Accessing EDID Information From UEFI Shell

Finnbarr P. Murphy

(fpm@fpmurphy.com)

EDID (Extended display Identification Data) is a defined data structure published by a digital monitor to describe its capabilities to a video source such as a graphics card on a computer. This enables a computer to easily determine what types of monitor(s) are connected to it. The EDID structure and possible values for the members of the structure is defined in a standard published by the Video Electronics Standards Association (VESA). Four versions of the EDID structure have been defined over the years. Currently v1.3 (E-EDID – Enhanced EDID) is probably the most common.

On modern Linux platforms you can access EDID information via the *sysfs* virtual filesystem which is mounted at */sys*. A number of utilities are available on Microsoft Windows that provide the same information. So far, I have not come across a similar utility for the UEFI Shell. Hence I decided to roll my own.

I have written a number of EDID parsers before. The main challenge on this particular platform was to provide support for the representation of decimal numbers as UCS2 strings. UEFI does not currently support floating point or real numbers as native types. I adopted a C language *floa* implementation I found from an Internet to use a fixed radix of 10 (decimal) and added the ability to specify the number of decimal places.

The other challenge was to find the GOP protocol handle which had the EDID information. See the code in *efi_main* for how I did it. I am sure there probably is a far more elegant way to do this but so far I have not found it. Please let me know if you know a better way of doing this.

