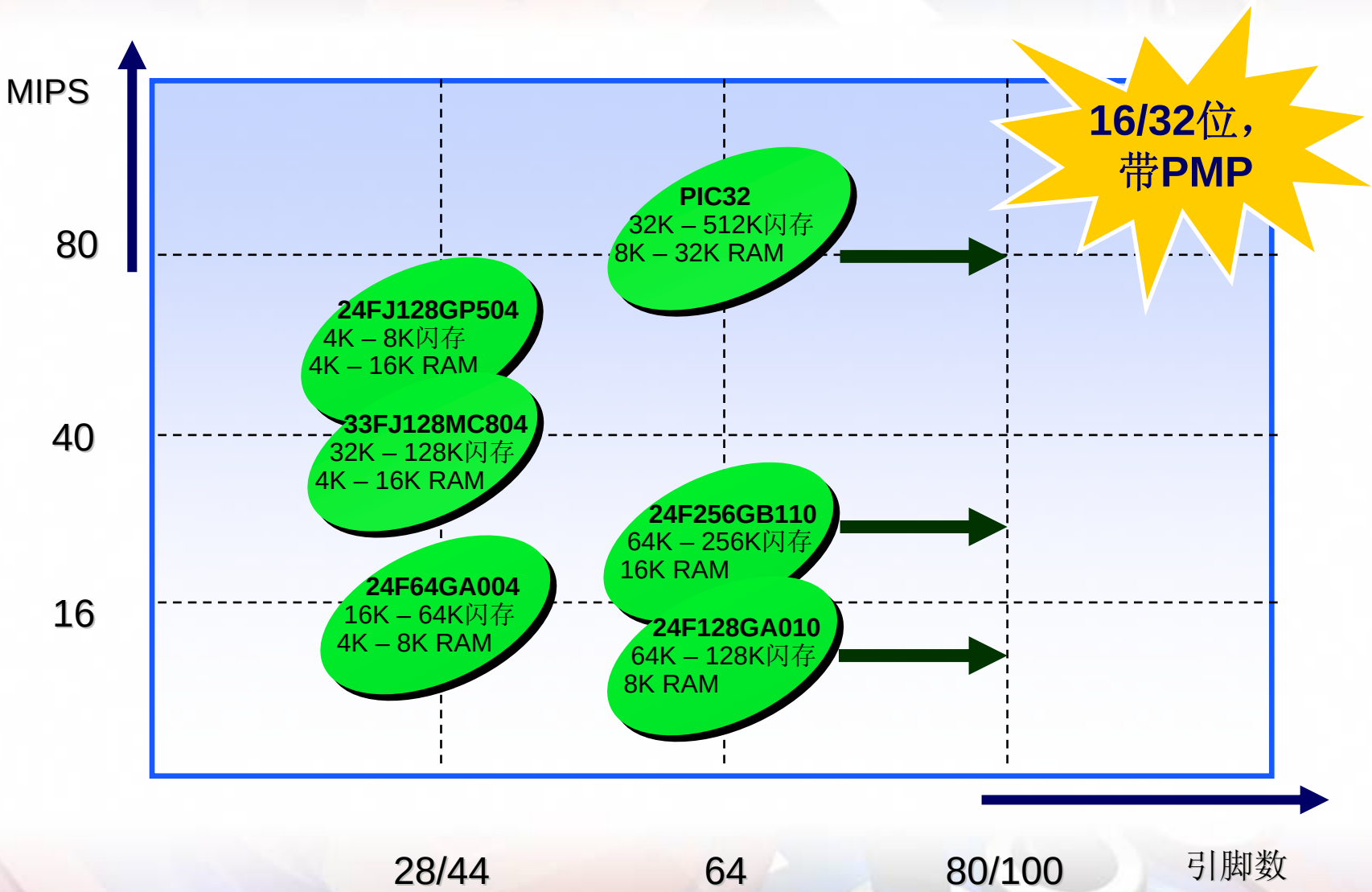
中文字体



关键优势：

- 对Microchip客户免费
- 模块化
- 存储器/MIPS要求低
- Microchip支持和培训

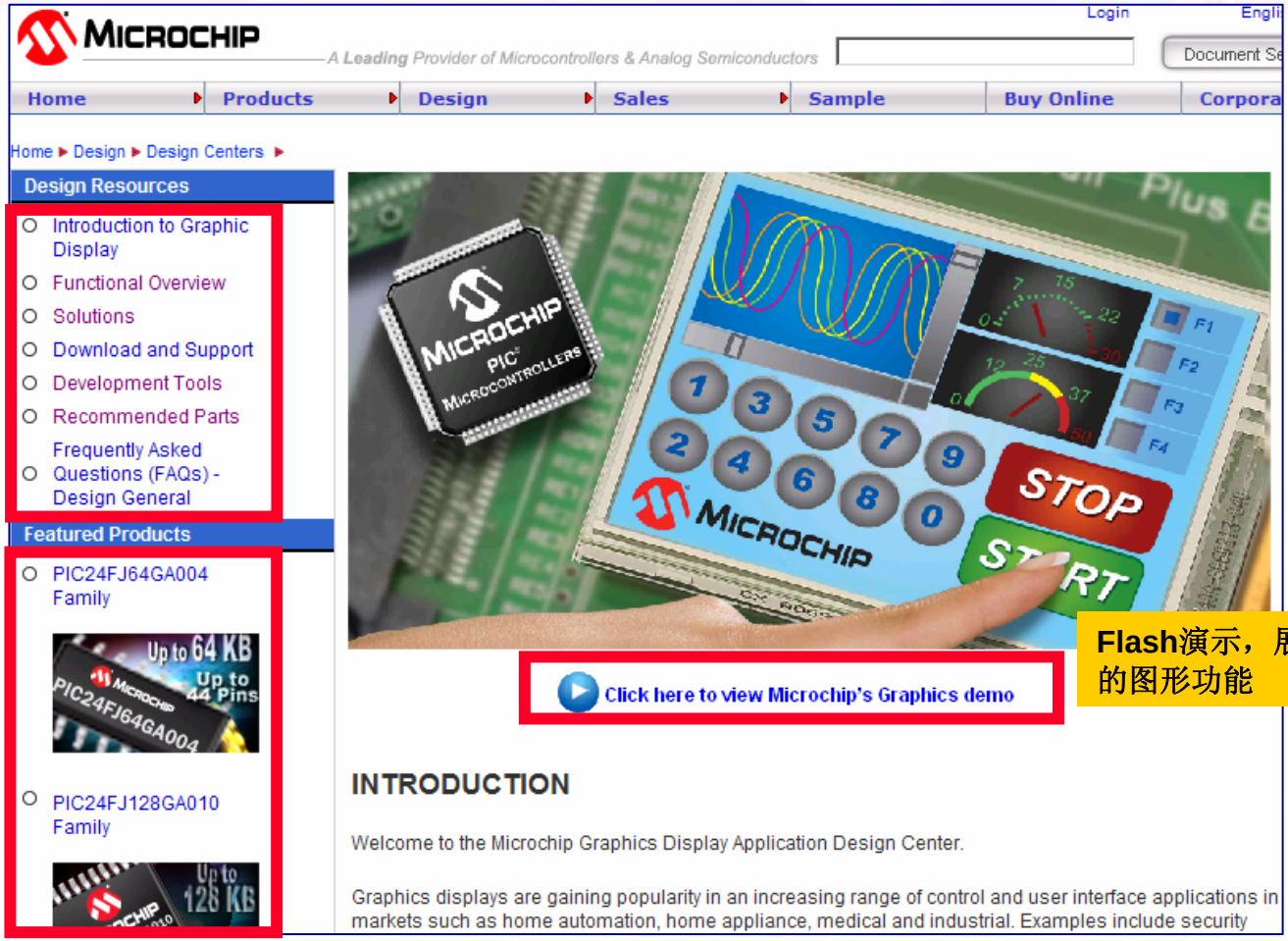
16位和32位PIC® MCU支持的图形



图形设计中心

站点导航

产品页面



Flash演示，展示Microchip的图形功能

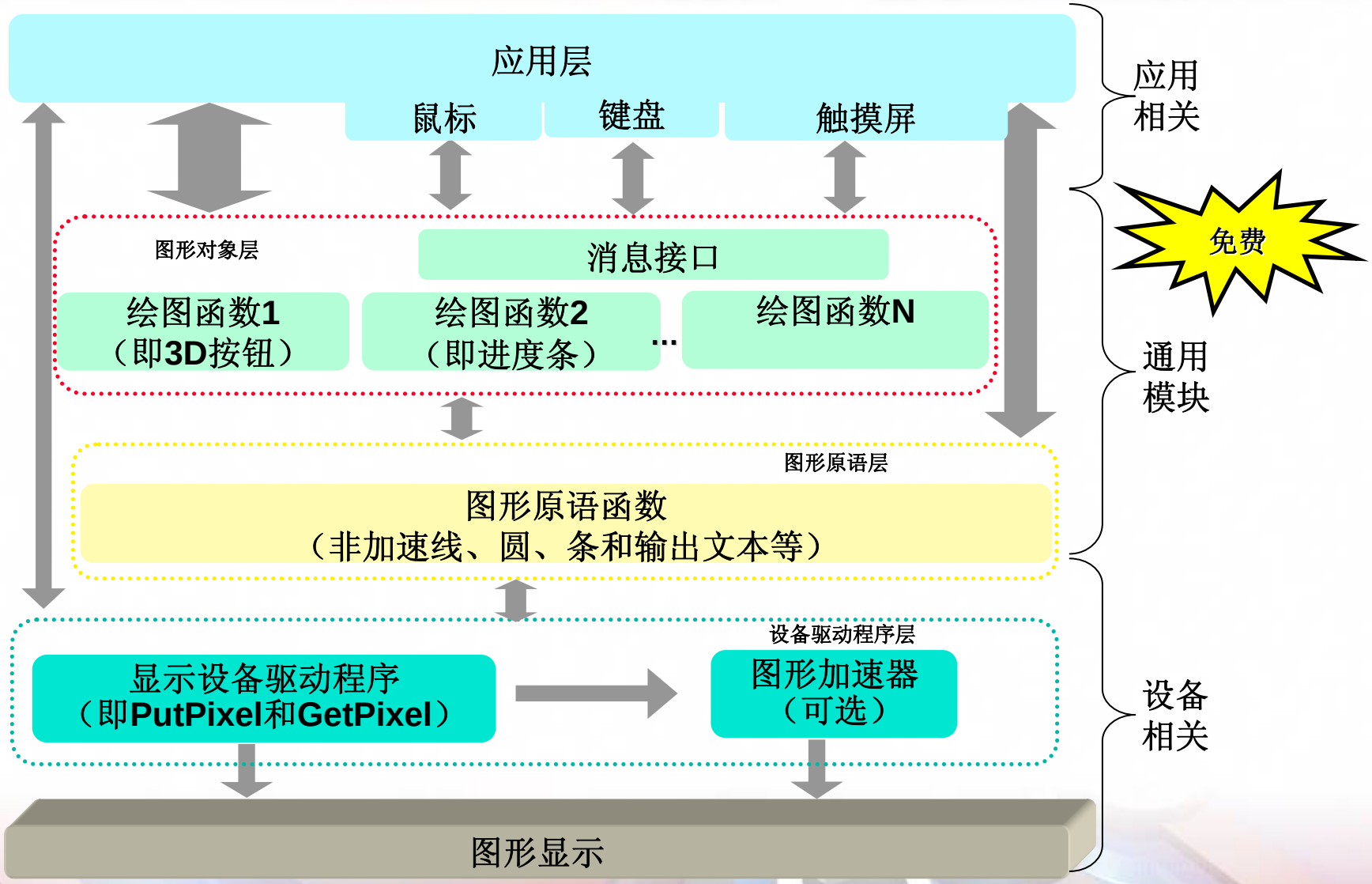
http://www.microchip.com/graphics

图形设计中心

- 图形库
  - 源代码、原理图、演示、文档和实用程序
- 应用笔记
  - AN1136——如何使用Microchip图形库中的控件
  - AN1182——Microchip图形库中的字体
- 网上研讨会
  - Microchip图形显示解决方案概述
  - 图形LCD工作方式介绍
  - 图形LCD系统与PIC24的接口
  - Microchip图形显示库架构
- Microchip RTC
  - 使用Microchip图形库设计（HIF 2131）

http://www.microchip.com/graphics

库概述



库配置

#defines位于GraphicsConfig.h中

- #include用于驱动程序头文件
- 选择输入设备
  - 触摸屏或键盘
  - *更多产品即将推出...*
- 选择要使用的控件
  - 按钮、复选框、滑动条等
- 允许/禁止焦点支持
- 图形模式选择
  - 单色
  - 纵向
  - Unicode支持（多字节字符）

库配置
#defines位于GraphicsConfig.h中

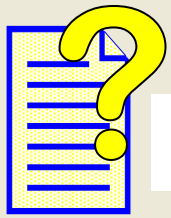
- 设置图像位置
  - 内部闪存、外部存储器或两者皆有
- 非阻断式配置
  - 基于状态的对象渲染
  - 有效使用处理器时间
  - `#define USE_NONBLOCKING_CONFIG`
- 阻断式配置
  - 简单的线性流程
  - 直接控制刷新或重新绘制
  - `// #define USE_NONBLOCKING_CONFIG`

库帮助

- Introduction
- Software License Agreement
- Release Notes
- Getting Started
- Library Configuration
  - Configuration Setting
  - Input Device Selection
  - Focus Support Selection
  - Graphics Object Selection
  - Font Source Selection
  - Bitmap Source Selection
  - Graphics Mode
  - Hardware Profiles
  - Library Structure
- Graphics Object Layer
- Graphics Primitive Layer
- Device Driver Layer
- Miscellaneous Topics
- Files

帮助文件作为**MPLAB**图形库安装的一部分，位于以下目录：

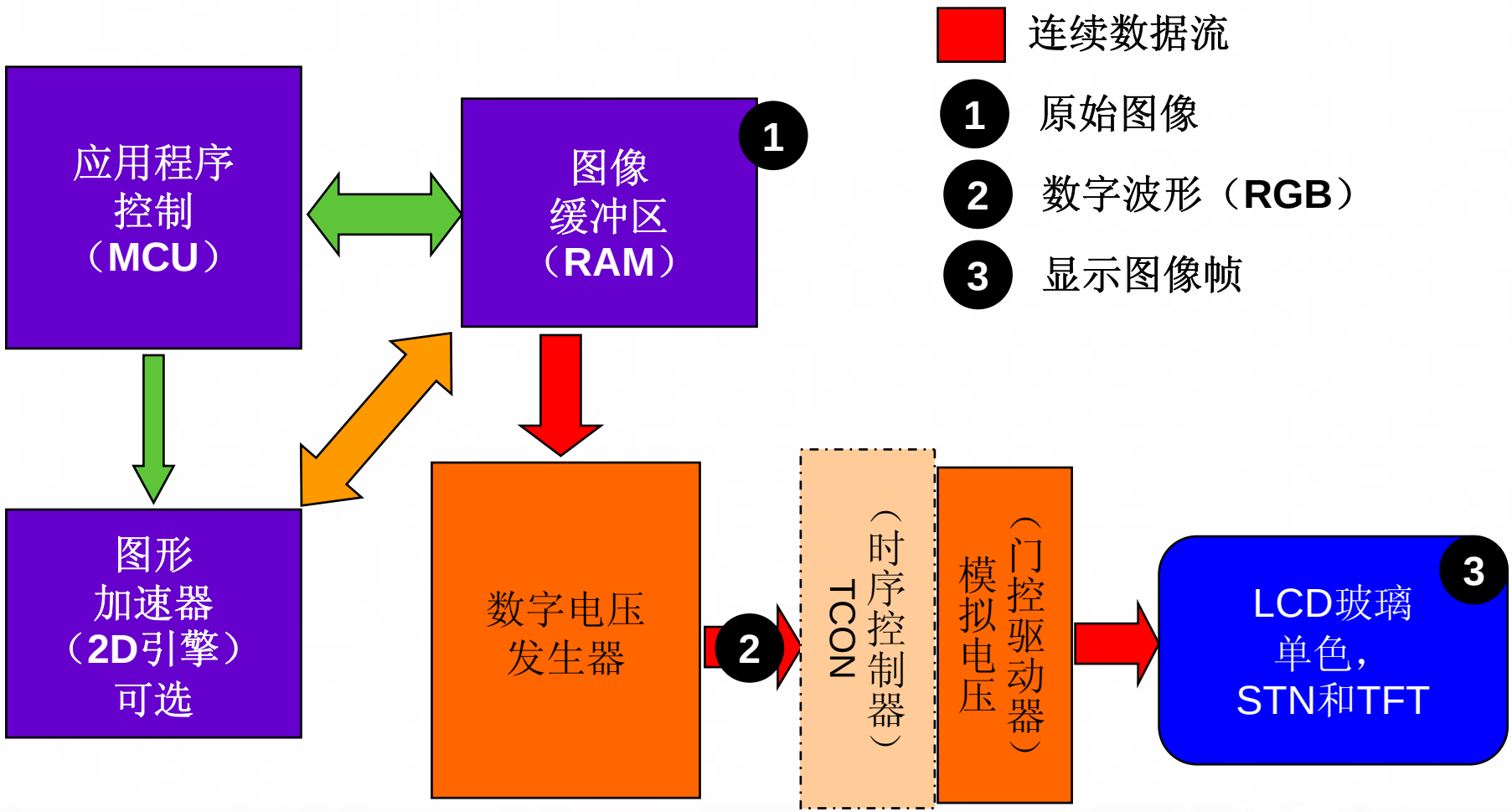
C:\Microchip Solutions\Microchip\Help



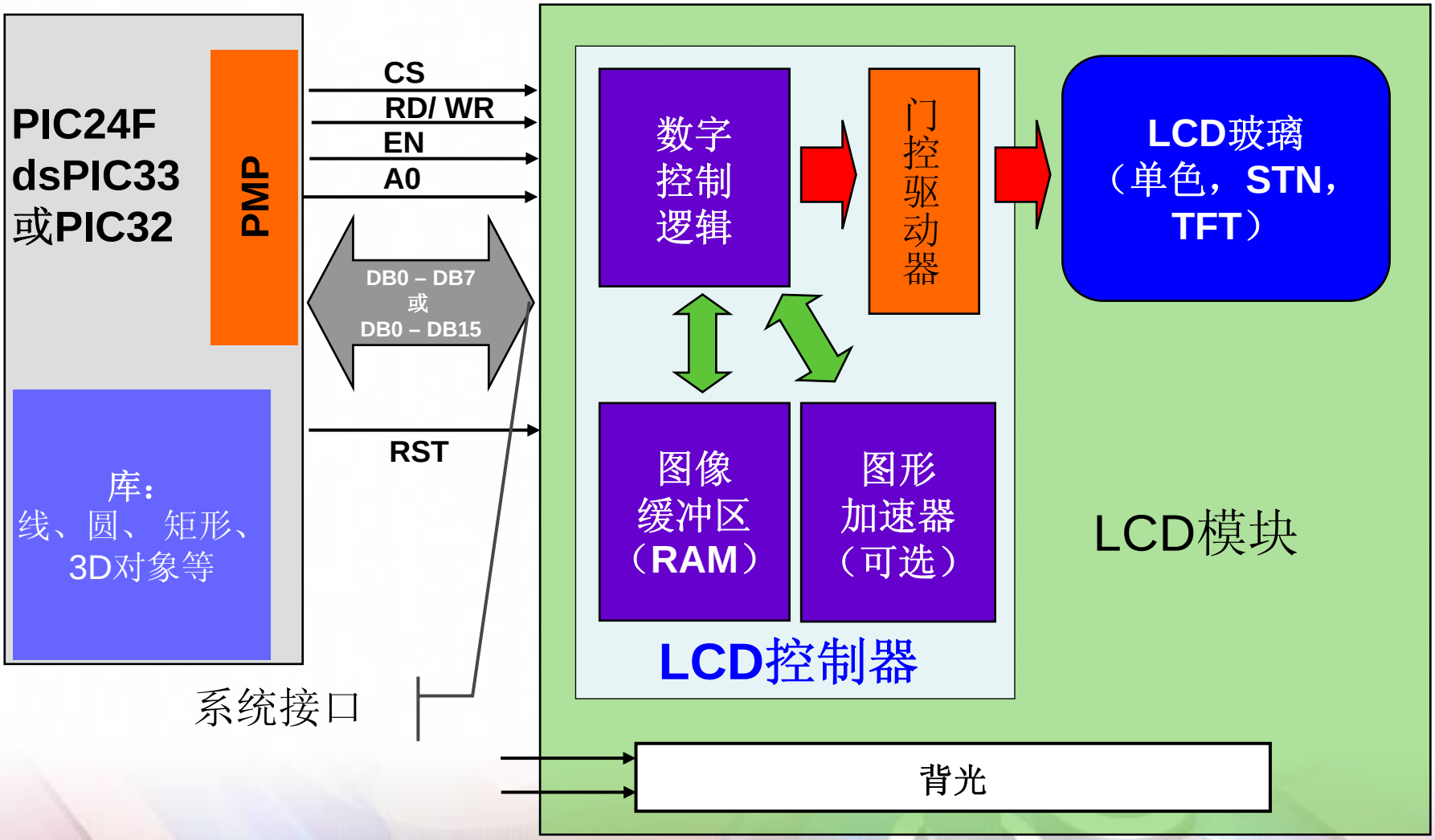
Graphics Library Help.chm

开始了解 LCD系统硬件

图形LCD系统

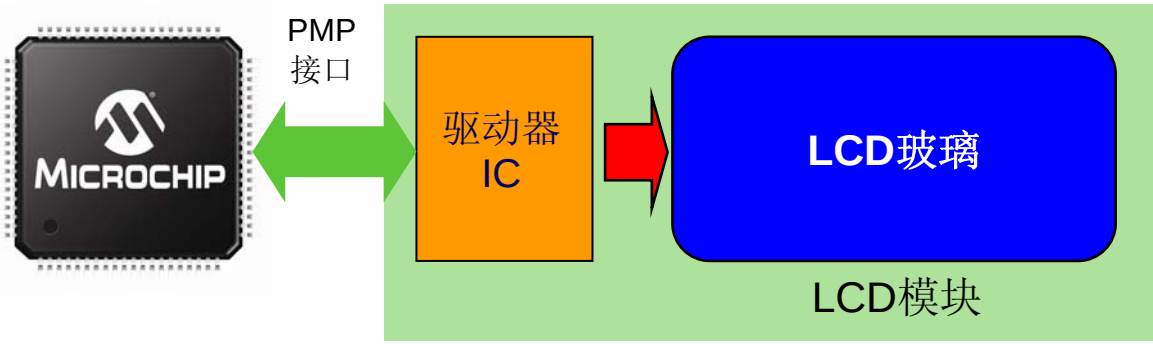


LCD模块
并行系统接口



图形解决方案

基于PMP的解决方案1

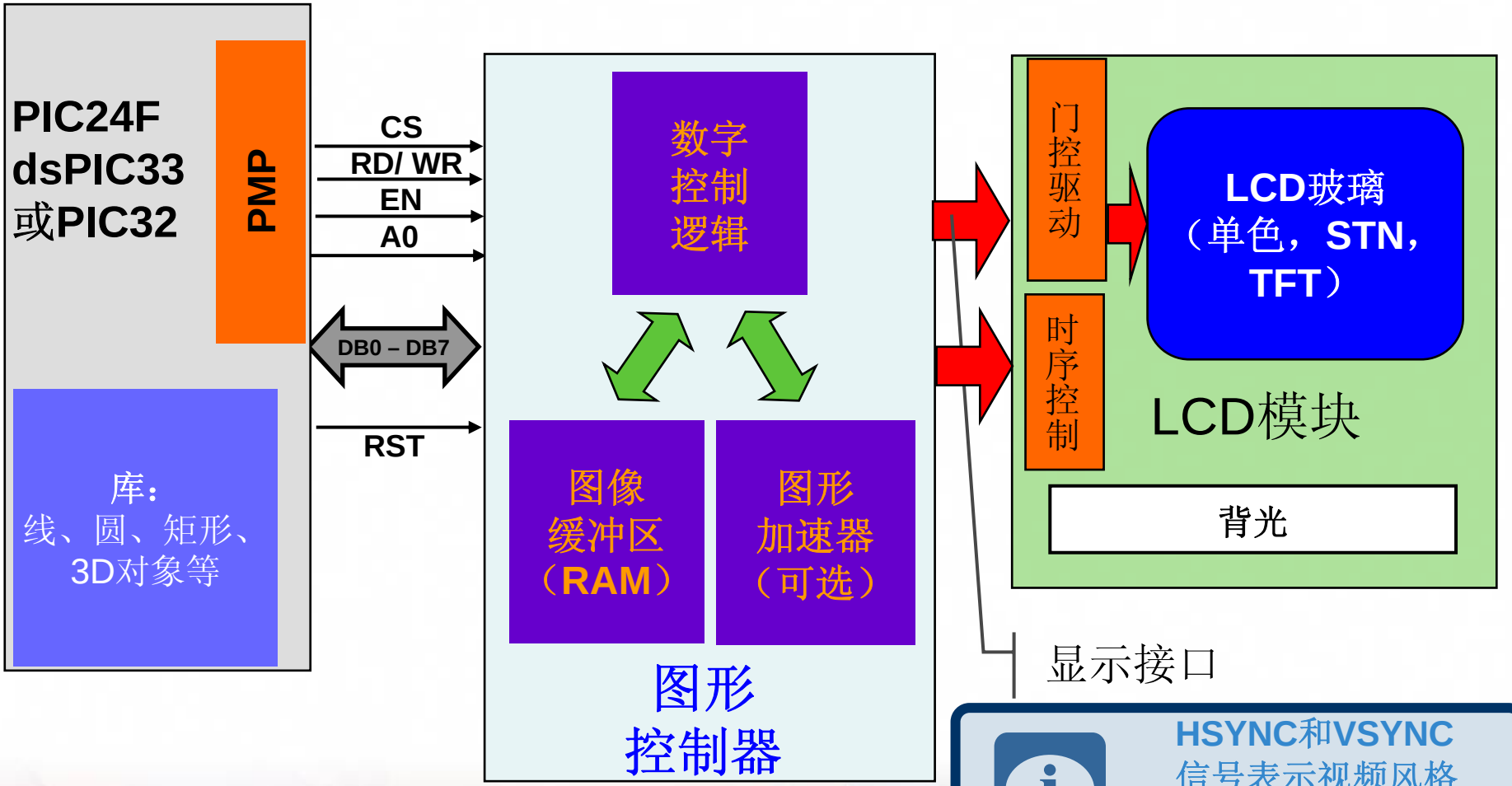


注：LCD模块由LCD供应商单独销售

- 用PMP作为与LCD模块的接口
- 很适合<2.8”的显示器
- Microchip库适用于任何LCD玻璃和驱动器IC
- 对某些供应商提供驱动程序固件

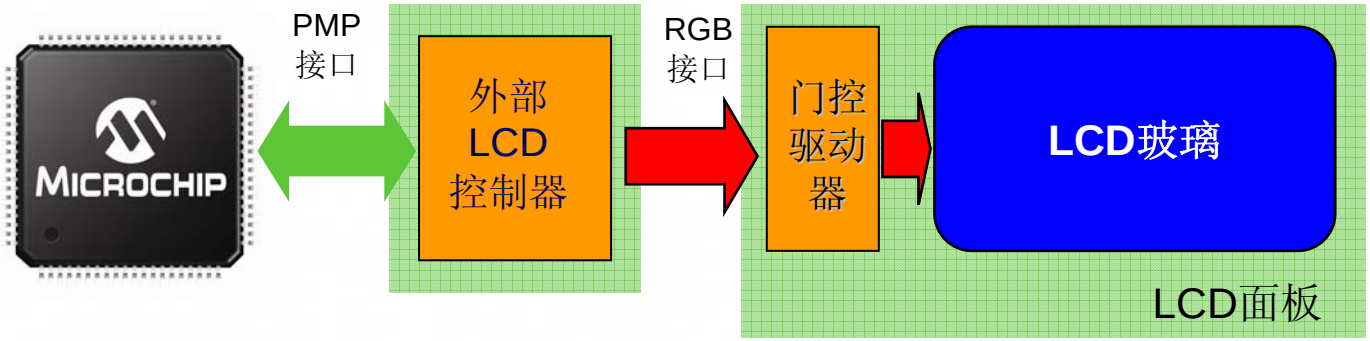
当前支持的驱动器IC	
供应商	部件编号
HiTech	HIT1270
LG	LGDP4531
Renesas	R61505U
Orise Tech	SPFD5408A
Samsung	S6D0129/0139
Solomon Systech	SSD1339

