[UEFI Practice] UEFI Graphics Display (Display Driver)

原创 jiangwei0512 Modified on 2023-06-25 00:48:32 Read 4.2k Collection 14 Likes 2

Category Column: UEFI Development Basics Article Tags: uefi

摘要 This article introduces how the QemuVideoDxe driver works in the UEFI environment, especially how to use EFI_DRIVER_BINDING_PROTOCOL to support and start the graphics driver. The QemuVideoDetect function is used to detect and identify the graphics card emulated by QEMU. EFI_GRAPHICS_OUTPUT_PROTOCOL is the key protocol for subsequent graphics display, including mode query, setting, and BlockTransfer operations. The Blt function is used for pixel operations to achieve display o…

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

Expand

Display Driver

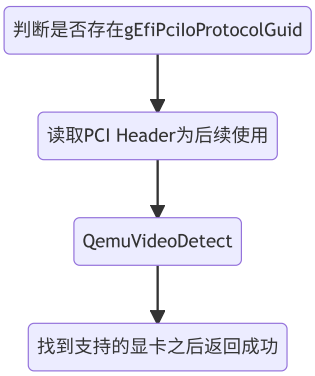
OVMF BIOS uses this as the graphics driver. The underlying implementation of specific graphics display is not the focus, so only a brief introduction is given here.

QemuVideoDxe is a UEFI Driver Model, corresponding to `EFI_DRIVER_BINDING_PROTOCOL` :

c AI generated projects 登录复制 run

```c
EFI_DRIVER_BINDING_PROTOCOL  gQemuVideoDriverBinding = {
  QemuVideoControllerDriverSupported,
  QemuVideoControllerDriverStart,
  QemuVideoControllerDriverStop,
  0x10,
  NULL,
  NULL
};
```

- The process of **Supported function :**



The key point is `QemuVideoDetect()` the implementation, find the graphics card:

bash AI generated projects 登录复制

```bash
QemuVideo: QEMU Standard VGA detected
```

This is the graphics card simulated by QEMU, and you don't need to pay attention to the specific implementation.

- **The initialization part in the Start function** does not need to be paid special attention to. The focus is on the Protocol installed here:

c AI generated projects 登录复制 run

```c
struct _EFI_GRAPHICS_OUTPUT_PROTOCOL {
  EFI_GRAPHICS_OUTPUT_PROTOCOL_QUERY_MODE    QueryMode;
  EFI_GRAPHICS_OUTPUT_PROTOCOL_SET_MODE      SetMode;
  EFI_GRAPHICS_OUTPUT_PROTOCOL_BLT           Blt;
  ///
  /// Pointer to EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE data.
  ///
  EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE          *Mode;
};
```

The corresponding GUID is `gEfiGraphicsOutputProtocolGuid` , this Protocol will be used by the subsequent UEFI display module, referred to as **GOP** .

The members of this Protocol can be divided into two categories, one is related to Mode and related operations, and the other is Blt. The former is the setting of the mode, while the full name of the latter is Block Transfer, and its real function is to display output. The reason for this name is that what it actually does is to transfer data, which will be further explained later.

`Mode` The structure is as follows:

