Application Note: AN00127

USB Video Class Device

This application note shows how to create a USB device compliant to the standard USB Video Class (UVC) on an XMOS multicore microcontroller.

The code associated with this application note provides an example of using the XMOS USB Device Library (XUD) and associated USB class descriptors to provide a framework for the creation of USB video devices like webcam, video player, camcorders etc.

This example USB video class implementation provides a video camera device running over high speed USB. It supports standard requests associated with the class. The application doesn't connect a camera sensor device but emulates it by creating simple video data which is streamed to the host PC. Any host software that supports viewing UVC compliant video capture devices can be used to view the video streamed out of the XMOS device. This demonstrates the simple way in which USB video devices can easily be deployed using an xCORE-USB device.

Note: This application note provides a standard USB video class device and as a result does not require external drivers to run on Windows, Mac or Linux.

Required tools and libraries

- xTIMEcomposer Tools - Version 14.0.0
- XMOS USB library - Version 2.0.0

Required hardware

This application note is designed to run on an XMOS xCORE-USB series device.

The example code provided with the application has been implemented and tested on the xCORE-USB sliceKIT (XK-SK-U16-ST) but there is no dependency on this board and it can be modified to run on any development board which uses an xCORE-USB series device.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the Universal Serial Bus 2.0 Specification and related specifications, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[1].
- For the full API listing of the XMOS USB Device (XUD) Library please see the document XMOS USB Device (XUD) Library[2].
- For information on designing USB devices using the XUD library please see the XMOS USB Device Design Guide for reference[3].

[1] http://www.xmos.com/published/glossary
[2] http://www.xmos.com/published/xuddg
[3] http://www.xmos.com/published/xmos-usb-device-design-guide

1 Overview

1.1 Introduction

USB Video Class (UVC) is a standard class specification that standardizes video streaming functionality on the USB. It enables devices like webcams, digital camcorders, analog video converters, analog and digital television tuners etc to connect seamlessly with host machines.

UVC supports streaming multiple video formats including YUV, MJPEG, MPEG-2 TS, H.264, DV etc. It provides structures for describing the functionalities of the video device to the host and defines USB requests to control different parameters of the device and characteristics of the video stream. It also provides flexibility for a video device to support multiple video resolutions, formats and frame rates, which highly influences the bandwidth negotiation between the device and the host.

Many OS platforms have native support for UVC drivers which greatly reduces the time required for developers to create USB video devices.

In this application note, the UVC implementation for xCORE-USB device is explained in detail which will help you to build your own USB video device. The demo example doesn't interface a camera sensor but you can easily extend it to add a camera.

The standard USB Video Class specification can be found in the USB-IF website.

(http://www.usb.org/developers/docs/devclass_docs/USB_Video_Class_1_1_090711.zip)

1.2 Block diagram

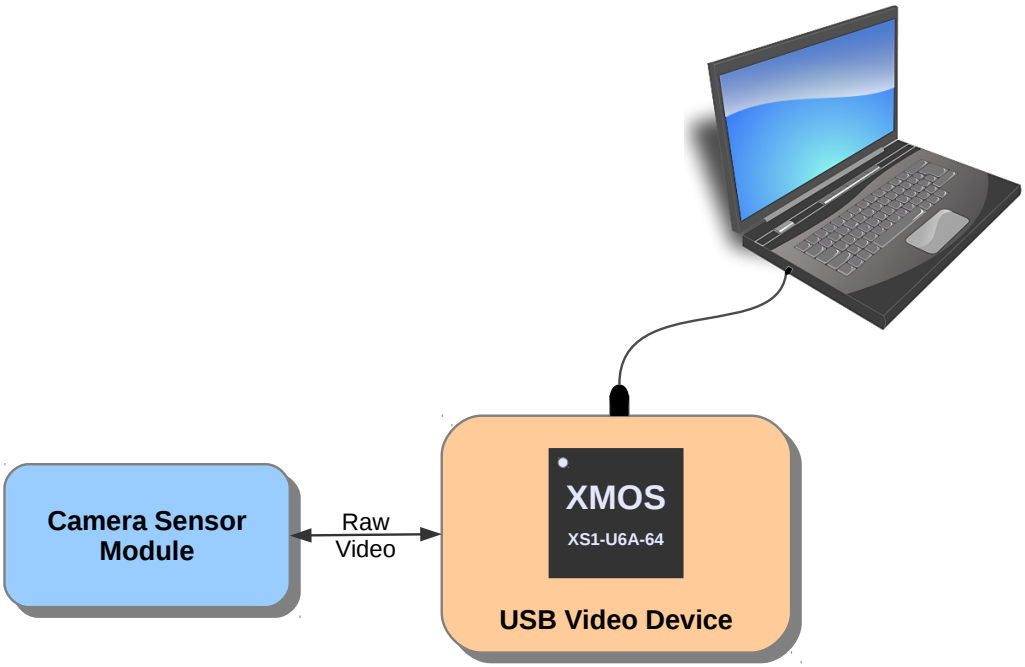

Figure 1: Block diagram of USB video class application

The 'Camera Sensor' shown in the above figure is not interfaced in the demo example but it is emulated by creating color video frames inside the device.

2 USB Video Class application note

The example in this application note uses the XMOS USB device library and shows a simple program that enumerates a USB Video Class device in a host machine and streams uncompressed video frames in YUV format with thirty frames per second to a video capture host software.

For this USB Video device application example, the system comprises three tasks running on separate logical cores of an xCORE-USB multicore microcontroller.

The tasks perform the following operations.

- A task containing the USB library functionality to communicate over USB.
- A task implementing Endpoint0 responding to both standard and video class-specific USB requests.
- A task implementing the application code to send video data over streaming endpoints.

These tasks communicate via the use of xCONNECT channels which allow data to be passed between application code running on separate logical cores.

The following diagram shows the task and communication structure for this USB video class application example.

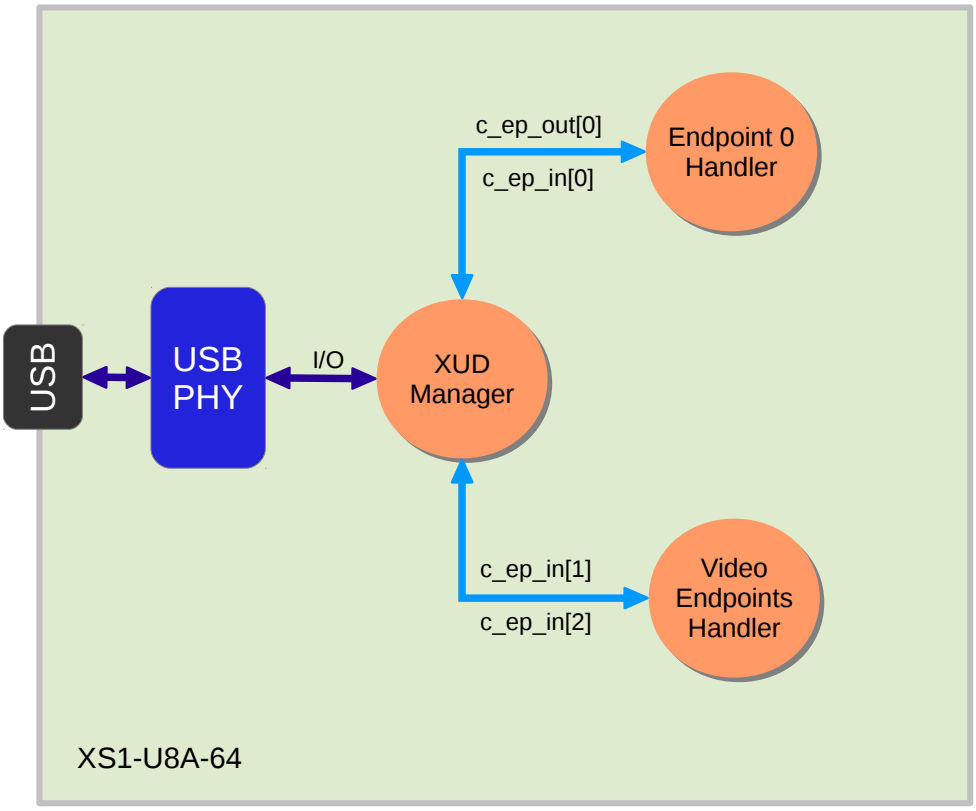

Figure 2: Task diagram of the USB video device example

2.1 Makefile additions for this example

To start using the USB library, you need to add **lib_usb** to your makefile:

```
USED_MODULES = ... lib_usb ...
```

You can then access the USB functions in your source code via the *usb.h* header file:

```
#include <usb.h>
```

2.2 Source code files

The example application consists of multiple source code files and the following list provides an overview of how the source code is organized.