LCD模块
视频（如RGB）类型接口



图形解决方案

基于PMP的解决方案2



- 很适合>2.8”的显示器
- LCD控制器还包括缓冲区RAM和图形加速
- Microchip库适用于具有RGB接口的LCD玻璃

支持的LCD控制器	
供应商	部件编号
Solomon Systech	SSD1906/1905

注:
- 外部LCD控制器和LCD面板由其他供应商单独销售
- SSD1906可从Microchipdirect.com获得

开始了解 LCD系统硬件

编写底层驱动程序的技巧和诀窍

底层驱动程序

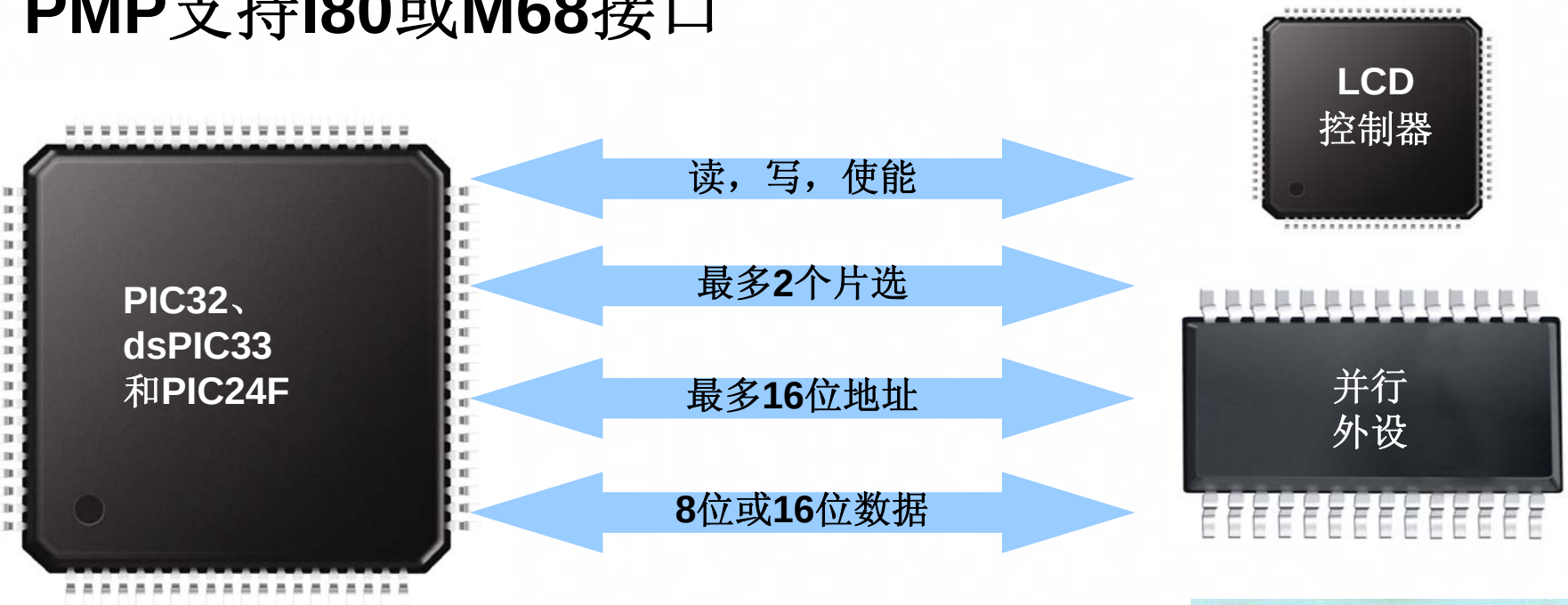
- **LCD模块控制器IC：**
  - HiTech：HIT1270
  - LG：LGDP4531
  - Renesas：R61505U
  - Orise Tech：SPFD5408A *（新）*
  - Samsung：S6D0129、S6D0139
  - Solomon Systech：SSD1339
- 外部**LCD**控制器：
  - Solomon Systech：SSD1905、SSD1906 *（新）*
- *更多产品即将推出…*

底层驱动程序技巧和诀窍

- 验证LCD至CPU的接口与I80或M68兼容
  - 如果是RGB类型的总线，请使用图形控制器IC
- 复制严格匹配您的控制器的`[driver].c`和`[driver].h`文件
  - 显示类型
  - 色深
  - 分辨率
- 检查控制寄存器的访问和地址
  - 变址寻址寄存器（像LGDP4531）
    - 不使用**PMP**地址线
  - 映射的存储器（像SSD1906）
    - 使用**PMP**地址线
- 如果**LCD/**控制器对原语函数进行了加速，请将函数置于`[driver].c`中

编写驱动程序
并行主端口——PMP

- `ResetDevice()`初始化PMP
- PMP支持I80或M68接口



在PIC32和dsPIC上使用DMA模块，以提高性能

底层驱动程序技巧和诀窍

- 编辑[Driver].h文件
  - 设置屏幕分辨率设置 ...

```
…
// 定义水平屏幕尺寸。
#define SCREEN_HOR_SIZE          240
// 定义垂直屏幕尺寸。
#define SCREEN_VER_SIZE          320
```

  - 如果需要，编辑颜色定义...

```
…
#define BLACK                RGB565CONVERT(0,0,0)
#define BRIGHTBLUE           RGB565CONVERT(0,0,255)
#define BRIGHTGREEN          RGB565CONVERT(0,255,0)
#define BRIGHTCYAN           RGB565CONVERT(0,255,255)
…
```

底层驱动程序技巧和诀窍

- 定义在驱动程序中实现的原语函数...
  - 原语层中实现的未定义函数

```
//#define USE_DRV_FONT
//#define USE_DRV_LINE
//#define USE_DRV_CIRCLE
//#define USE_DRV_FILLCIRCLE
#define USE_DRV_BAR
#define USE_DRV_CLEARDEVICE
#define USE_DRV_PUTIMAGE
```

- 设置行存储间距（图像缓冲区）...
  - 将x,y坐标转换为像素（图像缓冲区）地址

```
…
// 行存储间距
#define LINE_MEM_PITCH                256
…
```

240(RGB) x 320 LINE_MEM_PITCH

- 以下信息来自驱动器IC数据手册中的RAM地址设置控制寄存器

AD[16:0]	GRAM数据映射
17’h00000 … 17’h000EF	第1行的GRAM数据
17’h00100 … 17’h001EF	第2行的GRAM数据
……	…..
17’h13D00 … 17’h13DEF	第318行的GRAM数据
17’h13E00 … 17’h13EEF	第319行的GRAM数据
17’h13F00 … 17’h13FEF	第320行的GRAM数据

来源：LGDP4531数据手册

- 每行256字节

底层驱动程序技巧和诀窍

- 从制造商处获取初始化代码
- 修改复制文件中的**API**：
  - **ResetDevice()**——此处是初始化代码
  - **SetIndex()**——用于变址寻址寄存器访问
- 可能无需更改的其他**API**：
  - **PutPixel()**
  - **GetPixel()**
  - **WriteData()**
  - **SetAddress()**
- 桩（**Stub**）**API**——如果需要编写函数：
  - **SetActivePage()**
  - **SetVisualPage()**
  - **SetPalette()**

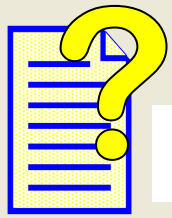
底层驱动程序总结

- 需要用I80或M68总线直接连接到PIC
- 如果是RGB类型的总线，请使用图形控制器芯片（例如SSD1906）
- 复制和控制器最匹配的驱动程序文件
- 从制造商处获取初始化代码
- 编辑所需的API
  - 至少`ResetDevice()`
- 编辑driver.h文件以反映
  - 驱动器IC图像缓冲区
  - 屏幕分辨率
  - 驱动程序中的原语函数
- 初始化PMP以适应系统接口

驱动程序帮助

- Device Driver Layer
  - Device Level Primitives
    - GetPixel Function
    - PutPixel Function
    - GetColor Macro
    - SetColor Macro
    - GetMaxX Macro
    - GetMaxY Macro
    - GetClipBottom Macro
    - GetClipLeft Macro
    - GetClipRight Macro
    - GetClipTop Macro
    - SetClip Macro
    - SetClipRgn Macro
    - CLIP_DISABLE Macro
    - CLIP_ENABLE Macro
    - Color Definition
  - Device Control
    - IsDeviceBusy Macro
    - DelayMs Function
    - SetActivePage Macro
    - SetPalette Macro
    - SetVisualPage Macro
    - ResetDevice Function
  - Adding New Device Driver
    - USE_DRV_BAR Macro
    - USE_DRV_CLEARDEVICE Macro
    - USE_DRV_PUTIMAGE Macro
    - USE_DRV_CIRCLE Macro
    - USE_DRV_FILLCIRCLE Macro
    - USE_DRV_FONT Macro
    - USE_DRV_LINE Macro

帮助文件中的“设备驱动程序”部分提供了驱动程序文件中所用的所有函数的说明。



图形库帮助

Microchip图形原语层

原语层

- 直接与设备驱动程序对话
- 编译时间选项：**GraphicsConfig.h**
  - 选择设备驱动程序
  - 字体来源 （内部或外部）
  - Unicode支持（AN1182）
  - 位图来源（内部或外部）

关键原语API

- 设置函数
  - **InitGraph()**——初始化显示
    - **Primitive.c**中的默认设置
  - **ClearDevice()**——
    - 用当前颜色清屏
    - 将光标置于**(0,0)**处

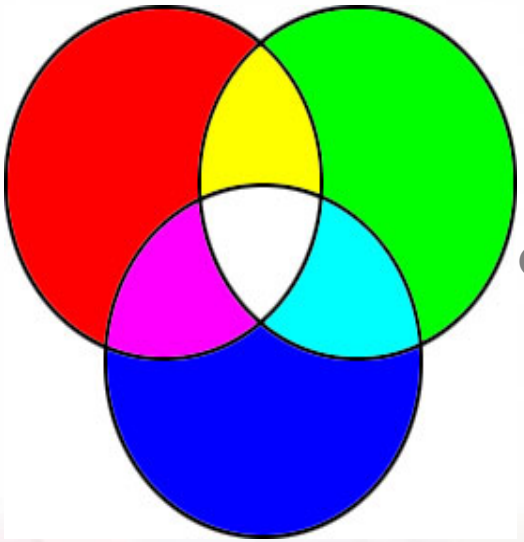
原语API

- 通用函数
  - 影响下个*Set*之前的函数
    - **SetColor(*COLOR*)**——设置绘图颜色
      - **driver.h**文件中的***COLOR***值
    - **SetFont(*&fontimage*)**——设置字体映像
    - **SetLineType(*key*)**
      - ***SOLID_LINE***
      - ***DOTTED_LINE***
      - ***DASHED_LINE***
    - **SetLineThickness(*key*)**
      - ***NORMAL_LINE***
      - ***THICK_LINE***

RGB颜色模型

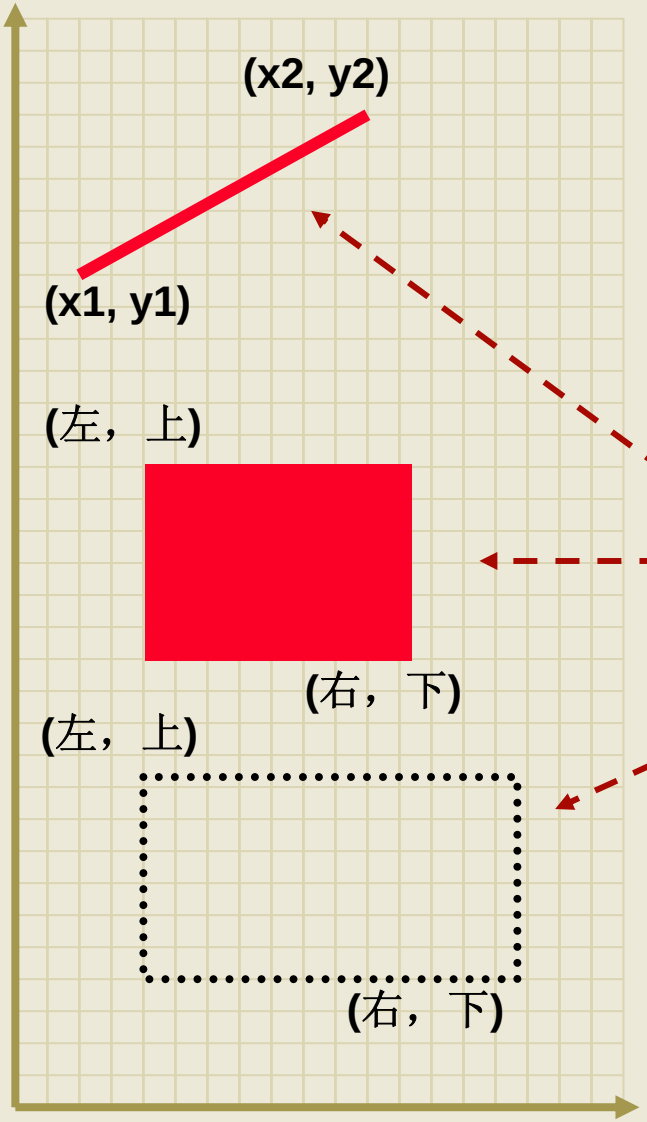
定义：

RGB颜色模型是一种叠加式颜色模型，将红、绿和蓝色光以不同方式叠加来产生众多颜色。颜色以三原色（RGB）形式表现。 来源：维基百科



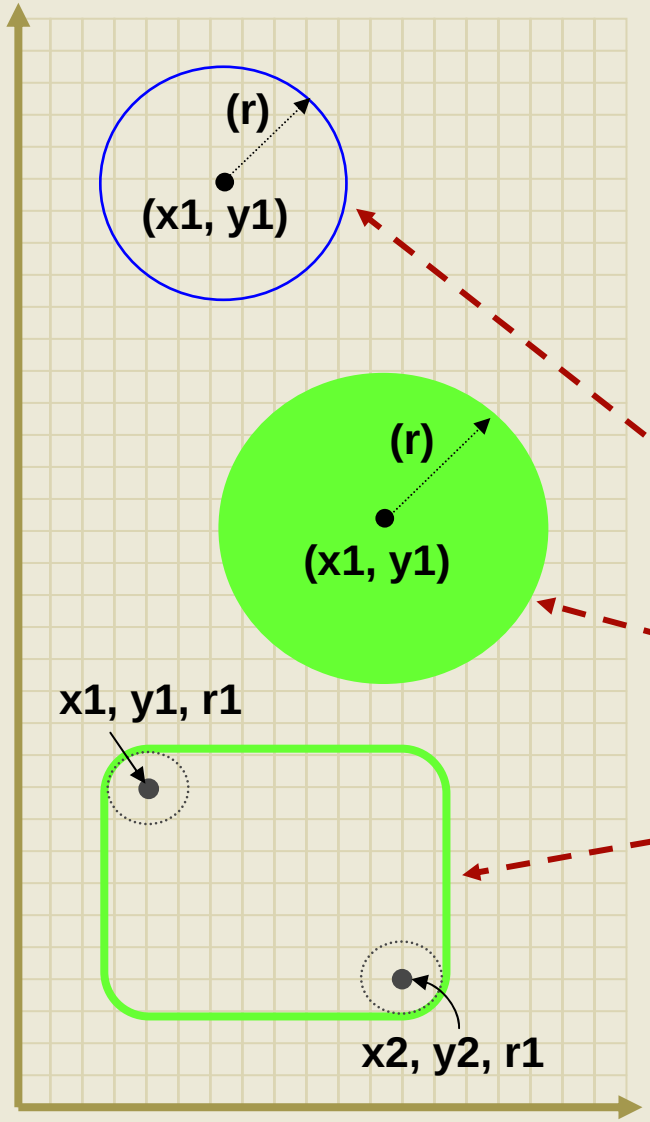
- 16位（65,536色）
  - 红色：5位
  - 绿色：6位
  - 蓝色：5位
- 18位（262,144色）
  - 红色：6位
  - 绿色：6位
  - 蓝色：6位
- 每种成分的值代表相应颜色的深浅

原语绘图函数



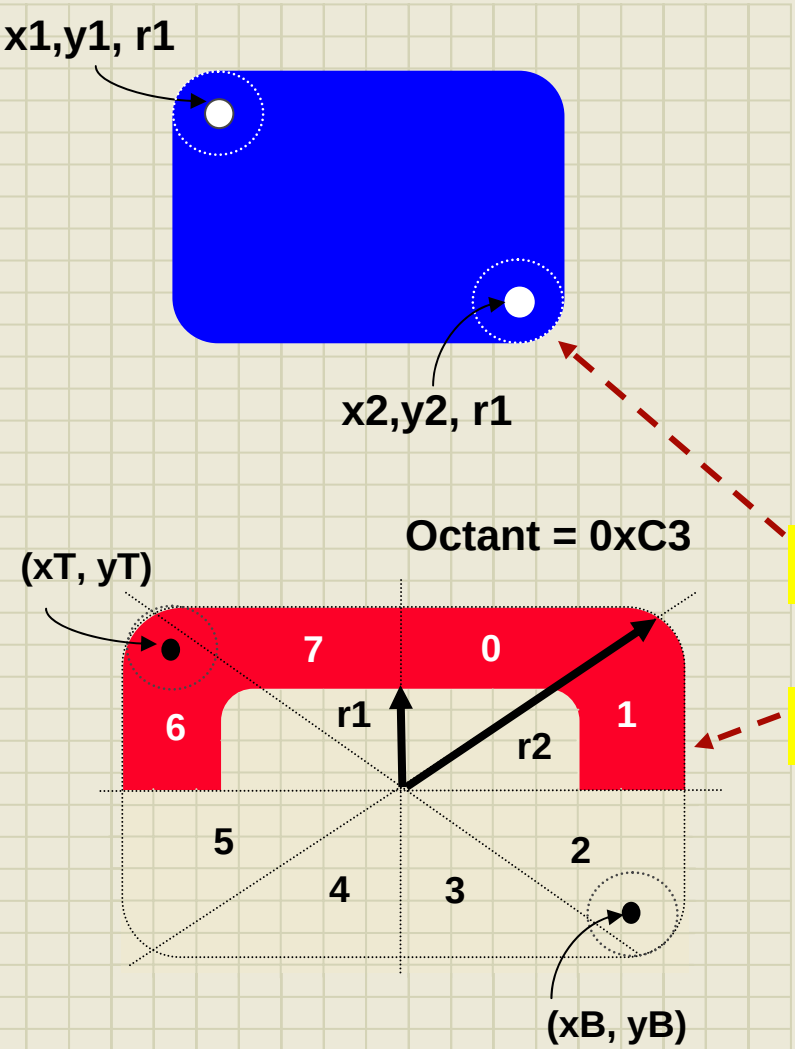

```
int main(void)
{
….
SetColor(BRIGHTRED);
SetLineType(SOLID_LINE);
SetLineThickness(THICK_LINE);
Line(x1, y1, x2, y2);
Bar(left, top, right, bottom);
SetColor(BLACK);
SetLineType(DOTTED_LINE);
Rectangle(left, top, right, bottom);
…
}
```

原语绘图函数




```
int main(void)
{
….
SetColor(BRIGHTBLUE);
SetLineType(SOLID_LINE);
SetLineThickness(NORMAL_LINE);
Circle(x1, y1, r);
SetColor(BRIGHTGREEN);
FillCircle(x1, y1, r);
SetLineThickness(THICK_LINE);
Bevel(x1, y1, x2, y2, r1);
….
}
```

原语绘图函数




```
int main(void)
{
….
SetColor(BRIGHTBLUE);
SetLineType(SOLID_LINE);
SetLineThickness(NORMAL_LINE);
FillBevel(x1,y1,x2,y2,r1);
SetColor(BRIGHTRED);
Arc(xT,yT,xB,yB,r1,r2,Octant);
….
}
```

