Future Technology Devices International Ltd.

Application Note

AN_158

Vinculum-II Webcam Application

Using OLED Display

Document Reference No.: FT_000355

Version 1.0

Issue Date: 2010-11-01

This application note provides an example of using the FTDI Vinculum-II (VNC2) to communicate with a Webcam and an OLED display. Drivers and source code are also provided (downloadable from the FTDI website)

Future Technology Devices International Limited (FTDI)

Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom
Tel.: +44 (0) 141 429 2777 Fax: + 44 (0) 141 429 2758
E-Mail (Support): **support1@ftdichip.com** Web: http://www.ftdichip.com

Table of Contents

1 Introduction

This application note describes an application that uses Vinculum-II to display images, obtained from a webcam, on an OLED display.

The source code for the application is provided as an example and is neither guaranteed nor supported by FTDI. All source code for the application can be downloaded from the following location on the FTDI website: http://www.ftdichip.com/Support/SoftwareExamples/VinculumIIProjects/Vinculum-II_Webcam_Application_Using_OLED_Display.zip

1.1 Overview

This application note describes the design and implementation of the webcam application. The functionality of the application is tightly coupled to the hardware used, and as a result, this note covers only the minimum steps necessary to display images on the chosen hardware. There is no attempt at generality: that will be the subject of a future enhancement.

The application was written in C and developed using the Vinculum-II IDE.

1.2 Hardware Requirements

- V2EVAL board
- 64-pin Vinculum II daughter board
- Logitech Webcam Pro 9000
- Densitron OLED display module
- SEPS525 160 RGB x 128 dots, 262K Colours PM-OLED Display Driver and Controller
- 5V:12V DC to Dc converter to power OLED display

(See Appendix A for references to the datasheets)

With this hardware, a frame rate of 4.4 frames/sec at a resolution of 160x120 pixels is achieved.

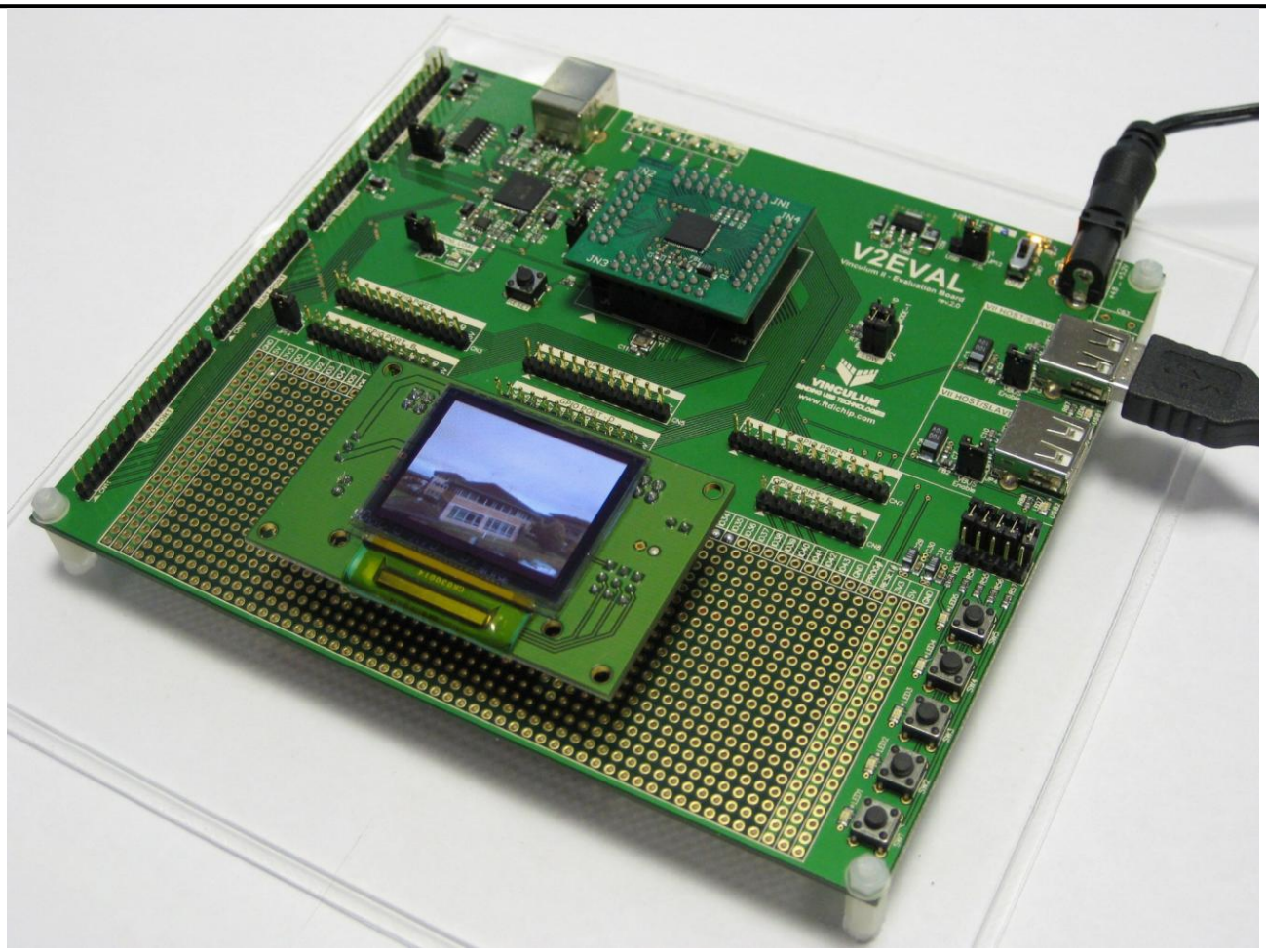

Photo 1 : V2EVAL Board with OLED Display

2 Application Architecture

The application corresponds to the VNC2 firmware model defined in [7]. It consists of two threads: ***usbReader*** is responsible for attaching to the webcam which has been enumerated by the USB host, configuring the webcam, and reading and storing the webcam data stream; ***DisplayOutput*** is responsible for converting the webcam data, in YUV format, to RGB format, and sending the RGB data to the display. Dual buffers are used for storing data, and access to the data is synchronised using semaphores.

The architecture shown in Figure 1 is that of a standard producer-consumer application:

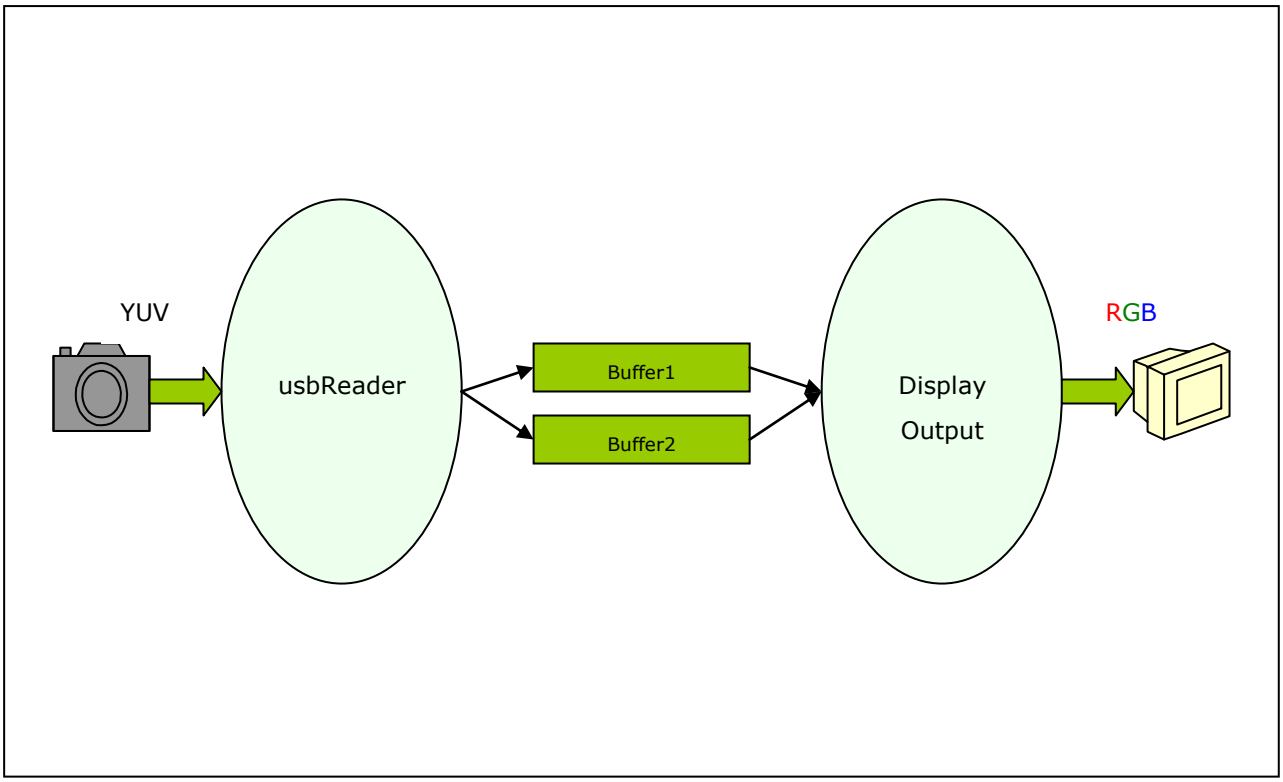

Figure 1: Application Architecture

3 Hardware Interface

The Densitron OLED has two operational modes, both used by the application. Bus interface mode is used at startup to configure the display characteristics of the OLED. When the OLED is configured, the application switches it to RGB interface mode to send RGB data to the OLED. RGB interface mode is the operational mode used when the OLED is displaying RGB data.

Each operational mode has its own set of control signals and these are described in the following sections.

3.1.1 Bus Interface

The Densitron OLED is configured to operate in 8-bit bus interface mode. Pins on the SEPS525 (driver and controller module) are interfaced to GPIO lines on Vinculum-II as shown in Figure 2. See [2] for the SEPS5215 pin descriptions.

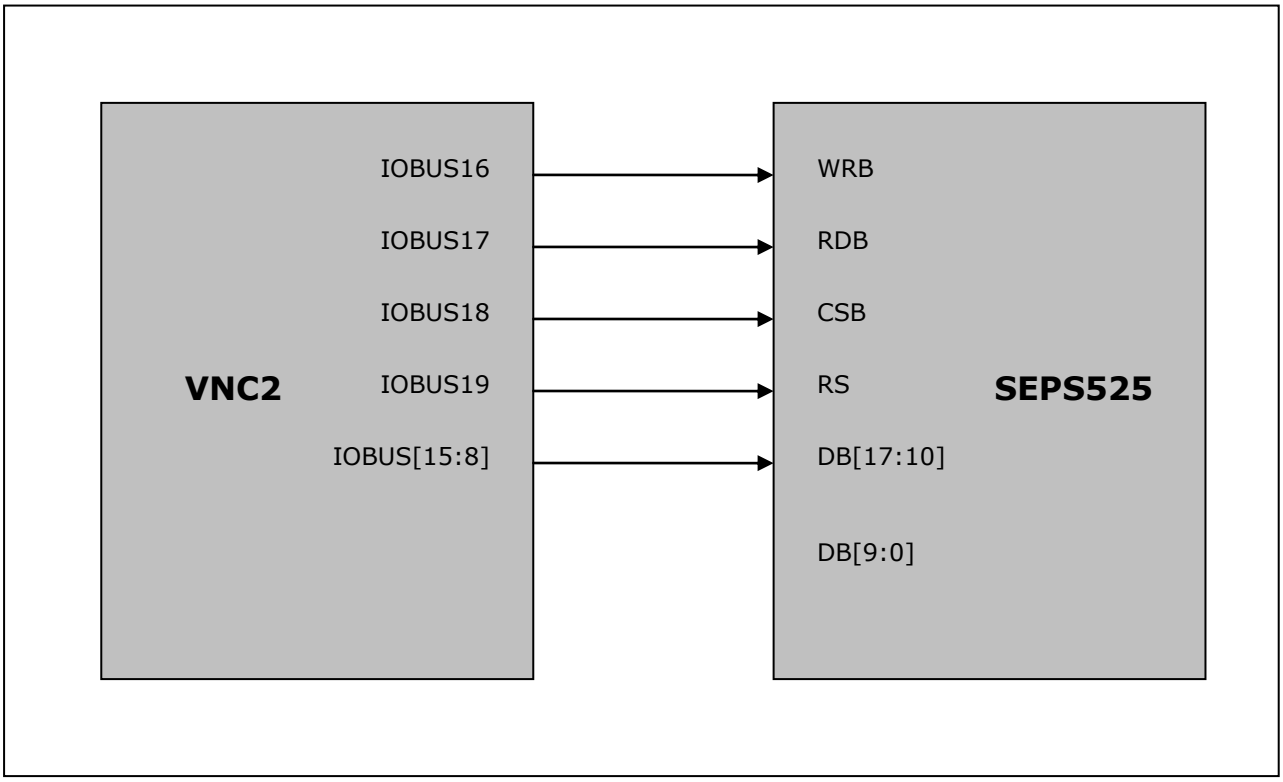

Figure 2: 8-bit Bus Interface Mode

3.1.2 RGB Interface

The Densitron OLED is configured to operate in 6-bit RGB interface mode. Pins on the SEPS525 are interfaced to GPIO lines on Vinculum-II as shown in Figure 3. See [2] for the SEPS5215 pin descriptions.