Without further ado, here is the source code:

```
//
//  Copyright (c) 2012  Finnbarr P. Murphy.   All rights reserved.
//
//  Display core EDID v1.3 information.
//
//  License: BSD License
//
#include <efi.h>
#include <efilib.h>
#include <stdio.h>
#include "efi_gop.h"
#include "efi_edid.h"
EFI_GUID GV_GUID = EFI_GLOBAL_VARIABLE;
EFI_GUID SIG_DB = { 0xd719b2cb, 0x3d3a, 0x4596, {0xa3, 0xbc, 0xda, 0xd0,  0xe, 0x67, 0x65,
 0x6f }};
#define EFI_SHELL_INTERFACE_GUID \
   (EFI_GUID) {0x47c7b223, 0xc42a, 0x11d2, {0x8e,0x57,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
#define EFI_OPEN_PROTOCOL_GET_PROTOCOL    0x00000002
static struct efi_gop *gop;
EFI_HANDLE *Image_Handle;
VOID *Interface;
//
// Based on code found at http://code.google.com/p/my-itoa/
//
int
```



```
Integer2AsciiString(int val, char* buf)
{
    const unsigned int radix = 10;
    char* p = buf;
    unsigned int a;
    int len;
    char* b;
    char temp;
    unsigned int u;
    if (val < 0) {
        *p++ = '-';
        val = 0 - val;
    }
    u = (unsigned int)val;
    b = p;
    do {
        a = u % radix;
        u /= radix;
        *p++ = a + '0';
    } while (u > 0);
    len = (int)(p - buf);
    *p-- = 0;
    // swap
    do {
       temp = *p; *p = *b; *b = temp;
       --p; ++b;
    } while (b < p);
    return len;
}
//
// Based on code found on the Internet (author unknown)
// Search for ftoa implementations
//
int
Float2AsciiString(float f, char *buffer, int numdecimals)
{
    int status = 0;
    char *s = buffer;
    long mantissa, int_part, frac_part;
    short exp2;
    char *p;
    char m;
    typedef union {
        long L;
        float F;
    } LF_t;
    LF_t x;
    if (f == 0.0) {           // return 0.00
       *s++ = '0'; *s++ = '.'; *s++ = '0'; *s++ = '0';
       *s = 0;
      return status;
    }
    x.F = f;
    exp2 = (unsigned char)(x.L >> 23) - 127;
    mantissa = (x.L &amp;amp; 0xFFFFFF) | 0x800000;
    frac_part = 0;
    int_part = 0;
    if (exp2 >= 31 || exp2 < -23) {
        *s = 0;
        return 1;
    }
    if (exp2 >= 0) {
        int_part = mantissa >> (23 - exp2);
        frac_part = (mantissa << (exp2 + 1)) &amp;amp; 0xFFFFFF;
    } else {
        frac_part = (mantissa &amp;amp; 0xFFFFFF) >> -(exp2 + 1);
    }
    if (int_part == 0)
```



```
       *s++ = '0';
    else {
        Integer2AsciiString(int_part, s);
        while (*s) s++;
    }
    *s++ = '.';
    if (frac_part == 0)
        *s++ = '0';
    else {
        for (m = 0; m < numdecimals; m++) {                        // print BCD
            frac_part = (frac_part << 3) + (frac_part << 1);      // frac_part *= 10
            *s++ = (frac_part >> 24) + '0';
            frac_part &amp;= 0xFFFFFF;
        }
    }
    *s = 0;
    return status;
}
VOID
Ascii2UnicodeString(CHAR8 *String, CHAR16 *UniString)
{
    while (*String != '&#92;&#48;') {
        *(UniString++) = (CHAR16) *(String++);
    }
    *UniString = '&#92;&#48;';
}
CHAR16 *
DisplayGammaString(UINT8 gamma)
{
    char str[8];
    static CHAR16 wstr[8];
    float g1 = (float)gamma;
    float g2 = 1.00 + (g1/100);
    Float2AsciiString(g2, str, 2);
    Ascii2UnicodeString(str, wstr);
    return wstr;
}
CHAR16 *
ManufacturerAbbrev(UINT16 *ManufactureName)
{
    static CHAR16 mcode[8];
    UINT8 *block = (UINT8 *)ManufactureName;
    UINT16 h = EDID_COMBINE_HI_8LO(block[0], block[1]);
    mcode[0] = (CHAR16)((h>>10) &amp; 0x1f) + 'A' - 1;
    mcode[1] = (CHAR16)((h>>5) &amp; 0x1f) + 'A' - 1;
    mcode[2] = (CHAR16)(h &amp; 0x1f) + 'A' - 1;
    mcode[3] = (CHAR16)'&#92;&#48;';
    return mcode;
}
int
CheckForValidEdid(EFI_EDID_ACTIVE_PROTOCOL *Edid)
{
    EDID_DATA_BLOCK *EdidDataBlock = (EDID_DATA_BLOCK *)Edid->Edid;
    UINT8 *edid = (UINT8 *)Edid->Edid;
    UINT8 i;
    UINT8 checksum = 0;
    const UINT8 EdidHeader[] = {0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00};
    for (i = 0; i < EDID_LENGTH; i++) {
        checksum += edid[i];
    }
    if (checksum != 0) {
        return(1);
    }
    if (*edid == 0x00) {
       checksum = 0;
       for (i = 0; i < 8; i++) {
           if (*edid++ == EdidHeader[i])
               checksum++;
```



```
       }
       if (checksum != 8) {
           return(1);
       }
    }
    if (EdidDataBlock->EdidVersion != 1 || EdidDataBlock->EdidRevision > 4) {
        return(1);
    }
    return(0);
}
void
PrintDetailedTimingBlock(UINT8 *dtb)
{
    Print(L"Horizonal Image Size: %d mm\n", EDID_DET_TIMING_HSIZE(dtb));
    Print(L" Vertical Image Size: %d mm\n", EDID_DET_TIMING_VSIZE(dtb));
    Print(L"HoriImgSzByVertImgSz: %d\n", dtb[14]);
    Print(L"   Horizontal Border: %d\n", EDID_DET_TIMING_HBORDER(dtb));
    Print(L"     Vertical Border: %d\n", EDID_DET_TIMING_VBORDER(dtb));
}
void
PrintEdid(EFI_EDID_ACTIVE_PROTOCOL *Edid)
{