- usb_video.xc, usb_video.h - Contains the USB video class descriptors and endpoint handler tasks (functions).
- uvc_req.c, uvc_req.h - Contains functions and data structures to handle class-specific USB requests.
- uvc_defs.h - This header file has defines that are used for USB descriptors, class-specific requests and video details like resolution, payload size and frame rate etc.
- main.xc - Contains main() function and some USB related defines.

2.3 Declaring resource and setting up the USB components

main.xc has some defines in it that are used to configure the XMOS USB device library. These are displayed below.

```
/* USB Endpoint Defines */
#define XUD_EP_COUNT_OUT   1    // 1 OUT EP0
#define XUD_EP_COUNT_IN    3    // (1 IN EP0 + 1 INTERRUPT IN EP + 1 ISO IN EP)
```

The above set of defines describe the endpoint configurations for this device.. This example has bi-directional communication with the host machine via the standard endpoint0 and two other endpoints for implementing the part of our video class.

All these defines are passed to the setup function for the USB library which is called from **main()**.

2.4 The application main() function

Below is the source code for the main function of this application, which is taken from the source file main.xc

```
int main() {

    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    /* 'Par' statement to run the following tasks in parallel */
    par
    {
        on USB_TILE: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                         null, XUD_SPEED_HS, XUD_PWR_SELF);

        on USB_TILE: Endpoint0(c_ep_out[0], c_ep_in[0]);

        on USB_TILE: VideoEndpointsHandler(c_ep_in[1], c_ep_in[2]);
    }
    return 0;
}
```

Looking at this in a more detail you can see the following:

- The par statement starts three separate tasks in parallel.
- There is a task to configure and execute the USB library: xud(). This library call runs in an infinite loop and handles all the underlying USB communications and provides abstraction at the endpoints level.
- There is a task to startup and run the Endpoint0 code: Endpoint0(). It handles the control endpoint zero and must be run in a separate logical core inorder to provide timely response to control requests from the host.
- There is a task to handle two other endpoints required for the Video class: VideoEndpointsHandler(). This function handles one Isochronous IN endpoint for video streaming and one interrupt IN endpoint for sending notifications to host.
- The define USB_TILE describes the tile on which the individual tasks will run.
- In this example all tasks run on the same tile as the USB PHY although this is only a requirement of xud().
- The xCONNECT communication channels used for inter-task communication are setup at the beginning of main() and passed on to respective tasks.
- The USB defines discussed earlier are passed into the function xud().

2.5 Configuring the USB device ID

The USB ID values used for vendor ID, product ID and device version number are defined in the file uvc_defs.h. These are used by the host machine to determine the vendor of the device (in this case XMOS) and the product plus the firmware version.

```
/* USB Video device product defines */
#define BCD_DEVICE  0x0100
#define VENDOR_ID   0x20B1
#define PRODUCT_ID  0x1DE0
```

2.6 Video device topology

This section provides a brief overview of the representation of video device in a topology. It introduces you to the terms used in the video class specification, which helps you to understand the further sections of this application note.

A video device is represented as an interconnection of multiple addressable entities. Each entity represents a functionality and has properties which are controlled by the USB host. The following are the different entities:

- **Units**
  - Selector Unit
  - Processing Unit
  - Extension Unit
- **Terminals**
  - Input Terminal
  - Output Terminal
  - Special Terminals (extends the I/O terminal)
    - Media Transport Terminal
    - Camera Terminal

These entities are interconnected by means of *Input Pins* and *Output Pins*. A Unit has one or more Input Pins and a single Output Pin, where each Pin represents logical data streams inside the video device. A Terminal has either a single Input Pin or a single Output Pin. An *Input Terminal(IT)* represents a starting point for data streams of the video device. An *Output Terminal(OT)* represents an ending point for data streams.

The functionality of a Unit or Terminal is further described through Video Controls. A Control typically provides access to a specific video property. Video properties include brightness, contrast, sharpness, digital zoom etc. Each Control has a set of attributes that can be manipulated or that provide additional information, they are:

- Current setting
- Minimum setting
- Maximum setting
- Resolution
- Size
- Default

For example, the brightness of the video stream can be controlled by the USB host by changing the current setting of the Brightness Control inside a Processing unit.

The following diagram shows the topology of the demo application

No Units are involved in the demo application example. More information on Units can be found from the USB video class specification documents.

This video device topology is communicated to the host through USB descriptors which is discussed in the following section.

2.7 USB Descriptors

USB Video class device has to support class-specific descriptors apart from the standard descriptors defined in the USB specifications. The class specific descriptors are customized according to the need of the USB Video device.

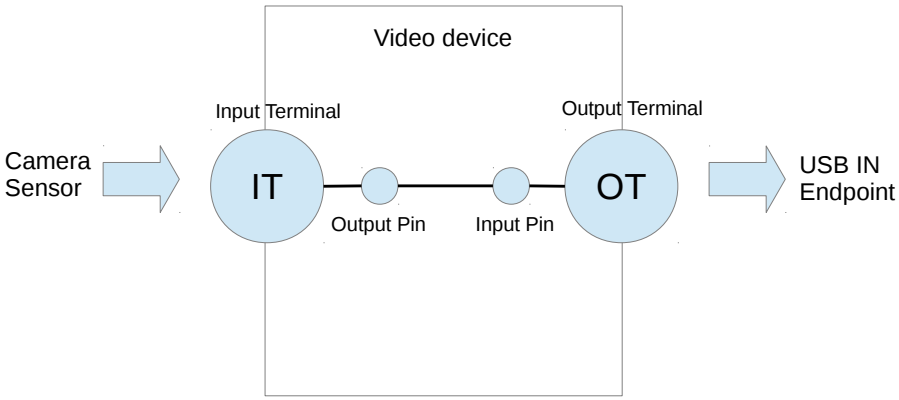

Figure 3: Topology of the UVC example

The following figure shows the descriptors used in the example code.

The above figure is discussed in detail in the following sections.

2.7.1 USB Device Descriptor

usb_video.xc is where the standard USB device descriptor is declared for the Video class device. Below is the structure which contains this descriptor. This will be requested by the host when the device is enumerated on the USB bus.

```
/* USB Device Descriptor */
static unsigned char devDesc[] =
{
    0x12,                   /* 0  bLength */
    USB_DESCTYPE_DEVICE,    /* 1  bdescriptorType - Device*/
    0x00,                   /* 2  bcdUSB version */
    0x02,                   /* 3  bcdUSB version */
    0xEF,                   /* 4  bDeviceClass - USB Miscellaneous Class */
    0x02,                   /* 5  bDeviceSubClass  - Common Class */
    0x01,                   /* 6  bDeviceProtocol  - Interface Association Descriptor */
    0x40,                   /* 7  bMaxPacketSize for EP0 - max = 64*/
    (VENDOR_ID & 0xFF),     /* 8  idVendor */
    (VENDOR_ID >> 8),       /* 9  idVendor */
    (PRODUCT_ID & 0xFF),    /* 10 idProduct */
    (PRODUCT_ID >> 8),      /* 11 idProduct */
    (BCD_DEVICE & 0xFF),    /* 12 bcdDevice */
    (BCD_DEVICE >> 8),      /* 13 bcdDevice */
    0x01,                   /* 14 iManufacturer - index of string*/
    0x02,                   /* 15 iProduct  - index of string*/
    0x00,                   /* 16 iSerialNumber  - index of string*/
    0x01                    /* 17 bNumConfigurations */
};
```

From this descriptor you can see that product, vendor and device firmware revision are all coded into this structure. This will allow the host machine to recognise the video device when it is connected to the USB bus.

