我们知道，一块显卡一般都会提供一个或多个输出接口。为了利用我们手中的显卡，我们都希望尽可能多的让所有的输出接口都工作。而对于AMD显卡而言，完美驱动所有输出接口的关键当然是一个合适的Framebuffer。本文将向您阐述AMD的Framebuffer每个字段的详细定义。

一、什么是Framebuffer？

Framebuffer即帧缓冲。它是Unix/Linux为显示设备提供的一个接口，把显存抽象后的一种设备，他允许上层应用程序在图形模式下直接对显示缓冲区进行读写操作。这种操作是抽象的，统一的。用户不必关心物理显存的位置、换页机制等等具体细节。这些都是由Framebuffer设备驱动来完成的。

Mac OS X的AMD显卡kext中定义的Framebuffer只适用苹果设备所采用的AMD显卡。我们修改Framebuffer的数值，就是为了让Mac的显卡驱动能够完美驱动我们的显卡。越来越多的黑苹果用户，在采用了Chameleon(变色龙)的GraphicsEnabler=Yes参数或者Clover(四叶草)的InjectATI=Yes之后，某个显卡输出接口就正常工作了，便把自己的显卡称为免驱卡，这是不正确的。而真正所有的接口都能驱动的显卡，也是所谓的免驱卡，除了苹果原装显卡之外，数量微乎其微。

二、基础知识

1. 输出接口种类

LVDS	一般笔记本的LCD显示屏幕一般采用的这种接口。如果您使用笔记本电脑，则很有可能内置一个这样的接口。	暂无图片
DVI	DVI接口根据通道数量分为单通道(Single-link DVI, 简写SDVI)和双通道(Dual-link DVI, 简写DDVI)。同时，根据信号传输方式的不同，分为DVI-D(Digital数字信号)，DVI-A(Analog模拟信号)和DVI-I(Integrated混合式)三种，现在常见的是DVI-D和DVI-I。不同的通道数，不同的信号传输方式，对应的接口也不一样，您可以通过接口的外观来判断接口的类型。	
VGA	较为古老的接口，有些电脑仍然有配备这种接口。在笔记本电脑上，可能没有左右两边的固定螺丝孔位。	
HDMI	是一种新型接口，用来传输高清视频和音频信号。	
DP	DisplayPort是一种正在发展的新型接口。其中Mini DisplayPort被苹果设备使用。甚至有些笔记本电脑也采用了eDP作为内屏。	

2. 字节序

在计算机中，1个字节(byte)由8个比特位(bit)组成，所以我们可以用一个两位的十六进制数来表示一个

字节，十六进制数的前缀用"0x"表示。为了方便表示数据的存储，我们对每个字节进行编址，规定从左到右，地址逐渐增大。假设内存中有这么一串序列：02 08 1F 7A，最左边字节的地址最低，最右边的地址最高。假设最左边的地址为0x12，那么最右边的地址为0x12+0x3=0x15。

Mac OS X中，字节序采用的小端模式(Little-Endian)，即低地址储存的数据存放低位，高地址储存的数据存高位。我们把02 08 1F 7A转化成相应的十六进制数，就应该是0x7A1F0802。在本文的后面，需要用到这种转化。

3. 显卡显示原理

显卡卡在渲染好一个显示结果后,把渲染结果传递给编码器(encoder)进行编码，然后传递给发射器(transmitter)，发射器再把信号发送到不同的接口(port)上，然后接口通过数据线连接到显示器，我们就可以看到画面了。这其中编码器(encoder) 负责是把渲染结果转变成显示设备可以读懂的信号或者说编码，显示器才能显示出我们看到的色彩斑斓的画面。如果采用了错误的编码器编码渲染结果，显示器在解码时就会产生花屏。同样的，如果有两个不同的编码器同时发送信号到同一个显示器，也可能产生花屏。如果发射器发射信号到错误的显示器，就会造成期望使用的显示器没有接受的信号，就会出现黑屏。

