OSD Engine Design

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修改人 | 日期 | 修改内容 |
| 0.1 | 陈长虹 | 2018/3/8 | 初始版本 |
| 0.2 | 陈长虹 | 2018/3/21 | 添加了OSD Memory部分  去掉了Modifier部分,使用软件实现更简单灵活  去掉Circle, rotate等硬件比较难实现的部分 |
| 0.3 | 陈长虹 | 2018/3/30 | 修改了BITMAP部分,增加了平铺功能  增加了有MASK的BITMAP功能,BITMAP位图中只存放Alpha值,然后将mask\_color与背景色blending  添加了bitmap\_data数据结构 |

目录

[1. Overview 3](#_Toc510184186)

[1.1 OSD Definition 3](#_Toc510184187)

[1.2 Entity relationship 4](#_Toc510184188)

[1.3 Hardware memory layout 4](#_Toc510184189)

[1.4 Features 5](#_Toc510184190)

[1.5 Design consideration 5](#_Toc510184191)

[1.6 Data path 6](#_Toc510184192)

[2. Entity Attribute Definition 6](#_Toc510184193)

[Scene(场景) 6](#_Toc510184194)

[Palette(调色板) 7](#_Toc510184195)

[Window(窗口) 7](#_Toc510184196)

[Block(绘图块) 8](#_Toc510184197)

[Ingredient(素材) 8](#_Toc510184198)

[Bitmap(位图素材) 9](#_Toc510184199)

[Character(字符素材) 9](#_Toc510184200)

[Rectangle(矩形素材) 10](#_Toc510184201)

[Line(线段素材) 11](#_Toc510184202)

[Glyph(字符点阵位图) 12](#_Toc510184203)

[bitmap\_data(位图数据) 12](#_Toc510184204)

[3. Software Design 13](#_Toc510184205)

[Software Archtecture 13](#_Toc510184206)

[4. OSD File Memory Layout 15](#_Toc510184207)

[5. 其他设计 17](#_Toc510184208)

[Multi Palette 17](#_Toc510184209)

[Color index compress 17](#_Toc510184210)

[Fixed Palette 17](#_Toc510184211)

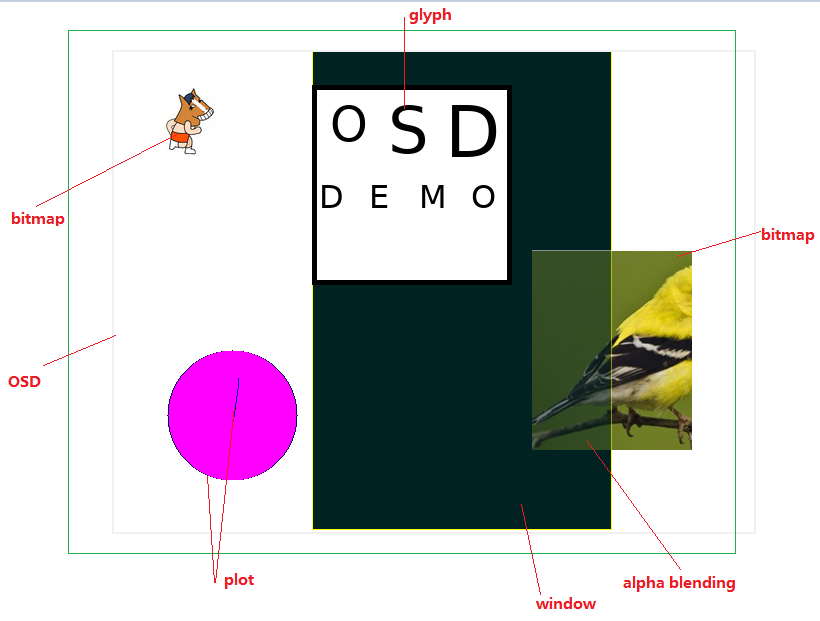
[Window分类 17](#_Toc510184212)

[WINDOW中快速定位BLOCK问题 18](#_Toc510184213)