Microchip图形原语层（字体和图像）

字体

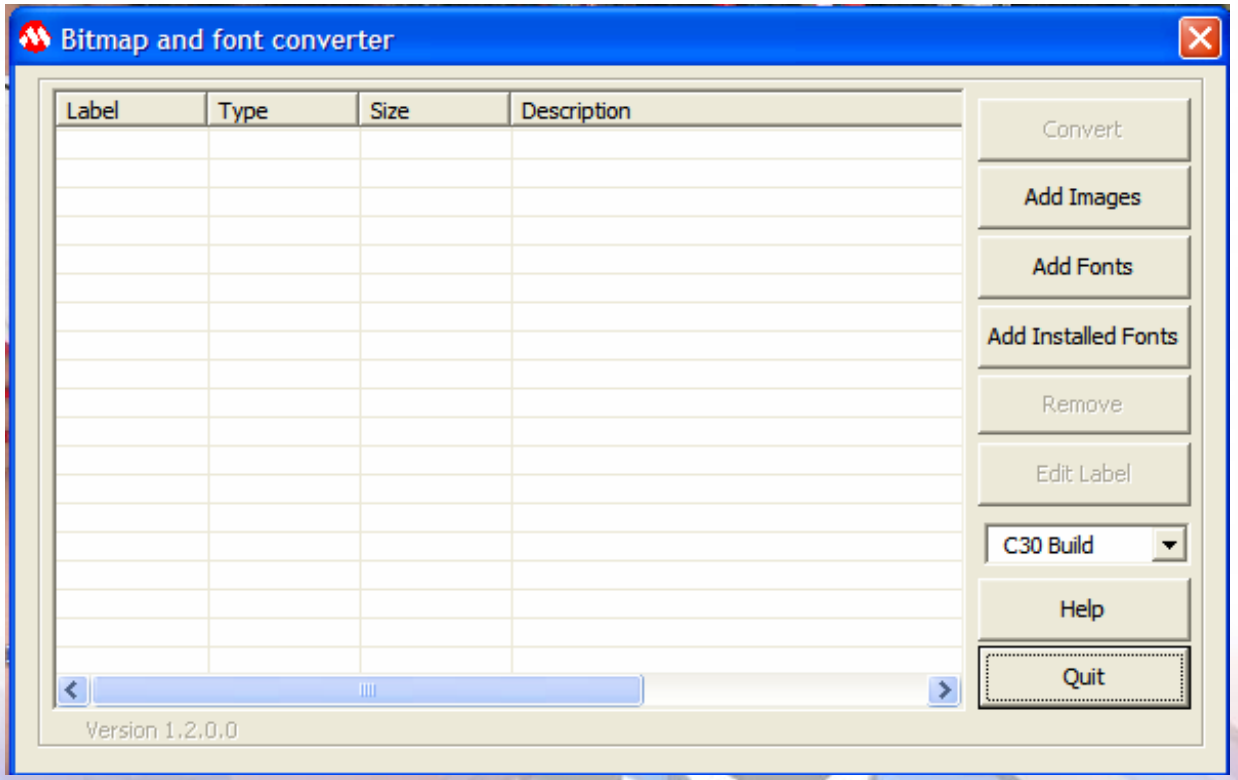
定义：

字体是一种电子数据文件，包含一组字形、字符和符号。字体是用字体编辑器创建的，常被视为艺术作品。预创建的字体可从许多来源获得，但可能需要许可证。它们经常是有版权的。

- 库支持已转换
  - True Type和Open Type字体
  - 光栅（位图）字体
- 字体映像
  - 可存储在内部或外部闪存中
  - 有经过过滤的字体映像可用
- 通过多字节字符支持**Unicode**
  - AN1182——Microchip图形库中的字体

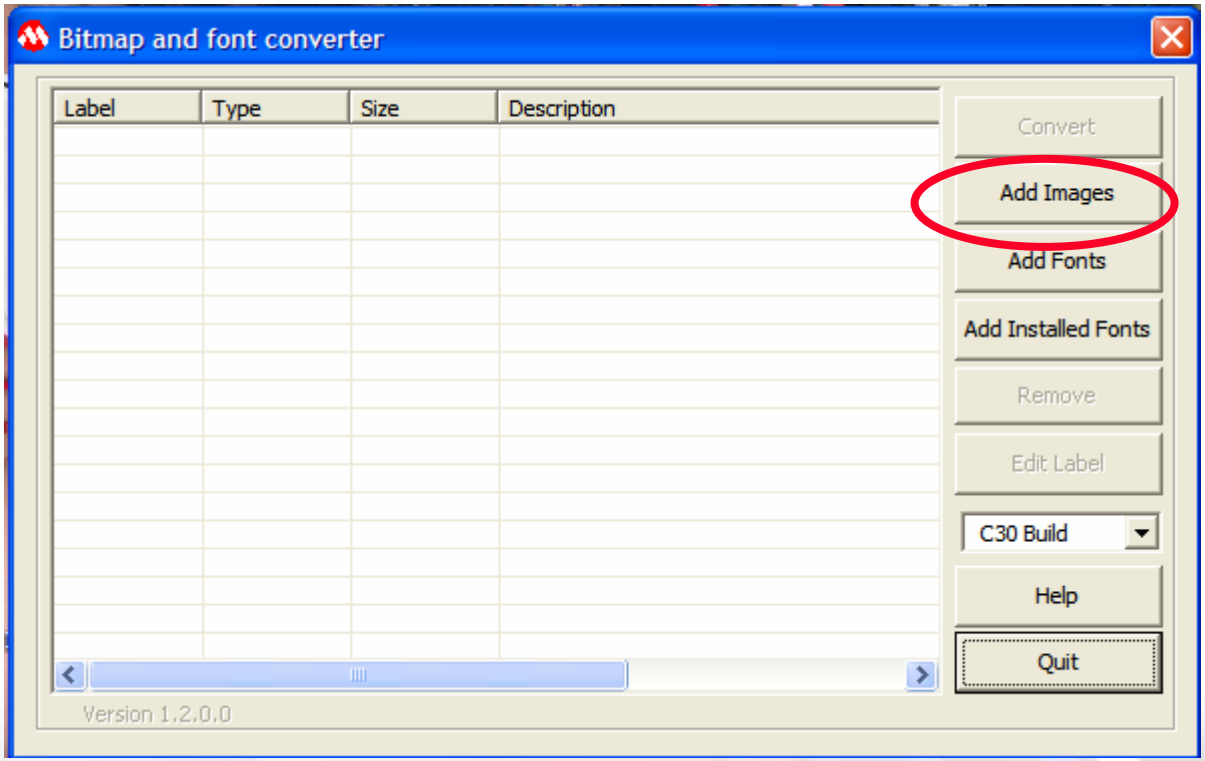
使用字体

- 位图和字体转换器
  - 随库提供免费的实用程序
  - 转换为库能理解的格式



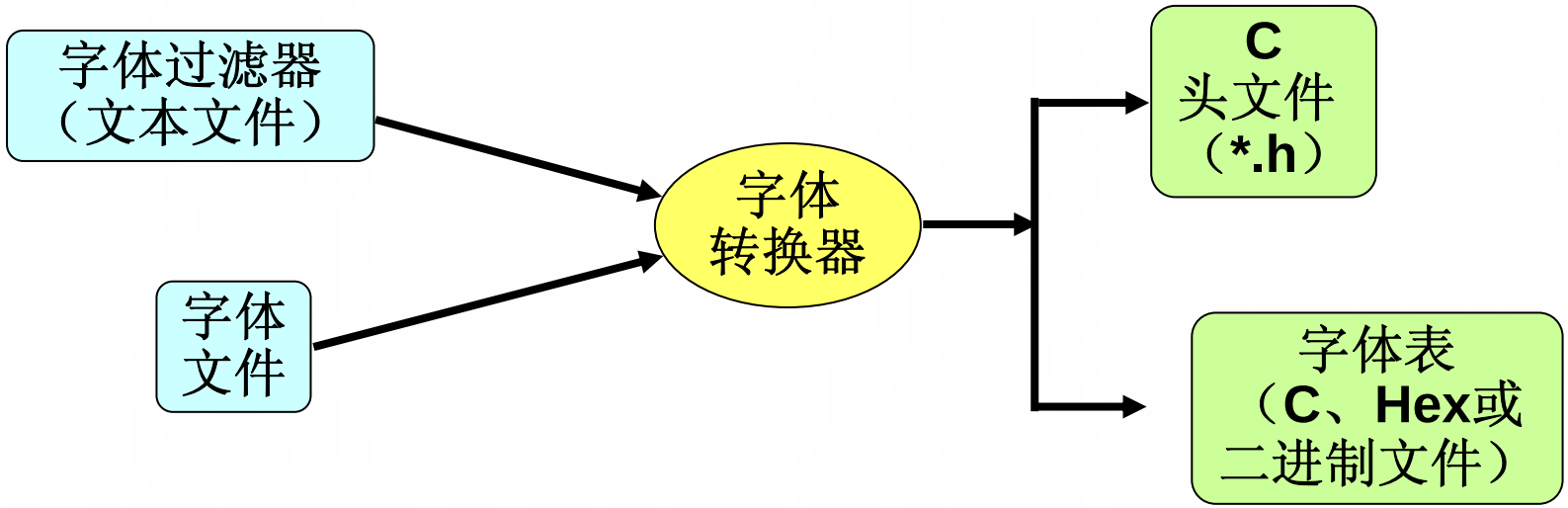
转换字体

- 转换字体
  - 存储在内部存储器中——生成c文件
  - 存储在外部存储器中——生成hex和c文件



字体过滤器

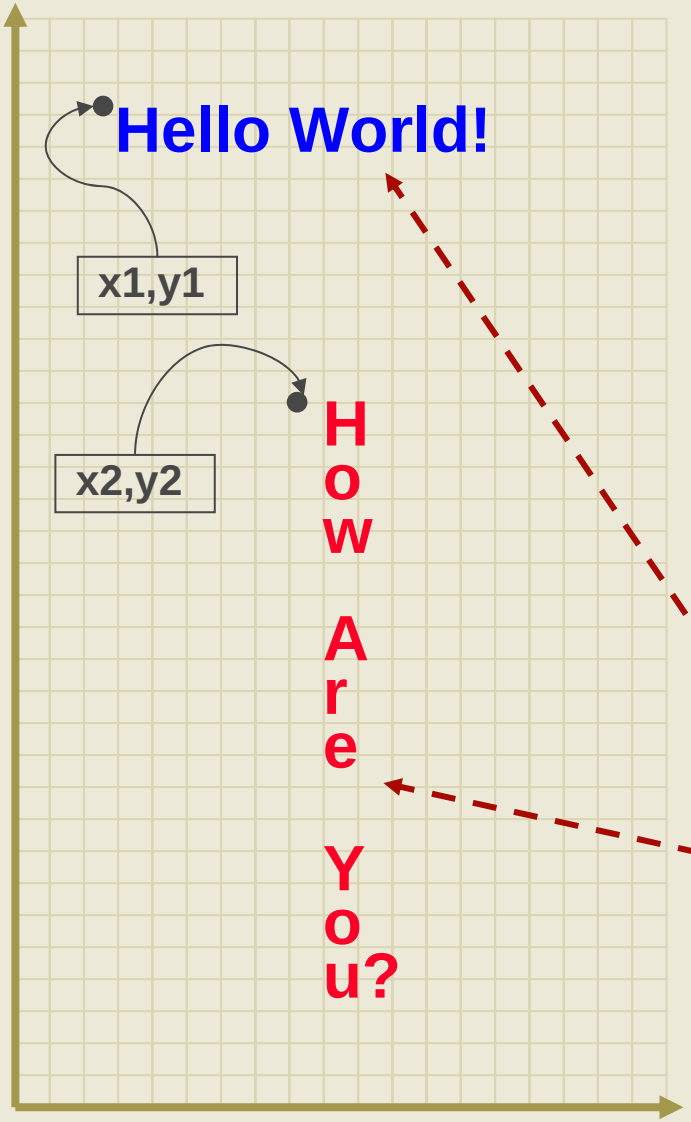
- 用于节约字体映像的存储空间
  - 字形被重映射到新的字符ID
  - 需要过滤器文本文件（字体过滤器文件）
  - 必须在项目中包含已生成的引用文件



字体过滤器文本文件

- 创建于**unicode**编辑器中（例如**Wordpad**）
- 必须保存为**16**位**unicode**格式
- 每行都必须有**3**段：
  - **<字符串标签>:<字符串>//<注释>**
    - 需要“*//*”指示
    - 注释是可选的
- 必须重新创建字体映射才能编辑字符串
- 请参见应用笔记**1182**（**Microchip**图形库中的字体）

使用字体




```
int main(void)
{
extern const FONT_FLASH myFont;
XCHAR myString1[]="Hello World!";
XCHAR myString2[]="How are you?";
….
SetFont((void*)&myFont);
SetColor(BLUE);
MoveTo(x1,y1); // 移动光标
OutText(myString1);
SetColor(BRIGHTRED);
SetFontOrientation(ORIENT_VER);
OutTextXY(x2, y2, myString2);
….
}
```

图像

定义：

位图就是存储器内图像的逐点描述。每个点的值按照色深或每像素位数（**bpp**）存储在一个或多个数据位中。

1 bpp

4 bpp

8 bpp

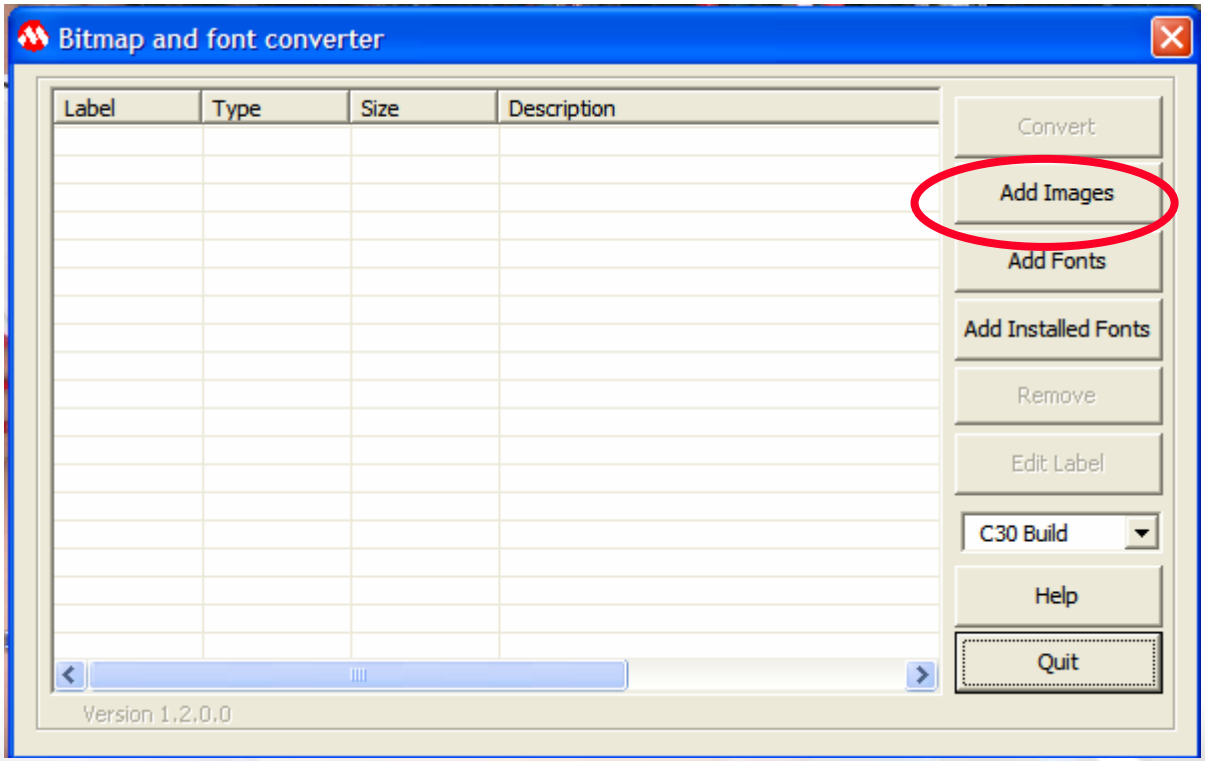
16 bpp

分辨率和图像缓冲区大小

	X分辨率	Y分辨率	像素	色深（bpp）	颜色	所需字节
单色						
1/16 VGA	160	120	19,200	1	2	2,400
1/8 VGA	240	160	38,400	1	2	4,800
QVGA	320	240	76,800	1	2	9,600
彩色STN						
1/16 VGA	160	120	19,200	8	256	19,200
1/8 VGA	240	160	38,400	8	256	38,400
QVGA	320	240	76,800	8	256	76,800
彩色TFT						
1/16 VGA	160	120	19,200	16	65,536	38,400
1/8 VGA	240	160	38,400	16	65,536	76,800
QVGA	320	240	76,800	16	65,536	153,600
1/16 VGA	160	120	19,200	18	262,144	43,200
1/8 VGA	240	160	38,400	18	262,144	86,400
QVGA	320	240	76,800	18	262,144	172,800

转换图像

- 转换位图图像
  - 存储在内部存储器中——生成c文件
  - 存储在外部存储器中——生成hex文件



使用图像

- **Main.c——声明图像**

```
//////////////////// 字体和位图 ////////////////////////
// 该字体位于内部闪存中
extern const BITMAP_FLASH internal_bitmap;

// 该字体必须存储在外部闪存中，后者安装在
// 图形PICTail Plus电路板上
extern BITMAP_EXTERNAL  external_bitmap;
```

- **internalbitmap.c——由转换工具生成**

```
extern const char L11298[] __attribute__((aligned(2)));
// 名称可在此处更改。
const struct{short mem; const char* ptr;} internal_bitmap =
{0,L11298};
const char L11298[] __attribute__((aligned(2)))
={0x00,0x00,0x20,0x00,0x7F,0x00,0x00,0x23,0x00,0x06,0x88,0x01,0x0
0,0x08,0xAB,0x01,0x00,0x0C,0xCE,0x01,0x00,0x0E,0x14,0x02,0x00,…
```

使用图像

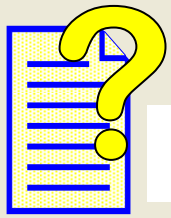

```
int main(void)
{
extern const BITMAP_FLASH image1;
BYTE stretch = IMAGE_NORMAL;
….
X = GetMaxX()-GetImageWidth((void*)&image1);
Y = GetMaxY()-GetImageHeight((void*)&image1);

// 将位图置于中心位置
PutImage((X >>1),(Y >> 1), &image1,stretch);
….
}
```

*拉伸*选项有：
IMAGE_NORMAL
IMAGE_X2

原语层帮助

- Graphics Primitive Layer
  - Set Up Functions
    - ClearDevice Function
    - InitGraph Function
  - Text Functions
    - SetFont Function
    - OutChar Function
    - OutText Function
    - OutTextXY Function
    - GetTextHeight Function
    - GetTextWidth Function
    - XCHAR Macro
    - FONT_HEADER Structure
    - FONT_FLASH Structure
    - FONT_EXTERNAL Macro
  - Line Functions
  - Rectangle Functions
  - Circle Functions
  - Graphic Cursor
  - Bitmap Functions
    - GetImageHeight Function
    - GetImageWidth Function
    - PutImage Function
    - BITMAP_HEADER Structure
    - Bitmap Settings
    - Bitmap Source
  - External Memory

帮助文件的“原语层”部分描述了各种不同的**API**，它们可帮助您绘制原语、使用字体以及使用图像。

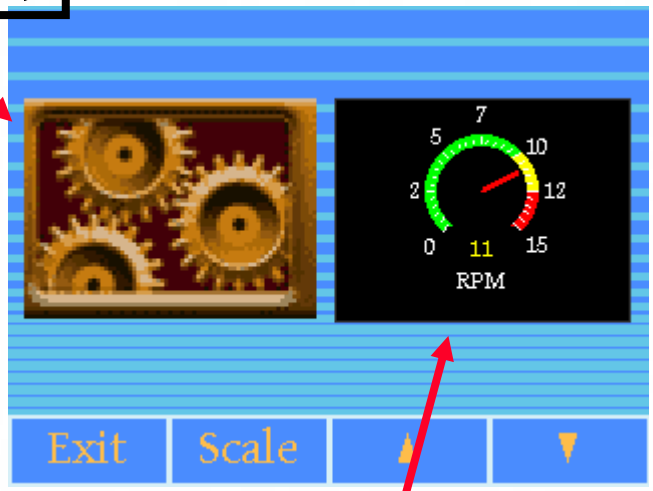


图形库帮助

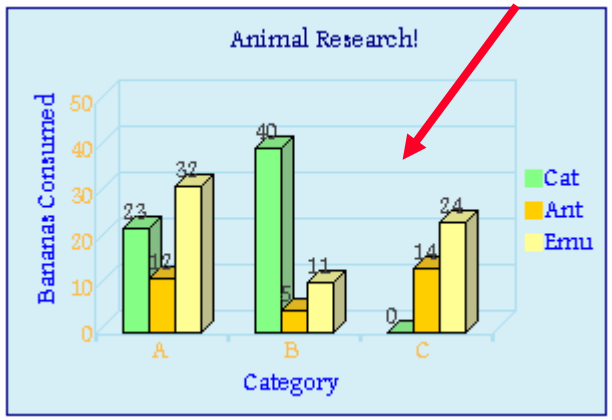
Microchip图形创建控件

库控件

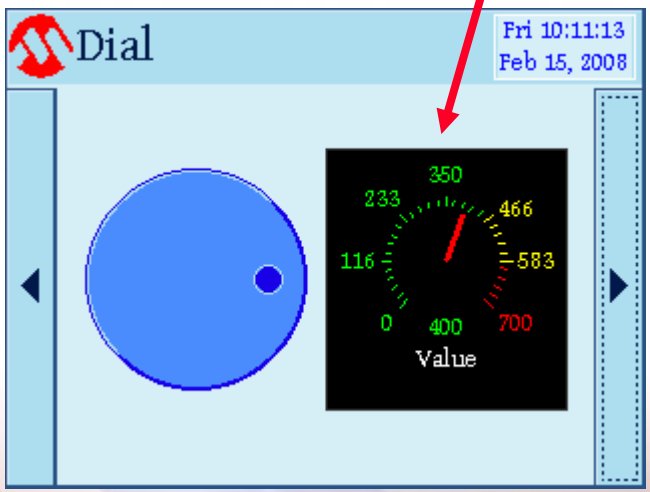
图片

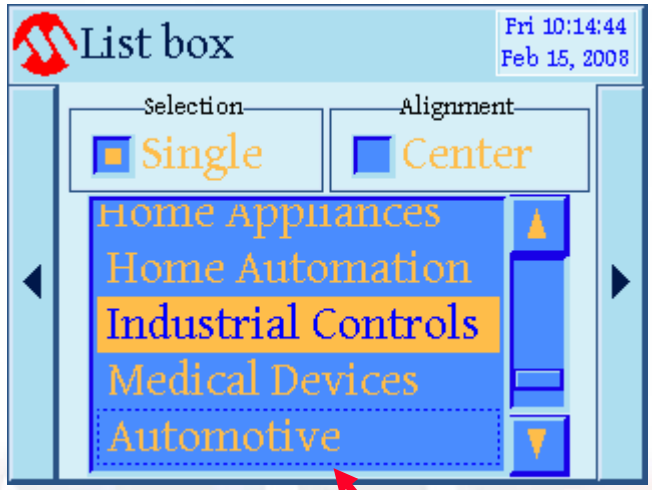


图表



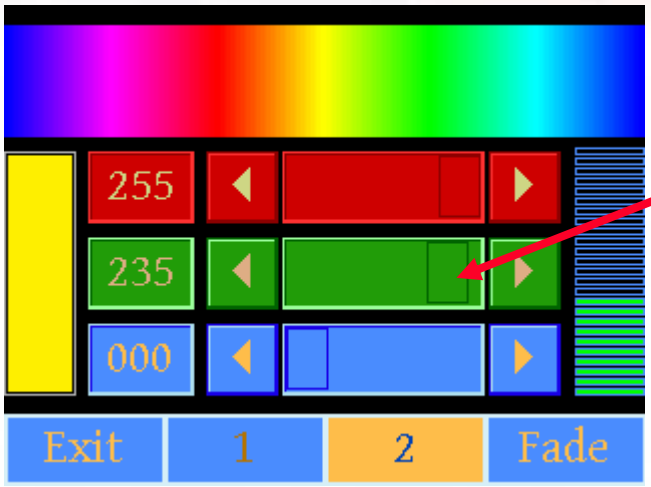
仪表





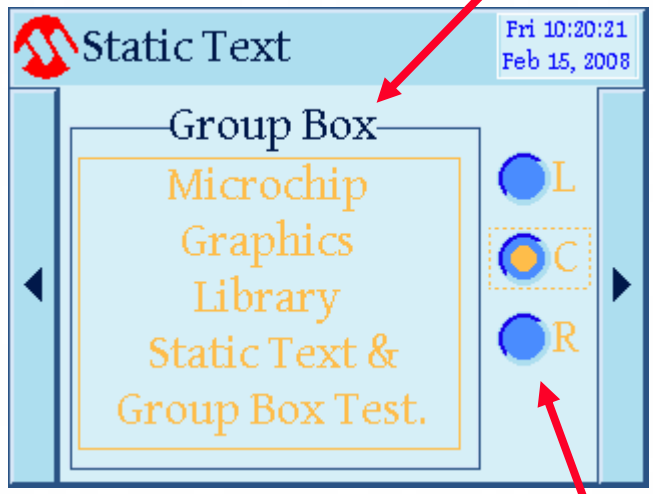
列表框

库控件

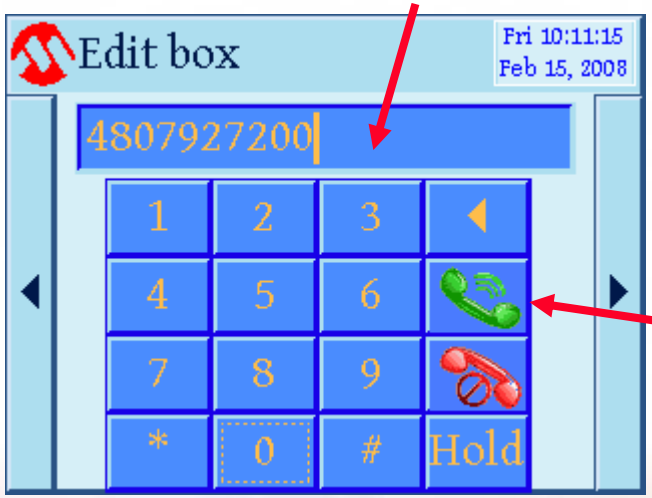


滑动条

组合框



编辑框



带图像的按钮

单选按钮

创建对象

定义：

ObjCreate(,,)是库用来以指定参数创建控件的函数。该函数自动填充控件的结构，将控件置于全局链表中，并返回指向创建控件的指针。

- **Obj** = 控件缩写
  - 在库帮助文件中定义
  - 示例：
    - **BtnCreate(,,,)**创建按钮
  - 示例：
    - **PbCreate(,,,)**创建进度条
- 可创建多个控件实例