三、Framebuffer结构

我们来看一个例子，这个Framebuffer是来自AMD7000Controller.kext中的Aji，最后的4行是从kext获取的原始数据。

Aji (4) DP, DP, DDVI, HDMI 00 04 00 00 04 03 00 00 00 01 01 01 12 04 05 01 00 04 00 00 04 03 00 00 00 01 02 01 22 05 04 02 04 00 00 00 14 02 00 00 00 01 03 00 00 00 03 06 00 08 00 00 04 02 00 00 00 01 04 00 11 02 01 04	Framebuffer名字 (接口数量) 所有接口的类型 接口1的属性值 接口2的属性值 接口3的属性值 接口4的属性值

我们针对原始数值进行划分：

00 04 00 00 00 04 00 00 04 00 00 00 00 08 00 00	04 03 00 00 04 03 00 00 14 02 00 00 04 02 00 00	00 01 00 01 00 01 00 01	01 01 02 01 03 00 04 00	12 22 00 11	04 05 00 02	05 04 03 01	01 02 06 04
接口类型 (ConnectorType)	控制标志 (ControlFlags)	显示特征 (Features)	占位符 (Placeholder)	发射器 (Transmitter)	解码器 (Encoder)	热拔插ID (hotplugID)	检测ID (senseID)

四、详细定义

对于原始数值的解释，仍然有不确定的地方，有些数值是确定的，而有些数值则是根据大量的分析猜测出来的。

1. 接口类型(ConnectorType)

对于这一段数值，已经可以确定了，对于每种接口，都是固定值。

```
#define CONNECTORTYPE_LVDS     0x00000002
#define CONNECTORTYPE_DVI_DUAL       0x00000004
#define CONNECTORTYPE_VGA       0x00000010
#define CONNECTORTYPE_SVIDEO     0x00000080
#define CONNECTORTYPE_DVI     0x00000200
#define CONNECTORTYPE_DP       0x00000400
#define CONNECTORTYPE_HDMI     0x00000800
```

所以，可以转化一下，得到相应的数值。

LVDS	02 00 00 00
DDVI	04 00 00 00
VGA	10 00 00 00
S-VIDEO	80 00 00 00
SDVI	00 02 00 00
DP	00 04 00 00
HDMI	00 08 00 00

2. 控制标志(ControlFlags)

对于每种类型，都有很多可能值：

```
LVDS        >   ControlFlag：0x0040 / 0x0100
DDVI        >   ControlFlag：0x0014 / 0x0214 / 0x0204 / 0x0016
VGA         >   ControlFlag：0x0010
S-VIDEO     >   ControlFlag：0x0002
SDVI        >   ControlFlag：0x0014 / 0x0214
DP          >   ControlFlag：0x0304 / 0x0604 / 0x0104 / 0x0204 / 0x0704 / 0x0314/ 0x0100
HDMI        >   ControlFlag：0x0204
```

对于这一段数值，不能完全确定。通过对所有Framebuffer进行分析和统计，并结合实际的驱动情况，我们可以作出如下推测。

LVDS	40 00 00 00 (默认)
DDVI	14 00 00 00 (DVI-D) 14 02 00 00 (DVI-I) 04 02 00 00 (DVI-I?有使用该数值驱动的例子)
VGA	10 00 00 00 (默认)

S-VIDEO	02 00 00 00 (默认)
SDVI	14 00 00 00 (DVI-D) 14 02 00 00 (DVI-I)
DP	04 03 00 00 (高清接口duallink = 0x2) 04 06 00 00 (普通接口duallink = 0x1)
HDMI	04 02 00 00 (默认)

3. 显示特征(Features)

前一个字节的值的定义：

```
#define FEATURE_USE_INTERNAL            0x01 /*内置*/
#define FEATURE_USE_RGB_ON_YUV          0x04
#define FEATURE_USE_BACKLIGHT           0x08 /*背光*/
#define FEATURE_BACKLIGHT_INVERTED      0x10
#define FEATURE_USE_CLAMSHELL           0x20
```