c AI generated projects 登录复制 run

```c
typedef struct {
  ///
  /// The version of this data structure. A value of zero represents the
  /// EFI_GRAPHICS_OUTPUT_MODE_INFORMATION structure as defined in this specification.
  ///
  UINT32                     Version;
  ///
  /// The size of video screen in pixels in the X dimension.
  ///
  UINT32                     HorizontalResolution;
  ///
  /// The size of video screen in pixels in the Y dimension.
  ///
  UINT32                     VerticalResolution;
  ///
  /// Enumeration that defines the physical format of the pixel. A value of PixelBltOnly
  /// implies that a linear frame buffer is not available for this mode.
  ///
  EFI_GRAPHICS_PIXEL_FORMAT    PixelFormat;
  ///
  /// This bit-mask is only valid if PixelFormat is set to PixelPixelBitMask.
  /// A bit being set defines what bits are used for what purpose such as Red, Green, Blue, or Reserved.
  ///
  EFI_PIXEL_BITMASK            PixelInformation;
  ///
  /// Defines the number of pixel elements per video memory line.
  ///
  UINT32                     PixelsPerScanLine;
} EFI_GRAPHICS_OUTPUT_MODE_INFORMATION;

typedef struct {
  ///
  /// The number of modes supported by QueryMode() and SetMode().
  ///
  UINT32                                MaxMode;
  ///
  /// Current Mode of the graphics device. Valid mode numbers are 0 to MaxMode -1.
  ///
  UINT32                                Mode;
  ///
  /// Pointer to read-only EFI_GRAPHICS_OUTPUT_MODE_INFORMATION data.
  ///
  EFI_GRAPHICS_OUTPUT_MODE_INFORMATION    *Info;
  ///
  /// Size of Info structure in bytes.
  ///
  UINTN                                 SizeOfInfo;
  ///
  /// Base address of graphics linear frame buffer.
  /// Offset zero in FrameBufferBase represents the upper left pixel of the display.
  ///
  EFI_PHYSICAL_ADDRESS                  FrameBufferBase;
  ///
  /// Amount of frame buffer needed to support the active mode as defined by
  /// PixelsPerScanLine xVerticalResolution x PixelElementSize.
  ///
  UINTN                                 FrameBufferSize;
} EFI_GRAPHICS_OUTPUT_PROTOCOL_MODE;
```

收起 ^

The above information is mainly the physical parameters of the graphics card, which will be determined during the initialization process of the graphics card itself, and the subsequent UEFI driver will use these parameters.

Here is a code example to get the above information (beni\BeniPkg\DynamicCommand\DisplayDynamicCommand\Display.c):

c AI generated projects 登录复制 run

```c
  for (Index = 0; Index < GopHandleCount; Index++) {
    Status = gBS->HandleProtocol (
                    GopHandleBuffer[Index],
                    &gEfiGraphicsOutputProtocolGuid,
                    (VOID *)&Gop
```

```
                  );
    if (EFI_ERROR (Status)) {
      continue;
    }
    Print (L"MaxMode                      : %d\r\n", Gop->Mode->MaxMode);
    Print (L"Mode                         : %d\r\n", Gop->Mode->Mode);
    Print (L"Info.Version                 : 0x%04x\r\n", Gop->Mode->Info->Version);
    Print (L"Info.HorizontalResolution    : %d\r\n", Gop->Mode->Info->HorizontalResolution);
    Print (L"Info.VerticalResolution      : %d\r\n", Gop->Mode->Info->VerticalResolution);
    Print (L"Info.PixelFormat             : %s\r\n", gPixelFormat[Gop->Mode->Info->PixelFormat]);
    Print (L"PixelInformation.RedMask     : 0x%04x\r\n",
            Gop->Mode->Info->PixelInformation.RedMask);
    Print (L"PixelInformation.GreenMask   : 0x%04x\r\n",
            Gop->Mode->Info->PixelInformation.GreenMask);
    Print (L"PixelInformation.BlueMask    : 0x%04x\r\n",
            Gop->Mode->Info->PixelInformation.BlueMask);
    Print (L"PixelInformation.ReservedMask  : 0x%04x\r\n",
            Gop->Mode->Info->PixelInformation.ReservedMask);
    Print (L"PixelsPerScanLine            : 0x%04x\r\n", Gop->Mode->Info->PixelsPerScanLine);
    Print (L"SizeOfInfo                   : %d\r\n", Gop->Mode->SizeOfInfo);
    Print (L"FrameBufferBase              : 0x%lx\r\n", Gop->Mode->FrameBufferBase);
    Print (L"FrameBufferSize              : 0x%lx\r\n", Gop->Mode->FrameBufferSize);
  }
```