创建控件
常见参数

ID

- 构成一个“句柄”的唯一标识符

Location

- 左、上、下、右
  - 定义位置和大小

State

- 用于控制控件的字

Style Scheme

- 定义控件外观

创建控件独特参数

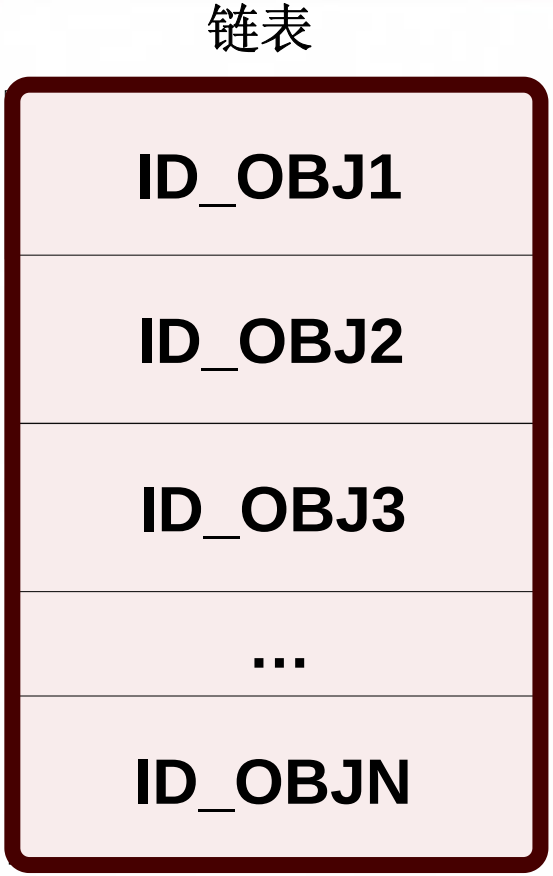
- **BtnCreate(,,)**——按钮
  - ***Radius***：圆边
  - ****pBitmap***：按钮上的图像（图标）
  - ****pText***：按钮上的文本
- **StCreate(,,)**——静态文本
  - ****pText***：框中的文本
- **PbCreate(,,)**——进度条
  - ***pos***：初始进度位置
  - ***range***：最大值100%

在应用程序代码中 …

- 示例：

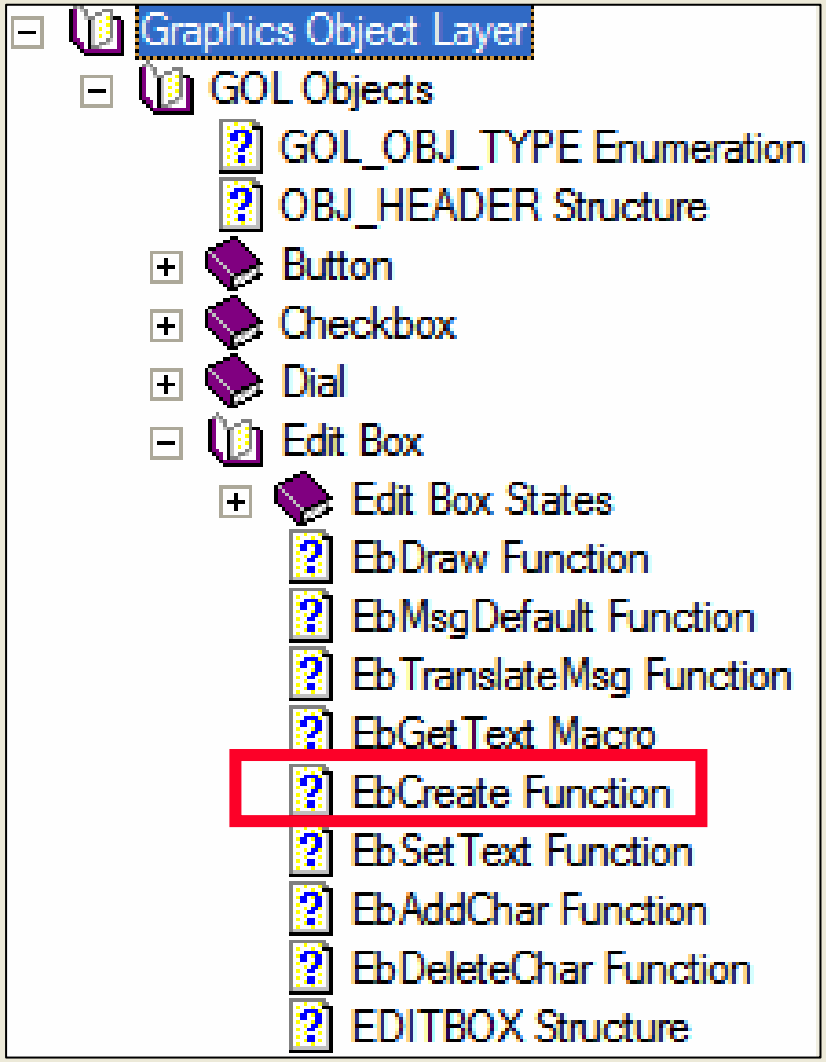
```
void CreateButtons(void)
{
…
BUTTON          *pBtn
#define          ID_BTN2       16
…
…
BtnCreate(     ID_BTN2,        // 第二个按钮ID
               x3, y3,         // 左，上
               x4, y4,         // 右，下
               Radius,         // 圆边
               BTN_DRAW,       // 显示按钮
               &arrow,         // 使用该位图
               NULL,           // 无文本
               altScheme);     // 样式方案
…
}
```

创建控件



- `ObjCreate(,,,)`
  - 填充控件的结构
  - 将控件添加到活动链表底部
  - 需要堆
  - 后面将说明更多关于使用链表的详细信息

控件帮助



要查找**ObjCreate API**，请展开所需的控件，并选择相应的创建函数。

图形库帮助

控件样式

定义：

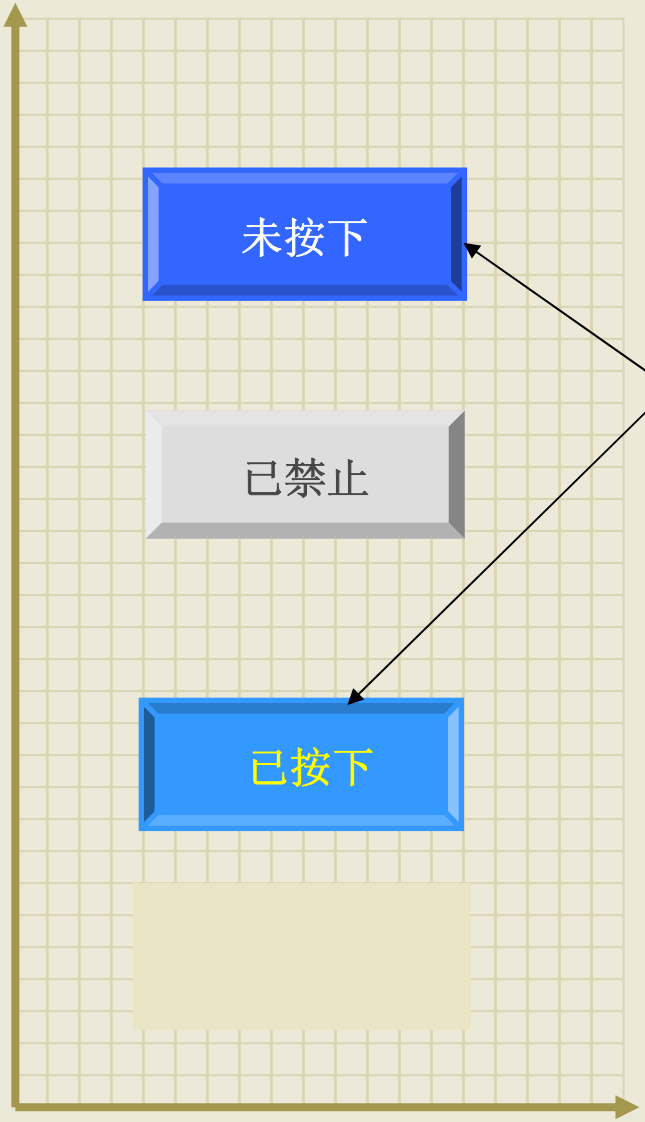
样式方案是库用来定义用于绘制控件的颜色和字体的结构。

- 可以定义多种方案
- 默认定义在**GOL.h**中
- **GOLCreateScheme()**
  - 创建方案结构
  - 返回指向结构的指针
- 结构中的**10**个成员
  - 样式方案以不同方式影响每个控件
  - 动态更改颜色会影响控件

样式方案结构

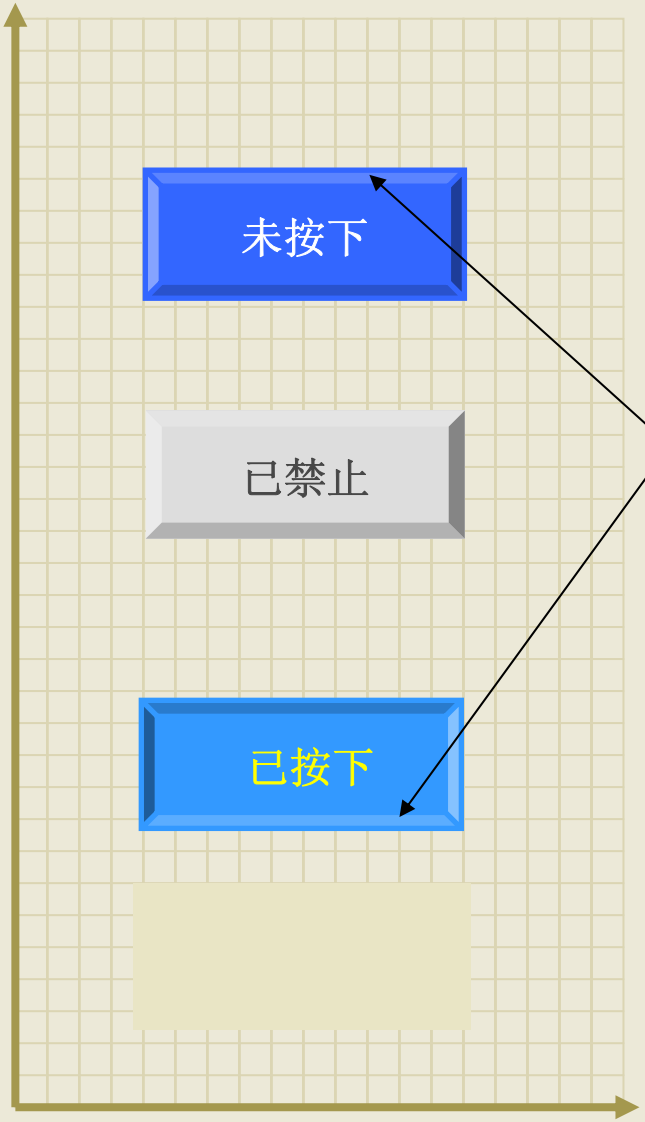

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = GREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTTAN;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案结构



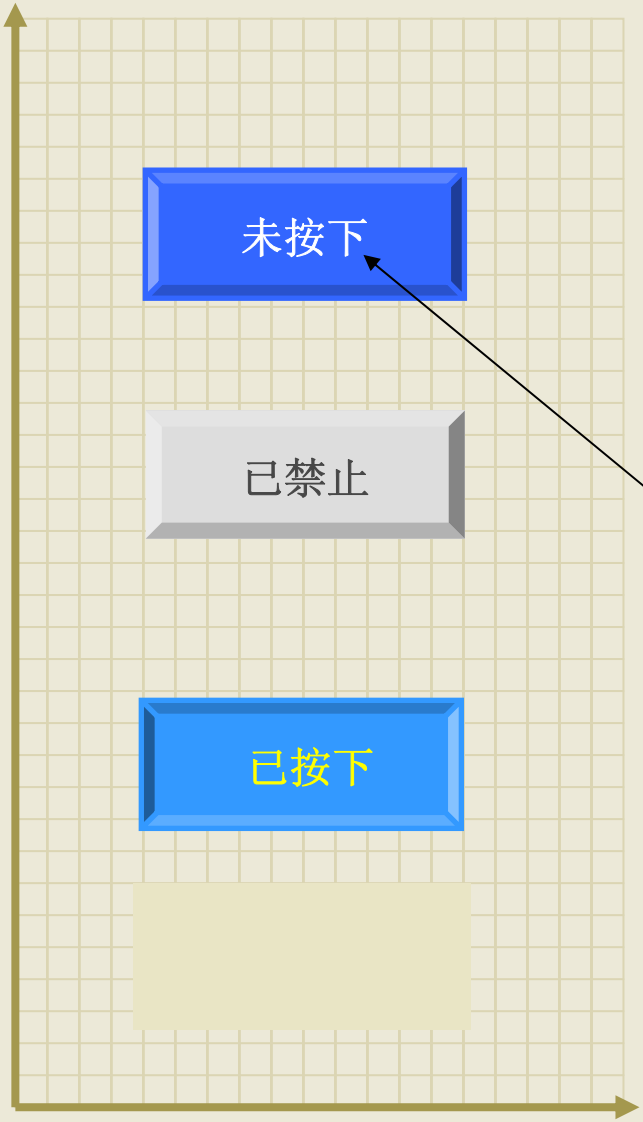

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = GREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTTAN;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



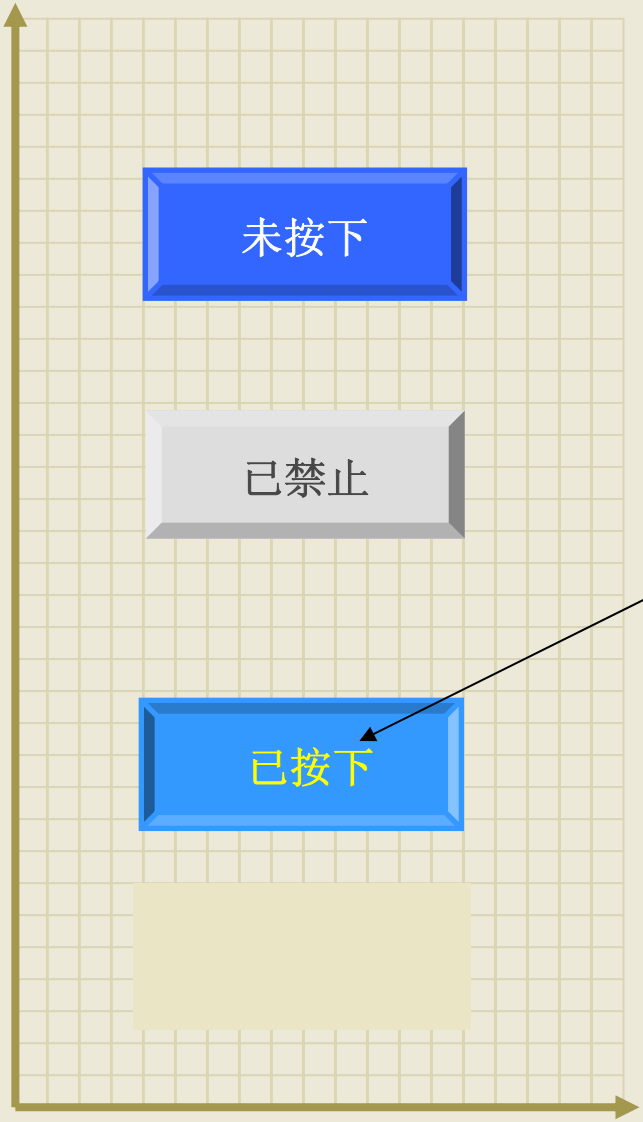

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = GREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTTAN;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



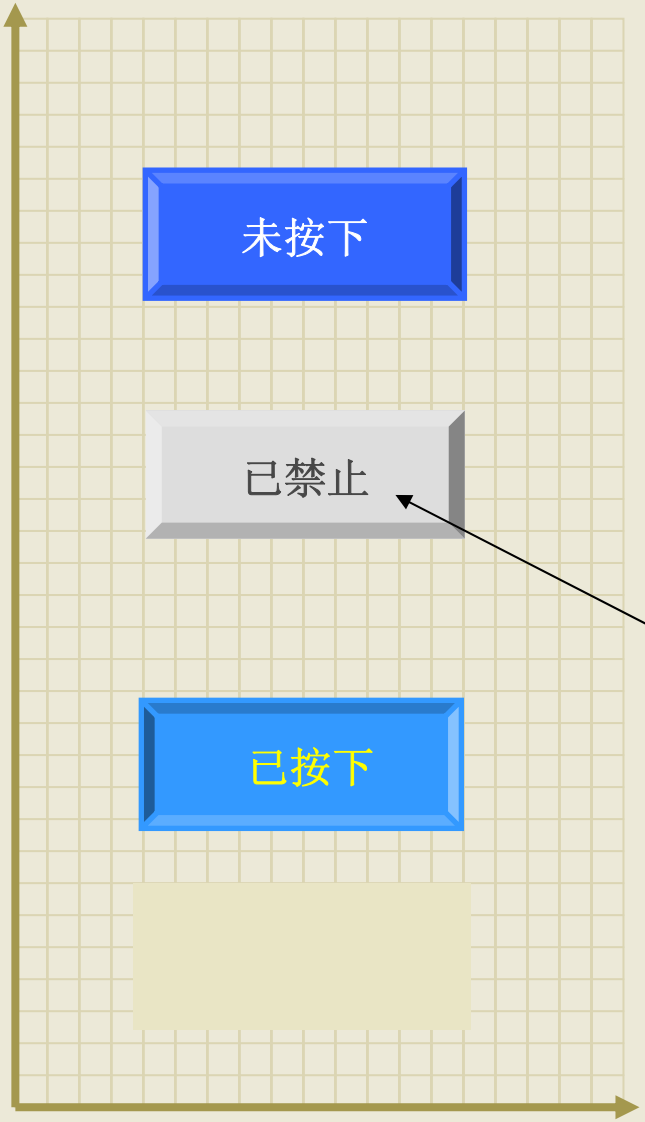

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = GREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTTAN;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



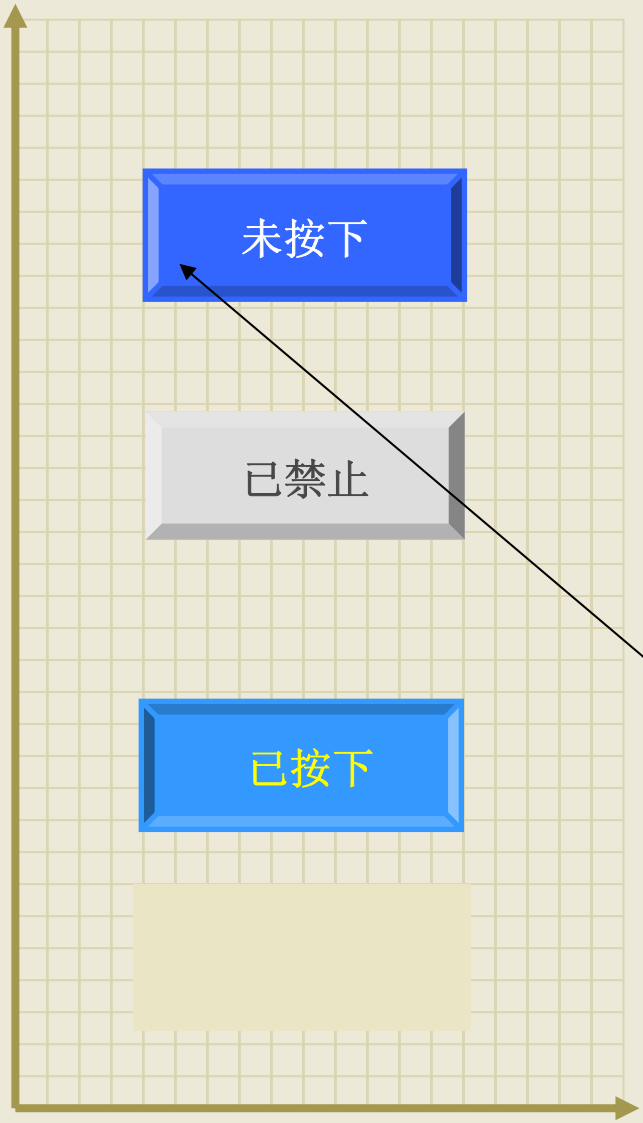

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = GREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