[半透明OSD 18](#_Toc510184214)

# Overview

## OSD Definition



OSD: 屏幕上面的显示

Window: 窗口系统,一个独立的绘图范围,多个窗口可以叠加,Blending

Palette: 调色板,系统所有的颜色都是调色板的索引

Block: 窗口内部的一个显示单元, Block= Ingredient + position(x, y)

Ingredient:绘图素材,绘制到Window上的基本单元,分为以下几种

Bitmap: 一副或者多副位图, 默认显示第一张

Character: 某个字对应的位图 = Glyph + Color

Rectangle:左上角坐标 + 右下角坐标

Line: Plot子类型,起始点 + 终止点

Glyph: 一个字符的点阵图

对象定义如下

OSD := Scene \* n

Scene := Window \* n + Palette \* m + Glyph \* p

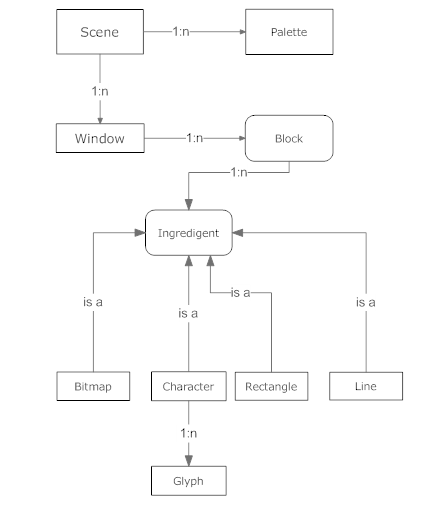
Window := Block \* n

Block := Ingredient + block\_x + block\_y(block\_x, block\_y指某个素材在Window中的相对位置,主要方便移动window时候,素材相对位置不用修改)

Ingredient := <Bitmap | Character | Rectangle | Line>

Character := Glyph + Color

## Entity relationship



## Hardware memory layout

* 软件产生数据结构
* 硬件读取数据结构以行为单位画图



## Features

* 多窗口管理
* 字体管理
* Bitmap绘制
* 绘图单元Rectangle, Line引入, 通过动态作图避免存储位图数据,节省内存

## Design consideration

* + 1. No Frame Buffer
    2. 尽量少的内存占用
    3. 硬件容易实现,效率高
    4. 主要的软件工作尽量在设计OSD的时候在Tool里面完成,执行时Firmware尽量少参与
    5. 软件Model与硬件同步, 容易验证硬件功能
    6. 考虑以后的存在Frame Buffer的设计, 软件工具和数据保持不变

## Data path



# Entity Attribute Definition

## Scene(场景)

表示一个OSD场景,系统中可能有多个Scene, 比如搜台作为一个Scene, 工厂菜单等等, 可以类比为windows程序的一个窗口, 系统的RAM应该能够放下最大的一个Scene.

一个时刻只有一个Scene被加载到RAM中,其他的Scene可以放到Flash或DDR中

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| width, height | UINT16 | OSD大小 |
| palette\_count | UINT8 | 调色板数量 |
| window\_count | UINT8 | 窗口数量 |
| ingredient\_count | UINT16 | 素材数量 |
| glyph\_count | UINT16 | 字符数量 |
| struct \_osd\_scene\_hw {  u16 width, height;  u32 ram\_offset;  u8 palette\_data\_size;  u8 ingredient\_data\_size;  u8 window\_data\_size;  u8 glyph\_header\_size;  u8 palette\_count;  u8 window\_count;  u16 ingredient\_count;  char title[12]; //3 bytes  u16 timer\_ms; //0 means no timer  u16 glyph\_count;  }; | | |

## Palette(调色板)

OSD的所有颜色都存放在查找表, 使用的颜色需要使用调色板来获取真正的RGB值

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| pixel\_format | UINT8 | 调色板类型,目前为0 |
| pixel\_bits | UINT8 | 1,2,4,8,使用当前调色板几个bit表示一个点 |
| entry\_count | UINT16 | 颜色数量 |
| luts\_addr | UINT32 | 查找表在RAM中的地址 |
| struct \_osd\_palette\_hw {  u8 pixel\_format;  u8 pixel\_bits; // 0, 1, 2, 4, 8, 16  u16 entry\_count;  u32 luts\_addr;  }; | | |

## Window(窗口)

表示一个窗口, 内部成员的坐标都是相对于窗口的坐标.