收起 ^

The result is:

```
0 vide card:
MaxMode                      : 30
Mode                         : 0
Version                      : 0x0000
HorizontalResolution         : 0x0500
VerticalResolution           : 0x0320
PixelFormat                  : PixelBlueGreenRedReserved8BitPerColor
PixelInformation.RedMask     : 0x0000
PixelInformation.GreenMask   : 0x0000
PixelInformation.BlueMask    : 0x0000
PixelInformation.ReservedMask  : 0x0000
PixelsPerScanLine            : 0x0500
SizeOfInfo                   : 36
FrameBufferBase              : 0xC0000000
FrameBufferSize              : 0x3E8000
1 vide card:
MaxMode                      : 30
Mode                         : 0
Version                      : 0x0000
HorizontalResolution         : 0x0500
VerticalResolution           : 0x0320
PixelFormat                  : PixelBlueGreenRedReserved8BitPerColor
PixelInformation.RedMask     : 0x0000
PixelInformation.GreenMask   : 0x0000
PixelInformation.BlueMask    : 0x0000
PixelInformation.ReservedMask  : 0x0000
PixelsPerScanLine            : 0x0500
SizeOfInfo                   : 36
FrameBufferBase              : 0xC0000000
FrameBufferSize              : 0x3E8000
Shell> _
```


Here we get two GOPs, but actually there is only one virtual graphics card in OVFM. This may seem strange, but in fact the two protocols point to the same graphics card. The reason why there are two is that it is installed twice `EFI_GRAPHICS_OUTPUT_PROTOCOL`. The reason why it needs to be installed twice is the ConSplitterDxe.inf module:

c AI generated projects 登录复制 run

```c
  //
  // If both ConOut and StdErr incorporate the same Text Out device,
  // their MaxMode and QueryData should be the intersection of both.
  //
  Status = ConSplitterTextOutAddDevice (&mConOut, TextOut, GraphicsOutput, UgaDraw);
```

Since this section only discusses graphics display, the Console part of UEFI is not introduced. If you want to obtain only the information of the graphics card itself, you can determine whether the Handle corresponding to the Protocol has a Device Path installed. The corresponding sample function is:

c AI generated projects 登录复制 run

```c
EFI_GRAPHICS_OUTPUT_PROTOCOL *
GetSpecificGop (
  VOID
  )
{
  EFI_STATUS                    Status = EFI_NOT_FOUND;
  EFI_GRAPHICS_OUTPUT_PROTOCOL  *Gop = NULL;
  UINTN                         Index = 0;
  UINTN                         GopHandleCount = 0;
  EFI_HANDLE                    *GopHandleBuffer = NULL;
  EFI_DEVICE_PATH_PROTOCOL      *GopDevicePath = NULL;

  //
  // Get all GOP responding independent video card.
  //
  Status = gBS->LocateHandleBuffer (
                 ByProtocol,
                 &gEfiGraphicsOutputProtocolGuid,
                 NULL,
                 &GopHandleCount,
                 &GopHandleBuffer
                 );
  if (EFI_ERROR (Status)) {
    DEBUG ((EFI_D_ERROR, "[%a][%d] Failed. - %r\n", __FUNCTION__, __LINE__, Status));
    goto DONE;
  }

  for (Index = 0; Index < GopHandleCount; Index++) {
    //
    // The video card should have device path.
    //
    Status = gBS->HandleProtocol (
                    GopHandleBuffer[Index],
                    &gEfiDevicePathProtocolGuid,
                    (VOID *)&GopDevicePath
                    );
    if (EFI_ERROR (Status)) {
      continue;
    }
    Status = gBS->HandleProtocol (
                    GopHandleBuffer[Index],
                    &gEfiGraphicsOutputProtocolGuid,
                    (VOID *)&Gop
                    );
    if (EFI_ERROR (Status)) {
      continue;
    } else {
      Print (L"Video card evice path: %s\r\n",
             ConvertDevicePathToText (GopDevicePath, TRUE, TRUE));
      break;
    }
  }

DONE:

  if (NULL != GopHandleBuffer) {
    FreePool (GopHandleBuffer);
    GopHandleBuffer = NULL;
  }

  return Gop;
}
```

收起 ^

The following are descriptions of the mode values:

```bash
MaxMode                     : 30 # 显卡支持的所有模式
Mode                        : 0  # 当前使用的模式
Info.Version                : 0x0000 # 模式版本信息
Info.HorizontalResolution   : 1280   # 分辨率
Info.VerticalResolution     : 800    # 分辨率
Info.PixelFormat            : PixelBlueGreenRedReserved8BitPerColor  # 像素相关的变量，跟具体的显卡有关，这里不用特别关注
PixelInformation.RedMask    : 0x0000
PixelInformation.GreenMask  : 0x0000
PixelInformation.BlueMask   : 0x0000
PixelInformation.ReservedMask  : 0x0000
PixelsPerScanLine           : 0x0500
SizeOfInfo                  : 36 # EFI_GRAPHICS_OUTPUT_MODE_INFORMATION的大小，共计36个字节
FrameBufferBase             : 0xC0000000 # 这个值实际上是显卡的MMIO Bar地址，跟下面的值一起，是底层操作需要关注的
FrameBufferSize             : 0x3E8000
```