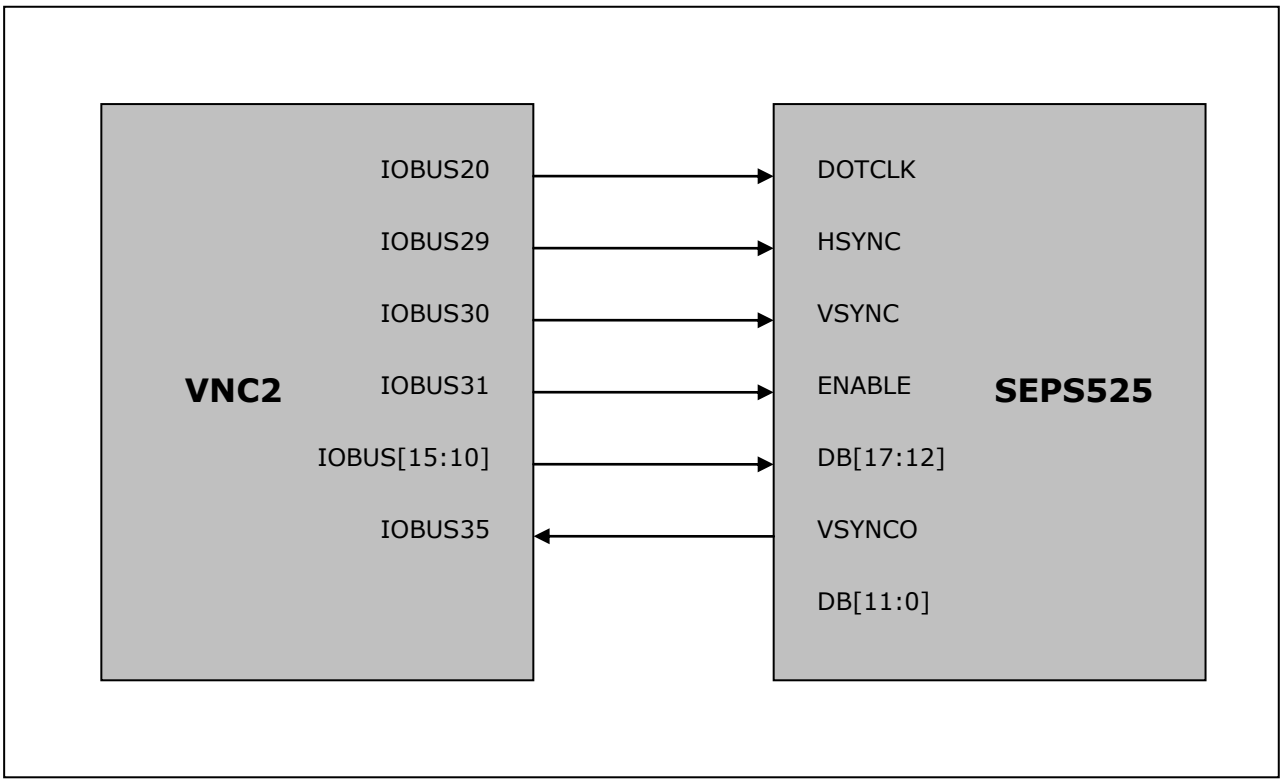

Figure 3: 6-bit RGB Interface Mode

4 Developing the Application

This section explains how to write an application interfacing a webcam to an OLED display. This section anticipates that the user has a sound understanding of application structure and the fundamental concepts of how the Vinculum Operating System (VOS) works. For an introduction to VinIDE, VOS and an overview of the general application structure please refer to the Vinculum-II tool-chain Getting Started Guide available from the FTDI website. The Vinculum-II tool-chain consists of a compiler, assembler, linker and debugger encapsulated in a graphical interface (the IDE) and is used to develop customised firmware for VNC2.

The following example demonstrates how to build a sample application from a blank project using the IDE. The sample demonstrates receiving YUV data from a webcam through an isochronous USB endpoint, converting the YUV data to RGB format, and outputting RGB data through a GPIO interface to an OLED display. A complete listing of the sample code is available at Appendix B at the end of this applications note. This sample source code has been tested but is provided for illustration only and is not supported by FTDI.

4.1 IOMux Setup

The VNC2 must be configured using the IOMux such that the signals coming from the device match the pins on the V2EVAL board. The following IOMux configuration source code is for the pin connections as defined within Section 3 connected to a 64 pin VNC2 daughter card. FTDI provides an IOMux configuration utility, as part of the IDE, to aid with configuring IOMux signals. For more information on IOMux and the IOMux configuration utility, see [4] and [6].

Pins 19, 20, 22, 23, 24, 25, 26 and 27 of VNC2 are the bidirectional data interface connections and are controlled by GPIO Port A. Pins 28, 29, 31, 32, 39, 48, 49 and 50 of VNC2 are outputs on the control interface and are controlled by GPIO Port B. Also on the control interface, VNC2 Pin 55 is an output and Pin 56 is an input. These pins are controlled by GPIO Port D.

```
unsigned char packageType = vos_get_package_type();

if (packageType == VINCULUM_II_64_PIN) {

    // GPIO Port A 0 to pin 19 as Bi-Directional, IOBUS8
    vos_iomux_define_bidi(19, IOMUX_IN_GPIO_PORT_A_0, IOMUX_OUT_GPIO_PORT_A_0);
    // GPIO Port A 1 to pin 20 as Bi-Directional, IOBUS9
    vos_iomux_define_bidi(20, IOMUX_IN_GPIO_PORT_A_1, IOMUX_OUT_GPIO_PORT_A_1);
    // GPIO Port A 2 to pin 22 as Bi-Directional, IOBUS10
    vos_iomux_define_bidi(22, IOMUX_IN_GPIO_PORT_A_2, IOMUX_OUT_GPIO_PORT_A_2);
    // GPIO Port A 3 to pin 23 as Bi-Directional, IOBUS11
    vos_iomux_define_bidi(23, IOMUX_IN_GPIO_PORT_A_3, IOMUX_OUT_GPIO_PORT_A_3);
    // GPIO Port A 4 to pin 24 as Bi-Directional, IOBUS12
    vos_iomux_define_bidi(24, IOMUX_IN_GPIO_PORT_A_4, IOMUX_OUT_GPIO_PORT_A_4);
    // GPIO Port A 5 to pin 25 as Bi-Directional, IOBUS13
    vos_iomux_define_bidi(25, IOMUX_IN_GPIO_PORT_A_5, IOMUX_OUT_GPIO_PORT_A_5);
    // GPIO Port A 6 to pin 26 as Bi-Directional, IOBUS14
    vos_iomux_define_bidi(26, IOMUX_IN_GPIO_PORT_A_6, IOMUX_OUT_GPIO_PORT_A_6);
    // GPIO Port A 7 to pin 27 as Bi-Directional, IOBUS15
    vos_iomux_define_bidi(27, IOMUX_IN_GPIO_PORT_A_7, IOMUX_OUT_GPIO_PORT_A_7);

    // CONTROL INTERFACE
    //
    // GPIO Port B 0 to pin 28 as Output. - WR
    vos_iomux_define_output(28, IOMUX_OUT_GPIO_PORT_B_0);   // IOBUS16
    // GPIO Port B 1 to pin 29 as Output. - RD
    vos_iomux_define_output(29, IOMUX_OUT_GPIO_PORT_B_1);   // IOBUS17
    // GPIO Port B 2 to pin 31 as Output. - CS
    vos_iomux_define_output(31, IOMUX_OUT_GPIO_PORT_B_2);   // IOBUS18
    // GPIO Port B 3 to pin 32 as Output. - RS
    vos_iomux_define_output(32, IOMUX_OUT_GPIO_PORT_B_3);   // IOBUS19
    // GPIO Port B 4 to pin 39 as Output. - DOTCLK
    vos_iomux_define_output(39, IOMUX_OUT_GPIO_PORT_B_4);   // IOBUS20
    // GPIO Port B 5 to pin 48 as Output. - HSYNC
    vos_iomux_define_output(48, IOMUX_OUT_GPIO_PORT_B_5);   // IOBUS29
    // GPIO Port B 6 to pin 49 as Output. - VSYNC
```

```
    vos_iomux_define_output(49, IOMUX_OUT_GPIO_PORT_B_6);    // IOBUS30
    // GPIO Port B 7 to pin 50 as Output. - ENABLE
    vos_iomux_define_output(50, IOMUX_OUT_GPIO_PORT_B_7);    // IOBUS31
    // GPIO Port D 6 to pin 55 as Output. - RES
    vos_iomux_define_output(55, IOMUX_OUT_GPIO_PORT_D_6);    // IOBUS34
    // GPIO Port D 7 to pin 56 as Input.
    vos_iomux_define_input(56, IOMUX_IN_GPIO_PORT_D_7);      // IOBUS35
}
```

4.2 UVC Driver

Support for the webcam is provided by the UVC driver. This driver integrates fully with the VOS Device Manager, but contains only the minimum functionality necessary for this application.

Note that the UVC driver component is not supplied with the IDE. Instead, it is implemented as application code, albeit using the VOS Device Manager driver model [7]. This section describes the functionality of the UVC driver.

4.2.1 Return Codes

The return codes used by the driver are as follows:

UVC_OK
: The command completed successfully.

UVC_INVALID_PARAMETER
: There was an error or problem with a parameter sent to the driver.

UVC_NOT_FOUND
: The USB Host was not found in an attach operation.

UVC_ERROR
: An unspecified error occurred.

4.2.2 uvc_init()

Syntax

```
unsigned char uvc_init(unsigned char devNum)
```

Description

Initialise the UVC driver and register the driver with the Device Manager.

Parameters

The device number to use when registering the driver with the Device Manager is passed in the devNum parameter.

Returns

The function returns zero if successful and non-zero if it could not initialise the driver or allocate memory for the driver.

Comments

The driver can be attached to an UVC device once the USB Host enumeration has completed.

4.2.3 Open and Close Operations

Before requests can be sent to the UVC driver, it must be opened and a valid handle obtained. Opening the UVC driver is performed by the standard Device Manager request, vos_dev_open(). To close the driver and invalidate the handle, call the standard Device Manager request, vos_dev_close().

