APPLICATION NOTE

Using the RGB624 Palette DAC Tagged Pixel Format

by Kirk Haskell

Introduction

The RGB family of Palette DACs from IBM Microelectronics are used in computer graphics adapters to take pixels from a frame buffer, stored in digital format, and convert the pixels to analog signals for display on a CRT.

The pixels in the frame buffer can be 4 BPP (bits per pixel), 8 BPP, 15 BPP (16 bits with 1 bit unused), 16 BPP or 24 BPP. These pixels are converted within the palette DAC to 24-bit RGB (8 bits of red, 8 bits of green, and 8 bits of blue.)

In general the smaller pixel sizes (4 and 8 BPP) are used as indices into a palette (color lookup table), which provides 24 bits to the DACs. The 15 and 16-bit pixels are expanded to 24 bits. 24-bit pixels are sent directly to the DACs. The 15, 16, and 24-bit pixels can also be used as indices into the palette, when a color transformation (such as gamma correction) is desired.

The RGB624 Palette DAC also supports the YUV pixel type. YUV pixels contain luminance (brightness) and chrominance (color difference) values. A special circuit on the Palette DAC converts the YUV pixels to 24-bit RGB pixels.

The RGB624 has support for a 16-bit “tagged RGB/YUV” pixel format. This format allows RGB and YUV pixels to be intermixed in the frame buffer by using one of the bits as a “tag”. The RGB624 uses the tag bit to interpret the remaining 15 bits “on the fly” as either RGB or YUV.

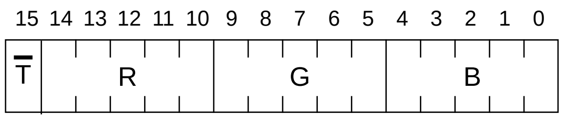
The 8, 15/16 and 24 BPP RGB formats are supported by most PC graphics software. Microsoft Windows supports these formats through its GDI (Graphics Device Interface).

In 1994 Microsoft and Intel released a joint specification called DCI (Display Control Interface). This is a driver level interface which provides for access to advanced graphics features, including YUV formats. This application note describes how to use DCI to exploit the RGB624 16-bit tagged RGB/YUV pixel format.

16-Bit Tagged RGB/YUV

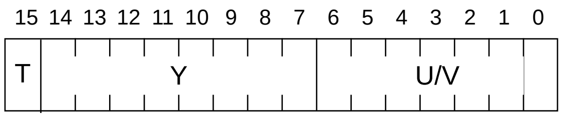
15 BPP RGB pixels use 15 bits of a 16-bit word, with 1 bit unused. The RGB624 tagged format uses this normally-unused bit as a “tag” bit that identifies the remaining fifteen bits as either RGB or YUV. The meaning of the tag bit is programmable. That is, the tag can be programmed such that a tag value of 0 indicates an RGB and a 1 indicates YUV, or the tag can be programmed so that a 0 indicates YUV and a 1 indicates RGB.

When the tag indicates an RGB pixel, the remaining fifteen bits are interpreted as 5:5:5 RGB (five bits each for red, green and blue.)

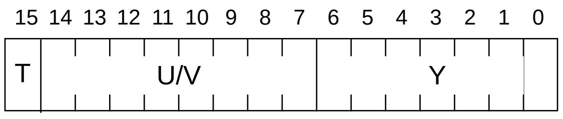


When the tag indicates a YUV pixel, the remaining 15 bits are interpreted as 15-bit YUV, with one of four variations selected by program control:

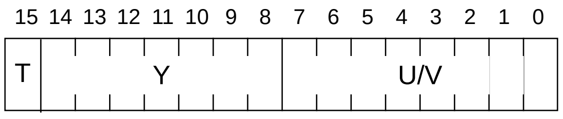
1. 8 bits of Y followed by 7 bits of U/V:



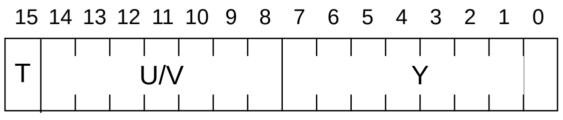
2. 8 bits of U/V followed by 7 bits of Y:



3. 7 bits of Y followed by 8 bits of U/V:



4. 7 bits of U/V followed by 8 bits of Y:



The tagged RGB/YUV format is useful for mixing pixel types in the frame buffer. For example, the background and various windows could be written and tagged as 555 RGB, while other windows (motion video, e.g.), could be written and tagged as YUV.