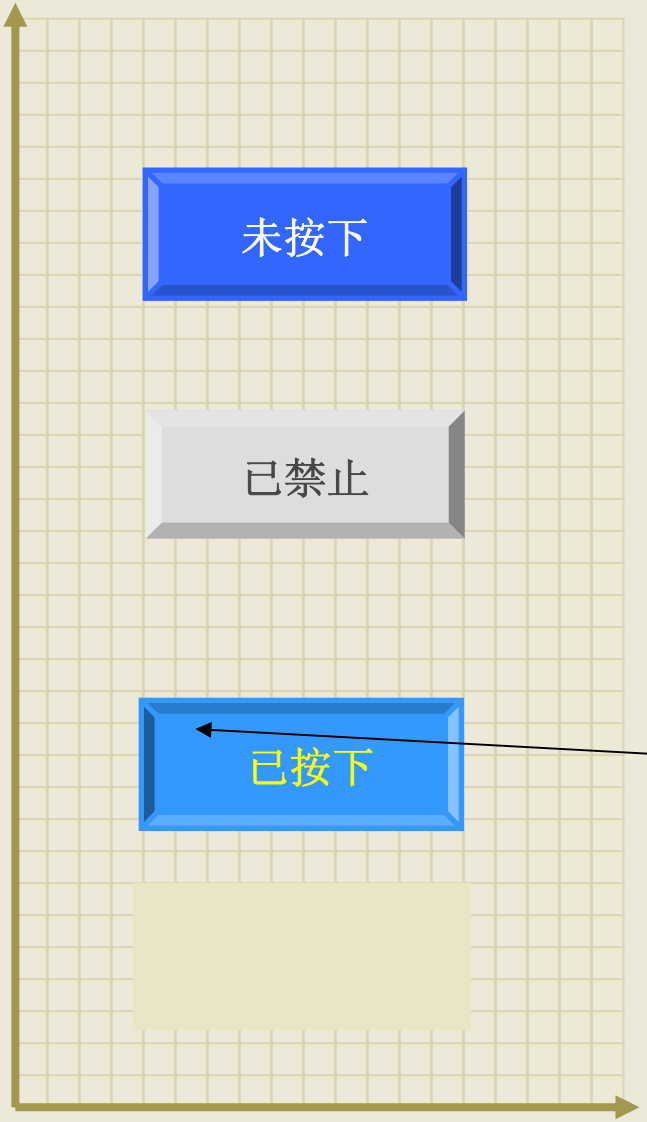

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



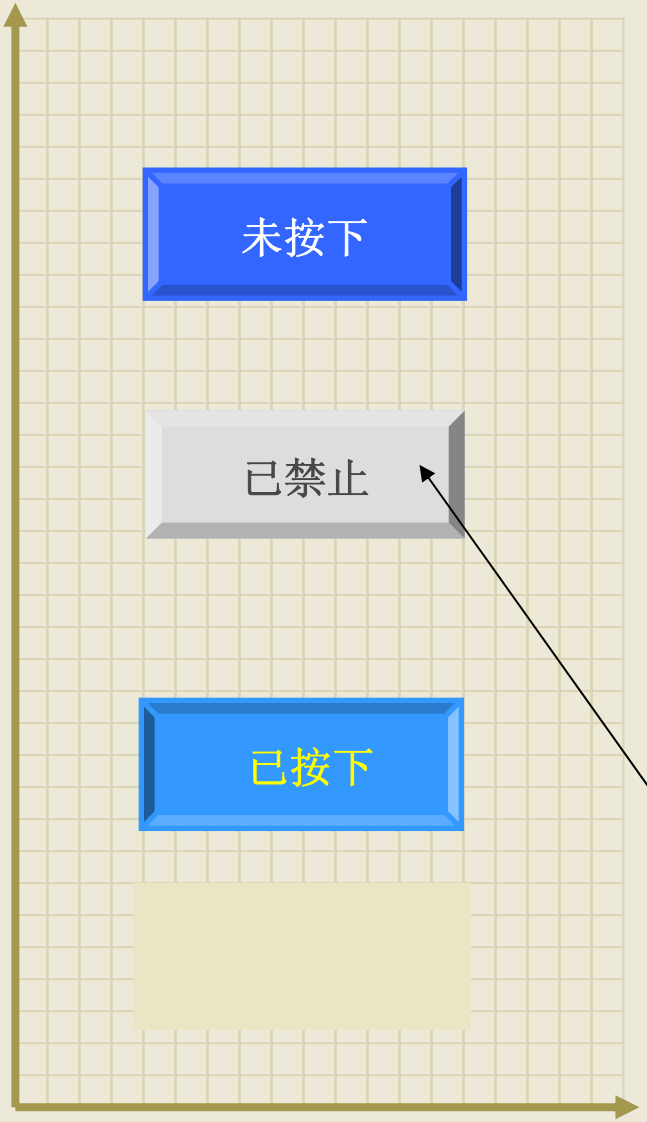

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



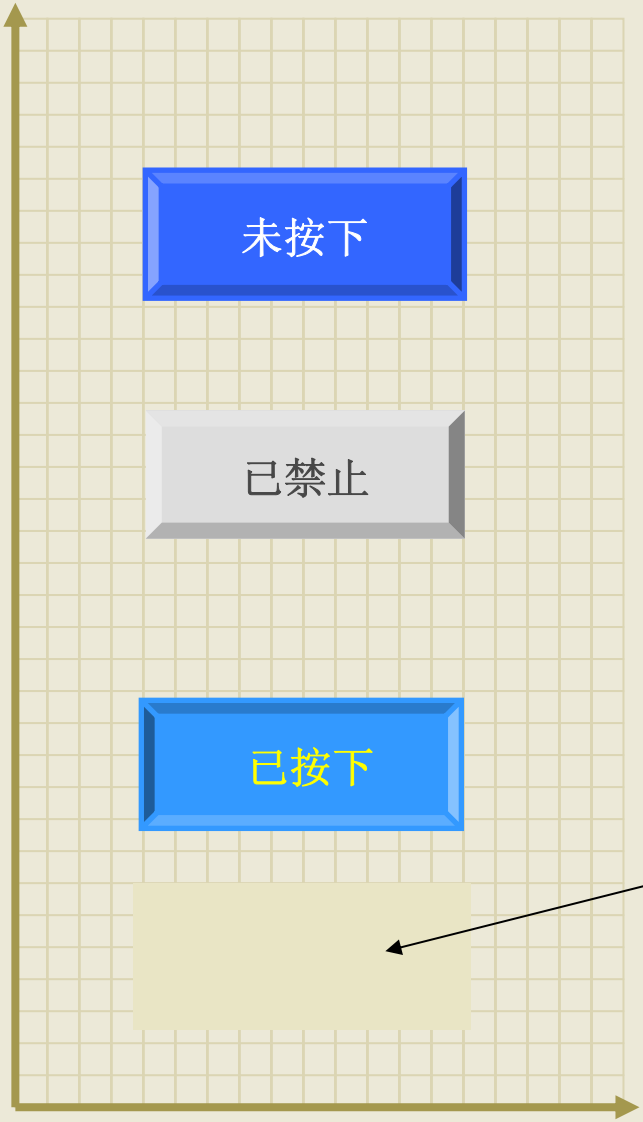

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



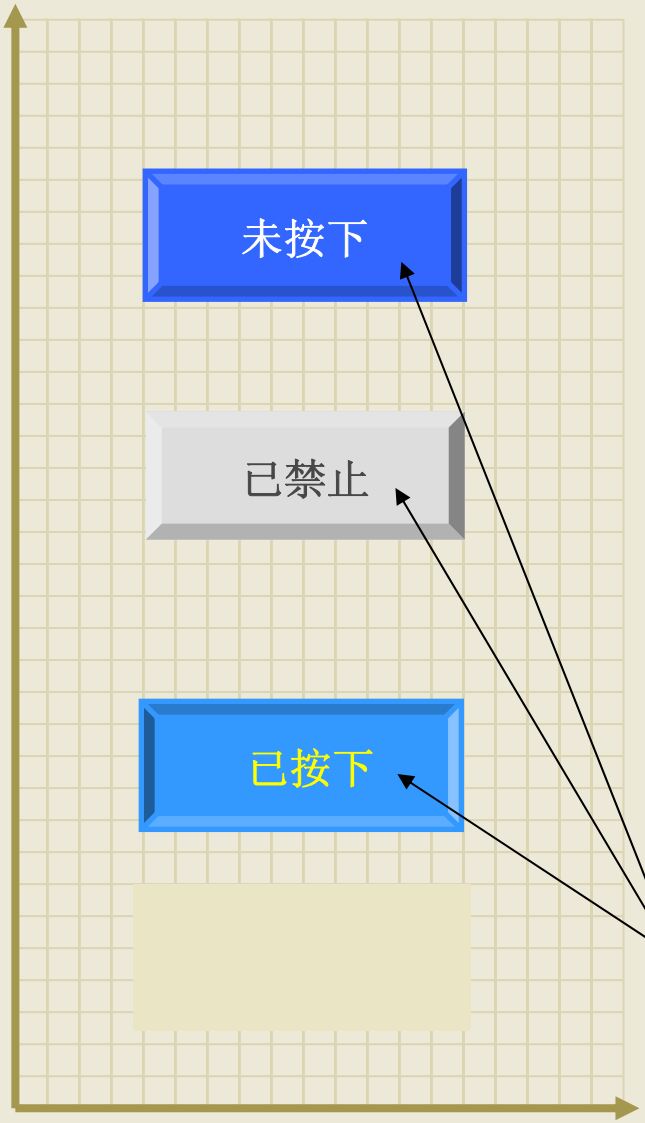

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



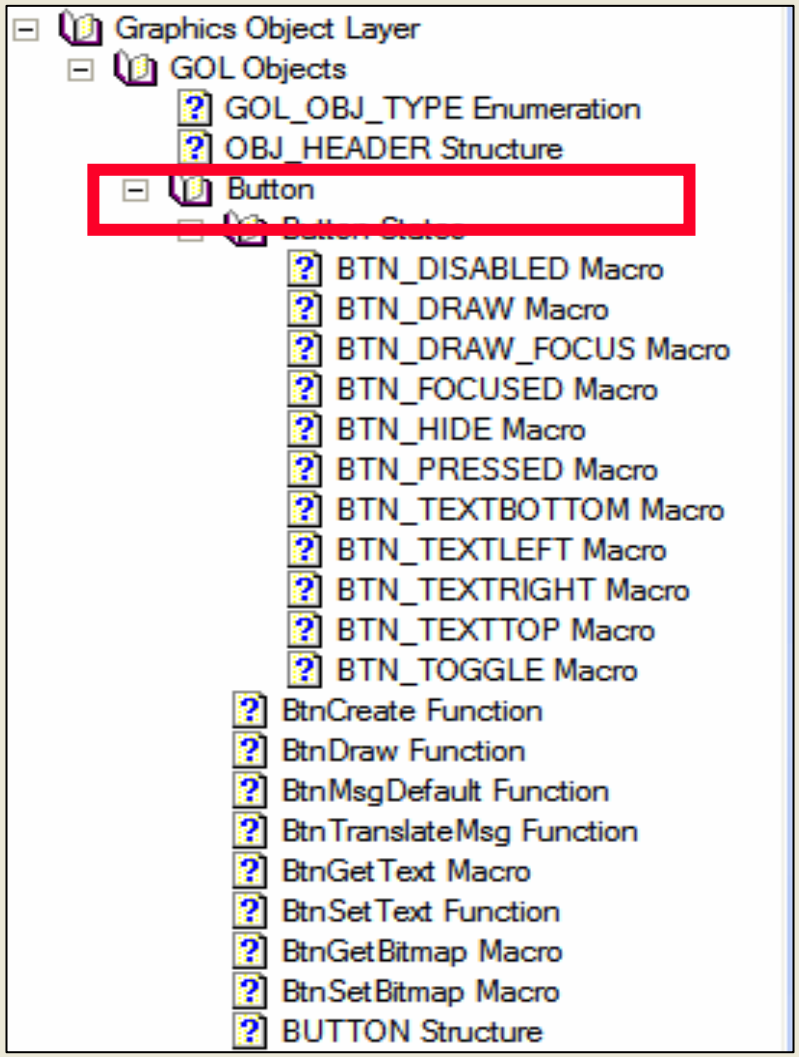

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

样式方案用法



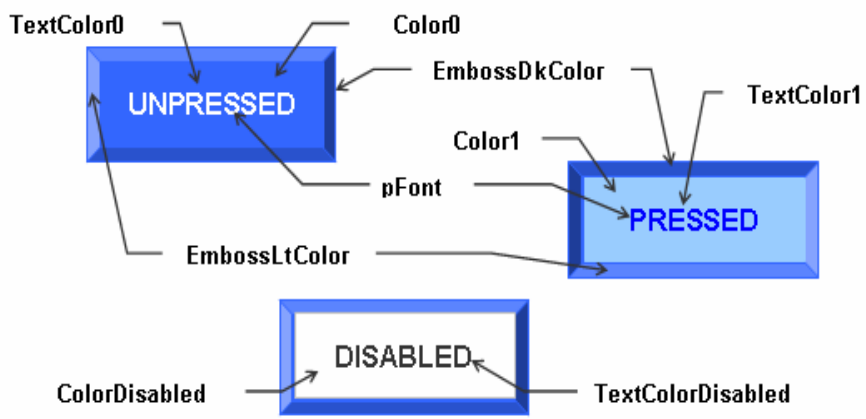

```
int main(void)
{
…
GOL_SCHEME     *altScheme;
altScheme = GOLCreateScheme();
…
altScheme -> EmbossDkColor = DKBLUE;
altScheme -> EmbossLtColor = BLUE;
altScheme -> TextColor0 = WHITE;
altScheme -> TextColor1 = YELLOW;
altScheme -> TextColorDisabled = DKGREY;
altScheme -> Color0 = BRIGHTBLUE;
altScheme -> Color1 = LTBLUE;
altScheme -> ColorDisabled = LTGREY;
altScheme -> CommonBkColor = TAN;
altScheme -> pFont = GOLFontDefault;
…
}
```

控件样式帮助



每个控件的顶层提供显示如何应用样式方案字段的图。

图形库帮助



CommonBkColor – used to hide the button on the screen.

样式方案API

- Style Scheme
  - GOLCreateScheme Function
  - GOLSetScheme Macro
  - GOLGetScheme Macro
  - GOLGetSchemeDefault Macro
  - GOL_SCHEME Structure
  - Default Style Scheme Settings
    - COLOR0DEFAULT Macro
    - COLOR1DEFAULT Macro
    - COLORDISABLEDDEFAULT Macro
    - COMMONBACKGROUNDCOLORDEFAULT Macro
    - TEXTCOLOR0DEFAULT Macro
    - TEXTCOLOR1DEFAULT Macro
    - EMBOSSDKCOLORDEFAULT Macro
    - EMBOSSLTCOLORDEFAULT Macro
    - FONTDEFAULT Macro
    - TEXTCOLORDISABLEDDEFAULT Macro
  - GOLFontDefault Variable
  - GOL_EMBOSS_SIZE Macro
  - RGB565CONVERT Macro

影响样式方案的其他**API**的描述可在图形库帮助文件中找到。

图形库帮助

要在创建控件后更改它的样式方案，请使用：

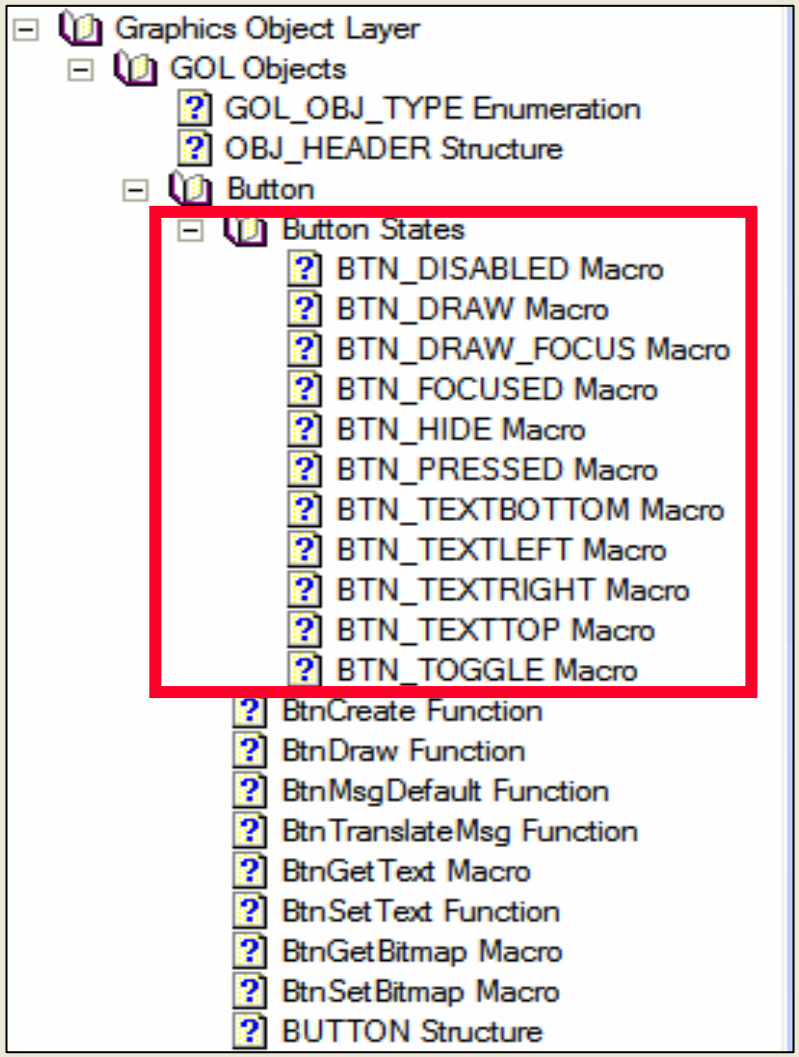
```
GOLSetScheme(*pObj, *pScheme)
```

Microchip图形绘制控件

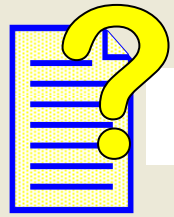
State字段
绘图位

- 控件结构的组成部分
- 指示对象需要被：
  - 隐藏
  - 部分重绘
  - 完全重绘
- 由所有控件使用
  - **OBJ_HIDE**：用CommonBkColor覆盖区域
  - **OBJ_DRAW**：重新绘制控件
    - 用 **GOLDraw()**自动清除
  - **OBJ_DRAW_FOCUS**：只重绘焦点
    - 由**GOLDraw()**自动设置

状态位帮助



每个控件还有独特的状态位。这些可以在库帮助文件中找到，左栏所示。

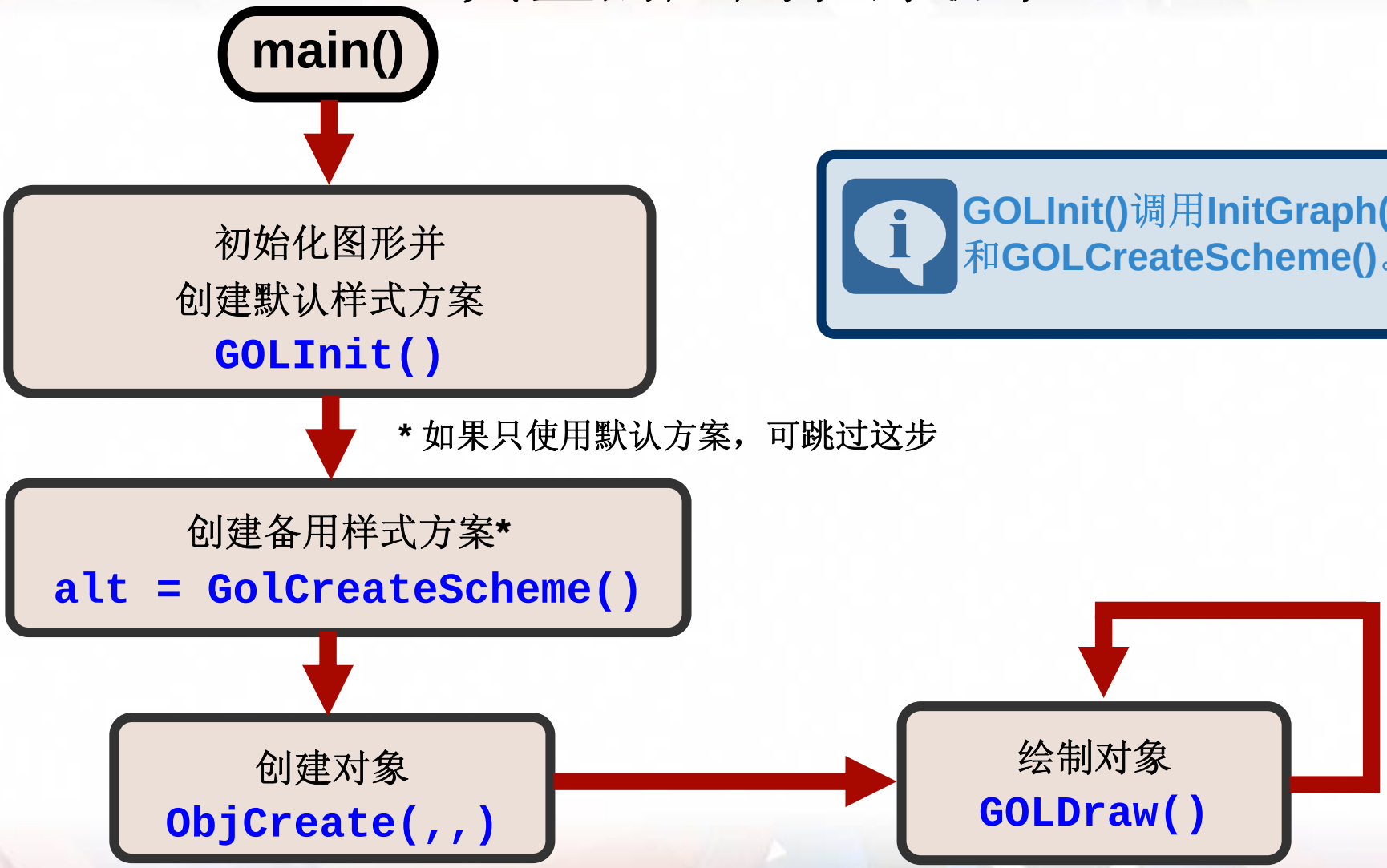


图形库帮助

绘制控件

- **GOLDraw()**
  - 解析链表
  - 检查列表中每个控件的状态位
  - 如果设置了绘图位，将渲染控件
  - 完成后将返回TRUE

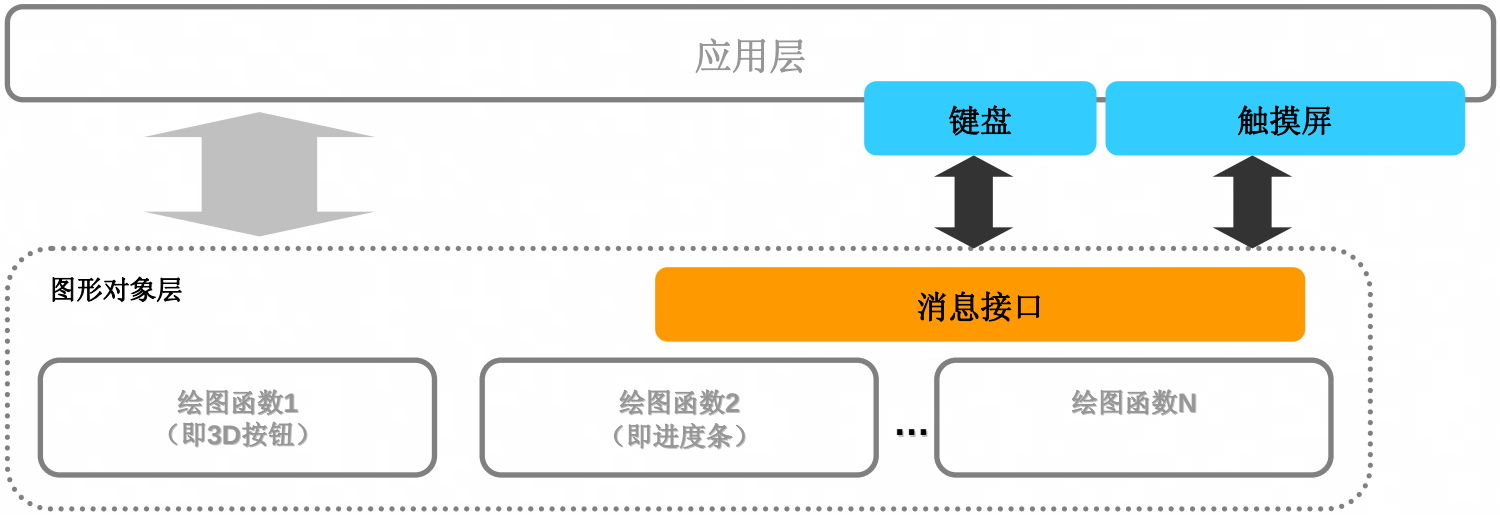
绘图示例
典型的应用程序流程




Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

与用户交互

消息接口



- 简化用户输入设备的集成
- 允许应用程序有效管理控件
- 为用户提供无缝接口
- 将来可能支持更多设备（如鼠标）

与用户交互
应用程序需求

- 检测用户输入
- 填充消息结构
- 调用**GOLMsg(&msg)**
  - **&msg**是消息结构的地址
- 提供回调函数（必需）
  - **GOLMsgCallback(,,,)**
    - 基于单个事件的系统/控件操作
    - 示例：按下按钮打开LED
    - 由 **GOLMsg()**调用
  - **GOLDrawCallback()**
    - 基于连续事件的系统/控件操作
    - 示例：按住按钮改变音量
    - 由 **GOLDraw()**调用
    - 修改绘图属性的唯一安全方式

消息结构

- 消息结构

```
typedef struct {
    BYTE                type;
    BYTE                uiEvent;
    SHORT               param1;
    SHORT               param2;
    } GOL_MSG;
```

- **type = *TYPE_KEYBOARD或TYPE_TOUCHSCREEN***
- **Param1和param2**取决于事件和类型
  - 对于触摸屏：
    - **Param1**：x位置
    - **param2**：y位置
  - 对于键盘：
    - **Param1**：接收消息的控件ID
    - **Param2**：取决于控件和事件

消息结构

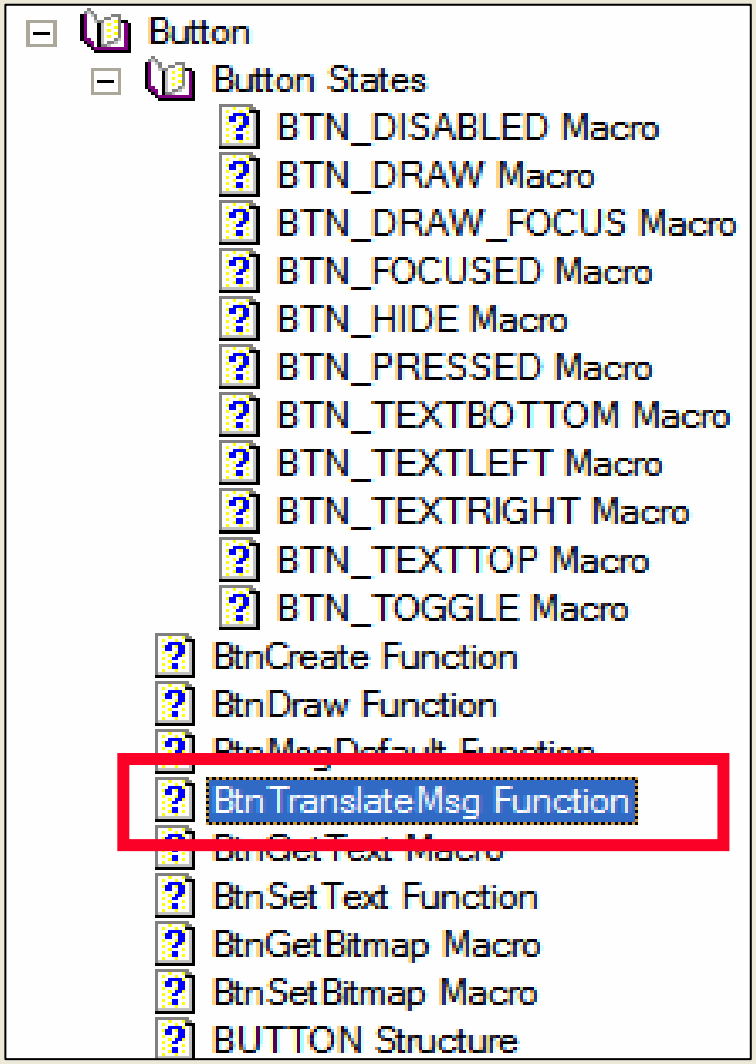
- **uiEvent**定义特定的用户操作
  - **uiEvent**值，用于触摸屏
    - **EVENT_PRESS**
    - **EVENT_RELEASE**
    - **EVENT_MOVE**
    - **EVENT_INVALID**
      - 触摸不影响控件
  - **uiEvent**值，用于键盘
    - **EVENT_KEYSCAN**
      - **param2**值随控件而异
      - 通常是一个扫描代码
    - **EVENT_CHARCODEe**（仅限编辑框）
      - **param2** = 要添加的字符
    - **EVENT_INVALID**