4.2.4 Read and Write Operations

The UVC driver must have been opened previously and a valid handle obtained. Reading data from the UVC device is performed by the standard Device Manager request, vos_dev_read().

Write requests are not required for this application.

4.2.5 IOCTL Calls

Calls to the UVC driver's IOCTL method take the form:

```
typedef struct _uvc_ioctl_cb_t
{
    unsigned char ioctl_code;
    // read buffer
    unsigned char *get;
    // write butter
    unsigned char *set;

} uvc_ioctl_cb_t;
```

The following IOCTL request codes are supported by the UVC driver:

VOS_IOCTL_UVC_ATTACH	Attaches the UVC driver to a USB interface device
VOS_IOCTL_UVC_DETACH	Detaches the UVC driver from a USB interface device
VOS_IOCTL_UVC_CLASS_REQUEST	Sends a UVC class request

4.2.5.1 VOS_IOCTL_UVC_ATTACH

Description

Attaches a UVC device to a USB interface device.

Parameters

The device interface handle must be passed in the *set* member of the *uvc_ioctl_cb_t* structure.

Returns

There is no data returned by this call, although the return indicates the success or otherwise of the attach.

Example

```
VOS_HANDLE hUsbHost, hUvc;
uvc_ioctl_cb_t uvc_iocb;

hUsbHost = vos_dev_open(VOS_DEV_USBHOST);
hUvc = vos_dev_open(VOS_DEV_UVC);


uvc_iocb.ioctl_code = VOS_IOCTL_UVC_ATTACH;
uvc_iocb.set = (void *) hUsbHost;
if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    // ERROR
}
```

4.2.5.2 VOS_IOCTL_UVC_DETACH

Description

Detaches a UVC device from a USB interface device.

Parameters

There are no parameters.

Returns

There is no data returned by this call, although the return indicates the success or otherwise of the detach.

Example

```
VOS_HANDLE hUsbHost, hUvc;
uvc_ioctl_cb_t uvc_iocb;

hUsbHost = vos_dev_open(VOS_DEV_USBHOST);
hUvc = vos_dev_open(VOS_DEV_UVC);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_ATTACH;
uvc_iocb.set = (void *) hUsbHost;
if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    // ERROR
}

...

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_DETACH;
if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    // ERROR
}
```

4.2.5.3 VOS_IOCTL_CLASS_REQUEST

Description

Sends a class request to the UVC device.

Parameters

The request data must be passed in a *usb_deviceRequest_t* structure whose address is passed in the *set* member of the *uvc_ioctl_cb_t* structure.

Returns

The returned data is placed in storage whose address is passed in the *get* member of the *uvc_ioctl_cb_t* structure.

Comments

Supported UVC requests are as follows (see [1], Section 4):

```
SET_CUR,              //0x01
GET_CUR= 0x81,        //0x81
GET_MIN,              //0x82
GET_MAX,              //0x83
GET_RES,              //0x84
GET_LEN,              //0x85
GET_INFO,             //0x86
GET_DEF               //0x87
```

Example

```
VOS_HANDLE hUsbHost, hUvc;
uvc_ioctl_cb_t uvc_iocb;
usb_deviceRequest_t desc_dev;
VideoProbeAndCommiteControl_t VProbeAndCom;

hUsbHost = vos_dev_open(VOS_DEV_USBHOST);
hUvc = vos_dev_open(VOS_DEV_UVC);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_ATTACH;
uvc_iocb.set = (void *) hUsbHost;
if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    // ERROR
}

desc_dev.bRequest = 0x81; //GET_CUR = 0x81
desc_dev.wValue = (1<<8);
desc_dev.wIndex = 1;
desc_dev.wLength = 34;

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
uvc_iocb.set = &desc_dev;
uvc_iocb.get = &VProbeAndCom;

if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    // ERROR;
}
```

4.3 Application Setup

The Webcam sample requires the following FTDI provided components: USBHOST driver, GPIO driver, UVC driver; VOS kernel and the stdio library.

4.3.1 Adding Driver Libraries

Each of the device drivers to be used in the application must be added to the current project within the IDE. Device drivers come in the form of library files that are all bundled with the IDE installer. Library files are easily added to the current project using the Project Libraries window within the IDE. For help with adding library files to VNC2 projects please refer to [5].

4.3.2 Initializing Drivers

Prior to starting the VNC2 RTOS scheduler (VOS) within the **main()** routine, all drivers to be used within the system must be initialized; that is, have the init routine called for each driver. The listing below demonstrates initializing each of the device drivers required for the sample.

After each of the drivers has been initialized the scheduler is called to start all application threads; there are two threads in the webcam example application.

```
void main(void)
{
    // GPIO IOCTL request block
    gpio_ioctl_cb_t gpio_iocb;


    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);


    gpioContext.port_identifier = GPIO_PORT_A;
    gpio_init(VOS_DEV_GPIO, &gpioContext);

    // configure USB Host port 1 only
    // use a max of 4 USB devices
    usbhostContext.if_count = 16;
    usbhostContext.ep_count = 10;
    usbhostContext.xfer_count = 2;
    usbhostContext.iso_xfer_count = 36;

    usbhost_init(VOS_DEV_USBHOST, -1, &usbhostContext);

    uvc_init(VOS_DEV_UVC);

    // do any required setup on devices
    tcbUsbRd = vos_create_thread(31,SIZEOF_FIRMWARE_TASK_MEMORY,usbReader,0);
    if (tcbUsbRd == 0xffff)
    {
        asm{HALT};
    }

    tcbDispOut = vos_create_thread(30,SIZEOF_FIRMWARE_TASK_MEMORY,displayOutput,0);

    if (tcbDispOut == 0xffff)
    {
        asm{HALT};
    }

    vos_start_scheduler();

main_loop:
    goto main_loop;
}
```

4.3.3 Opening Drivers

The thread usbReader, created in the main routine, contains the code for opening each of the device handles. The VOS_HANDLE obtained from a vos_dev_open is used throughout the application to reference each respective driver layer.

```
VOS_HANDLE hUsbHost, hUvc, hGpio;

void usbReader(void)
{

    hGpio = vos_dev_open(VOS_DEV_GPIO);
    hUsbHost = vos_dev_open(VOS_DEV_USBHOST);
    hUvc = vos_dev_open(VOS_DEV_UVC);

    while (1) {

        // usbReader code goes here

    }
}
```

4.3.4 Attaching to a UVC Device

The webcam is a USB Video Class (UVC) device and is supported by the UVC driver. When the webcam is connected to USB port 1, it is enumerated by the USB Host – which is VNC2 USB Port 1. The application makes a connection between the UVC driver and the USB Host by sending a VOS_IOCTL_UVC_ATTACH request to the UVC driver and passing the handle of the USB host as a parameter. VOS_IOCTL_UVC_ATTACH request blocks until the host has enumerated the webcam on its port. When the VOS_IOCTL_UVC_ATTACH request returns, the webcam is connected to the USB Host port, and the application is able to send requests via the UVC driver to the webcam. The driver architecture is shown in Figure 4.