For Video class device, it is mandatory to set the ‘bDeviceClass’, ‘bDeviceSubClass’ and ‘bDeviceProtocol’ fields to 0xEF, 0x02 and 0x01 respectively.

2.7.2 USB Configuration Descriptor

The USB configuration descriptor is used to configure the device in terms of the device class and the end-points setup. The hierarchy of descriptors under a configuration includes interface association descriptor, interfaces descriptors, class-specific descriptors and endpoints descriptors.

When a host requests a configuration descriptor, the entire configuration hierarchy including all the related descriptors are returned to the host. The following code shows the configuration hierarchy of the demo application.

```
/* USB Configuration Descriptor */
static unsigned char cfgDesc[] = {

  0x09,                       /* 0  bLength */
  USB_DESCTYPE_CONFIGURATION, /* 1  bDescriptorType - Configuration*/
  0xAE,00,                    /* 2  wTotalLength */
  0x02,                       /* 4  bNumInterfaces */
  0x01,                       /* 5  bConfigurationValue */
  0x03,                       /* 6  iConfiguration - index of string */
  0x80,                       /* 7  bmAttributes - Bus powered */
  0xFA,                       /* 8  bMaxPower (in 2mA units) - 500mA */

  /* Interface Association Descriptor */
  0x08,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE_ASSOCIATION,  /* 1 bDescriptorType - Interface Association */
  0x00,                       /* 2 bFirstInterface - VideoControl i/f */
  0x02,                       /* 3 bInterfaceCount - 2 Interfaces */
  USB_CLASS_VIDEO,            /* 4 bFunctionClass - Video Class */
  USB_VIDEO_INTERFACE_COLLECTION,  /* 5 bFunctionSubClass - Video Interface Collection */
  0x00,                       /* 6 bFunctionProtocol - No protocol */
  0x02,                       /* 7 iFunction - index of string */

  /* Video Control (VC) Interface Descriptor */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE,     /* 1 bDescriptorType - Interface */
  0x00,                       /* 2 bInterfaceNumber - Interface 0 */
  0x00,                       /* 3 bAlternateSetting */
  0x01,                       /* 4 bNumEndpoints */
  USB_CLASS_VIDEO,            /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_CONTROL,          /* 6 bInterfaceSubClass - VideoControl Interface */
  0x00,                       /* 7 bInterfaceProtocol - No protocol */
  0x02,                       /* 8 iInterface - Index of string (same as iFunction of IAD) */

  /* Class-specific VC Interface Header Descriptor */
  0x0D,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VC_HEADER,              /* 2 bDescriptorSubType - HEADER */
  0x10, 0x01,                 /* 3 bcdUVC - Video class revision 1.1 */
  0x28, 0x00,                 /* 5 wTotalLength - till output terminal */
  WORD_CHARS(100000000),      /* 7 dwClockFrequency - 100MHz (Deprecated) */
  0x01,                       /* 11 bInCollection - One Streaming Interface */
  0x01,                       /* 12 baInterfaceNr - Number of the Streaming interface */

  /* Input Terminal (Camera) Descriptor - Represents the CCD sensor (Simulated here in this demo) */
  0x12,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VC_INPUT_TERMINAL,      /* 2 bDescriptorSubType - INPUT TERMINAL */
  0x01,                       /* 3 bTerminalID */
  0x01, 0x02,                 /* 4 wTerminalType - ITT_CAMERA type (CCD Sensor) */
  0x00,                       /* 6 bAssocTerminal - No association */
  0x00,                       /* 7 iTerminal - Unused */
  0x00, 0x00,                 /* 8 wObjectiveFocalLengthMin - No optical zoom supported */
  0x00, 0x00,                 /* 10 wObjectiveFocalLengthMax - No optical zoom supported*/
  0x00, 0x00,                 /* 12 wOcularFocalLength - No optical zoom supported */
  0x03,                       /* 14 bControlSize - 3 bytes */
  0x00, 0x00, 0x00,           /* 15 bmControls - No controls are supported */

  /* Output Terminal Descriptor */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VC_OUPUT_TERMINAL,      /* 2 bDescriptorSubType - OUTPUT TERMINAL  */
  0x02,                       /* 3 bTerminalID */
  0x01, 0x01,                 /* 4 wTerminalType - TT_STREAMING type */
```

```
  0x00,                       /* 6 bAssocTerminal - No association */
  0x01,                       /* 7 bSourceID - Source is Input terminal 1 */
  0x00,                       /* 8 iTerminal - Unused */

  /* Standard Interrupt Endpoint Descriptor */
  0x07,                       /* 0 bLength */
  USB_DESCTYPE_ENDPOINT,      /* 1 bDescriptorType */
  (VIDEO_STATUS_EP_NUM | 0x80), /* 2 bEndpointAddress - IN endpoint*/
  0x03,                       /* 3 bmAttributes - Interrupt transfer */
  0x40, 0x00,                 /* 4 wMaxPacketSize - 64 bytes */
  0x09,                       /* 6 bInterval - 2^(9-1) microframes = 32ms */

  /* Class-specific Interrupt Endpoint Descriptor */
  0x05,                       /* 0 bLength */
  USB_DESCTYPE_CS_ENDPOINT,   /* 1 bDescriptorType - Class-specific Endpoint */
  0x03,                       /* 2 bDescriptorSubType - Interrupt Endpoint */
  0x40, 0x00,                 /* 3 wMaxTransferSize - 64 bytes */

  /* Video Streaming Interface Descriptor */
  /* Zero-bandwidth Alternate Setting 0 */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE,     /* 1 bDescriptorType - Interface */
  0x01,                       /* 2 bInterfaceNumber - Interface 1 */
  0x00,                       /* 3 bAlternateSetting - 0 */
  0x00,                       /* 4 bNumEndpoints - No bandwidth used */
  USB_CLASS_VIDEO,            /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_STREAMING,        /* 6 bInterfaceSubClass - VideoStreaming Interface */
  0x00,                       /* 7 bInterfaceProtocol - No protocol */
  0x00,                       /* 8 iInterface - Unused */

  /* Class-specific VS Interface Input Header Descriptor */
  0x0E,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_INPUT_HEADER,        /* 2 bDescriptorSubType - INPUT HEADER */
  0x01,                       /* 3 bNumFormats - One format supported */
  0x47, 0x00,                 /* 4 wTotalLength - Size of class-specific VS descriptors */
  (VIDEO_DATA_EP_NUM | 0x80), /* 6 bEndpointAddress - Iso EP for video streaming */
  0x00,                       /* 7 bmInfo - No dynamic format change */
  0x02,                       /* 8 bTerminalLink - Denotes the Output Terminal */
  0x01,                       /* 9 bStillCaptureMethod - Method 1 supported */
  0x00,                       /* 10 bTriggerSupport - No Hardware Trigger */
  0x00,                       /* 11 bTriggerUsage */
  0x01,                       /* 12 bControlSize - 1 byte */
  0x00,                       /* 13 bmaControls - No Controls supported */

  /* Class-specific VS Format Descriptor */
  0x1B,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_FORMAT_UNCOMPRESSED, /* 2 bDescriptorSubType - FORMAT UNCOMPRESSED */
  0x01,                       /* 3 bFormatIndex */
  0x01,                       /* 4 bNumFrameDescriptors - 1 Frame descriptor followed */
  0x59,0x55,0x59,0x32,
  0x00,0x00,0x10,0x00,
  0x80,0x00,0x00,0xAA,
  0x00,0x38,0x9B,0x71,        /* 5 guidFormat - YUY2 Video format */
  BITS_PER_PIXEL,             /* 21 bBitsPerPixel - 16 bits */
  0x01,                       /* 22 bDefaultFrameIndex */
  0x00,                       /* 23 bAspectRatioX */
  0x00,                       /* 24 bAspectRatioY */
  0x00,                       /* 25 bmInterlaceFlags - No interlaced mode */
  0x00,                       /* 26 bCopyProtect - No restrictions on duplication */

  /* Class-specific VS Frame Descriptor */
  0x1E,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_FRAME_UNCOMPRESSED,  /* 2 bDescriptorSubType */
  0x01,                       /* 3 bFrameIndex */
  0x01,                       /* 4 bmCapabilities - Still image capture method 1 */
  SHORT_CHARS(WIDTH),         /* 5 wWidth - 480 pixels */
  SHORT_CHARS(HEIGHT),        /* 7 wHeight - 320 pixels */
  WORD_CHARS(MIN_BIT_RATE),   /* 9 dwMinBitRate */
  WORD_CHARS(MAX_BIT_RATE),   /* 13 dwMaxBitRate */
  WORD_CHARS(MAX_FRAME_SIZE), /* 17 dwMaxVideoFrameBufSize */
  WORD_CHARS(FRAME_INTERVAL), /* 21 dwDefaultFrameInterval (in 100ns units) */
  0x01,                       /* 25 bFrameIntervalType */
```

```
  WORD_CHARS(FRAME_INTERVAL), /* 26 dwFrameInterval (in 100ns units) */

  /* Video Streaming Interface Descriptor */
  /* Alternate Setting 1 */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE,     /* 1 bDescriptorType - Interface */
  0x01,                       /* 2 bInterfaceNumber - Interface 1 */
  0x01,                       /* 3 bAlternateSetting -  1 */
  0x01,                       /* 4 bNumEndpoints */
  USB_CLASS_VIDEO,            /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_STREAMING,        /* 6 bInterfaceSubClass - VideoStreaming Interface */
  0x00,                       /* 7 bInterfaceProtocol - No protocol */
  0x00,                       /* 8 iInterface - Unused */

  /* Standard VS Isochronous Video Data Endpoint Descriptor */
  0x07,                       /* 0 bLength */
  USB_DESCTYPE_ENDPOINT,      /* 1 bDescriptorType */
  (VIDEO_DATA_EP_NUM | 0x80), /* 2 bEndpointAddress - IN Endpoint */
  0x05,                       /* 3 bmAttributes - Isochronous EP (Asynchronous) */
  0x00, 0x04,                 /* 4 wMaxPacketSize 1x 1024 bytes*/
  0x01,                       /* 6 bInterval */

};
```