AT键盘扫描代码在库帮助文件中提供

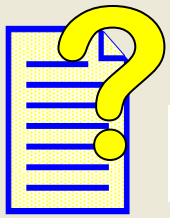
示例
填充侧面按钮消息

```
if(S5){
        msg->type    = TYPE_KEYBOARD;
        msg->uiEvent = EVENT_KEYSCAN;
        msg->param1  = obj->ID;
        msg->param2  = SCAN_CR_PRESSED;
}else{
        msg->type    = TYPE_KEYBOARD;
        msg->uiEvent = EVENT_KEYSCAN;
        msg->param1  = obj->ID;
        msg->param2  = SCAN_CR_RELEASED;
}return;
```

控件消息帮助



描述有效输入源、事件和默认行为的表可在每个控件的**ObjTranslateMsg**函数说明中找到。



图形库帮助

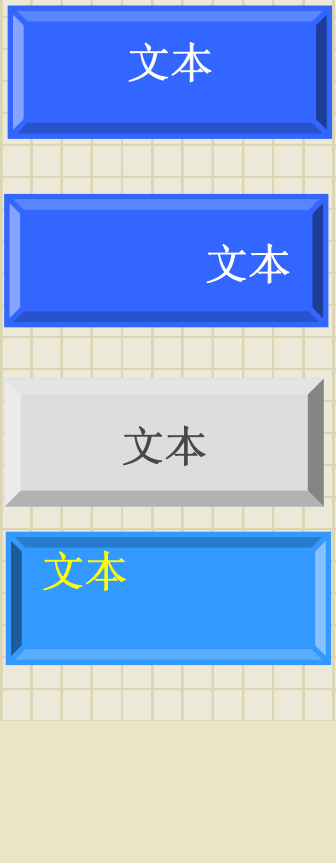

Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

与用户交互
控件操作

State字段属性位

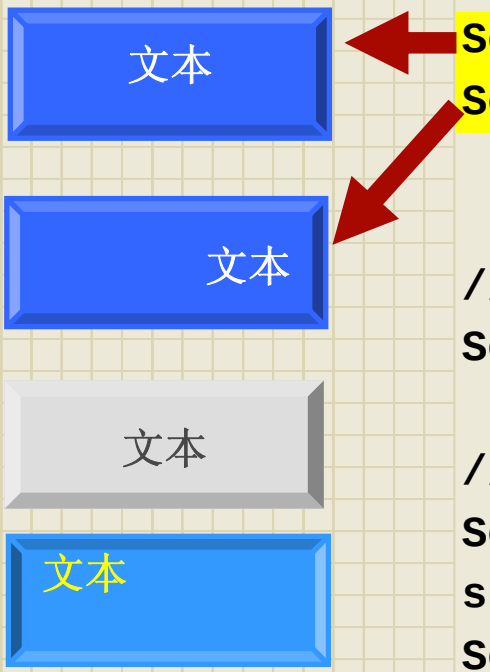
- 定义操作和外观
  - 不会自动受**GOLDraw()**影响
- 部分控件：
  - **OBJ_FOCUSED**：控件获取焦点
- 所有控件：
  - **OBJ_DISABLED**：控件已关闭
    - 所有消息都将被忽略

按钮状态



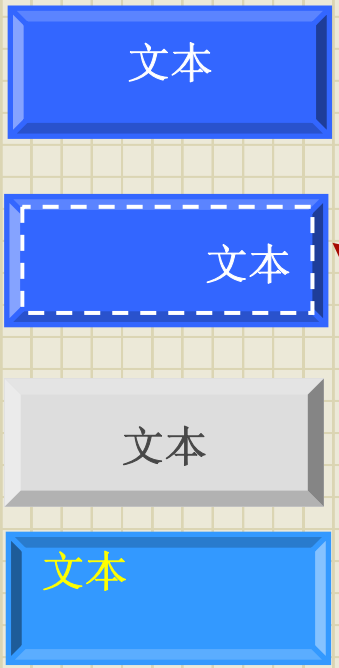

```
if (GOL_DRAW())
{
// 示例按钮文本对齐
SetState(pBtn, BTN_TEXTTOP);
SetState(pBtn, BTN_TEXTRIGHT);


// 示例按钮获取焦点
SetState(pBtn, BTN_FOCUSED);

// 示例按钮操作
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTRIGHT;
SetState(pBtn, state);

// 示例按钮隐藏
SetState(pBtn, BTN_HIDE)
}
```

按钮状态



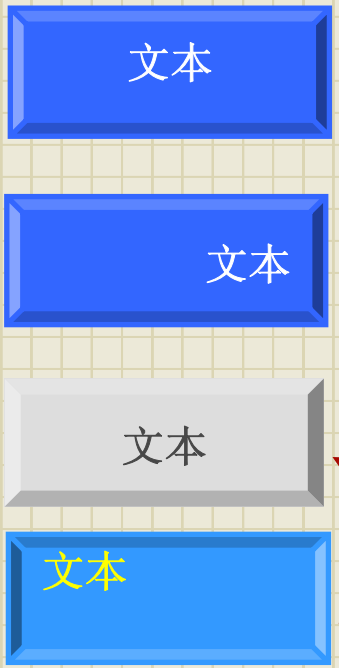

```
if (GOL_DRAW())
{
// 示例按钮文本对齐
SetState(pBtn, BTN_TEXTTOP);
SetState(pBtn, BTN_TEXTRIGHT);


// 示例按钮获取焦点
SetState(pBtn, BTN_FOCUSED);

// 示例按钮操作
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTRIGHT;
SetState(pBtn, state);

// 示例按钮隐藏
SetState(pBtn, BTN_HIDE)
}
```

按钮状态



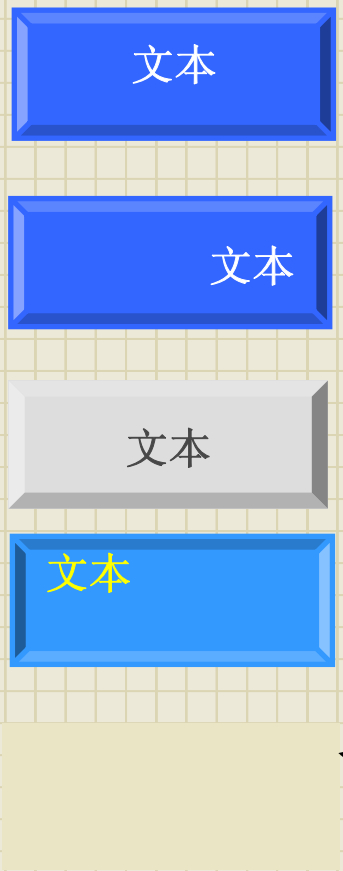

```
if (GOL_DRAW())
{
// 示例按钮文本对齐
SetState(pBtn, BTN_TEXTTOP);
SetState(pBtn, BTN_TEXTRIGHT);


// 示例按钮获取焦点
SetState(pBtn, BTN_FOCUSED);

// 示例按钮操作
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTRIGHT;
SetState(pBtn, state);

// 示例按钮隐藏
SetState(pBtn, BTN_HIDE)
}
```

按钮状态



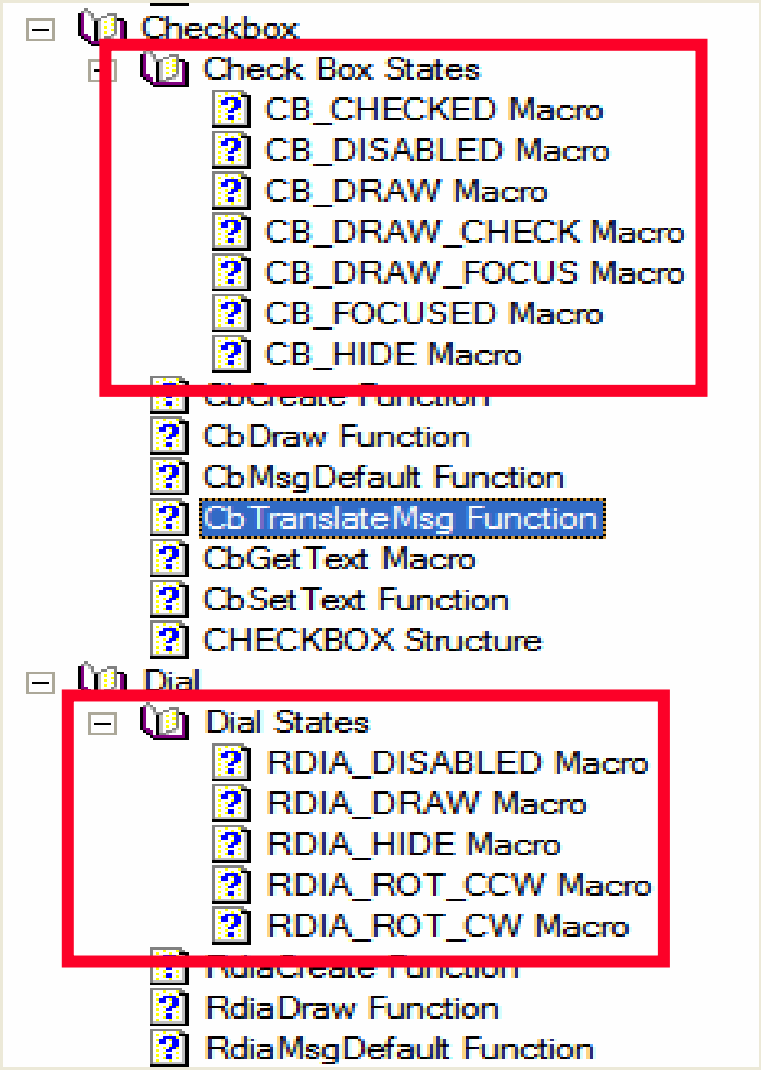

```
if (GOL_DRAW())
{
// 示例按钮文本对齐
SetState(pBtn, BTN_TEXTTOP);
SetState(pBtn, BTN_TEXTRIGHT);


// 示例按钮获取焦点
SetState(pBtn, BTN_FOCUSED);

// 示例按钮操作
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTRIGHT;
SetState(pBtn, state);

// 示例按钮隐藏
SetState(pBtn, BTN_HIDE)
}
```

按钮状态




```
if (GOL_DRAW())
{
// 示例按钮文本对齐
SetState(pBtn, BTN_TEXTTOP);
SetState(pBtn, BTN_TEXTRIGHT);

#define USE_FOCUS
// 示例按钮获取焦点
SetState(pBtn, BTN_FOCUSED);

// 示例按钮操作
SetState(pBtn, BTN_DISABLED);
state = BTN_PRESSED|BTN_TEXTTOP|BTN_TEXTRIGHT;
SetState(pBtn, state);

// 示例按钮隐藏
SetState(pBtn, BTN_HIDE);
}
```

状态位帮助

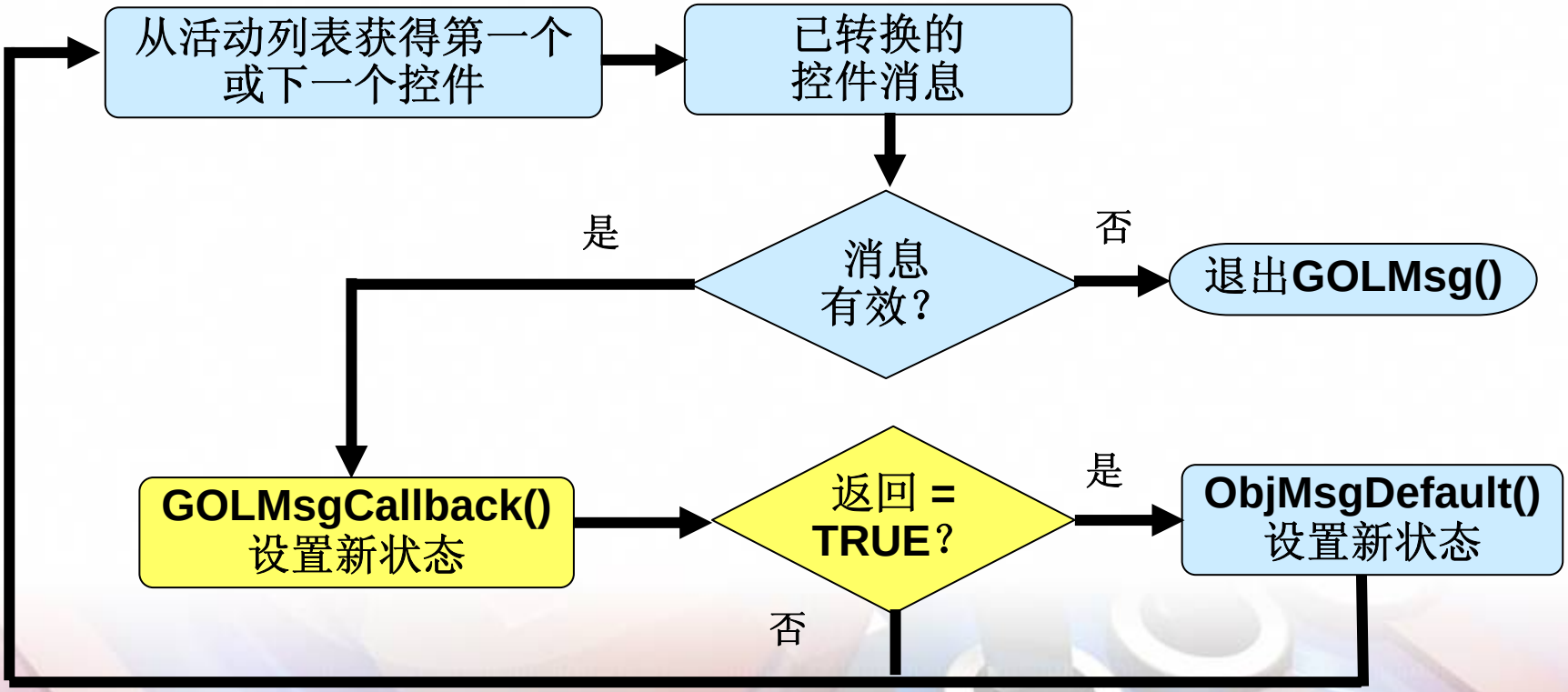


所有控件的状态位说明都可以在图形库帮助文件中找到。

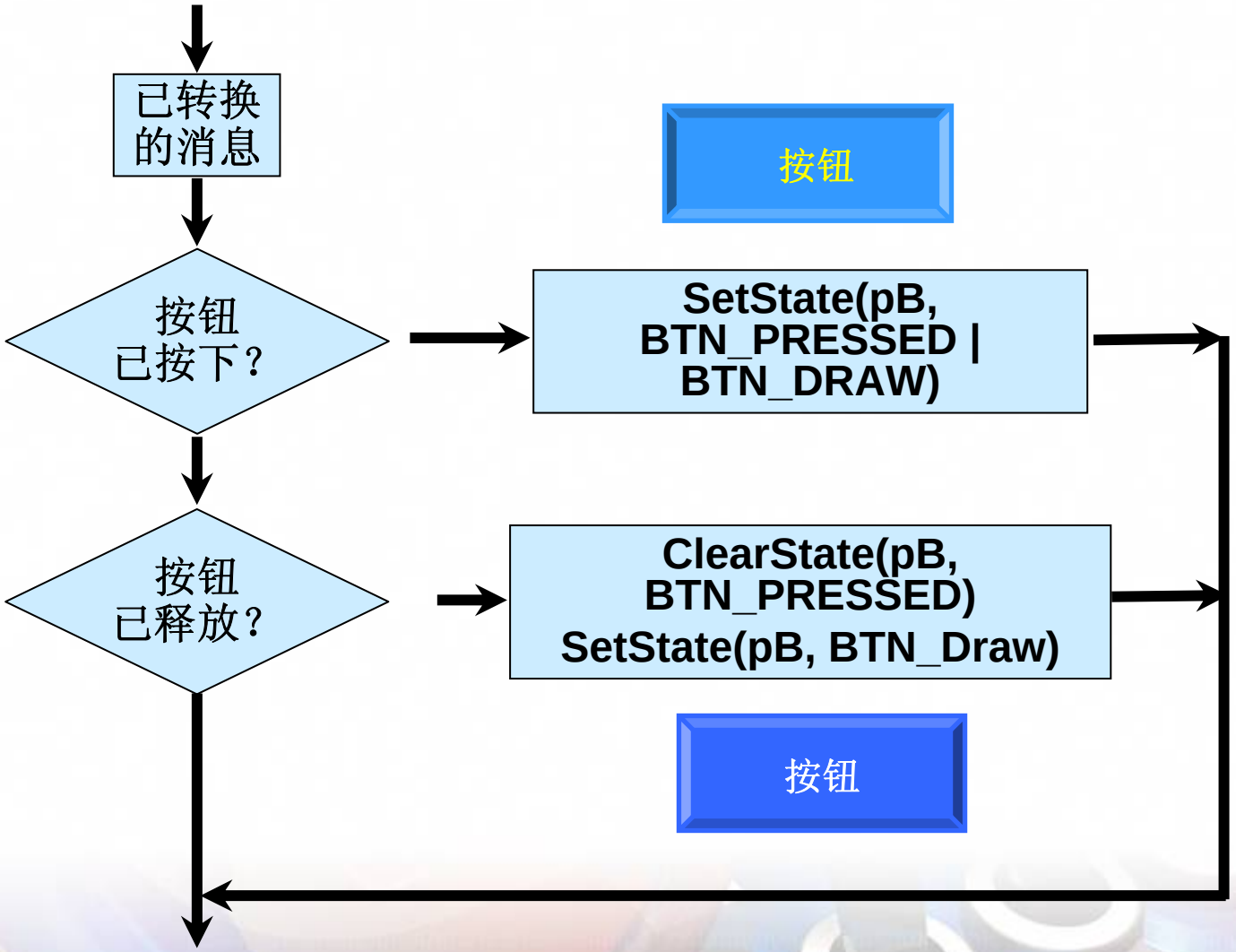
图形库帮助

GOLMsg（&msg）

- 在应用程序主循环中调用
- 不要调用，除非**GOLDraw()**已完成
  - **if (GOLDraw()) GOLMsg(&msg);**



控件默认操作
按钮

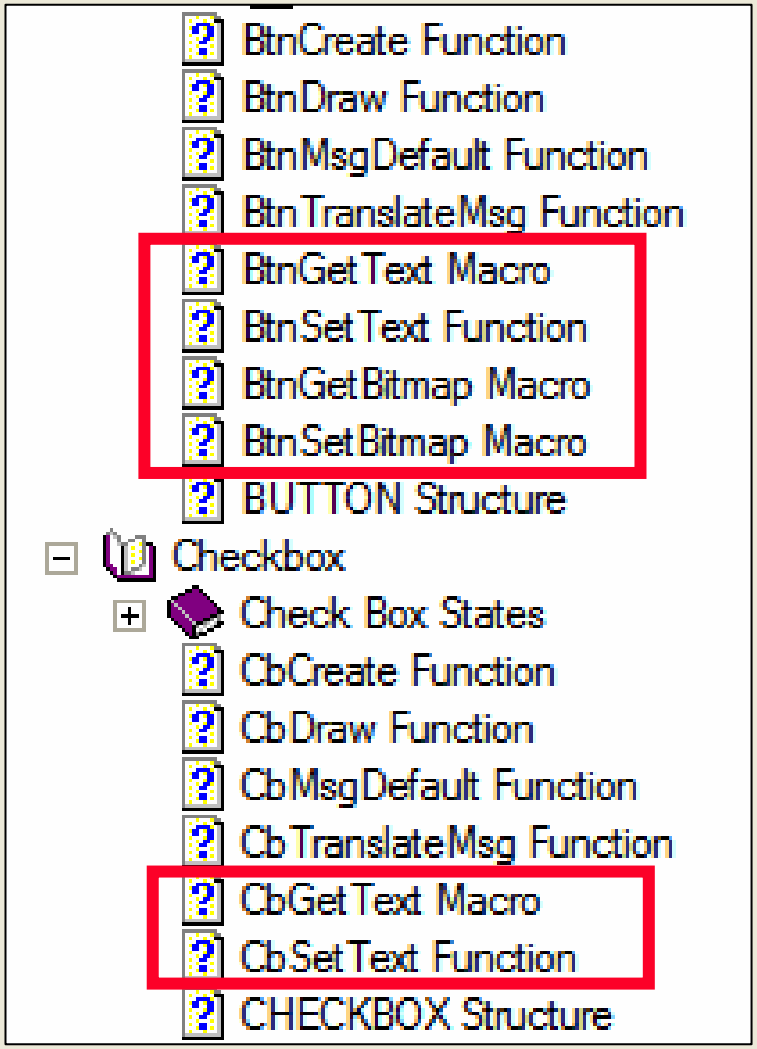

已转换的消息
按钮
按钮已按下？
SetState(pB, BTN_PRESSED | BTN_DRAW)
按钮已释放？
ClearState(pB, BTN_PRESSED)
SetState(pB, BTN_Draw)
按钮

技巧

- 在[Obj]MsgDefault()中找到默认控件值，前者位于[Obj].c中

```
void RbMsgDefault(WORD translatedMsg, RADIOBUTTON* pRb, GOL_MSG* pMsg)
{
…
    if(translatedMsg == RB_MSG_CHECKED){

        // 未选中组中的单选按钮
        pointer = (RADIOBUTTON*) pRb->pHead;

        while(pointer != NULL){
            if(GetState(pointer,RB_CHECKED)){
               ClrState(pointer, RB_CHECKED);      // 复位选中
               SetState(pointer, RB_DRAW_CHECK);   // 重新绘制
            }
            pointer = (RADIOBUTTON*)pointer->pNext;
        }
       // 设置选中和重新绘制
        SetState(pRb, RB_CHECKED|RB_DRAW_CHECK);
    }
}
```