    EDID_DATA_BLOCK *EdidDataBlock = (EDID_DATA_BLOCK *)Edid->Edid;
    UINT8 tmp;
    Print(L"        EDID Version: %d\n", EdidDataBlock->EdidVersion);
    Print(L"       EDID Revision: %d\n",  EdidDataBlock->EdidRevision);
    Print(L" Vendor Abbreviation: %s\n", ManufacturerAbbrev(&amp;(EdidDataBlock->Manufactu
reName)));
    Print(L"          Product ID: %0X\n", EdidDataBlock->ProductCode);
    Print(L"       Serial Number: %02X\n",  EdidDataBlock->SerialNumber);
    Print(L"    Manufacture Week: %d\n", EdidDataBlock->WeekOfManufacture);
    Print(L"    Manufacture Year: %d\n", EdidDataBlock->YearOfManufacture + 1990);
    tmp = EdidDataBlock->VideoInputDefinition;
    Print(L"         Video Input: ");
    if (CHECK_BIT(tmp, 7)) {
        Print(L"Analog\n");
    } else {
        Print(L"Digital\n");
    }
    if (tmp &amp; 0x1F) {
        Print(L"      Syncronization: ");
        if (CHECK_BIT(tmp, 4))
            Print(L"BlankToBackSetup ");
        if (CHECK_BIT(tmp, 3))
            Print(L"SeparateSync ");
        if (CHECK_BIT(tmp, 2))
            Print(L"CompositeSync ");
        if (CHECK_BIT(tmp, 1))
            Print(L"SyncOnGreen ");
        if (CHECK_BIT(tmp, 0))
            Print(L"SerrationVSync ");
        Print(L"\n");
    }
    tmp = EdidDataBlock->DpmSupport;
    Print(L"        Display Type: ");
    if (CHECK_BIT(tmp, 3) &amp;&amp; CHECK_BIT(tmp, 4)) {
        Print(L"Undefined");
    } else if (CHECK_BIT(tmp, 3)) {
        Print(L"RGB color");
    } else if (CHECK_BIT(tmp, 4)) {
        Print(L"Non-RGB multicolor");
    } else {
        Print(L"Monochrome");
    }
    Print(L"\n");
    Print(L"  Max Horizonal Size: %1d cm\n", EdidDataBlock->MaxHorizontalImageSize);
    Print(L"   Max Vertical Size: %1d cm\n", EdidDataBlock->MaxVerticalImageSize);
    Print(L"               Gamma: %s\n", DisplayGammaString(EdidDataBlock->DisplayGamma));
```



```
    PrintDetailedTimingBlock((UINT8 *)&amp;(EdidDataBlock->DescriptionBlock1[0]));
}
void *
efi_open_protocol (EFI_HANDLE Handle,
                   EFI_GUID *Protocol)
{
    EFI_STATUS Status;
    Status = uefi_call_wrapper(BS->OpenProtocol, 6,
                               Handle, Protocol, &amp;Interface,
                               Image_Handle,
                               NULL,
                               EFI_OPEN_PROTOCOL_GET_PROTOCOL);
    return Interface;
}
EFI_STATUS
efi_main (EFI_HANDLE image, EFI_SYSTEM_TABLE *systab)
{
    EFI_STATUS Status;
    EFI_GUID gEfiGopProtocolGuid = EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID;
    EFI_GUID gEfiEdidActiveProtocolGuid = EFI_EDID_ACTIVE_PROTOCOL_GUID;
    EFI_GUID gEfiEdidDiscoveredProtocolGuid = EFI_EDID_DISCOVERED_PROTOCOL_GUID;
    EDID_DATA_BLOCK *EdidDataBlock;
    EFI_EDID_ACTIVE_PROTOCOL *Edid;
    EFI_HANDLE *HandleList;
    EFI_HANDLE Handle;
    UINTN  Size = 0;
    UINTN  NumHandles = 0;
    int i;
    InitializeLib(image, systab);
    Status = uefi_call_wrapper(BS->LocateHandle, 5, ByProtocol,
                               &amp;gEfiGopProtocolGuid, NULL, &amp;Size, HandleList);
    if (Status == EFI_BUFFER_TOO_SMALL) {
         HandleList = AllocateZeroPool(Size + sizeof(EFI_HANDLE));
         if (HandleList == NULL) {
             Print(L"ERROR: Unable to allocate memory\n");
             return Status;
         }
        Status = uefi_call_wrapper(BS->LocateHandle, 5, ByProtocol,
                                   &amp;gEfiGopProtocolGuid, NULL, &amp;Size, HandleList)
;
         HandleList[Size/sizeof(EFI_HANDLE)] = NULL;
    }
    if (EFI_ERROR(Status)) {
        if (HandleList != NULL) {
            FreePool(HandleList);
            Print(L"ERROR: No handles\n");
            return Status;
        }
    }
    NumHandles = Size/sizeof(EFI_HANDLE);
    if (NumHandles == 0) {
        Print(L"No handles found\n");
        return EFI_SUCCESS;
    }
    for (i = 0; i < NumHandles; i++) {
        Handle = HandleList[i];
        VOID *Interface;
        unsigned mode;
        struct efi_gop_mode_info *info = NULL;
        UINTN size;
        gop = efi_open_protocol(Handle, &amp;gEfiGopProtocolGuid);
        if (!gop) {
            return EFI_SUCCESS;
        }
        for (mode = 0; mode < gop->mode->max_mode; mode++) {
            Status = uefi_call_wrapper(gop->query_mode, 4, gop, mode, &amp;size, &amp;info)
;
            if (Status) {
```



```
                info = 0;
                continue;
            }
        }
        Status = uefi_call_wrapper(BS->HandleProtocol, 3, Handle,
                        &amp;amp;gEfiEdidDiscoveredProtocolGuid, &amp;amp;Edid);
        if (Status == EFI_SUCCESS) {
            if (!CheckForValidEdid(Edid)) {
                PrintEdid(Edid);
            } else {
                Print(L"ERROR: Invalid EDID checksum\n");
            }
        }
    }
    return EFI_SUCCESS;
}
```