The RGB624 supports several options for 555 RGB:

❑ Palette lookup, or palette bypass

❑ Contiguous or sparse addressing of the palette

❑ When bypassing the palette, filling in the low order bits with ‘0’s (Zero Insertion), or using the high order bits to fill (“Linear” expansion)

These options are still available with tagged RGB/YUV for the pixels tagged as RGB.

(There is also an option that allows the high order bit to be used to select palette lookup or palette bypass on a pixel-by-pixel basis. This option is not available with tagged RGB/YUV since the high order bit is used as the tag.)

DCI Basics

DCI is a device level software interface that provides for:

❑ Frame buffer access

❑ Off-screen buffer access

❑ Video overlay

❑ Stretching

❑ Color space conversion

❑ Off-screen to on-screen “draw”

DCI is implemented by the DCI Manager and the DCI Display Module.

The DCI Manager is not hardware specific and provides general capabilities such as window overlap management.

The DCI Display Module (also called *DCI provider*) is the hardware specific portion of DCI. It can be a separate DLL or it may be part of the display driver.

Applications do not write to the DCI directly. Instead, requests are made of one or more DCI Clients (also called *DCI users*). DCI Clients are Windows subsystems that use graphics display features enabled by DCI. Examples are digital video codecs, 3-D graphics libraries and games support libraries.

DCI has the concept of *surfaces*. DCI clients write to surfaces, which can be the *primary surface* (the on-screen Windows display), *off-screen surfaces*, and *overlay surfaces*. Images drawn to off-screen surfaces may be transferred to the primary surface using the DCI `Draw` function.

All three surface types share in their characteristics, such as width, height in pixels, bit count (bits per pixel), stride (bytes per line), address, bitfield masks, capability flags (bank switched, dword aligned, etc.), and format.

The pixel formats (bitmap and compression types) are represented with a four character code known as FOURCC. The FOURCC format closest to the RGB624 tagged RGB/YUV format (YUYV) is “YUV2”. New FOURCC formats may be registered with Microsoft.

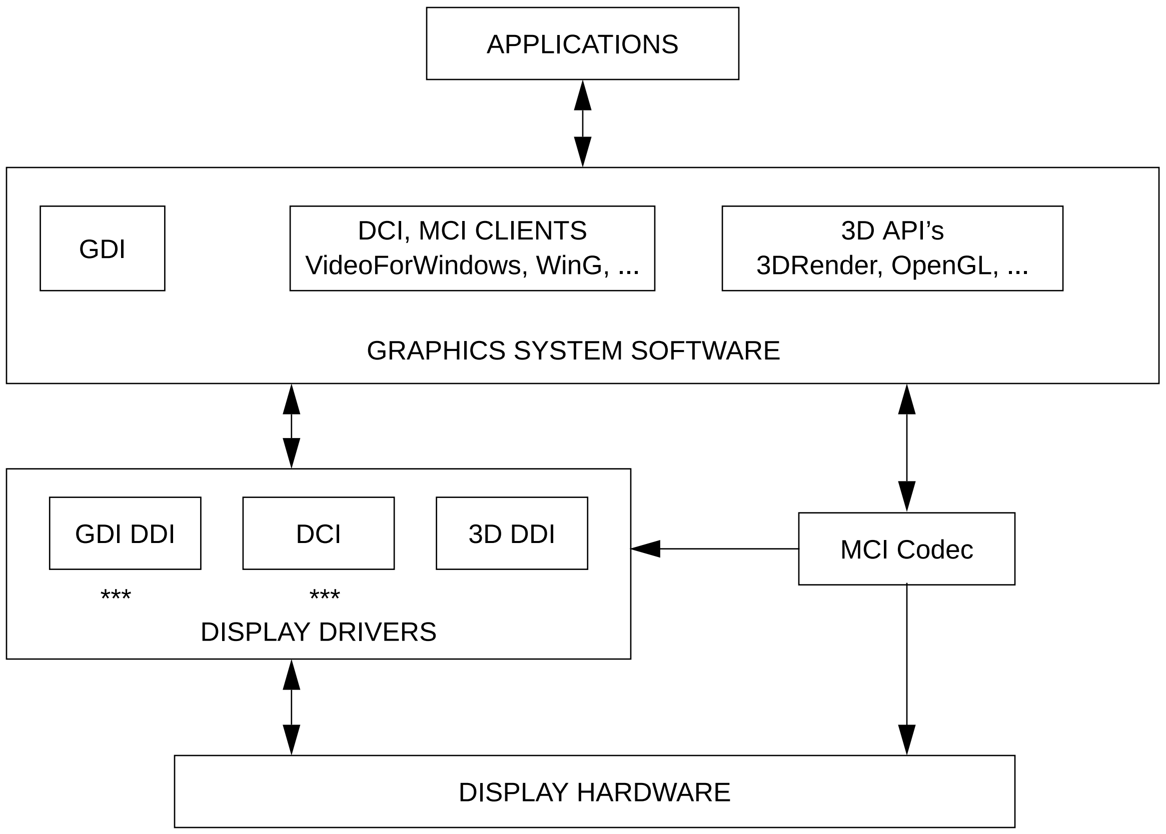
DCI Clients interrogate the DCI Manager to verify that a DCI Display Module is present and to determine the DCI capabilities provided by the DCI provider(s). The GDI is still used by applications and DCI Clients when desired DCI features are not provided.