Window之间可以层叠,有Z-Order,可以用Alpha做Blending

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| palette\_index | UINT8 | 调色板编号 |
| visible | UINT8 | 是否显示 |
| alpha | UINT8 | Alpha值, FF为不透明 |
| zorder | UINT8 | 窗口的层叠关系,大数代表在上面,最后画 |
| x, y | UINT16 | 所在位置 |
| width, height | UINT16 | 大小 |
| block\_count | UINT32 | 窗口的BLOCK数量 |
| block\_addr | UINT32 | BLOCK数组所在的RAM地址 |
| struct \_osd\_window\_hw {  u8 palette\_index;  u8 visible;  u8 alpha;  u8 z\_order;  u16 x, y;  u16 width, height;  u32 block\_count;  u32 blocks\_addr;  }; | | |

## Block(绘图块)

表示窗口内部的一个显示单元块, 由Ingredient + 所在Window位置(X,Y)组成

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| visible | 1BIT | 是否显示 |
| reserved | 7BIT |  |
| ingredient\_index | UINT16 | 素材编号,可能指向任何一种素材对象 |
| x, y | UINT16 | 在WINDOW内部的位置 |
| struct \_osd\_block {  u16 visible: 1; //lowest bits  u16 reserved: 15;  u16 ingredient\_index;  u16 x;  u16 y;  }; | | |

## Ingredient(素材)

绘图的基本单元,目前有四种素材,每个素材的前面32BIT数据结构相同,每种素材数据结构大小是相同的,可以根据索引方便找到内存中的位置

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| Type | UINT8 | 素材类型 1: 矩形, 2:线,3:字符,4:位图 |
| palette\_index | UINT8 | 调色板编号,如果设置为0xFF则使用所在WINDOW的palette\_index |
| Reserved | UINT16 |  |
| struct \_osd\_ingredient\_hw {  u8 type; //OSD\_INGREDIENT\_XXX  u8 palette\_index;  u16 flags;  union {  osd\_rectangle\_hw rectangle;  osd\_line\_hw line;  osd\_bitmap\_hw bitmap;  osd\_character\_hw character;  osd\_label\_hw label;  } data;  }; | | |

## Bitmap(位图素材)

图片对象, 可以包含一张或者多张图片,一个时刻只显示其中一幅图,通过切换显示的图来实现动画

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| type | UINT8 | 素材类型 1: 矩形, 2:线,3:字符,4:位图 |
| palette\_index | UINT8 | 调色板编号,如果设置为0xFF则使用所在WINDOW的palette\_index |
| reserved | UINT16 |  |
| tiled | UINT8 | 是否为平铺,为0时只显示一副,为1时平铺显示 |
| mask\_color | UINT8 | MASK颜色, 为0无效  其他值时,从palette查到颜色,然后bitmap data中存放8bit的灰度值 |
| bitmap\_count | UINT8 | 图片数量 |
| current\_bitmap | UINT8 | 当前要显示的图片 |
| data\_addr | UINT32 | 位图数据所在RAM,指向osd\_bitmap\_data |
| struct \_osd\_bitmap\_hw {  u16 width;  u16 height;  u8 tiled;  u8 mask\_color;  u8 bitmap\_count;  u8 current\_bitmap;  u32 data\_addr; //pointer to osd\_bitmap\_data  }; | | |