收起 ^

The smallest unit of GOP output is pixel, so here we will involve various pixel-related parameters, which together form the concept of mode. The current graphics card supports 30 modes, which can be read one by one through the code:

```c
  for (Index = 0; Index < Gop->Mode->MaxMode; Index++) {
    Status = Gop->QueryMode (Gop, Index, &SizeOfInfo, &ModeInfo);
    if (!EFI_ERROR (Status)) {
      Print (L"Mode                           : %d\r\n", Index);
      Print (L"Info.Version                   : 0x%04x\r\n", ModeInfo->Version);
      Print (L"Info.HorizontalResolution      : %d\r\n", ModeInfo->HorizontalResolution);
      Print (L"Info.VerticalResolution        : %d\r\n", ModeInfo->VerticalResolution);
      Print (L"Info.PixelFormat               : %s\r\n", gPixelFormat[ModeInfo->PixelFormat]);
      Print (L"PixelInformation.RedMask       : 0x%04x\r\n",
              ModeInfo->PixelInformation.RedMask);
      Print (L"PixelInformation.GreenMask     : 0x%04x\r\n",
              ModeInfo->PixelInformation.GreenMask);
      Print (L"PixelInformation.BlueMask      : 0x%04x\r\n",
              ModeInfo->PixelInformation.BlueMask);
      Print (L"PixelInformation.ReservedMask  : 0x%04x\r\n",
              ModeInfo->PixelInformation.ReservedMask);
      Print (L"PixelsPerScanLine              : 0x%04x\r\n", ModeInfo->PixelsPerScanLine);
      Print (L"SizeOfInfo                     : %d\r\n", SizeOfInfo);
      Print (L"----------------------------------------------\r\n");
      FreePool (ModeInfo);
twen    }
```

收起 ^

From the print information of the above code, we can see that the difference between the various modes mainly comes from the resolution, which will not be listed here.

Finally, we briefly explain how to operate pixels. The corresponding Blt function is declared as follows:

```c
typedef
EFI_STATUS
(EFIAPI *EFI_GRAPHICS_OUTPUT_PROTOCOL_BLT)(
  IN  EFI_GRAPHICS_OUTPUT_PROTOCOL            *This,
  IN  EFI_GRAPHICS_OUTPUT_BLT_PIXEL           *BltBuffer    OPTIONAL,
  IN  EFI_GRAPHICS_OUTPUT_BLT_OPERATION       BltOperation,
  IN  UINTN                                   SourceX,
  IN  UINTN                                   SourceY,
  IN  UINTN                                   DestinationX,
  IN  UINTN                                   DestinationY,
  IN  UINTN                                   Width,
  IN  UINTN                                   Height,
  IN  UINTN                                   Delta         OPTIONAL
  );
```

收起 ^

The function's input parameters are described as follows:

- `This` : The Protocol pointer itself.
- `BltBuffer` : `EFI_GRAPHICS_OUTPUT_BLT_PIXEL` Pointer, its structure is as follows:

```c
typedef struct {
  UINT8    Blue;
  UINT8    Green;
  UINT8    Red;
```

```
  UINT8    Reserved;
} EFI_GRAPHICS_OUTPUT_BLT_PIXEL;
```

It represents the three primary colors of a pixel. For example, "Blue=0, Green=0, Red=255" represents a red pixel.

As for `BltBuffer`, it can represent a pixel or point to a pixel array to represent the color of all pixels in an area.