A typical Multimedia Application written for Microsoft Windows will write to an API (Application Programming Interface) such as Microsoft Video For Windows. Since Video For Windows is a DCI Client, it can query the system for DCI Video capabilities such as support for the YUV2 pixel format. If an existing DCI Driver (or Provider) responds that it can support YUV2, then the DCI Client can set up a Video Codec (like MPEG or INDEO) to convert video streams to YUV2 frames and call the DCI provider to draw each frame. For an overall view of the data flow, see the illustration below. Note that the code required for enabling support of the Tagged YUV pixel format is located in the DCI and GDI display drivers. Note also that if tagged RGB/YUV pixel format were enabled, then the output pixel format from the GDI display driver would always be RGB555 whereas the output pixel format from the DCI Display Driver could be either RGB555 or tagged RGB/YUV.

Coding Support for Tagged Pixels

DCI Code Requirements

The bulk of the code required to support the tagged RGB/YUV format is in the DCI provider. Simple hardware assists can, however, make the code required very minimal. There are basically 3 ways for the video data to be converted from YUV2 to Tagged RGB/YUV:



*** Code changes required to support RGB624 tagged RGB/YUV pixel format

1. Let the codec do it (requires registering the tagged RGB/YUV format with Microsoft and that the codec can support it.)
2. Let hardware support it by doing a simple shift-right-1 and OR the tag bit when the video data is written to the frame buffer.
3. Let the DCI `Draw` function convert the frame from YUV2 to Tagged RGB/YUV.

Solution 1 is optimal but you cannot count on every video codec supporting the tagged RGB/YUV format. Solution 2 is the best since it requires no additional software in the DCI other than enabling the hardware function.

Note that solutions 1 and 3 require that the tagged RGB/YUV formats which have fields that cross a byte boundary (T+8Y+7U/V or T+8U/V+7Y) must set the least significant bit of the 8-bit field to a constant value. This is because hardware scalers will interpolate the byte fields with neighboring pixels.Note that other device-dependent code may be necessary depending on the functionality of the display controller. For instance, YUV to RGB conversion in the controller would need to be disabled.

DCI Code Implementation

The first step in the implementation is to determine if the Tagged RGB/YUV pixel format can be utilized. This is done at initialization of the Offscreen surface in DCI, `DCICreateOffScreenSurface`. If the primary surface (which had to be already created with `DCICreatePrimarySurface`) is RGB555, and the compression FOURCC format of the Offscreen Surface is YUV based (Indeo IF09, YUV2, YUV9, etc.), then it is possible to use the tagged RGB/YUV pixel format. A global flag should be set at this point to indicate that an Offscreen Surface needs to be converted to tagged RGB/YUV.

The next step is to check for this global flag in the DCI Draw Callback Function. If the flag is set, then the Draw function must convert the incoming FOURCC format to the appropriate tagged RGB/YUV format. If a hardware assist function is being utilized, then it must be reset upon completion of the Draw Function to insure proper GDI execution. This includes setting and resetting any available masks that are being utilized to manipulate the tag bit.