## Character(字符素材)

一个显示的字符, 字符素材 = 字符点阵(Glyph) + 颜色(Color), 这样某个字符多个颜色只用存一个点阵

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| type | UINT8 | 素材类型 1: 矩形, 2:线,3:字符,4:位图 |
| palette\_index | UINT8 | 调色板编号,如果设置为0xFF则使用所在WINDOW的palette\_index |
| reserved | UINT16 |  |
| color | UINT16 | 位图尺寸 |
| Reserved1 | UINT8 |  |
| Reserved2 | UINT16 |  |
| glyph\_addr | UINT32 | 字符点阵数据结构的地址,指向osd\_glyph |
| Reserved3 | UINT32 |  |
| struct \_osd\_character\_hw {  u8 color;  u8 reserved1;  u16 reserved2;  u32 glyph\_addr;  u32 reserved3;  }; | | |

## Rectangle(矩形素材)

一个矩形

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| type | UINT8 | 素材类型 1: 矩形, 2:线,3:字符,4:位图 |
| palette\_index | UINT8 | 调色板编号,如果设置为0xFF则使用所在WINDOW的palette\_index |
| reserved | UINT16 |  |
| width, height | UINT16 | 矩形宽度高度 |
| border\_color | UINT8 \* 4 | 上下左右边框的颜色 |
| border\_weight | UINT8 | 边框宽度 |
| Border\_style | 4BIT | 边框线段类型  #define OSD\_LINE\_STYLE\_SOLID 1  #define OSD\_LINE\_STYLE\_DASH 2  #define OSD\_LINE\_STYLE\_DOT1 3  #define OSD\_LINE\_STYLE\_DOT2 4  #define OSD\_LINE\_STYLE\_DOT3 5  #define OSD\_LINE\_STYLE\_DASH\_DOT 6  #define OSD\_LINE\_STYLE\_DASH\_DOT\_DOT 7 |
| Gradient\_mode | 4BIT | 渐变色类型  #define OSD\_GRADIENT\_MODE\_SOLID 1  #define OSD\_GRADIENT\_MODE\_LEFT\_TO\_RIGHT 2  #define OSD\_GRADIENT\_MODE\_TOP\_TO\_BOTTOM 3  #define OSD\_GRADIENT\_MODE\_TOP\_LEFT\_TO\_BOTTOM\_RIGHT 4  #define OSD\_GRADIENT\_MODE\_BOTTOM\_LEFT\_TO\_TOP\_RIGHT 5  #define OSD\_GRADIENT\_MODE\_CORNER\_TO\_CENTER 6 |
| bgcolor\_start, bgcolor\_end | UINT8 \* 2 | 矩形背景色开始颜色,结束颜色 |
| struct \_osd\_rectangle\_hw {  u16 width;  u16 height;  u8 border\_color\_top, border\_color\_bottom;  u8 border\_color\_left, border\_color\_right;  u8 border\_weight;  u8 border\_style: 4; //OSD\_LINE\_STYLE\_XXX, lower 4 bit  u8 gradient\_mode:4; //OSD\_GRADIENT\_MODE\_TOP\_TO\_BOTTOM\_XXX, higher 4 bit  u8 bgcolor\_start, bgcolor\_end;  }; | | |

## Line(线段素材)

一条线

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| type | UINT8 | 素材类型 1: 矩形, 2:线,3:字符,4:位图 |
| palette\_index | UINT8 | 调色板编号,如果设置为0xFF则使用所在WINDOW的palette\_index |
| reserved | UINT16 |  |
| X1,Y1,X2,Y2 | UINT16 \* 4 | 线段开始结束点 |
| weight | UINT8 | 宽度 |
| style | UINT8 | 线段类型  #define OSD\_LINE\_STYLE\_SOLID 1  #define OSD\_LINE\_STYLE\_DASH 2  #define OSD\_LINE\_STYLE\_DOT1 3  #define OSD\_LINE\_STYLE\_DOT2 4  #define OSD\_LINE\_STYLE\_DOT3 5  #define OSD\_LINE\_STYLE\_DASH\_DOT 6  #define OSD\_LINE\_STYLE\_DASH\_DOT\_DOT 7 |
| color | UINT8 | 线段颜色 |
| Reserved | UINT8 |  |
| struct \_osd\_line\_hw {  u16 x1, y1;  u16 x2, y2;  u8 weight;  u8 style; //OSD\_LINE\_STYLE\_XXX  u8 color;  u8 reserved2;  }; | | |