后一个字节中标志了是否支持音频传输，0x71为支持，0x01为不支持。所以该字段的值基本是确定的。

LVDS	09 01
DDVI	00 01
VGA	00 01
S-VIDEO	04 01
SDVI	00 01
DP	00 71
HDMI	00 71

4. 占位符(Placeholder)

这一段字符是未知的。在AMD4000，5000，6000中，这一段字符一般为00 00，所以原来被认为是编译时产生的对齐字符；而对于AMD6000+，这一段字符往往有值，前一字节可能是接口的某种顺序。一种建议的做法是，如果显卡是7系卡或更高版本，建议不要更改该字段，以防止出现问题。

5. 发射器(Transmitter, 简写txmit)

txmit值可以从显卡ROM中获取。具体定义如下：

```
/* Transmitter 低4位 (TransmitterID) */
#define UNIPHY     0x00 // 发射器使用的是UniPhy0号
#define UNIPHY1    0x01 // 发射器使用的是UniPhy1号
#define UNIPHY2    0x02 // 发射器使用的是UniPhy2号
```

```
/* Transmitter 高4位 (LinkID) */
#define DUALLINK  0x00 // 发射器同时使用线路A, B
#define LINKA      0x10// 发射器使用线路A
#define LINKB      0x20// 发射器使用线路B

/* Transmitter 整个字节  */
#define UNIPHYA      0x10 // = UNIPHY:LINKA
#define UNIPHYB      0x20 // = UNIPHY:LINKB
#define UNIPHYAB   0x00 // = UNIPHY:DUALLINK
#define UNIPHYC      0x11 // = UNIPHY1:LINKA
#define UNIPHYD      0x21 // = UNIPHY1:LINKB
#define UNIPHYCD   0x01 // = UNIPHY1:DUALLINK
#define UNIPHYE      0x12 // = UNIPHY2:LINKA
#define UNIPHYF      0x22 // = UNIPHY2:LINKB
#define UNIPHYEF     0x02 // = UNIPHY2:DUALLINK
#define DACA         0x00 // 双路支持的数字转模拟
#define DACB         0x10 // 数字转模拟
```

6. 解码器(Encoder, 简写enc)

enc值可以从显卡ROM中获取。具体定义如下：

```
/* Encoder 低4位 (DIG_ID：数字信号) */
#define DIG1   0x00 // = DIGA
#define DIG2   0x01 // = DIGB
#define DIG3   0x02 // = DIGC  仅针对5系卡及以后的版本
#define DIG4   0x03 // = DIGD  仅针对5系卡及以后的版本
#define DIG5   0x04 // = DIGE  仅针对5系卡及以后的版本
#define DIG6   0x05 // = DIGF  仅针对5系卡及以后的版本

/* Encoder Bits 高4位 (DAC_ID：模拟信号) */
#define DAC      0x10
```

7. 热拔插ID(HotplugID)

在过去的教程里，我们都是重新编排该字段。笔记本内屏LVDS为00，其它的从01开始依次递增。

而在新的7系卡以及更高的版本中，自定义该字段可能会导致问题。建议7系卡及更新的显卡，尽量保持原始值，只有在出现了显示问题后，再来调整该字段的值。

8. 检测ID(SenseID)

senseID值可以从显卡ROM中获取。具体定义如下：

计算公式：SenseID = (i2cid & 0xf) +1字段定义：

Bits 0-3: Sense Line

Bit 4: Use hw i2c flag

五、参考资料及引用

从零开始完美玩转苹果ATI驱动+QE/CI+多屏,理论上所有A卡可行,以4860(RV790GT)为例

ATI 5系和6系显卡驱动&修改FB探讨

Mobility Radeon HD 4650:Full Resolution with QE & CI working on Internal LVDS screen