- `BltOperation`: Output operation, its value is as follows:

c AI generated projects 登录复制 run

```c
///
/// actions for BltOperations
///
typedef enum {
  ///
  /// Write data from the BltBuffer pixel (0, 0)
  /// directly to every pixel of the video display rectangle
  /// (DestinationX, DestinationY) (DestinationX + Width, DestinationY + Height).
  /// Only one pixel will be used from the BltBuffer. Delta is NOT used.
  ///
  EfiBltVideoFill,

  ///
  /// Read data from the video display rectangle
  /// (SourceX, SourceY) (SourceX + Width, SourceY + Height) and place it in
  /// the BltBuffer rectangle (DestinationX, DestinationY )
  /// (DestinationX + Width, DestinationY + Height). If DestinationX or
  /// DestinationY is not zero then Delta must be set to the length in bytes
  /// of a row in the BltBuffer.
  ///
  EfiBltVideoToBltBuffer,

  ///
  /// Write data from the BltBuffer rectangle
  /// (SourceX, SourceY) (SourceX + Width, SourceY + Height) directly to the
  /// video display rectangle (DestinationX, DestinationY)
  /// (DestinationX + Width, DestinationY + Height). If SourceX or SourceY is
  /// not zero then Delta must be set to the length in bytes of a row in the
  /// BltBuffer.
  ///
  EfiBltBufferToVideo,

  ///
  /// Copy from the video display rectangle (SourceX, SourceY)
  /// (SourceX + Width, SourceY + Height) to the video display rectangle
  /// (DestinationX, DestinationY) (DestinationX + Width, DestinationY + Height).
  /// The BltBuffer and Delta are not used in this mode.
  ///
  EfiBltVideoToVideo,

  EfiGraphicsOutputBltOperationMax
} EFI_GRAPHICS_OUTPUT_BLT_OPERATION;
```

收起 ^

- `SourceX/SourceY`: Indicates `BltBuffer` or displays the coordinates of the upper left corner of an area in the Buffer.
- `DestinationX/DestinationY`: Indicates `BltBuffer` the coordinates of the upper left corner of an area in the source or display buffer. Whether the source and destination are `BltBuffer` the display buffer or not needs to `BltOperation` be determined.
- `Width/Height`: Indicates the length and width of the displayed rectangular area.
- `Delta`: It is invalid in the `EfiBltVideoFill` OR `EfiBltVideoToVideo` operation, otherwise it represents `BltBuffer` the number of bytes in a row of pixels.

Here is an example:

c AI generated projects 登录复制 run

```c
  SetMem (&FillColour, sizeof (EFI_GRAPHICS_OUTPUT_BLT_PIXEL), 0x0);
  FillColour.Red = 255;
  Status = Gop->Blt (
                  Gop,
                  &FillColour,
                  EfiBltVideoFill,
                  0,
                  0,
                  0,
                  0,
                  Gop->Mode->Info->HorizontalResolution,
                  Gop->Mode->Info->VerticalResolution,

```

```
                    0
                    );
```

收起 ^

At this time, a red full screen will be displayed:

It is used here `EfiBltVideoFill` , and it fills a rectangular area (here the entire screen is specified according to the subsequent parameters) with a pixel (here a red pixel).