## Glyph(字符点阵位图)

表示一个字符的点阵信息

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| left, top | UINT8 | 字符位图起始画点位置  一般来说字符位图大小<字符大小,旁边会有空白间距 |
| width,height | Width, height | 字符位图宽度高度 |
| char\_code | UINT16 | 字符Unicode编码,软件使用 |
| font\_id | UINT8 | 字体编号,软件使用 |
| font\_size | UINT8 | 字号,软件使用 |
| pitch | UINT8 | 每行点阵使用的byte数 |
| advance\_x | UINT8 | 画下一个字X坐标移动的距离 |
| data\_size | 15BIT | 点阵数据大小 |
| monochrome | 1BIT | 是否为单色位图  单色位图时每个BIT表示一个点  不是单色位图时8BIT表示一个点的Alpha值,用于表示抗锯齿字 |
| struct \_osd\_glyph {  u8 left, top;  u8 width, height;  u16 char\_code;  u8 font\_id;  u8 font\_size;  u8 pitch;  u8 advance\_x;  u16 data\_size:15;  u16 monochrome:1;  };  内存中接着就是字体的点阵数据 | | |

## Bitmap\_data(位图数据)

表示一个字符的点阵信息

|  |  |  |
| --- | --- | --- |
| 属性/操作 | 类型 | 说明 |
| Bitmap\_width, bitmap\_height | UINT16 | 位图数据宽度和高度 |
| data\_size | UINT32 | 位图数据大小 |
| struct \_osd\_bitmap\_data {  u16 bitmap\_width;  u16 bitmap\_height;  u32 data\_size;  };  内存中接着就是位图的颜色数据 | | |

备注: 为了软件编程方便,目前给的尾款都比较大,这个可以根据实际需求缩小.比如glyph\_addr目前是UINT32,如果确认RAM SIZE < 64K, 那么可以用UINT16来表示

# Software Design

## Software Archtecture



|  |  |
| --- | --- |
| OML | OSD Markup Language  描述OSD数据的文本格式，采用 YAML或者XML格式 |
| GUI Tool | 图形编辑工具，用于生成OSD描述文件(OML) |
| Text Editor | 可以采用普通文本编辑工具来手动编辑OML文件 |
| OML2BIN | 命令行工具，将OSD YAML文件转换为寄存器描述文件+存放在Flash中的OSD Data文件 |
| Register File | OSD Engine寄存器配置，主要是Window的配置信息 |
| OSD Data File | OSD Data所有资源的数据信息,将OML里面描述的OSD转换为二进制格式，供Firmware和HW读取，可能存放在Flash或者RAM中 |
| OSD Simulation | OSD模拟器，运行在Windows上面，可以根据OSD数据模拟最终的运行效果，可以用于RTL的验证和OSD的调试 |