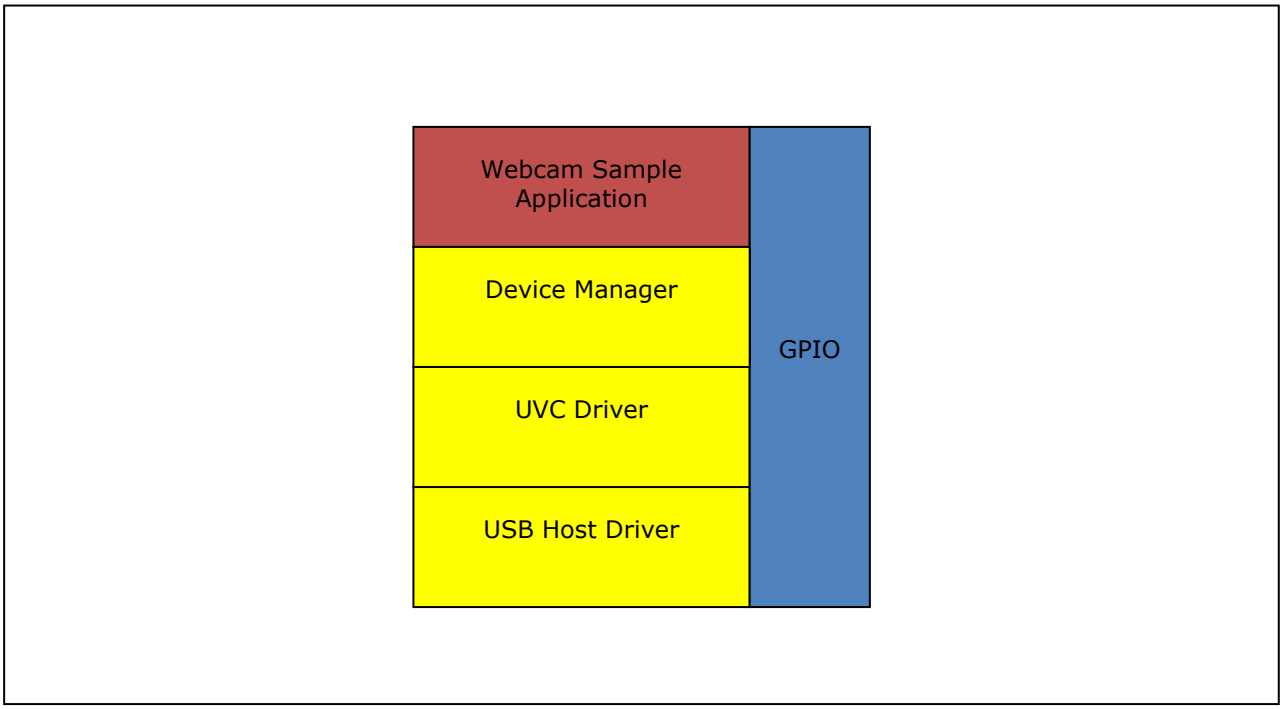

Figure 4: Driver Architecture

4.3.5 Configuring a UVC Device

As described above, the control code VOS_IOCTL_UVC_CLASS_REQUEST sends a request to a UVC device. VOS_IOCTL_UVC_CLASS_REQUEST IOCTLs are handled in bus interface mode. When configuration has been completed, the application switched to RGB interface mode to process YUV data from the webcam.

4.3.6 Processing the YUV Data

YUV data is received in a frame of 192 bytes: the first 12 bytes constitute the payload header, and the remaining 180 bytes are YUV data. The application uses the information in the header to synchronise with the data stream. The remaining bytes in the data stream are YUV data; the application converts this to RGB format suitable for displaying on the OLED display.

The payload header is defined in [1], Section 2.4.3.3, Table 2-5. For this application, the significant bits in the header are contained in the **bmHeaderInfo** field: D1 – End-of-Frame; D6 – Error.

To synchronise with the frame data, the application waits for End-of-Frame; during this phase, all other data is ignored. When the application is synchronised with the frame data, the reception of Error means that a re-synchronisation is required; during this phase, the application ignores all data until End-of-Frame is received, and synchronisation is re-established.

In addition, because of the nature of the data stream for this particular application configuration, it is necessary to perform a re-synchronisation after 1494 frames have been received.

The relationship between the YUV processing states is shown in Figure 5.

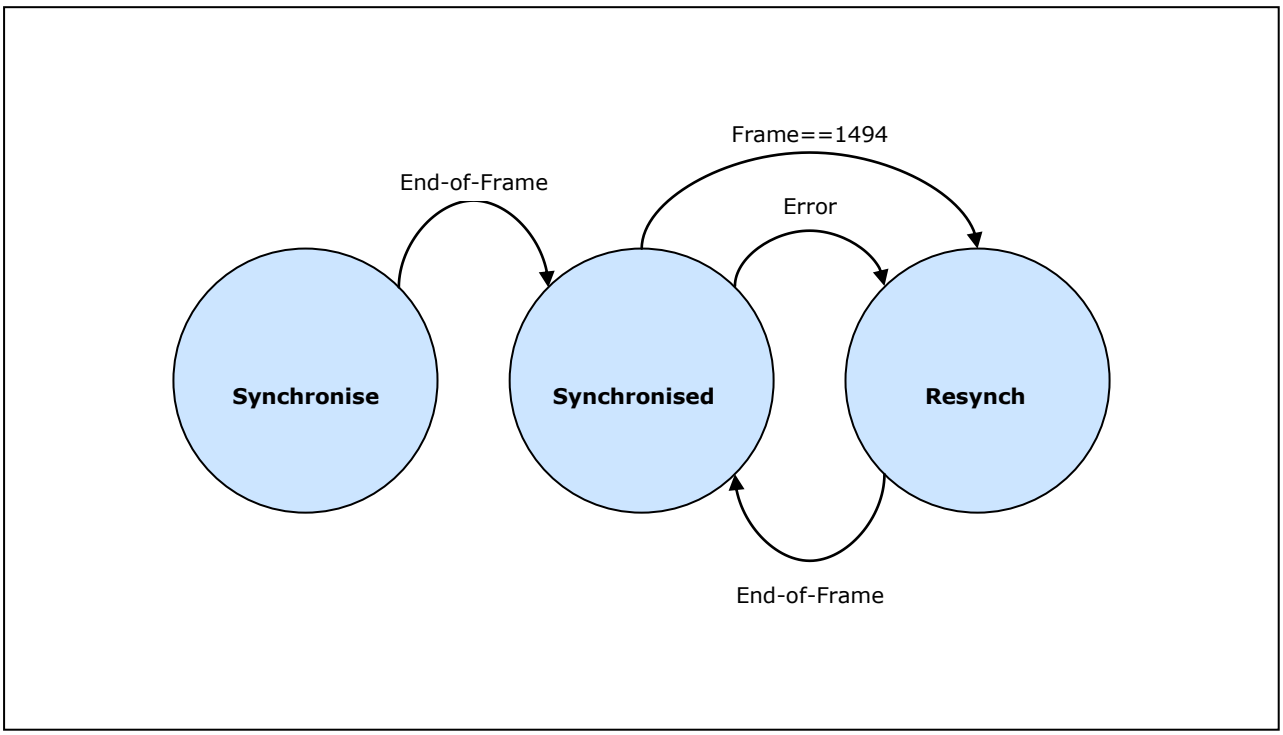

Figure 5: YUV Processing States

4.4 YUV to RGB Conversion

Data from the webcam is received in YUV pixel format. It is necessary to convert this to RGB format for the OLED display. The YUV format used is Y'UV444: 4 bytes Y'UV are converted to 6 bytes RGB. The webcam example code that approximates the conversion algorithm is as follows:

```
unsigned char Blue, Green, Red;
unsigned char Y0, Y1, U, V;
unsigned char *inBuf;
unsigned char size;

void YUY2RGBConvert (unsigned char *inputBuffer1 )
{
    inBuf = inputBuffer1;

     while (size)
     {
         //size -= 4;
         asm { DEC8 size $4; };

         //  Y0 = (*inBuf++) - 16;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $16
            CPY8    Y0    %r0
            }
           //U = (*inBuf++) - 128;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $128
            CPY8    U    %r0
            }
           //Y1 = (*inBuf++) - 16;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $16
            CPY8    Y1    %r0
            }
           //V = (*inBuf++) - 128;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $128
            CPY8    V    %r0
            }

           //Red = Y0 + V;
        asm {
            ADD8    Red    Y0    V;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Red;
        gpio_port_b = 0x70;

           //Green = Y0 - V - U;
        asm {
            SUB8    Green    Y0    V;
            SUB8    Green    U;
```

```
            }
        gpio_port_b = 0x60;
        gpio_port a = Green;
        gpio_port b = 0x70;

           //Blue = Y0 + U ;
        asm {
            ADD8    Blue    Y0    U;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Blue;
        gpio_port_b = 0x70;

           //Red = Y1 + V;
        asm {
            ADD8    Red    Y1    V;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Red;
        gpio_port_b = 0x70;

           //Green = Y1 - V - U;
        asm {
            SUB8    Green    Y1    V;
            SUB8    Green    U;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Green;
        gpio_port_b = 0x70;

           //Blue = Y1 + U;
        asm {
            ADD8    Blue    Y1    U;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Blue;
        gpio_port_b = 0x70;

       }
       return;

}
```

In addition to converting YUV data to RGB format, the example code shows data being output to the OLED display. This is done with direct access to the appropriate GPIO port. The timing for the 6-bit RGB interface is shown in [2], p.13.