The *configuration descriptor* tells host about the power requirements of the device and the number of interfaces it supports.

Multiple interfaces together provides the video functionality. This group of interfaces is called Video Interface Collection. The Video Interface Collection is described by an *interface association descriptor* (IAD). In the example application, the IAD defines that the interface zero and one groups to form the USB Video device. These two interfaces are:

- Video Control Interface (VC Interface)
- Video Streaming Interface (VS Interface)

Note: A video function must have one VideoControl interface and zero or more VideoStreaming interfaces.

2.7.3 VideoControl Interface

This interface controls the functional behavior of the video device. It is described by both standard and class-specific descriptors.

The *Standard VC interface descriptor* identifies the interface number and class and provides the number of endpoints that belongs to this interface. The default *Endpoint 0* is used by this interface for control purpose through class-specific requests. Another optional endpoint called *Status Interrupt Endpoint* is used to send asynchronous status notifications to the host. This interrupt endpoint is described by both standard and class-specific endpoint descriptors.

The *Class-Specific VC interface descriptor* describes the whole topology of the video device. It includes *Unit descriptors* and *Terminal descriptors*. The example application doesn't include any Units and hence only Terminal descriptors can be found in the descriptors hierarchy structure.

The Class-Specific descriptors starts with a header called VC Interface Header descriptor. This descriptor mentions the version of UVC specification followed and the collection of streaming interfaces to which this VideoControl interface belongs.

The *Input Terminal descriptor* provides information on the functional aspects of the input source of the video device. Following code shows the fields of this descriptor:

```
USB_VC_INPUT_TERMINAL,      /* 2 bDescriptorSubType - INPUT TERMINAL */
0x01,                       /* 3 bTerminalID */
0x01, 0x02,                 /* 4 wTerminalType - ITT_CAMERA type (CCD Sensor) */
0x00,                       /* 6 bAssocTerminal - No association */
```

In the above code, the 'bTerminalID' is an unique identifier of this terminal and 'bTerminalType' declares camera as the input type.

The *Output Terminal descriptor* is shown in the following code.

```
USB_VC_OUPUT_TERMINAL,      /* 2 bDescriptorSubType - OUTPUT TERMINAL  */
0x02,                       /* 3 bTerminalID */
0x01, 0x01,                 /* 4 wTerminalType - TT_STREAMING type */
0x00,                       /* 6 bAssocTerminal - No association */
0x01,                       /* 7 bSourceID - Source is Input terminal 1 */
0x00,                       /* 8 iTerminal - Unused */
```

The above descriptor shows that the 'bSourceID' is defined as 0x01 which is the 'bTerminalID' of the input terminal. This information shows the interconnection between the entities, which the host uses to identify the topology of the video device.

2.7.4 VideoStreaming Interface

VideoStreaming interfaces are used to interchange video data streams between the Host and the Video device. Each interface can have one isochronous or bulk data endpoint. Interfaces supporting isochronous video transfer must have alternate settings which enables host to change the bandwidth requirements imposed by an active isochronous pipe. It is also mandatory to provide a zero-bandwidth alternate setting as the default alternate setting(alternate setting zero) that provides the host software the option to temporarily relinquish USB bandwidth by switching to this alternate setting.

In the UVC example, the zero-bandwidth alternate setting of the VideoStreaming interface is described by standard interface descriptor and class-specific VS interface descriptors.

The *Standard VS interface descriptor* provides the interface number, the number of endpoints that belongs to this interface etc. In case of zero-bandwidth alternate setting the number of endpoints is set to zero.

The *Class-Specific VS interface descriptors* are used to describe the supported video stream formats, video frame details, still image frame details, color profile of video data etc. The following is the list of these class-specific descriptors:

- Input Header descriptor
- Output Header descriptor
- Payload Format descriptor
- Video Frame descriptor
- Still Image frame descriptor
- Color Matching descriptor

The *Input Header descriptor* is meant for interfaces that contain IN endpoint and *Output Header* is for interfaces that contain OUT endpoint.

The following code shows the fields of Input Header descriptor:

```
USB_VS_INPUT_HEADER,        /* 2 bDescriptorSubType - INPUT HEADER */
0x01,                       /* 3 bNumFormats - One format supported */
0x47, 0x00,                 /* 4 wTotalLength - Size of class-specific VS descriptors */
(VIDEO_DATA_EP_NUM | 0x80), /* 6 bEndpointAddress - Iso EP for video streaming */
0x00,                       /* 7 bmInfo - No dynamic format change */
0x02,                       /* 8 bTerminalLink - Denotes the Output Terminal */
```

The above code shows the number of formats supported, the address of endpoint which streams video data and the output terminal ID which links to this streaming interface.

The *Payload Format descriptor* describes the video format. The fields of this descriptor is shown below:

```
USB_VS_FORMAT_UNCOMPRESSED, /* 2 bDescriptorSubType - FORMAT UNCOMPRESSED */
0x01,                       /* 3 bFormatIndex */
0x01,                       /* 4 bNumFrameDescriptors - 1 Frame descriptor followed */
0x59,0x55,0x59,0x32,
0x00,0x00,0x10,0x00,
0x80,0x00,0x00,0xAA,
0x00,0x38,0x9B,0x71,        /* 5 guidFormat - YUY2 Video format */
BITS_PER_PIXEL,             /* 21 bBitsPerPixel - 16 bits */
0x01,                       /* 22 bDefaultFrameIndex */
```

The above code shows that the video stream is of uncompressed YUY2 format and uses 16-bits per pixel.