Another requirement of the DCI provider, depending upon the implementation, may be to manage the tag bit plane completely. For example, if a DCI surface in tagged RGB/YUV format is re-positioned on the display, thereby exposing GDI regions, it may be the DCI provider's responsibility to erase the tag bit plane of the exposed GDI region. This is possible to do in DCI with the `SetClipList` and `SetDestination` Callback functions. The DCI Client will call these functions in the DCI provider whenever the DCI surface window moves, re-sizes, or has its cliplist changed. The DCI provider would need to save its window (or bounding window) dimensions and make sure that when they changed through subsequent calls to `SetDestination` and `SetCliplist` that the exposed GDI regions would get the tag bit cleared.

GDI Code Requirements

The other code changes required for supporting the tagged RGB/YUV pixel format exist in the GDI display driver. Again, a simple hardware assist function would make the code required very minimal. To support tagged RGB/YUV pixel format, the GDI driver must insure that the tag bit always equal zero. If a “Hardware Modify Mask” that always wrote zero into the tag bit for every frame buffer write is available, the only code required is to enable this hardware function at initialization (DCI would need to disable this function and re-enable it upon completion of a DCI Draw Operation). A “Hardware Protect Mask” could also be utilized and always set to `0x7fff` during GDI operations. This would require DCI (which would always operate with the protect mask equal to `0xffff`) to clear tag bit regions in the DCI window which become overwritten by a GDI window; this would occur whenever a DCI window moves or when its cliplist changes.

GDI Code Implementation

If a hardware assist as previously described was utilized, then the code required for the GDI implementation would be to simply enable the function at GDI initialization (Enable Function). Code would also be placed in Enable to check that the Pixel Depth requested is 15 bits (RGB555 -- 32768 colors) and that the RGB624 Palette DAC was present; both must be true for tagged RGB/YUV pixel format to be enabled.

If the tagged RGB/YUV pixel format was enabled and no hardware assists were available, then the driver software would need to play a larger role. The extent of the code required is dependent on the hardware capabilities of the graphics controller. To guarantee that the tag bit would always be equal to zero in the frame buffer, the GDI driver needs to handle 3 things:

1. When converting logical colors to physical colors, make sure that the physical color always has the tag bit cleared.
2. When converting DIBs (Device Independent Bitmaps) to physical (or device dependent) bitmaps, again make sure that the pixel data has the tag bit cleared.
3. For BitBlt or Output Functions which have a ROP3 or ROP2 of the types described below, the tag bit is in danger of flipping from 0 to 1. In the following, “S” stands for Source, “D” stands for Destination, and “P” stands for Pattern.

   - ROP2 :
     ```
     if ((D == 0) & (S == 0))-> (D = 1)
     ```

   This occurs for all even numbered ROP2 codes.

   - ROP3:
     ```
     if ((D == 0) & (S == 0) &(P == 0))
     -> (D = 1)
     ```

   This occurs for all odd numbered ROP3 codes.

Common ROPs that are in this class are XOR or NotD and ONE or “set D to all ones” which both are independent of a Source or Pattern. Drawing Operations that could potentially use one of these types of ROP codes can be broken down into 2 categories:

1. Those handled entirely in the driver and pixel-by-pixel written directly into a linearly accessed frame buffer.
2. Those that use some device dependent hardware acceleration to enhance the performance of the operation.

For those handled by linear access to a frame buffer, the tag bit must be always set to zero for each write. For those handled by hardware acceleration, if the hardware also supports a Protect Mask, then this mask can be temporarily set to `0x7fff` during the operation and then reset back to `0xffff` after the operation. In the worst case, the hardware acceleration may not be utilized and the drawing operation would be handled completely by direct access to the frame buffer.

Related Documentation

1. RGB624 Data Sheet (IBM Microelectronics document number IOG624DSU)
2. Display Control Interface (DCI) Specification
3. Microsoft Windows Software Development Kit
4. Microsoft Windows Device Driver Development Kit
5. Microsoft Video for Windows 1.1 Developer’s Kit

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY
12533-6531

The IBM home page can be found at
http://www.ibm.com.

The IBM Microelectronics Division home page can be found at
http://www.chips.ibm.com.

Fast Fax Service 415-885-4121

Document No. SK10-3012-00