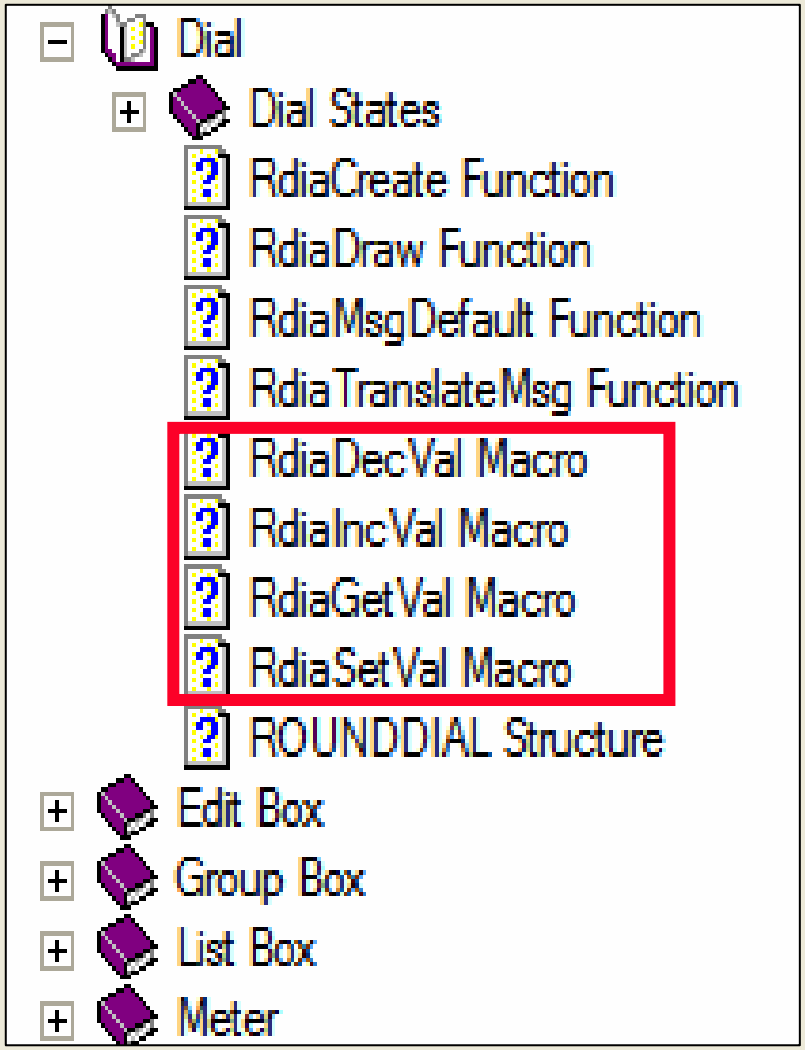
控件API帮助



控件**API**位于帮助文件的“图形对象层”部分的各个控件下。

图形库帮助

控件API帮助



控件**API**位于帮助文件的“图形对象层”部分的各个控件下。

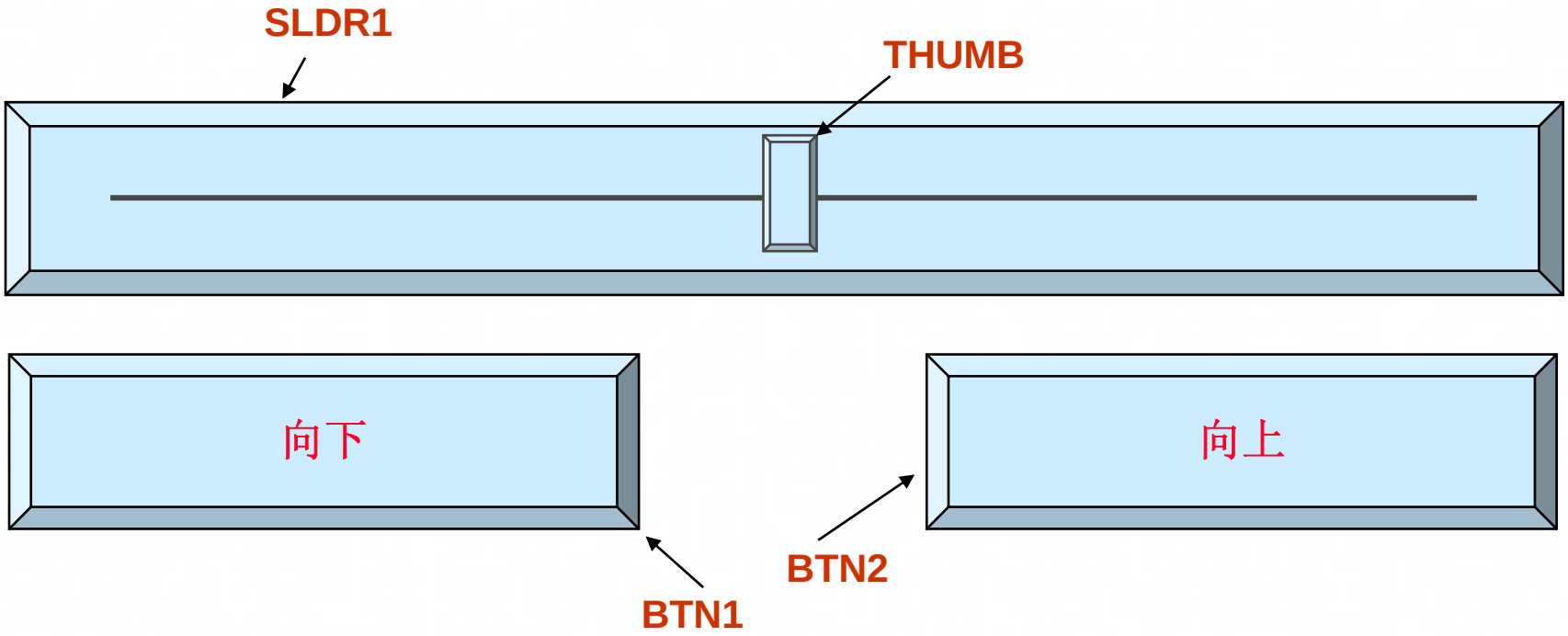
图形库帮助

与用户交互
GOLMsgCallback()

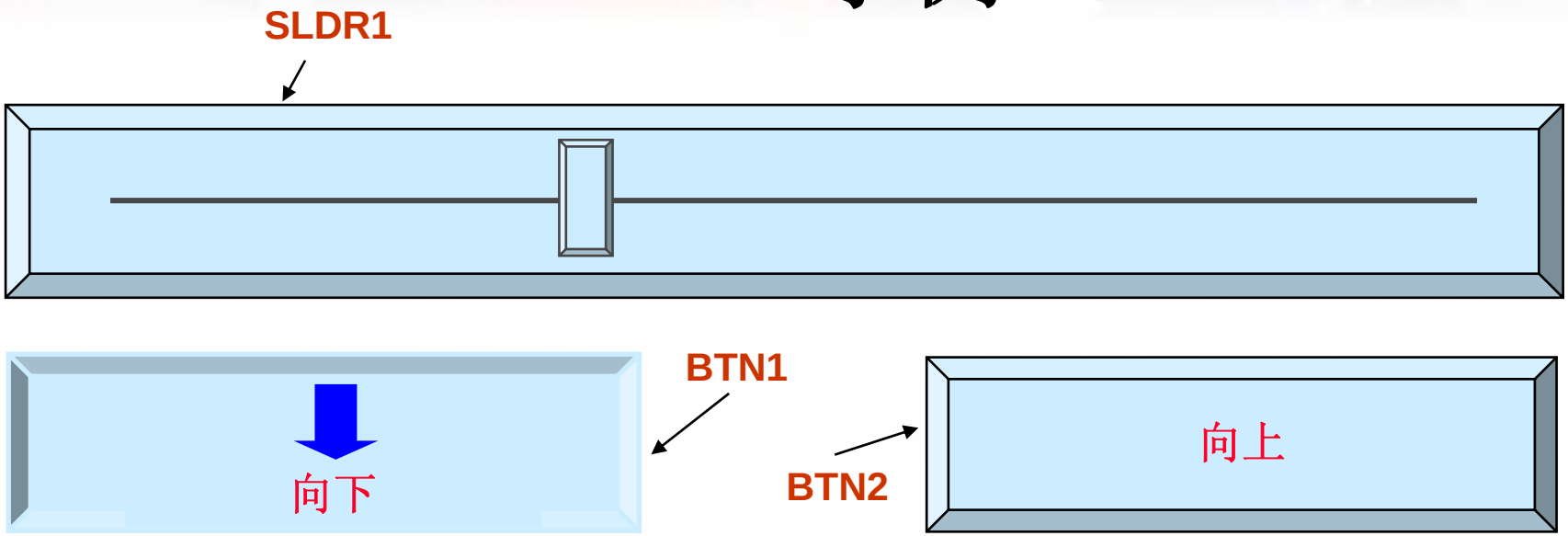
GOLMsgCallback()

- 必须在应用程序代码中提供
- 执行定制控件操作
  - 示例：按下按钮时更改位图
- 与系统交互
  - 示例：按下按钮时点亮LED
- 输入参数：
  - **objMsg**：　控件的已转换消息
  - **pObj**：　指向控件的指针
  - **pMsg**：　指向消息结构的指针
- 输出：
  - **TRUE**：　也执行默认操作
  - **“0”**：　跳过默认操作

GOLMsgCallback()示例

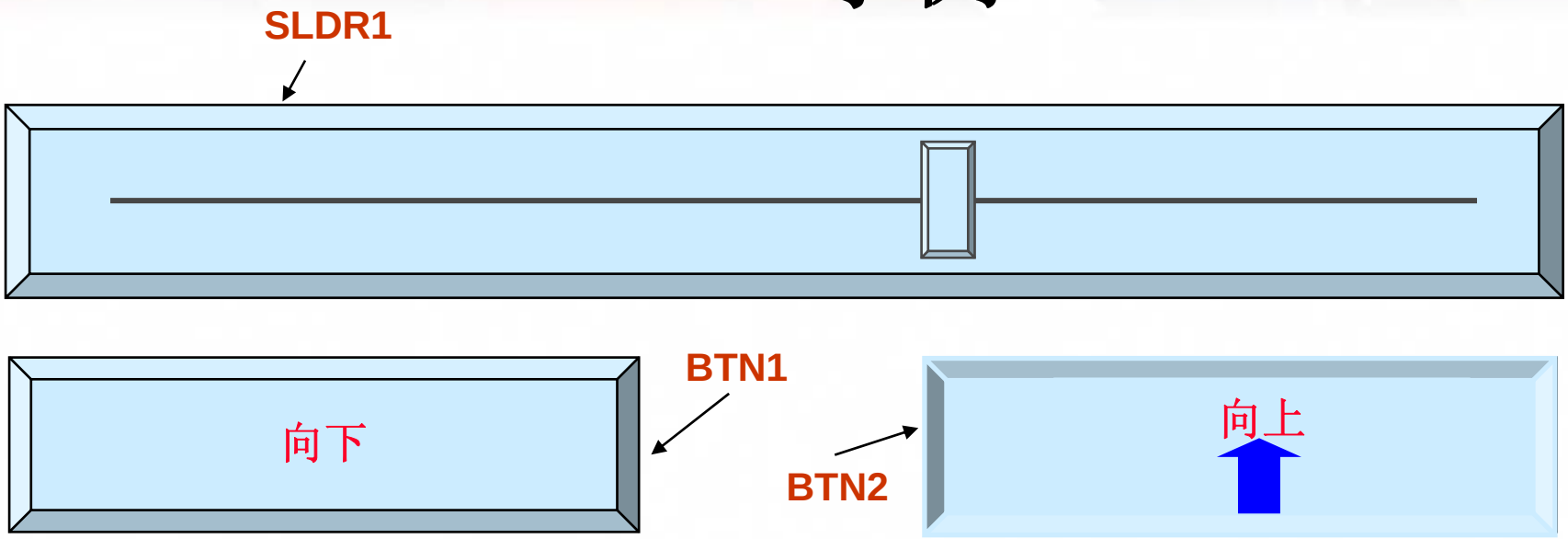


GOLMsgCallback()示例



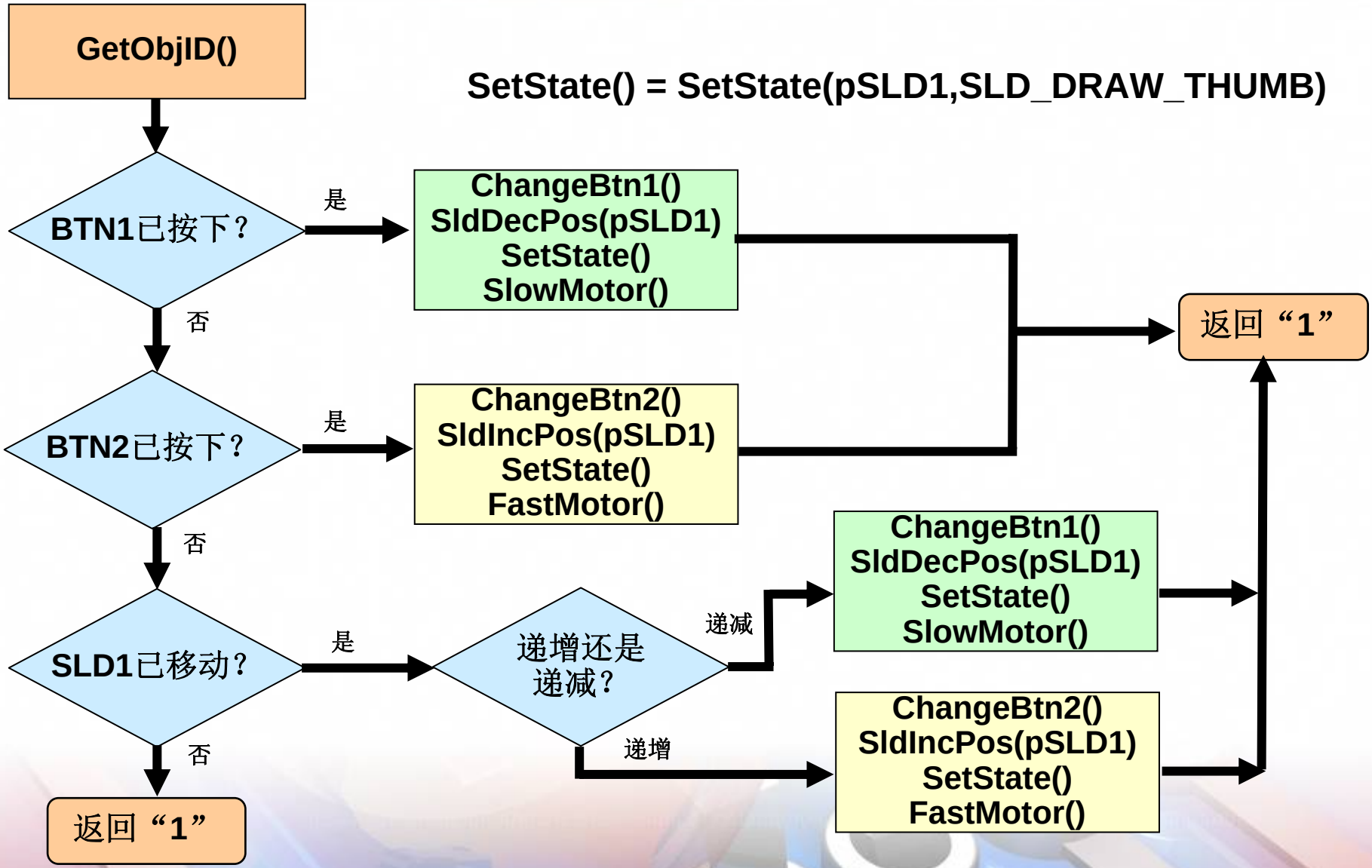
- 控件操作：
  - 使滑块左移
  - 下移**BTN1**文本
  - 添加向下箭头位图
- 系统操作：
  - 降低电机速度

GOLMsgCallback()示例



- 控件操作：
  - 将滑块右移
  - 上移**BTN2**文本
  - 添加向上箭头位图
- 系统操作：
  - 提高电机速度

GOLMsgCallback()示例

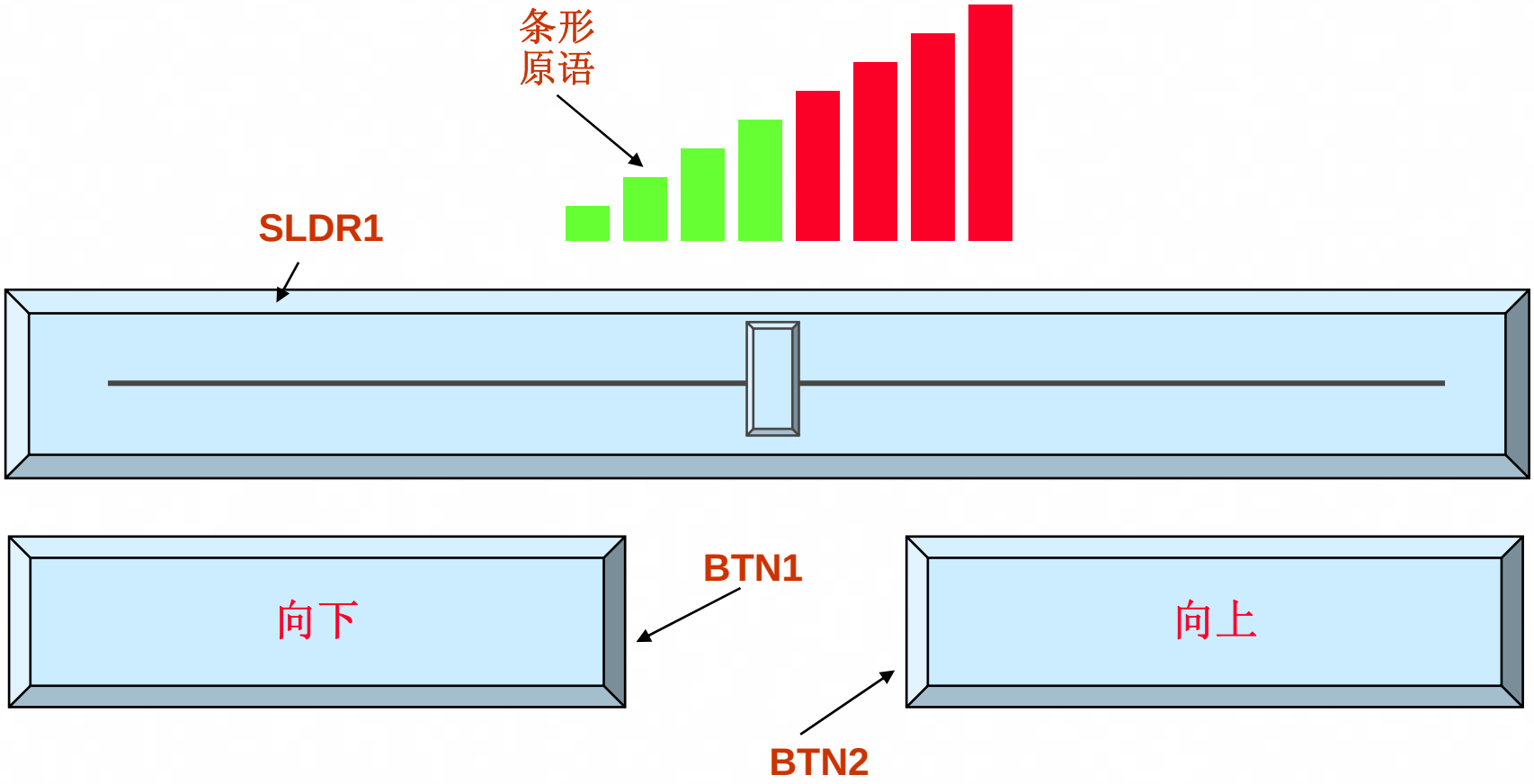


GOLDrawCallback()

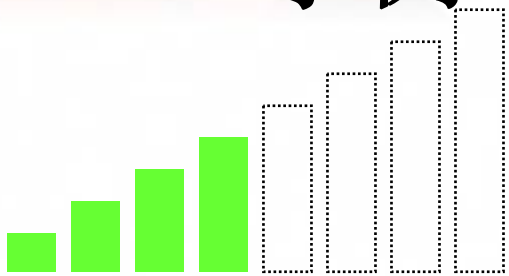
定制绘图
GOLDrawCallback()

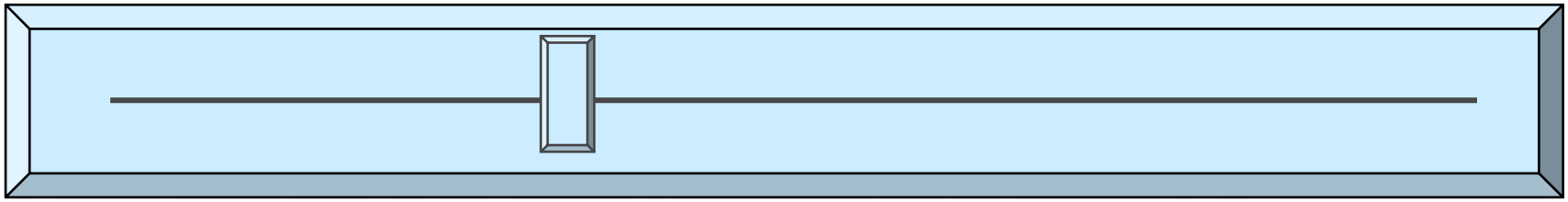
- 绘图完成后由**GOLDraw()**调用
- 必须包含在应用程序代码中
  - 返回**TRUE**将控制权交还**GOLDraw()**
  - 返回**“0”**保持绘图控制
- 执行定制绘图
  - 示例：信号强度指示器
- 连续事件监控
  - 示例：按住按钮更改音量
- 进行以下操作的唯一安全途径：
  - 更改绘图参数
  - 修改活动链表

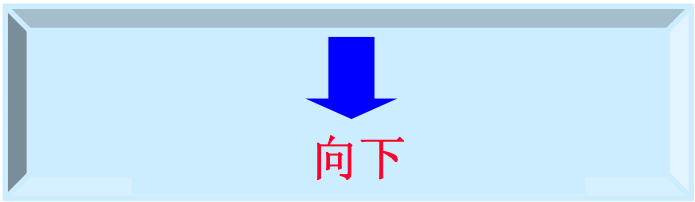
GOLDrawCallback()
示例

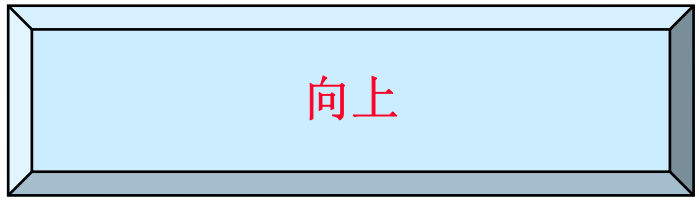


GOLDrawCallback()示例



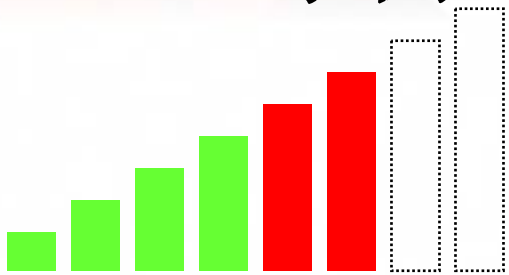


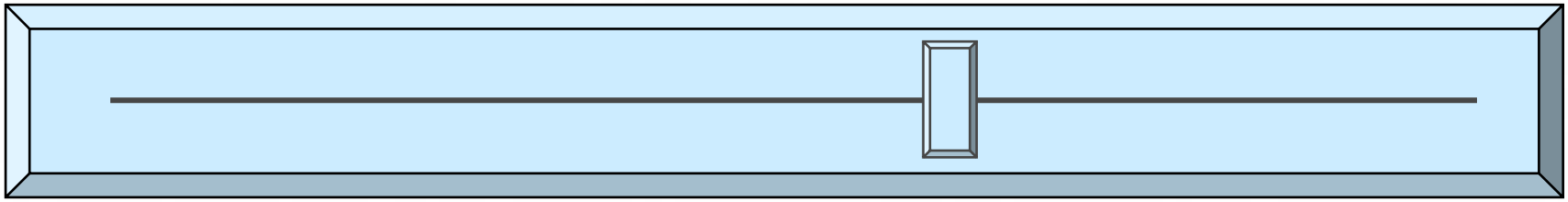


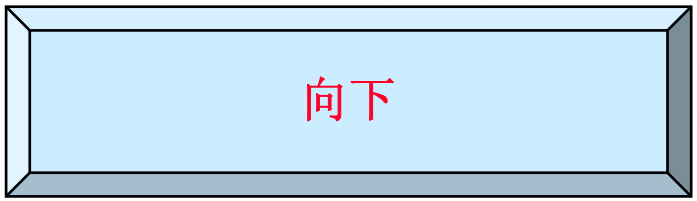


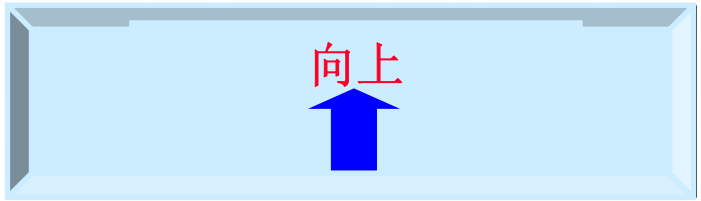
- 控件操作：
  - 将滑动块左移
  - 下移**BTN1**文本并添加图像
  - 擦除条
- 系统操作：
  - 降低音量