The *Video Frame descriptor* mentions the frame resolution, frame rate, video buffer size etc. The following code shows the fields of this descriptor:

```
USB_VS_FRAME_UNCOMPRESSED,  /* 2 bDescriptorSubType */
0x01,                       /* 3 bFrameIndex */
0x01,                       /* 4 bmCapabilities - Still image capture method 1 */
SHORT_CHARS(WIDTH),         /* 5 wWidth - 480 pixels */
SHORT_CHARS(HEIGHT),        /* 7 wHeight - 320 pixels */
WORD_CHARS(MIN_BIT_RATE),   /* 9 dwMinBitRate */
WORD_CHARS(MAX_BIT_RATE),   /* 13 dwMaxBitRate */
WORD_CHARS(MAX_FRAME_SIZE), /* 17 dwMaxVideoFrameBufSize */
WORD_CHARS(FRAME_INTERVAL), /* 21 dwDefaultFrameInterval (in 100ns units) */
0x01,                       /* 25 bFrameIntervalType */
WORD_CHARS(FRAME_INTERVAL), /* 26 dwFrameInterval (in 100ns units) */
```

The defines used in the above code are present in the *uvc_defs.h* file and they are shown below:

```
/* USB Video resolution */
#define BITS_PER_PIXEL 16
#define WIDTH  480
#define HEIGHT 320

/* Frame rate */
#define FPS 30

#define MAX_FRAME_SIZE (WIDTH * HEIGHT * BITS_PER_PIXEL / 8)
#define MIN_BIT_RATE   (MAX_FRAME_SIZE * FPS * 8)
#define MAX_BIT_RATE   (MIN_BIT_RATE)
#define PAYLOAD_SIZE   (1 * 1024)

/* Interval defined in 100ns units */
#define FRAME_INTERVAL       (10000000/FPS)
```

The other alternate setting of this interface has the data streaming isochronous endpoint and it is the operational alternate setting. The class-specific descriptors are not repeated in this alternate setting.

The *Standard VS Isochronous Endpoint descriptor* of the alternate setting 1 of the UVC example is shown below:

```
/* Standard VS Isochronous Video Data Endpoint Descriptor */
0x07,                       /* 0 bLength */
USB_DESCTYPE_ENDPOINT,      /* 1 bDescriptorType */
(VIDEO_DATA_EP_NUM | 0x80), /* 2 bEndpointAddress - IN Endpoint */
0x05,                       /* 3 bmAttributes - Isochronous EP (Asynchronous) */
0x00, 0x04,                 /* 4 wMaxPacketSize 1x 1024 bytes*/
0x01,                       /* 6 bInterval */
```

The above code shows that the maximum packet size of the endpoint is 1024 bytes and the 'bInterval' of 0x01 requests host to poll the endpoint every microframe(125us).

In general, USB video devices supports a set of video parameter combinations(including video format, frame size and frame rate) and multiple alternate settings with different maximum packet size endpoints. This enables the host to select the appropriate alternate setting that provides only the required bandwidth for a given video parameter combination.

2.7.5 USB String Descriptors

String descriptors provide human readable information for your device and you can configure them with your USB product information. The descriptors are placed in an array as shown in the below code.

```
/* String table - unsafe as accessed via shared memory */
static char * unsafe stringDescriptors[]=
{
  "\x09\x04",             /* Language ID string (US English) */
  "XMOS",                 /* iManufacturer */
  "XMOS USB Video Device",/* iProduct */
  "Config",               /* iConfiguration string */
};
```

The XMOS USB library will take care of encoding the strings into Unicode and structures the content into USB string descriptor format.

2.8 USB Standard and Class-Specific requests

In *usb_video.xc* there is a function *Endpoint0()* which handles all the USB control requests sent by host to the control endpoint 0. USB control requests includes both standard USB requests and the UVC class-specific requests.

In *Endpoint0()* function, a USB request is received as a setup packet by calling *USB_GetSetupPacket()* library function. The setup packet structure is then examined to distinguish between standard and class-specific requests.

The XMOS USB library provides a function *USB_StandardRequests()* to handle the standard USB requests. This function is called with setup packet and descriptors structures as shown below

```
/* Returns  XUD_RES_OKAY if handled okay,
 *          XUD_RES_ERR if request was not handled (STALLed),
 *          XUD_RES_RST for USB Reset */
unsafe{
result = USB_StandardRequests(ep0_out, ep0_in, devDesc,
            sizeof(devDesc), cfgDesc, sizeof(cfgDesc),
            null, 0, null, 0, stringDescriptors, sizeof(stringDescriptors)/sizeof(stringDescriptors[0]),
            sp, usbBusSpeed);
 }
```

The video class interfaces use endpoint 0 as the control element and receives all class-specific requests on it. The class-specific requests are used to set and get video related controls. These request are divided into:

- VideoControl requests
- VideoStreaming requests

The function *UVC_InterfaceClassRequests()* present in *uvc_req.c* handles the class-specific requests. The defines corresponding to the class-specific request codes are present in *uvc_defs.h* as shown below.

```
/* Video Class-specific Request codes */
#define SET_CUR     0x01
#define GET_CUR     0x81
#define GET_MIN     0x82
#define GET_MAX     0x83
#define GET_RES     0x84
#define GET_LEN     0x85
#define GET_INFO    0x86
#define GET_DEF     0x87

/* Video Streaming Interface Control selectors */
#define VS_PROBE_CONTROL        0x01
#define VS_COMMIT_CONTROL       0x02
```

In the UVC example, the SET and GET requests for Video Probe and Commit Controls are handled. The Video Probe and Commit Controls are involved in the negotiation of streaming parameters between the host and the device. The following code shows the structure of the streaming parameters that are negotiated with those Controls.

```
/* Video Probe and Commit Controls (Table 4-47 , UVC 1.1) */
typedef struct
{
    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;
} __attribute__((packed)) UVC_ProbeCommit_Ctrl_t;
```

The demo application doesn't have multiple set of streaming parameters and therefore the GET_DEF, GET_MIN, GET_MAX and GET_CUR requests are handled similarly and return same values to the host.

This source code can be easily extended to support more class-specific requests.

2.9 Video data streaming

Streaming video data between device and host takes place through the streaming endpoint of the VideoStreaming interface. The video is streamed by continuously transmitting the video samples at a particular rate. A video sample refers to an encoded block of video data that the format-specific decoder is able to accept and interpret in a single transmission.

In the UVC example, the video data is in packed 4:2:2 YUV format (YUY2) and a video sample corresponds to a single video frame of 480x320 pixels. Each video sample is split into multiple class-defined Payload Transfers. A Payload Transfer is composed of the class-defined payload header followed by the video payload data. The payload format is as shown below:

For an isochronous endpoint, each (micro)frame will contain a single payload transfer. The maximum packet size of the isochronous endpoint in the example code is 1024 bytes, therefore excluding the payload header length 1012 bytes are available for the video data in a single payload transfer.

The function *VideoEndpointsHandler()* present in *usb_video.xc* handles the isochronous video data endpoint. Each payload transfer is carried out by using the *XUD_SetBuffer()* API of the USB library.

For demonstartion, video data is generated in the device by filling up the buffers with red, green and blue color values (YUV format) as shown in the following code.

```
/* Fill video buffers with different color data */
for(int i = 0; i < (PAYLOAD_SIZE/4); i++) {
    /* Set RED color */
    gVideoBuffer[0][i] = 0x7010D010;
    /* Set GREEN color */
    gVideoBuffer[2][i] = 0x00000000;
    /* Set BLUE color */
    gVideoBuffer[1][i] = 0xDC206020;
}
```

These buffers are used together to create a video frame. The following code from *VideoEndpointsHandler()* shows the transmission of a single frame.

```
/* Transmits single frame */
while(expectedPixels > 0)
{
    if(expectedPixels < (PAYLOAD_SIZE - PAYLOAD_HEADER_LENGTH)) {
        /* Payload transfer */
        result = XUD_SetBuffer(episo_in, (gVideoBuffer[index], unsigned char[]), expectedPixels+
          ↪ PAYLOAD_HEADER_LENGTH);
    } else {
        /* Payload transfer */
        result = XUD_SetBuffer(episo_in, (gVideoBuffer[index], unsigned char[]), 1024);
    }
    /* Note down the SOF counts */
    sofCounts++;

    expectedPixels -= ((PAYLOAD_SIZE)- PAYLOAD_HEADER_LENGTH);

    if(expectedPixels <= (MAX_FRAME_SIZE - split)) {
        index = (index + 1) % 3;
        split += (MAX_FRAME_SIZE / 6);
    }
}
```