5 Contact Information

Head Office – Glasgow, UK

Future Technology Devices International Limited
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales1@ftdichip.com
E-mail (Support) support1@ftdichip.com
E-mail (General Enquiries) admin1@ftdichip.com
Web Site URL http://www.ftdichip.com
Web Shop URL http://www.ftdichip.com

Branch Office – Taipei, Taiwan

Future Technology Devices International Limited (Taiwan)
2F, No. 516, Sec. 1, NeiHu Road
Taipei 114
Taiwan , R.O.C.
Tel: +886 (0) 2 8791 3570
Fax: +886 (0) 2 8791 3576

E-mail (Sales) tw.sales1@ftdichip.com
E-mail (Support) tw.support1@ftdichip.com
E-mail (General Enquiries) tw.admin1@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Hillsboro, Oregon, USA

Future Technology Devices International Limited (USA)
7235 NW Evergreen Parkway, Suite 600
Hillsboro, OR 97123-5803
USA
Tel: +1 (503) 547 0988
Fax: +1 (503) 547 0987

E-Mail (Sales) us.sales@ftdichip.com
E-Mail (Support) us.support@ftdichip.com
E-Mail (General Enquiries) us.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Branch Office – Shanghai, China

Future Technology Devices International Limited (China)
Room 408, 317 Xianxia Road,
Shanghai, 200051
China
Tel: +86 21 62351596
Fax: +86 21 62351595

E-mail (Sales) cn.sales@ftdichip.com
E-mail (Support) cn.support@ftdichip.com
E-mail (General Enquiries) cn.admin@ftdichip.com
Web Site URL http://www.ftdichip.com

Distributor and Sales Representatives

Please visit the Sales Network page of the FTDI Web site for the contact details of our distributor(s) and sales representative(s) in your country.

6 Appendix A – References

Document References

[1] USB Implementers Forum, *Universal Serial Bus Device Class Definition for Video Devices Revision 1.1*, June 1, 2005.

[2] SYNCOAM, *SEPS525 160 RGB x 128 Dots, 262K Colors PM-OLED Display Driver and Controller, Version 0.20*, SYNCOAM Co., Ltd, April 14th 2006.

[3] Densitron Displays, *OLED Display Module DD-160128FC-1A/2A with EVK Board*, Densitron Technologies, 2006.

[4] FTDI Application Note 139, *IO_MUX Explained*, FTDI, 2010.

[5] FTDI Application Note 142, *Vinculum-II_Tool_Chain_Getting_Started_Guide*, FTDI, 2010.

[6] FTDI Application Note 144, *Vinculum-II_IO_Mux_Config_Utility_User_Guide*, FTDI, 2010.

[7] FTDI Application Note 151, *Vinculum-II User Guide*, FTDI, 2010.

Acronyms and Abbreviations

Terms	Description
UVC	Universal Serial Bus Video Class
IOMux	Input Output Multiplexer – Used to configure pin selection on different package types of the VNC2.
V2EVAL	Vinculum II Evaluation Board- Customer evaluation board for the VNC2 allowing prototype development.
VOS	Vinculum Operating System
IDE	Integrated Development Environment
VNC2	Vinculum II