Lots more useful data is available from EDID. I only parsed and outputted a small amount of the data that is available to keep the demo code to a reasonable size. If you are reading this post, I assume that you are reasonably familiar with UEFI concepts including the GOP protocol, and how to build command line applications using the *gnu_efi* package, so I am not going to explain the above code in detail.

Using the above code, here is what was outputted for a 14″ Lenovo laptop:

```
       EDID Version: 1
      EDID Revision: 3
 Vendor Abbreviation: LGD
         Product ID: 00000362
      Serial Number: 00000000
   Manufacture Week: 0
   Manufacture Year: 2011
        Video Input: Digital
       Display Type: Monochrome
  Max Horizonal Size: 31 cm
   Max Vertical Size: 17 cm
              Gamma: 2.20
Horizonal Image Size: 309 mm
 Vertical Image Size: 174 mm
HoriImgSzByVertImgSz: 16
   Horizontal Border: 0
     Vertical Border: 0
```

I am not sure why the display type is monochrome rather than color. I need to investigate that issue further when I get some spare time (think longhaul flight!)

Feel free to use this code. I would however appreciate it if you would respect my copyright notice and include it at the top of the code. I have placed a copy of this code together with a Makefile and the two necessary headers on GitHub under the *showedid* subdirectory. To compile the code, you need to have the *gnu_efi* package installed.

By the way, no future versions of the EDID standard are planned. DisplayID is a VESA standard first published in 2009 which is intended to replace EDID and E-EDID. It features variable-length structures which encompass all existing EDID extensions as well as new extensions for 3D displays, embedded displays and more. However, I suspect that EDID will live on for a long time simply because of the enormous number of EDID-enabled monitors that are out there.