The above code shows that 'gVideoBuffer[]' holds the payload data and is sent continuously to the host till the expected number of pixels per frame is over. These payload buffers are populated with payload header as shown in the following code.

```
/* Fill the buffers with payload header */
for(int i=0; i<3; i++)
{
    /* Make the Payload header */
    (gVideoBuffer[i], unsigned char[])[0] = PAYLOAD_HEADER_LENGTH;
    (gVideoBuffer[i], unsigned char[])[1] = frame;
    /* Set dwPresentationTime */
    (gVideoBuffer[i], unsigned short[])[1] = pts;
    (gVideoBuffer[i], unsigned short[])[2] = pts>>16;
    /* Set scrSourceClock */
    (gVideoBuffer[i], unsigned short[])[3] = pts;
    (gVideoBuffer[i], unsigned short[])[4] = pts>>16;
    (gVideoBuffer[i], unsigned short[])[5] = (sofCounts>>3) & 2047;
}
```

In the above code, the 'pts' is Presentation timestamp and it is obtained from a timer running in the xCORE device at 100MHz. The 'pts' and start of frame counter (count of USB SOF) are used to arrive at the Source Clock reference field.

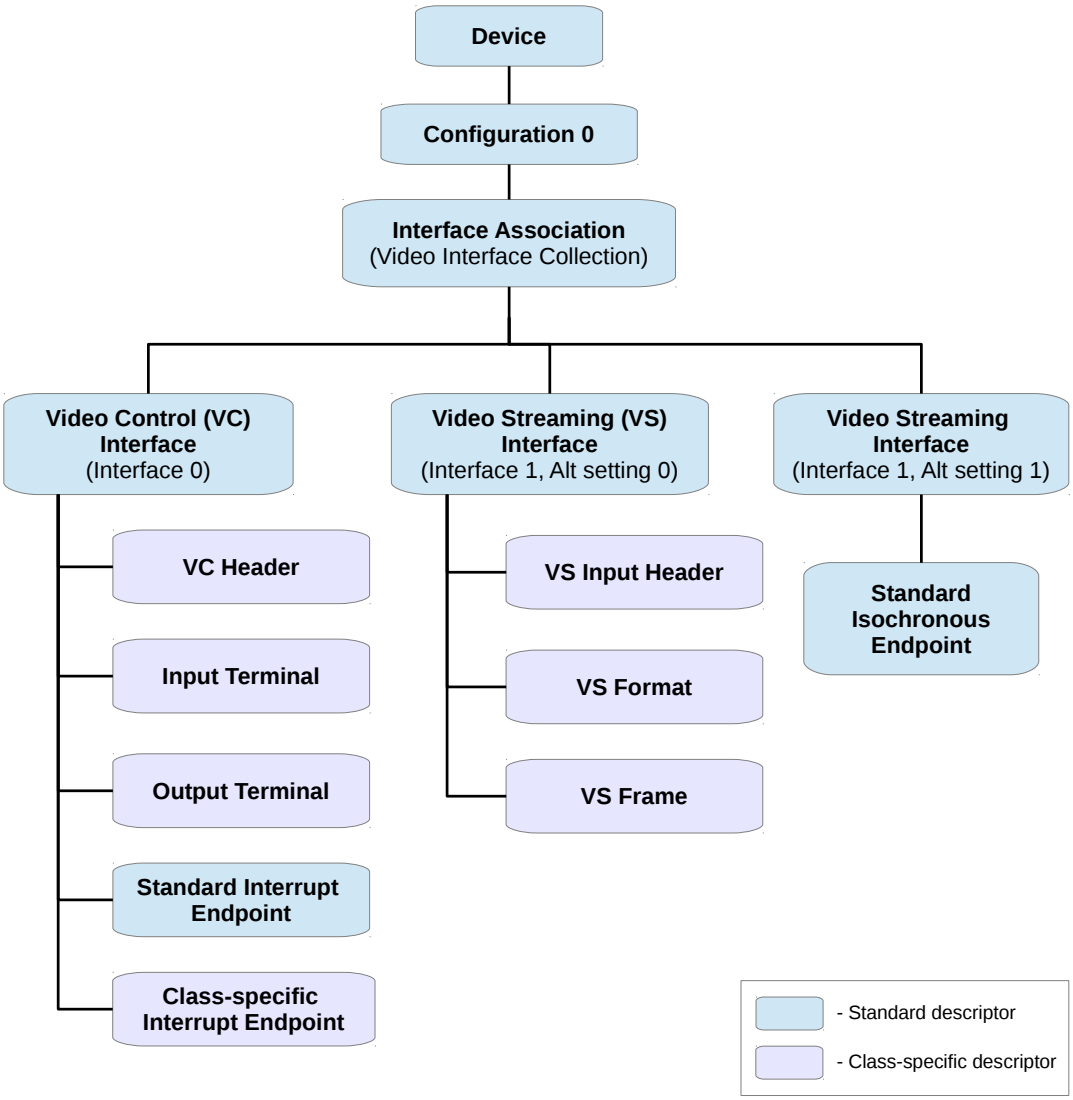

Figure 4: Hierarchical structure of USB descriptors of UVC example

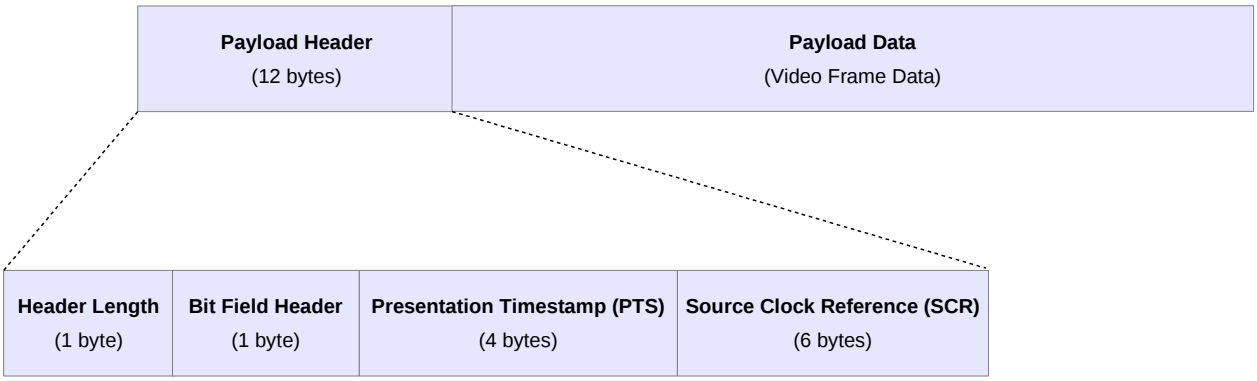

Figure 5: Payload format for uncompressed streams

APPENDIX A - Demo Hardware Setup

To setup the demo hardware the following boards are required.