GOLDrawCallback()
示例



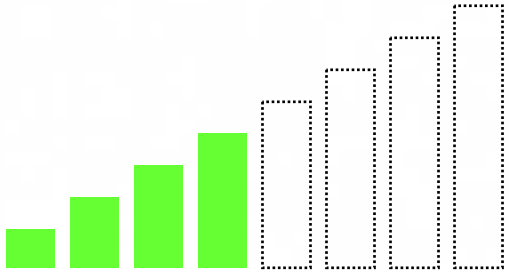






- 控件操作：
  - 将滑动块右移
  - 上移**BTN2**文本并添加图像
  - 添加条
- 系统操作：
  - 增大音量

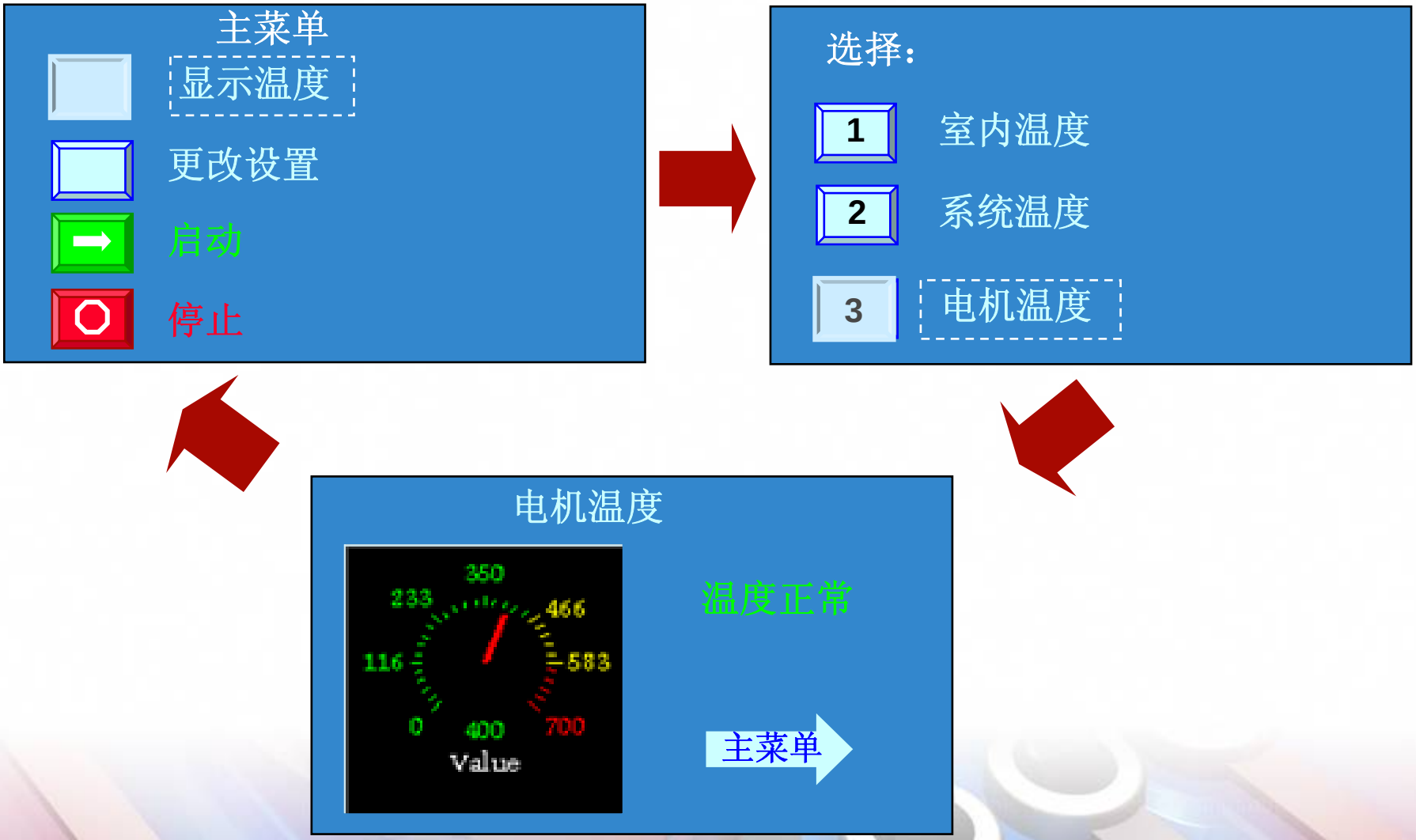
GOLDrawCallback()
示例——流程图



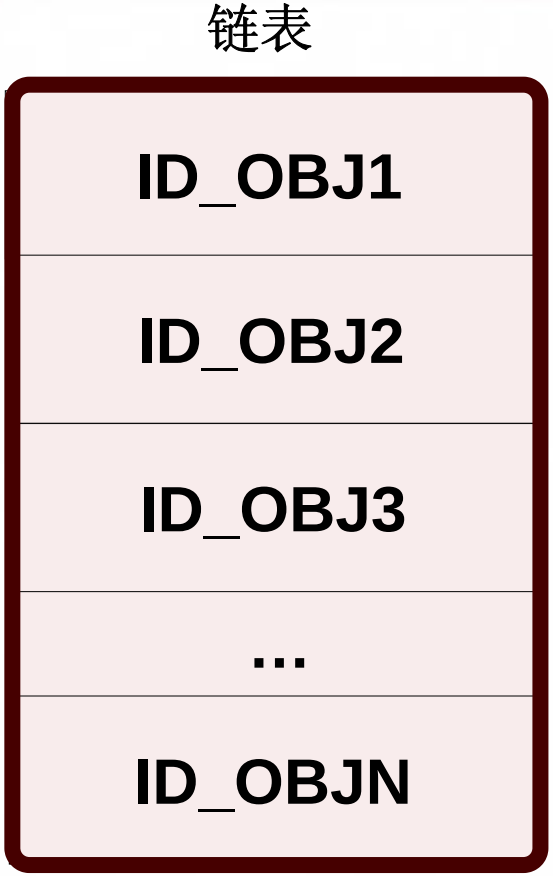

Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

汇总

GUI应用程序


主菜单
显示温度
更改设置
启动
停止
选择:
1
室内温度
2
系统温度
3
电机温度
电机温度
350
233
466
116
583
0
400
700
Value
温度正常
主菜单

创建控件



重新调用：

- **ObjCreate(,,,)**
  - 填充控件的结构
  - 将控件添加到链表底部
- **GOLDraw()**
  - 解析链表
  - 根据状态位设置重新绘制

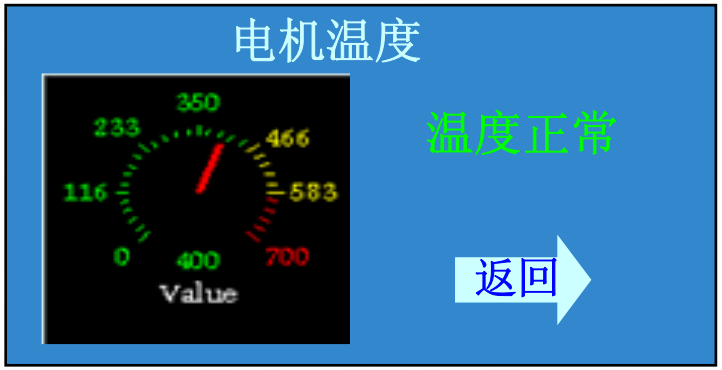
屏幕管理选项
使用一个列表

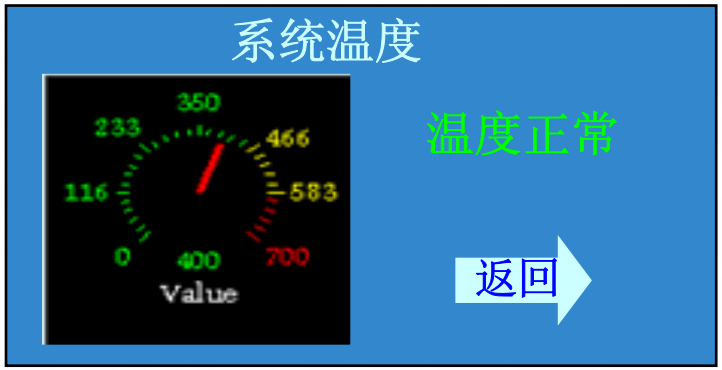
- 隐藏旧控件并绘制新控件
  - 可能增加绘制时间
- 覆盖旧控件
  - 只绘制屏幕的一部分
  - 绘制较快
  - 应禁止底部控件

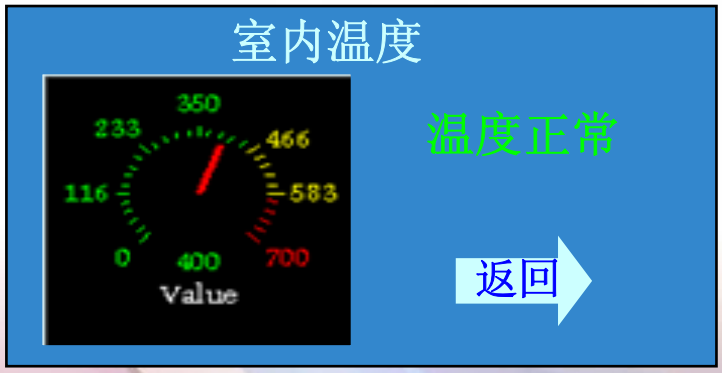
屏幕管理选项
使用多个列表

- 绘制更快
- 每屏幕一个列表
  - 使用GOL管理 API
    - **GOLGetList()**——保存活动列表
    - **GOLNewList()**——设置指针为**null**
    - **GOLSetList(*pList*)**——切换活动列表
  - **ObjCreate(,,,)**用于生成新列表
  - 为所有列表分配堆
    - 示例**GUI**

一个列表/屏幕堆需求







MotorTemp

1个按钮	（1 * 28 = 28字节）
2个静态文本	（2 * 22 = 44字节）
1个仪表	（1 * 40 = 40字节）
共	112字节

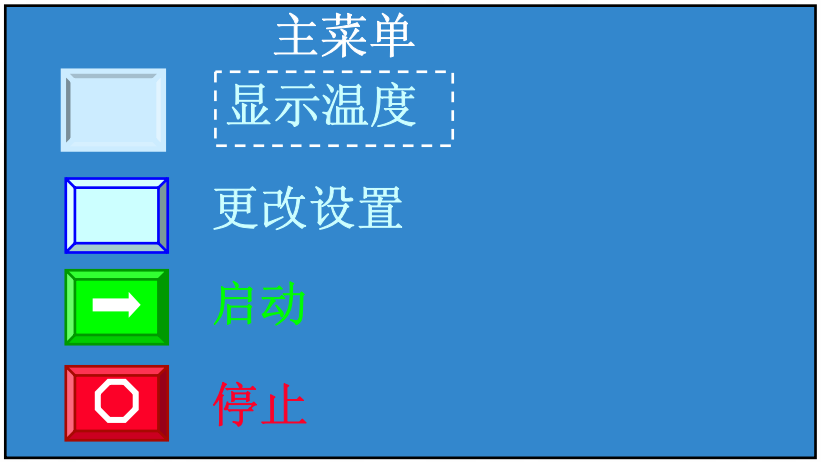
SystemTemp

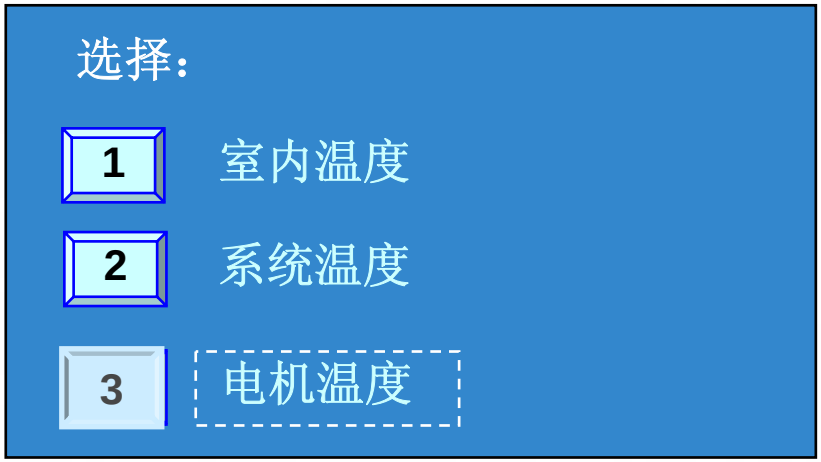
●1个按钮	（1 * 28 = 28字节）
●2个静态文本	（2 * 22 = 44字节）
●1个仪表	（1 * 40 = 40字节）
●	共 112字节

RoomTemp

●1个按钮	（1 * 28 = 28字节）
●2个静态文本	（2 * 22 = 44字节）
●1个仪表	（1 * 40 = 40字节）
●	共 112字节

一个列表/屏幕堆需求





MainMenu

4个按钮	（4 * 28 = 112字节）
5个静态文本	（5 * 22 = 110字节）
共	222字节

TempSelect

3个按钮	（3 * 28 = 84字节）
4个静态文本	（4 * 22 = 88字节）
共	172字节

需要的总堆数

5个屏幕	730字节
3个样式方案	60字节
共	790字节

屏幕管理选项
使用多个列表

- 动态创建列表
  - 使用**GOLFree()**函数从堆中释放存储空间
  - 使用**ObjCreate(,,,)**生成新列表
  - 为最大列表分配堆
    - 示例**GUI：222**字节
  - 存储器效率最高的方式


Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

总结

总结

今天我们学习了如何：

- 应用技巧来协助编写用于图形库的底层驱动程序
- 编写在**LCD**面板上显示图像、字体和原语的程序
- 编写在**LCD**面板上显示和控制控件的程序
- 创建能充分利用**Microchip**图形库的**GUI**应用程序代码

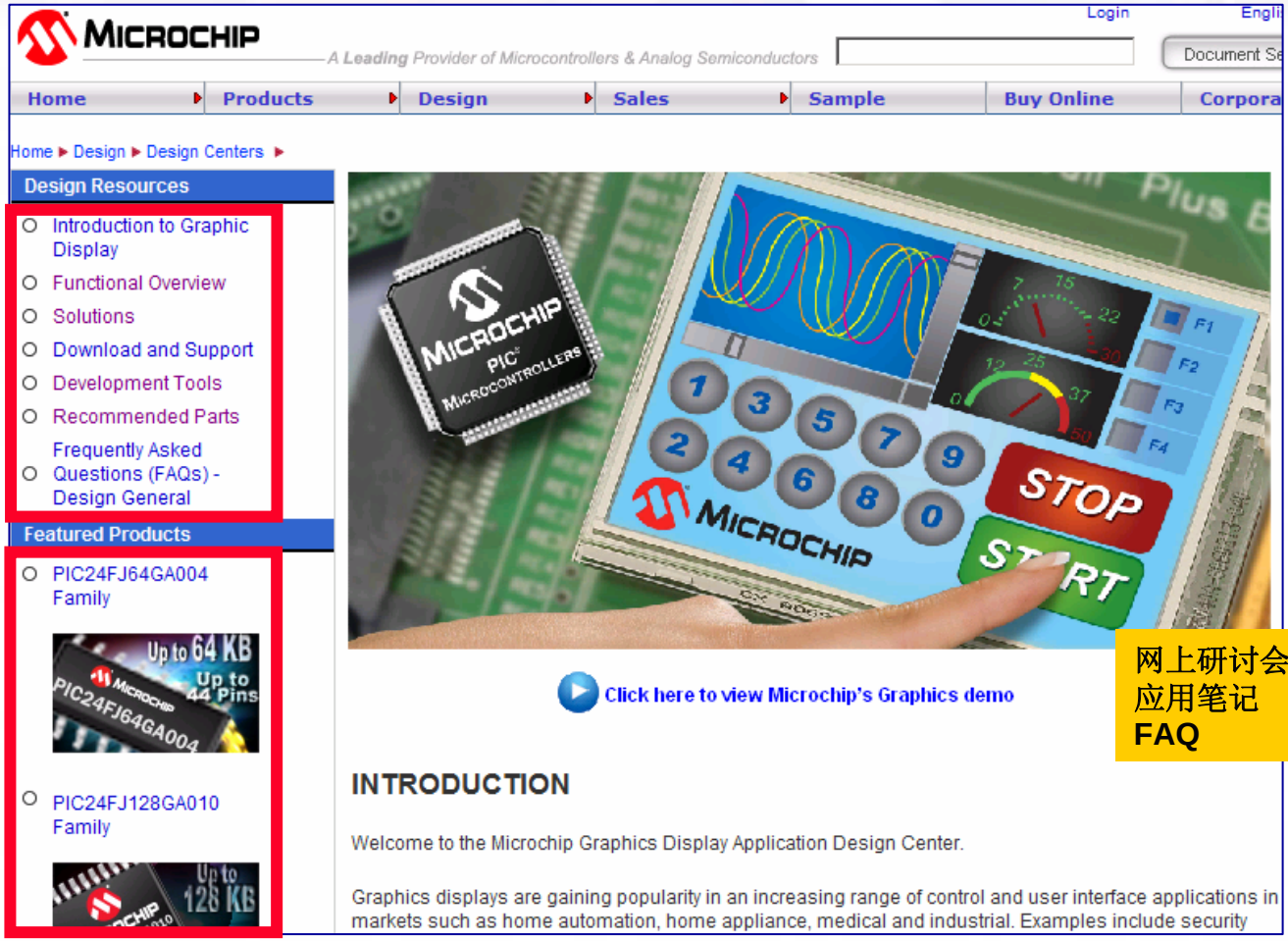
经验提示

- 避免堆不完整
  - 使用GOLFree()
  - 切勿
    - 手动从活动列表删除控件
  - 切勿
    - 修改列表，除非**GOLDraw()**已完成
- 避免不合常理的绘图效果
  - 切勿
    - 修改**ISR**内的绘图属性
    - 在**GOLDraw()**完成前修改控件状态位或样式方案

资源

站点导航

产品页面



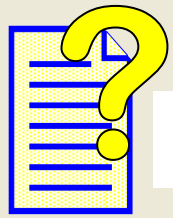
网上研讨会
应用笔记
FAQ

http://www.microchip.com/graphics

资源
库帮助文件

- Introduction
- Software License Agreement
- Release Notes
- Getting Started
- Library Configuration
  - Configuration Setting
  - Input Device Selection
  - Focus Support Selection
  - Graphics Object Selection
  - Font Source Selection
  - Bitmap Source Selection
  - Graphics Mode
  - Hardware Profiles
- Library Structure
- Graphics Object Layer
- Graphics Primitive Layer
- Device Driver Layer
- Miscellaneous Topics
- Files

帮助文件作为**MPLAB**图形库安装的一部分，位于以下目录：

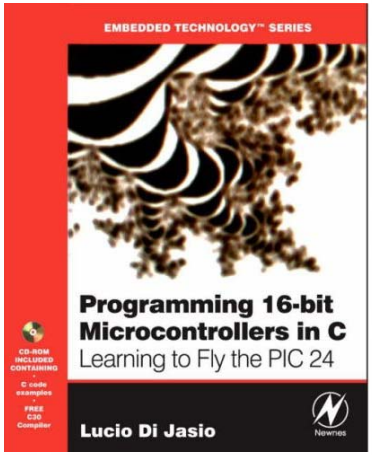


图形库帮助

工具

- **Microchip图形库v1.4**
  - http://www.microchip.com/graphics
- **MPLAB® IDE（v8.10）**
  - http://www.microchip.com/mplab
- **MPLAB C30和C32编译器**
  - http://www.microchip.com/mplabc
- **Color Schemer**
  - http://www.colorschemer.com

推荐读物



Programming 16-bit Microcontrollers in C

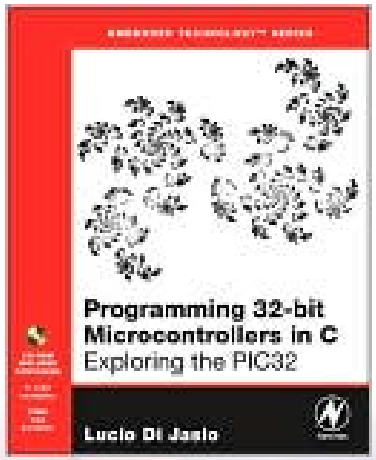
作者：**Lucio Di Jasio**

ISBN-10：0750682922

ISBN-13：978-0750682923

http://www.flyingpic24.com

新的出版物



Programming 32-bit Microcontrollers in C

作者：**Lucio Di Jasio**

ISBN-10：0750687096

ISBN-13：978-0750687096


Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

谢谢！！


Microchip Technology
MASTERS 08
CONFERENCE
YOU + MICROCHIP
ENGINEERING THE FUTURE TOGETHER

补充信息

240RGB x 320 GRAM地址映射示例

	RL=1	S0	S1	S2	S3	S4	S5	S6	S7	S8	…	S714	S715	S716	S717	S718	S719	
	RL=0	S719	S718	S717	S716	S715	S714	S713	S712	S711	…	S5	S4	S3	S2	S1	S0	
	BGR=0	R	G	B	R	G	B	R	G	B	…	R	G	B	R	G	B	Vertical address
	BGR=1	B	G	R	B	G	R	B	G	R	…	B	G	R	B	G	R	
TB=1	TB=0																	
G0	G319	0000H,0000H			0000H, 0001H			0000H, 0010H			…	0000H, 00EEH			0000H, 00EFH			0
G1	G318	0001H,0000H			0001H, 0001H			0001H, 0010H			…	0001H, 00EEH			0001H, 00EFH			1
G2	G317	0010H,0000H			0010H, 0001H			0010H, 0010H			…	0010H, 00EEH			0010H, 00EFH			2
G3	G316	0011H,0000H			0011H, 0001H			0011H, 0010H			…	0011H, 00EEH			0011H, 00EFH			3
G4	G315	0100H,0000H			0100H, 0001H			0100H, 0010H			…	0100H, 00EEH			0100H, 00EFH			4
.	.	.			.			.			.	.			.			.
.	.	.			.			.			.	.			.			.
.	.	.			.			.			.	.			.			.
G316	G3	013CH, 0000H			013CH, 0001H			013CH, 0010H			…	013CH, 00EEH			013CH, 00EFH			316
G317	G2	013DH, 0000H			013DH, 0001H			013DH, 0010H			…	013DH, 00EEH			013DH, 00EFH			317
G318	G1	013EH, 0000H			013EH, 0001H			013EH, 0010H			…	013EH, 00EEH			013EH, 00EFH			318
G319	G0	013FH, 0000H			013FH, 0001H			013FH, 0010H			…	013FH, 00EEH			013FH, 00EFH			319

Horizontal address	0	1	2	…	238	239

176RGB x 132 示例（R61506）

- R61506数据手册中的其他示例…

AD[16:0]	GRAM数据映射
16’h0000 … 16’h00AF	第1行的GRAM数据
16’h0100 … 16’h01AF	第2行的GRAM数据
……	…..
16’h8100 … 16’h81AF	第130行的GRAM数据
16’h8200 … 16’h82AF	第131行的GRAM数据
16’h8300 … 16’h83AF	第132行的GRAM数据

- 仍然是每行256字节

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自**ResetDevice()**，该函数位于**LGDP4531.c**驱动程序中 …

MODE1:MODE0
Mode = 1（对于M68）
Mode = 2（对于180）

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自**ResetDevice()**，该函数位于**LGDP4531.c**驱动程序中 …

WAITB：
数据建立至读/写操作开始时间

WAITM：
读操作至字节使能选通的时间

WAITE：
使能/选通后保持数据的时间

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

编写驱动程序
并行主端口——PMP

- 等待状态计算
  - 取决于驱动器IC写和读选通最小次数

等待状态 = $(T_{MIN} / T_{CY}) - 1$

对运行于**16MIPS**的**PIC 24F**：

T_{CY} = 62.5 ns

来自驱动器**IC**数据手册

T_{WRMIN} = 25 ns => 0 等待状态
T_{RDMIN} = 250 ns => 1.4 等待状态

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自**ResetDevice()**，该函数位于**LGDP4531.c**驱动程序中 …

CSF1:CFS0
片选功能位
0 – PMSC2:1用作Addr15:14
1 – PMCS2 = 片选
PMCS1 = Addr14
2 – PMCS2:1 = 片选

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自**ResetDevice()**，该函数位于**LGDP4531.c**驱动程序中 …

PTRDEN：读/写选通使能
0 = 禁止
1 = 使能

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自**ResetDevice()**，该函数位于**LGDP4531.c**驱动程序中 …

PTWREN：
写使能选通使能
0 = 禁止
1 = 使能

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

编写驱动程序
并行主端口——PMP

- 为LCD系统接口配置PMP
  - PMP模式寄存器（PMMODE）
  - PMP控制寄存器（PMCON）
- 示例来自 **ResetDevice()**，该函数位于 **LGDP4531.c**驱动程序中 **…**

PMPEN：PMP使能
0 = 禁止
1 = 使能

```
// PMP设置
   PMMODE = 0;
   PMAEN = 0;
   PMCON = 0;
   PMMODEbits.MODE  = 2;
   PMMODEbits.WAITB = 0;
   PMMODEbits.WAITM = 1;
   PMMODEbits.WAITE = 0;
   PMCONbits.CSF    = 0;
   PMCONbits.PTRDEN = 1;
   PMCONbits.PTWREN = 1;
   PMCONbits.PMPEN  = 1;
```

商标

Microchip的名称和徽标组合、Microchip徽标、Accuron、dsPIC、KeeLoq、KeeLoq徽标、MPLAB、PIC、PICmicro、PICSTART、rfPIC和SmartShunt均为Microchip Technology Inc.在美国和其他国家或地区的注册商标。

FilterLab、Linear Active Thermistor、MXDEV、MXLAB、SEEVAL、SmartSensor和The Embedded Control Solutions Company 均为Microchip Technology Inc.在美国的注册商标。

Analog-for-the-Digital Age、Application Maestro、CodeGuard、dsPICDEM、dsPICDEM.net、dsPICworks、dsSPEAK、ECAN、ECONOMONITOR、FanSense、In-Circuit Serial Programming、ICSP、ICEPIC、Mindi、MiWi、MPASM、MPLAB Certified徽标、MPLIB、MPLINK、mTouch、PICkit、PICDEM、PICDEM.net、PICtail、PIC32徽标、PowerCal、PowerInfo、PowerMate、PowerTool、REAL ICE、rfLAB、Select Mode、Total Endurance、UNI/O、WiperLock和ZENA均为Microchip Technology Inc.在美国和其他国家或地区的商标。

SQTP是Microchip Technology Inc.在美国的服务标记。

在此提及的所有其他商标均为各持有公司所有。