# OSD File Memory Layout

|  |  |  |  |
| --- | --- | --- | --- |
| **Part** | **Data structure** | **Count** | **Unit**  **size** |
| Scene Header | struct \_osd\_scene\_hw {  u16 width, height;  u32 ram\_offset;  u8 palette\_data\_size;  u8 ingredient\_data\_size;  u8 window\_data\_size;  u8 glyph\_header\_size;  u8 palette\_count;  u8 window\_count;  u16 ingredient\_count;  char title[12]; //3 bytes  u16 timer\_ms; //0 means no timer  u16 glyph\_count;  }; | 1 | 32B |
| Palettes Header | struct \_osd\_palette\_hw {  u8 pixel\_format;  u8 pixel\_bits;  u16 entry\_count;  u32 luts\_addr;  }; | palette\_count | 20B |
| Ingredients Header | struct \_osd\_ingredient\_hw {  u8 type;  u8 palette\_index;  u16 flags;  union {  osd\_rectangle\_hw rectangle;  osd\_line\_hw line;  osd\_bitmap\_hw bitmap;  osd\_character\_hw character;  osd\_label\_hw label;  } data;  }; | ingredient\_count | 16B |
| Window header | struct \_osd\_window\_hw {  u8 palette\_index;  u8 visible;  u8 alpha;  u8 z\_order;  u16 x, y;  u16 width, height;  u32 block\_count;  u32 blocks\_addr;  }; | window\_count | 16B |
| Glyph binary | struct \_osd\_glyph {  u8 left, top;  u8 width, height;  u16 char\_code;  u8 font\_id;  u8 font\_size;  u8 pitch;  u8 advance\_x;  u16 data\_size:15;  u16 monochrome:1;  };  u8 glyph\_data[data\_size]; | variant | 12 + data\_size |
| Palette binary | binary | 4 | entry\_count |
| Ingredient binary | binary | variant | Depends on type |
| Window binary | struct \_osd\_block {  u16 visible: 1;  u16 reserved: 15;  u16 ingredient\_index;  u16 x;  u16 y;  }; | block\_count | 4 |

# 其他设计

## Multi Palette

采用多个Palette的方式, 可以减少每个color所需要存储的颜色位数

比如图片1(100 x 100)有16种颜色, 图片2(100 x 100)有200种颜色, 此时若采用同一个Palette, 都需要用8BIT来存放数据一个Pixel 值, 那么需要使用的BYTE数目为100 \* 100 + 100 \* 100 = 200000 bytes.

如果图片1采用一个4BIT的Palette, 图片2采用8bit的Palette, 则需要的Byte数据为(4 \* 100 \* 100)/8 + 100 \* 100 = 150000 bytes

## Color index compress

可以考虑对较大的BITMAP进行无损压缩,可以采用如下压缩方式,在工具中计算出最优的压缩方式,然后保存到图片数据中. 数据头需要保存每行的BYTE数, 用于在绘图的时候快速定位到某一行所在的BYTE. 在BITMAP数据转换查找LUT之前,先进行解压缩.

1. Huffman Code
2. RLE算法

需要考虑硬件解压缩的成本

## Fixed Palette

可以将常用的Palette用运算电路表示,避免存放所有RGB值, 比如灰度, 还有常用的颜色表

## Window分类

为了硬件设计方便, 可以将window分为两类, 一类window只能画字符和Bitmap, 这种window可以限制为4-8个, 第二类window值可以画矩形和线段,这种可以为16-32个

第一类Window的RAM总数应该>=32K, 每一个Window的内存可以通过寄存器配置, 比如window0使用0-8K, window1使用9-16K,不得重叠

第二类window由于占用RAM较小,可以使用固定分配,比如每个分配是1K, 512字节等等

这样设计主要考虑到硬件在同一个周期里只能对一段RAM进行访问,如果有两个字符叠加绘制会不能满足需求。这时候采用两个WINDOW对应自己的RAM，这样就不会产生冲突。

所有素材都与Window相关联,比如字符A需要在Window0和Window1中都要画出来,就得存放两次在两块WINDOW各自的RAM中。

Window之间可以做层叠和Alpha Blending.

## WINDOW中快速定位BLOCK问题

为了硬件实现简单,可以规定在同一个Window中所有素材不得重叠,也就是说,一个WINDOW中,一个点最多对应一个素材, 这样比较容易算出当前点使用的素材.

这样我们素材在WINDOW中的位置可以用相对位置来表示

比如在一个(300x200)的Window中, 要画以下素材

(10,10)的位置画一个(8x8)的字符A

(30,20)的位置画一个(10x10)的字符B

(0,100)的位置画一个(30,30)的位图C

在RAM中可以这样表示

(+10,+10, A), (+20, +10, B), (-30,+80, C), 这样硬件定位起来比较容易.

## 半透明OSD

最后绘制出来的的LineBuffer中每一个点应该是有Alpha参数的,需要与Video做最后的Blending, 用于制作半透明的OSD