- xCORE-USB slicekKIT (XK-SK-U16-ST)
  - xCORE-USB Core board.
  - USB A/B sliceCARD.
  - xTAG-2 debug adaptor
  - Power supply

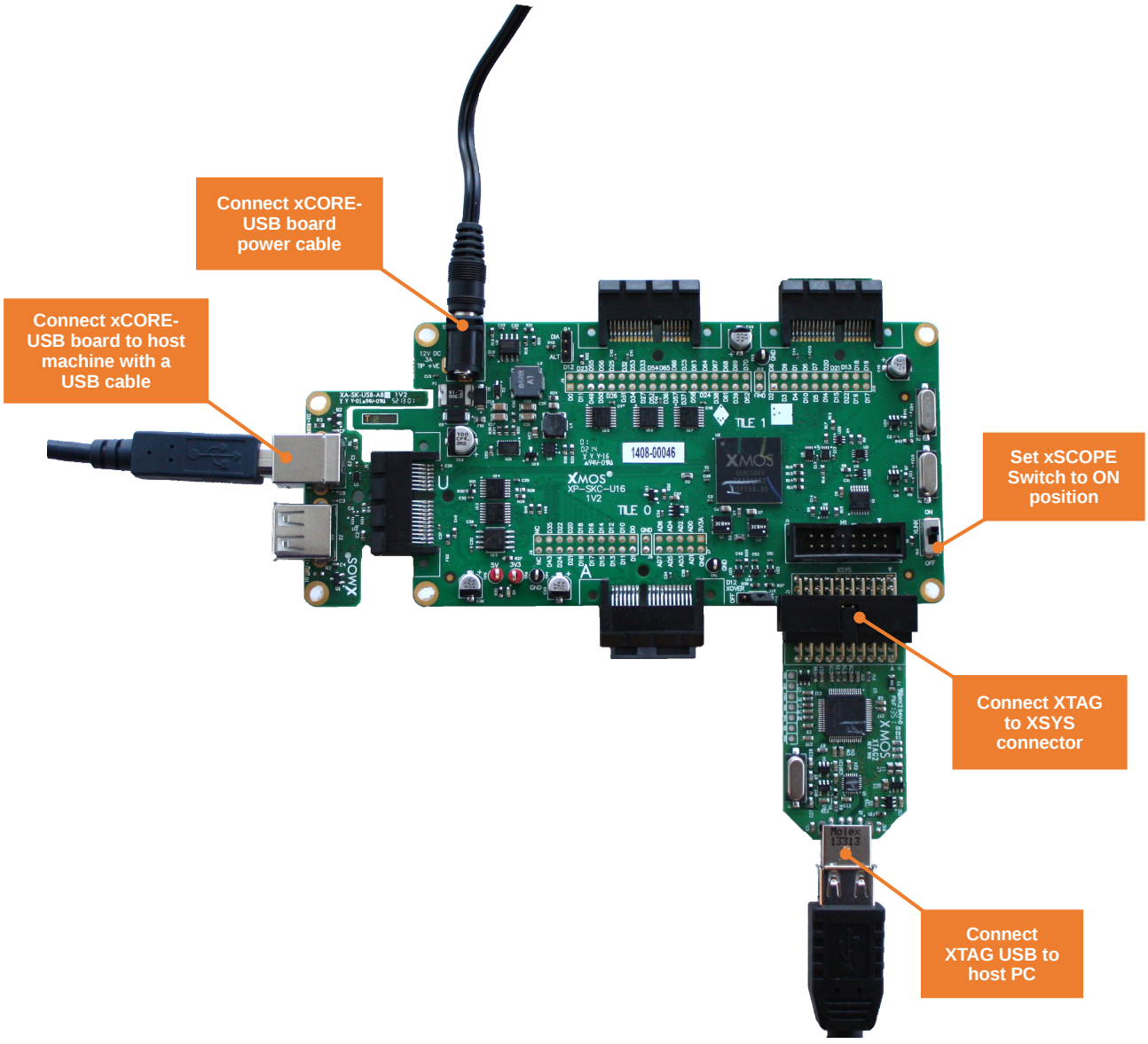

Figure 6: XMOS xCORE-USB sliceKIT

The hardware should be configured as displayed above for this demo:

- The XTAG debug adapter should be connected to the XSYS connector and the XTAG USB should be connected to the host machine.
- The USB sliceCARD should be connected to the U slot (J4 header) of the xCORE-USB Core board and the other end of USB sliceCARD should be connected to host machine using a USB cable.
- The XLINK switch on the core board should be set to the ON position.
- The xCORE-USB core board should have the power cable connected.

APPENDIX B - Launching the demo application

Once the demo example has been built either from the command line using xmake or via the build mechanism of xTIMEcomposer studio we can execute the application on the xCORE-USB sliceKIT.

Once built there will be a `bin` directory within the project which contains the binary for the xCORE device. The xCORE binary has a XMOS standard .xe extension.

B.1 Launching from the command line

From the command line the `xrun` tool is used to download code to the xCORE device. Changing into the bin directory of the project we can execute the code on the xCORE microcontroller as follows:

```
> xrun --xscope app_usb_video.xe          <-- Download and execute the xCORE code
```

Once this command has executed the USB video device should have enumerated on your machine.

B.2 Launching from xTIMEcomposer Studio

From xTIMEcomposer Studio the run mechanism is used to download code to xCORE device. Select the xCORE binary from the bin directory, right click and then 'Run As'-> 'xCORE application' to execute the code on the xCORE device.

Once this command has executed the USB video device should have enumerated on your machine.

B.3 Running the demo

The demo can be run on any OS that has support for USB Video class driver. Windows, Linux and Mac OS have native support for UVC driver. The following sections describe in detail on how to run the demo on those OS platforms.

B.3.1 Running on Windows

- In Microsoft Windows, When the USB Video device enumerates the host driver will be installed to get the device ready for operation. The following figure shows the dialog that completes installation of driver for the *XMOS USB Video Device*.

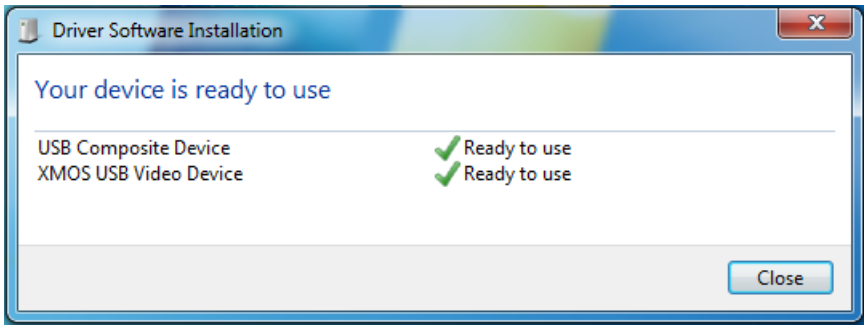

Figure 7: Driver installation for the Video device

- After the driver is installed properly, you can use any video capture softwares like VLC Media player, AmCap etc to open the *XMOS USB Video Device*.
- Open VLC Media player, select *Media* menu and click *Open Capture Device...*. This will open a dialog window on which you can select the Video device as shown in the following picture.

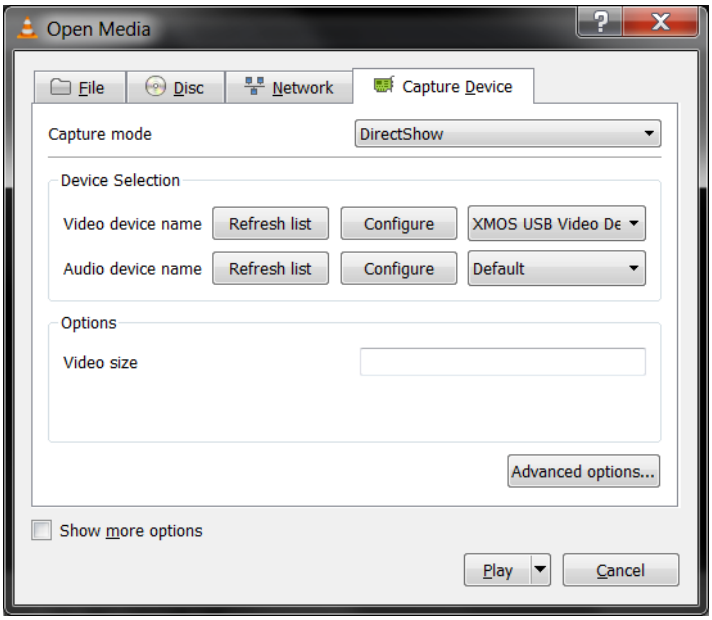

Figure 8: Open Video device in VLC Media player

- Click *Play* to see the demo video streamed out of the Video device. The video is a sequence of colored rows that scroll upwards. The following figure shows a snapshot of the video.