7 Appendix B – Code Listing

```
/*
This software is provided by Future Technology Devices International Limited "as is" and
any express or implied warranties, including, but not limited to, the implied warranties
of merchantability and fitness for a particular purpose are disclaimed. In no event shall
future technology devices international limited be liable for any direct, indirect,
incidental, special, exemplary, or consequential damages (including, but not limited to,
procurement of substitute goods or services; loss of use, data, or profits; or business
interruption) however caused and on any theory of liability, whether in contract, strict
liability, or tort (including negligence or otherwise) arising in any way out of the use
of this software, even if advised of the possibility of such damage.
*/

#include "vos.h"
#include "USBHost.h"
#include "USB.h"
#include "GPIO.h"
#include "FIFO.h"
#include "UVC.h"
#include "SEPS525.h"


#define SIZEOF_FIRMWARE_TASK_MEMORY 0x800
#define VOS_QUANTUM                 50
#define VOS_TICK_INTERVAL           1


#define NUMBER_OF_DEVICES           4
#define VOS_DEV_USBHOST             0
#define VOS_DEV_FIFO                1
#define VOS_DEV_GPIO                2
#define VOS_DEV_UVC                 3


VOS_HANDLE hUsbHost, hFifo, hGpio;
VOS_HANDLE hUvc;

vos_tcb_t *tcbUsbRd, *tcbDispOut;


// GPIO context structure
gpio_context_t gpioContext;
usbhost_context_t usbhostContext;

fifo_context_t fifo_ctx;

void usbReader(void);
void displayOutput(void);


#define ISO_TRANSFER_SIZE     192
#define ISO_TRANSFER_COUNT    1
#define WEBCAM_HEADER_SIZE    12
#define WEBCAM_PAYLOAD_SIZE   (ISO_TRANSFER_SIZE - WEBCAM_HEADER_SIZE)


#define BUF_SIZE              (ISO_TRANSFER_SIZE * ISO_TRANSFER_COUNT)

// buffers and mutexes to protect them
unsigned char buffer1[BUF_SIZE];
```

```
unsigned char buffer2[BUF_SIZE];
vos_semaphore_t sb1_avail, sb1_full;
vos_semaphore_t sb2_avail, sb2_full;


#ifdef _VNC2
port gpio_port_a@0x186;
port gpio_port_b@0x187;
port gpio_port_c@0x188;
#endif

unsigned int data;
unsigned int pxcount;
unsigned int xfercount;


typedef enum _VS_InterfaceControlStatus_e
{
    VS_CONTROL_UNDEFINED=0x00,        //0x00
    VS_PROBE_CONTROL,                 //0x01
    VS_COMMIT_CONTROL,                //0x02
    VS_STILL_PROBE_CONTROL,           //0x03
    VS_STILL_COMMIT_CONTROL,          //0x04
    VS_STILL_IMAGE_TRIGGER_CONTROL,   //0x05
    VS_STREAM_ERROR_CODE_CONTROL,     //0x06
    VS_GENERATE_KEY_FRAME_CONTROL,    //0x07
    VS_UPDATE_FRAME_SEGMENT_CONTROL,  //0x08
    VS_SYNCH_DELAY_CONTROL            //0x09
}VS_InterfaceControlStatus_e;

typedef enum _VS_ControlRequest_e
{
    RC_UNDEFINED=0x00,    //0x00
    SET_CUR,              //0x01
    GET_CUR= 0x81,        //0x81
    GET_MIN,              //0x82
    GET_MAX,              //0x83
    GET_RES,              //0x84
    GET_LEN,              //0x85
    GET_INFO,             //0x86
    GET_DEF               //0x87
}VS_ControlRequest_e;

typedef struct _VideoProbeAndCommiteControl_t
{

    unsigned short bmHint;
    unsigned char bFormatIndex;
    unsigned char bFrameIndex;
    unsigned int dwFrameInterval;
    unsigned short wKeyFrameRate;
    unsigned short wPFrameRate;
    unsigned short wCompQuality;
    unsigned short wCompWindowSize;
    unsigned short wDelay;
    unsigned int dwMaxVideoFrameSize;
    unsigned int dwMaxPayloadTransferSize;
    unsigned int dwClockFrequency;
    unsigned char bmFramingInfo;
    unsigned char bPreferedVersion;
    unsigned char bMinVersion;
    unsigned char bMaxVersion;
```

```
} VideoProbeAndCommiteControl_t;

VideoProbeAndCommiteControl_t VProbeAndCom;
VideoProbeAndCommiteControl_t VProbeAndComMax;
VideoProbeAndCommiteControl_t VProbeAndComMin;


//void yuv2rgb(void);
void yuv2rgb(unsigned int addr,unsigned char short_packet);
void process_data_stream(unsigned char *buf,unsigned short num_bytes);


void main(void)
{
    // GPIO IOCTL request block
    gpio_ioctl_cb_t gpio_iocb;
    unsigned char packageType;


    vos_set_clock_frequency(VOS_48MHZ_CLOCK_FREQUENCY);
    vos_init(VOS_QUANTUM, VOS_TICK_INTERVAL, NUMBER_OF_DEVICES);


    packageType = vos_get_package_type();

    if (packageType == VINCULUM_II_32_PIN) {
        asm{HALT};  // 32-pin package not supported for the current board setup
    }
    else if (packageType == VINCULUM_II_48_PIN){
        asm{HALT};  // 48-pin package not supported for the current board setup
    }
    else if (packageType == VINCULUM_II_64_PIN) {
    // DATA INTERFACE
        // GPIO Port A 0 to pin 28 as Bi-Directional.
        vos_iomux_define_bidi(19, IOMUX_IN_GPIO_PORT_A_0,
IOMUX_OUT_GPIO_PORT_A_0);   // IOBUS8
        // GPIO Port A 1 to pin 29 as Bi-Directional.
        vos_iomux_define_bidi(20, IOMUX_IN_GPIO_PORT_A_1,
IOMUX_OUT_GPIO_PORT_A_1);   // IOBUS9
        // GPIO Port A 2 to pin 31 as Bi-Directional.
        vos_iomux_define_bidi(22, IOMUX_IN_GPIO_PORT_A_2,
IOMUX_OUT_GPIO_PORT_A_2);   // IOBUS10
        // GPIO Port A 3 to pin 32 as Bi-Directional.
        vos_iomux_define_bidi(23, IOMUX_IN_GPIO_PORT_A_3,
IOMUX_OUT_GPIO_PORT_A_3);   // IOBUS11
        // GPIO Port A 4 to pin 39 as Bi-Directional.
        vos_iomux_define_bidi(24, IOMUX_IN_GPIO_PORT_A_4,
IOMUX_OUT_GPIO_PORT_A_4);   // IOBUS12
        // GPIO Port A 5 to pin 40 as Bi-Directional.
        vos_iomux_define_bidi(25, IOMUX_IN_GPIO_PORT_A_5,
IOMUX_OUT_GPIO_PORT_A_5);   // IOBUS13
        // GPIO Port A 6 to pin 41 as Bi-Directional.
        vos_iomux_define_bidi(26, IOMUX_IN_GPIO_PORT_A_6,
IOMUX_OUT_GPIO_PORT_A_6);   // IOBUS14
        // GPIO Port A 7 to pin 42 as Bi-Directional.
        vos_iomux_define_bidi(27, IOMUX_IN_GPIO_PORT_A_7,
IOMUX_OUT_GPIO_PORT_A_7);   // IOBUS15
    // CONTROL INTERFACE
        // GPIO Port B 0 to pin 43 as Output. - WR
        vos_iomux_define_output(28, IOMUX_OUT_GPIO_PORT_B_0);   // IOBUS16
        // GPIO Port B 1 to pin 44 as Output. - RD
        vos_iomux_define_output(29, IOMUX_OUT_GPIO_PORT_B_1);   // IOBUS17
        // GPIO Port B 2 to pin 45 as Output. - CS
```

```
        vos_iomux_define_output(31, IOMUX_OUT_GPIO_PORT_B_2);   // IOBUS18
        // GPIO Port B 3 to pin 46 as Output. - RS
        vos_iomux_define_output(32, IOMUX_OUT_GPIO_PORT_B_3);   // IOBUS19
        // GPIO Port B 4 to pin 47 as Output. - DOTCLK
        vos_iomux_define_output(39, IOMUX_OUT_GPIO_PORT_B_4);   // IOBUS20
//        vos_iomux_define_output(57, IOMUX_OUT_GPIO_PORT_B_4);
        // GPIO Port B 5 to pin 48 as Output. - HSYNC
        vos_iomux_define_output(48, IOMUX_OUT_GPIO_PORT_B_5);   // IOBUS29
        // GPIO Port B 6 to pin 49 as Output. - VSYNC
        vos_iomux_define_output(49, IOMUX_OUT_GPIO_PORT_B_6);   // IOBUS30
        // GPIO Port B 7 to pin 50 as Output. - ENABLE
        vos_iomux_define_output(50, IOMUX_OUT_GPIO_PORT_B_7);   // IOBUS31
        // GPIO Port D 6 to pin 55 as Output. - RES
        vos_iomux_define_output(55, IOMUX_OUT_GPIO_PORT_D_6);   // IOBUS34
        // GPIO Port D 7 to pin 56 as Input.
        vos_iomux_define_input(56, IOMUX_IN_GPIO_PORT_D_7);   // IOBUS35
    }

    gpioContext.port_identifier = GPIO_PORT_A;
    gpio_init(VOS_DEV_GPIO, &gpioContext);

    // configure USB Host port 1 only
     // use a max of 4 USB devices
    usbhostContext.if_count = 16;
    usbhostContext.ep_count = 10;
    usbhostContext.xfer_count = 2;
    usbhostContext.iso_xfer_count = 36;

    usbhost_init(VOS_DEV_USBHOST, -1, &usbhostContext);

    uvc_init(VOS_DEV_UVC);

    // do any required setup on devices
    tcbUsbRd = vos_create_thread(31,SIZEOF_FIRMWARE_TASK_MEMORY,usbReader,0);
    if (tcbUsbRd == 0xffff)
    {
        asm{HALT};
    }

    tcbDispOut =
vos_create_thread(30,SIZEOF_FIRMWARE_TASK_MEMORY,displayOutput,0);

    if (tcbDispOut == 0xffff)
    {
        asm{HALT};
    }

    vos_init_semaphore(&sb1_avail, 1);
    vos_init_semaphore(&sb1_full, 0);

    vos_init_semaphore(&sb2_avail, 1);
    vos_init_semaphore(&sb2_full, 0);

//    SEPS525_init();

    vos_start_scheduler();

main_loop:
    goto main_loop;
}
```

```
void usbReader(void)
{
   // USBHost ioctl request block
    usbhost_ioctl_cb_t hc_iocb;

    usbhost_ioctl_cb_class_t hc_iocb_class;
    usbhost_ioctl_cb_class_t hc_class;
   // endpoint handles
    usbhost_ep_handle *epIsoIn, *epIsoOut, *epCtrl;
    // endpoint info
    usbhost_ioctl_cb_ep_info_t epInfo;
    // device request
    usb_deviceRequest_t desc_dev;
    usbhost_xfer_iso_t xfer;

    unsigned short frame;
    usbhost_ioctl_cb_dev_info_t devInfo;