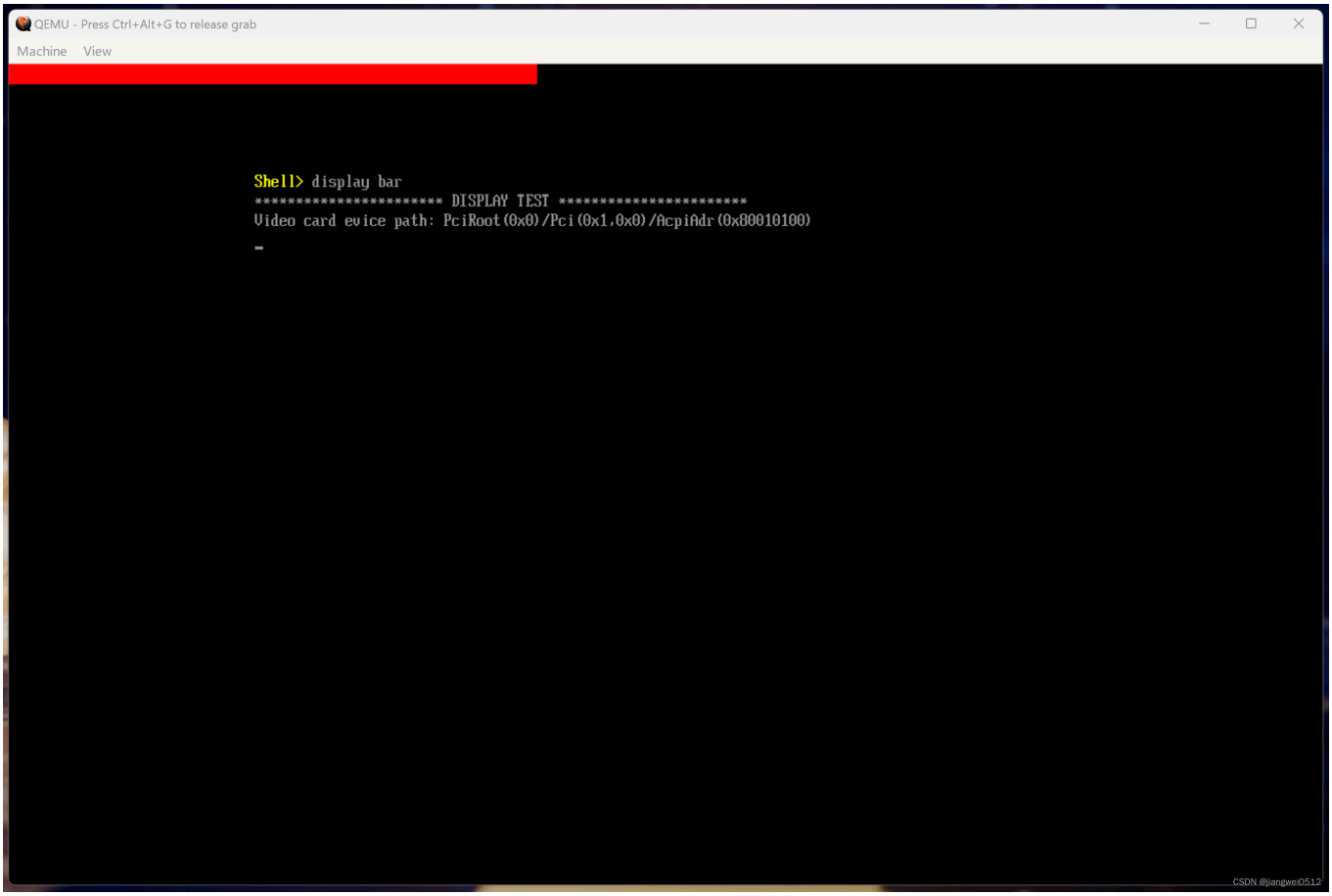
Here `Delta` is an example of a progress bar simulation:

c AI generated projects 登录复制 run

```c
    Width = Gop->Mode->Info->HorizontalResolution;  // The width of bar.
    Height = BAR_HEIGHT;  // The height of bar.
    Blt = AllocateZeroPool (sizeof (EFI_GRAPHICS_OUTPUT_BLT_PIXEL) * Width * Height);
    if (NULL == Blt) {
      DEBUG ((EFI_D_ERROR, "[%a][%d] Out of memory\n", __FUNCTION__, __LINE__));
      Status = EFI_OUT_OF_RESOURCES;
      goto DONE;
    }
    //
    // Buffer for a red process bar.
    //
    for (IndexW = 0; IndexW < Width; IndexW++) {
      for (IndexH = 0; IndexH < Height; IndexH++) {
        Blt[IndexH * Width + IndexW].Red = 255;
      }
    }
    for (IndexW = 0; IndexW < Width; IndexW++) {
      Status = Gop->Blt (
                      Gop,
                      Blt,
                      EfiBltBufferToVideo,
                      0,
                      0,
                      IndexW,
                      0,
                      1,
                      Height,
                      sizeof (EFI_GRAPHICS_OUTPUT_BLT_PIXEL) * Width
                      );
      gBS->Stall (1000 * 10);
    }
  }
```

收起 ^

The result is:



The red part in the upper part of the picture above is the progress bar, which will keep moving until it fills the entire width.

The specific code implementation can be found at https://gitee.com/jiangwei0512/edk2-beni.git .