    int ifs, aset;

    vos_semaphore_t semDum;
    unsigned char num_dev;
    unsigned char status, ret;


    // device handle
    usbhost_device_handle *ifDev;
    unsigned char i;
    char *buf, c;
    static int ind = 0;

    hGpio = vos_dev_open(VOS_DEV_GPIO);
    hUsbHost = vos_dev_open(VOS_DEV_USBHOST);
    hUvc = vos_dev_open(VOS_DEV_UVC);
    do
    {
        uvc_ioctl_cb_t uvc_iocb;

        uvc_iocb.ioctl_code = VOS_IOCTL_UVC_ATTACH;
        uvc_iocb.set = (void *) hUsbHost;

        if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
        {
            asm{HALT};
        }

        //GET_CUR = 0x81
        set_class_request(desc_dev,1,0,1,GET_CUR,26);

        uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
        uvc_iocb.set = &desc_dev;
        uvc_iocb.get = &VProbeAndCom;

        if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
        {
            asm{HALT};
        }

        //GET_MAX = 0x83
        set_class_request(desc_dev,1,0,1,GET_MAX,26);

        uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
        uvc_iocb.set = &desc_dev;
```

```
uvc_iocb.get = &VProbeAndComMax;

if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    asm{HALT};
}

//GET_MIN = 0x82
set_class_request(desc_dev,1,0,1,GET_MIN,26);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
uvc_iocb.set = &desc_dev;
uvc_iocb.get = &VProbeAndComMin;

if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    asm{HALT};
}

//SET_CUR Change the interval to 2000000 i.e 5 frames per second
//change format descriptor to 1 Uncompressed  Video Format
//By default the Frame Index is 2(i.e)160*120
//Probe to make sure ISO rate is ISO_TRANSFER_SIZE

// maximum frame rate
VProbeAndCom.dwFrameInterval = 2000000;
VProbeAndCom.bFormatIndex = 1;
VProbeAndCom.bFrameIndex = 2;
// request data in the format that the Linux app expects it
VProbeAndCom.dwMaxVideoFrameSize = 38400;
VProbeAndCom.dwMaxPayloadTransferSize = ISO_TRANSFER_SIZE;

set_class_request(desc_dev,1,0,1,SET_CUR,26);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
uvc_iocb.set = &desc_dev;
uvc_iocb.get = &VProbeAndCom;

if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    asm{HALT};
}

//GET_CUR
set_class_request(desc_dev,1,0,1,GET_CUR,26);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
uvc_iocb.set = &desc_dev;
uvc_iocb.get = &VProbeAndCom;

if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
{
    asm{HALT};
}

// at this stage can check returned values to ensure they are set
// however the dwMaxPayloadTransferSize depends on the endpoint
// we are using so may differ from that expected

//SET_CUR Commite
set_class_request(desc_dev,1,0,2,SET_CUR,26);

uvc_iocb.ioctl_code = VOS_IOCTL_UVC_CLASS_REQUEST;
```

```
        uvc_iocb.set = &desc_dev;
        uvc_iocb.get = &VProbeAndCom;

        if (vos_dev_ioctl(hUvc, &uvc_iocb) != UVC_OK)
        {
            asm{HALT};
        }

        break;
    } while (1);


    vos_delay_msecs(300);

    vos_wait_semaphore(&sb1_avail);

    status = vos_dev_read(hUvc,buffer1,ISO_TRANSFER_SIZE *
ISO_TRANSFER_COUNT,NULL);

    vos_signal_semaphore(&sb1_full);

    do
    {
        vos_wait_semaphore(&sb2_avail);
        // now start the read from the ISO endpoint
        status = vos_dev_read(hUvc,buffer2,ISO_TRANSFER_SIZE *
ISO_TRANSFER_COUNT,NULL);
        vos_signal_semaphore(&sb2_full);

        vos_wait_semaphore(&sb1_avail);
        // read from the ISO endpoint
        status = vos_dev_read(hUvc,buffer1,ISO_TRANSFER_SIZE *
ISO_TRANSFER_COUNT,NULL);
        vos_signal_semaphore(&sb1_full);

    } while (1);
}



void displayOutput(void)
{

   SEPS525_reset();
   OLED_Init();

   RGB_IF_Init();


    do
    {
        vos_wait_semaphore(&sb1_full);
        process_data_stream(buffer1,BUF_SIZE);
        vos_signal_semaphore(&sb1_avail);

        vos_wait_semaphore(&sb2_full);
        process_data_stream(buffer2,BUF_SIZE);
        vos_signal_semaphore(&sb2_avail);

    }while(1);
}

unsigned char Blue, Green, Red;
```

```
unsigned char Y0, Y1, U, V;
unsigned char *inBuf;
unsigned char size;

void YUY2RGBConvert (unsigned char *inputBuffer1 )
{
    inBuf = inputBuffer1;

     while (size)
     {
         //size -= 4;
         asm { DEC8 size $4; };

         //  Y0 = (*inBuf++) - 16;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $16
            CPY8    Y0    %r0
            }
           //U = (*inBuf++) - 128;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $128
            CPY8    U    %r0
            }
           //Y1 = (*inBuf++) - 16;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $16
            CPY8    Y1    %r0
            }
           //V = (*inBuf++) - 128;
        asm {
            CPY16    %r0    inBuf
            INC16    inBuf    $1
            CPY8    %r0    (%r0)
            SUB8    %r0    $128
            CPY8    V    %r0
            }

           //Red = Y0 + V;
        asm {
            ADD8    Red    Y0    V;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Red;
        gpio_port_b = 0x70;

           //Green = Y0 - V - U;
        asm {
            SUB8    Green    Y0    V;
            SUB8    Green    U;
            }
        gpio_port_b = 0x60;
        gpio_port_a = Green;
        gpio_port_b = 0x70;
```

```
            //Blue = Y0 + U ;
          asm {
              ADD8    Blue    Y0    U;
              }
          gpio_port_b = 0x60;
          gpio_port_a = Blue;
          gpio_port_b = 0x70;

            //Red = Y1 + V;
          asm {
              ADD8    Red    Y1    V;
              }
          gpio_port_b = 0x60;
          gpio_port_a = Red;
          gpio_port_b = 0x70;

            //Green = Y1 - V - U;
          asm {
              SUB8    Green    Y1    V;
              SUB8    Green    U;
              }
          gpio_port_b = 0x60;
          gpio_port_a = Green;
          gpio_port_b = 0x70;

            //Blue = Y1 + U;
          asm {
              ADD8    Blue    Y1    U;
              }
          gpio_port_b = 0x60;
          gpio_port_a = Blue;
          gpio_port_b = 0x70;

        }
        return;

}



enum {
    SYNCING,
    SYNCED,
    RESYNC,

};

#define WEBCAM_EOF_BIT        0x02
#define WEBCAM_TOG_BIT        0x01
#define WEBCAM_ERR_BIT        0x40

static unsigned char state = SYNCING;
static unsigned short numXfers = 0;
void process_data_stream(unsigned char *buf,unsigned short num_bytes)
{
    unsigned char c;

    while (num_bytes != 0) {

        switch (state) {

        case SYNCING : // syncing with frame header
            if (*buf == 0x0c) {
```

```
            // could be a start of a webcam frame
            if ((*(buf+1) & WEBCAM_EOF_BIT) == WEBCAM_EOF_BIT) {
                // Yup - it's synced to the webcam frames
                // and goto SYNCED
                numXfers = 0;
                state = SYNCED;
                num_bytes = 0;
                break;
            }
            // Nope - it's not synced yet
            // consume the data
            num_bytes = 0;
        }
        else {
            asm{HALT};
        }
        break;

    case SYNCED : // synced to webcam frames
        if (*buf == 0x0c) {
            if ((*(buf+1) & WEBCAM_ERR_BIT) == WEBCAM_ERR_BIT) {
                state = RESYNC;
                num_bytes = 0;
                break;
            }
            ++numXfers;
            size = numXfers==1494?60:180;
            YUY2RGBConvert(buf + 12);
            if (numXfers==1494) {
                numXfers = 0;
                state = RESYNC;
                num_bytes = 0;
                break;
            }

            // consume the payboad
            num_bytes = 0;
        }
        else {
            asm{HALT};
        }
        break;

    case RESYNC : // syncing with frame header
        if (*buf == 0x0c) {
            // could be a start of a webcam frame
            if ((*(buf+1) & WEBCAM_EOF_BIT) == WEBCAM_EOF_BIT) {
                // Yup - it's synced to the webcam frames
                // and goto SYNCED
                numXfers = 0;
                state = SYNCED;
                num_bytes = 0;
                break;
            }
            // Nope - it's not synced yet
            // consume the data
            num_bytes = 0;
        }
        else {
            asm{HALT};
        }
        break;
```

```
            default :
                break;

            }
        }
    }
```


8 Appendix C – Revision History

Revision	Changes	Date
1.0	First Release	2010-11-01