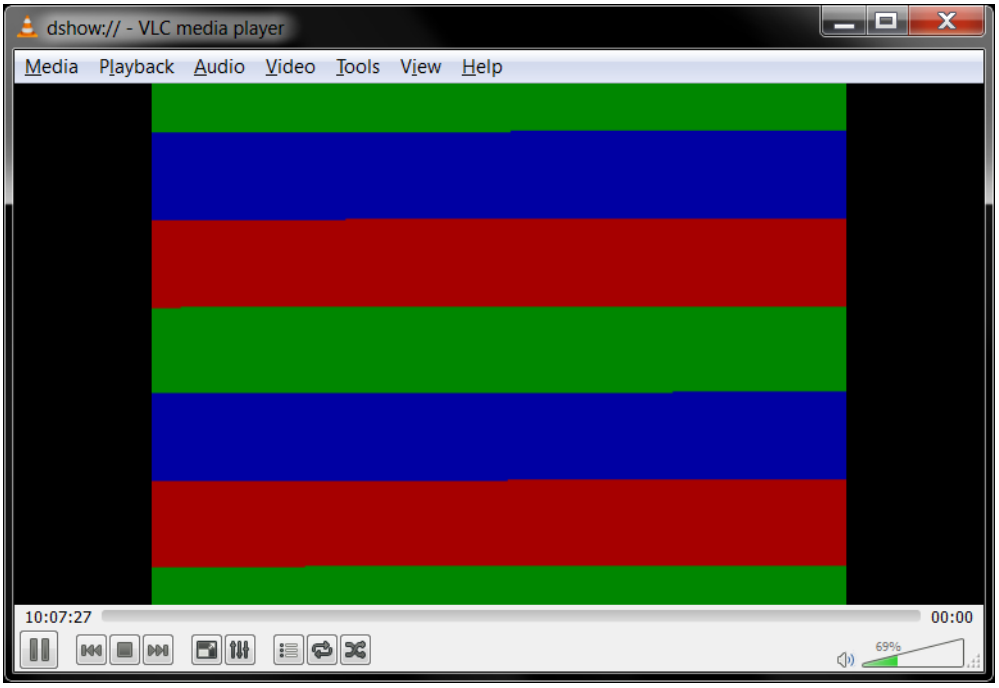

Figure 9: Video streamed from the XMOS USB Video device

B.3.2 Running on Mac OSX

- In Mac OSX, once the USB Video device is enumerated the UVC driver will be loaded by the host to get the device ready for operation. The device will have enumerated as *XMOS USB Video Device*.
- Now you can open this Video device using any video capture software. *Photo Booth* is one such application that comes by default with Mac. Open Photo Booth application, click on 'Camera' menu and then select *XMOS USB Video Device*. The application will then show the video streamed out of the USB device. The video is a sequence of colored rows that scroll upwards.

B.3.3 Running on Linux

- Under linux, when the device enumerates the native UVC driver will be loaded and the device will be mounted as */dev/videoX* where 'X' is a number.
- Now you can use any video capture software like VLC Media player, Cheese, luvcview etc to open the Video device.
- Open VLC Media player, select *Media* menu and click *Open Capture Device...*. This will open a dialog window on which you can select the Video device as shown in the following picture.

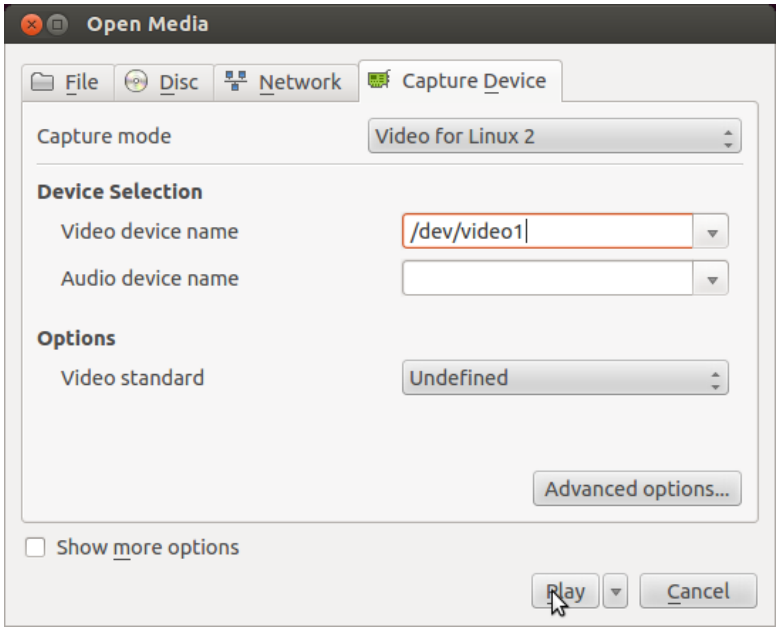

Figure 10: Open Video device in VLC Media player

- Click *Play* to see the demo video streamed out of the *XMOS USB Video Device*. The demo video looks like sequence of colored rows scrolling up.

APPENDIX C - References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS xCORE-USB Device Library

http://www.xmos.com/published/xuddg

XMOS USB Device Design Guide

http://www.xmos.com/published/xmos-usb-device-design-guide

USB Video Class Specification, USB.org:

http://www.usb.org/developers/docs/devclass_docs/

USB 2.0 Specification

http://www.usb.org/developers/docs/usb20_docs/usb_20_081114.zip

YUV Video Format

http://en.wikipedia.org/wiki/YUV

http://linuxtv.org/downloads/v4l-dvb-apis/V4L2-PIX-FMT-YUYV.html

APPENDIX D - Full source code listing

D.1 Source code for main.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

/* Includes */
#include <platform.h>
#include <xs1.h>
#include <xscope.h>
#include <xccompat.h>

#include "usb_video.h"

/* xSCOPE Setup Function */
#if (USE_XSCOPE == 1)
void xscope_user_init(void) {
    xscope_register(0, 0, "", 0, "");
    xscope_config_io(XSCOPE_IO_BASIC); /* Enable fast printing over XTAG */
}
#endif

/* USB Endpoint Defines */
#define XUD_EP_COUNT_OUT   1    // 1 OUT EP0
#define XUD_EP_COUNT_IN    3    // (1 IN EP0 + 1 INTERRUPT IN EP + 1 ISO IN EP)

int main() {

    chan c_ep_out[XUD_EP_COUNT_OUT], c_ep_in[XUD_EP_COUNT_IN];

    /* 'Par' statement to run the following tasks in parallel */
    par
    {
        on USB_TILE: xud(c_ep_out, XUD_EP_COUNT_OUT, c_ep_in, XUD_EP_COUNT_IN,
                         null, XUD_SPEED_HS, XUD_PWR_SELF);

        on USB_TILE: Endpoint0(c_ep_out[0], c_ep_in[0]);

        on USB_TILE: VideoEndpointsHandler(c_ep_in[1], c_ep_in[2]);
    }
    return 0;
}
```

D.2 Source code for usb_video.xc

```
// Copyright (c) 2015, XMOS Ltd, All rights reserved

#include "string.h"
#include "usb.h"
#include "usb_video.h"

/* Definition of Descriptors */
/* USB Device Descriptor */
static unsigned char devDesc[] =
{
    0x12,                   /* 0  bLength */
    USB_DESCTYPE_DEVICE,    /* 1  bdescriptorType - Device*/
    0x00,                   /* 2  bcdUSB version */
    0x02,                   /* 3  bcdUSB version */
    0xEF,                   /* 4  bDeviceClass - USB Miscellaneous Class */
```

```
    0x02,                   /* 5  bDeviceSubClass  - Common Class */
    0x01,                   /* 6  bDeviceProtocol  - Interface Association Descriptor */
    0x40,                   /* 7  bMaxPacketSize for EP0 - max = 64*/
    (VENDOR_ID & 0xFF),     /* 8  idVendor */
    (VENDOR_ID >> 8),       /* 9  idVendor */
    (PRODUCT_ID & 0xFF),    /* 10 idProduct */
    (PRODUCT_ID >> 8),      /* 11 idProduct */
    (BCD_DEVICE & 0xFF),    /* 12 bcdDevice */
    (BCD_DEVICE >> 8),      /* 13 bcdDevice */
    0x01,                   /* 14 iManufacturer - index of string*/
    0x02,                   /* 15 iProduct  - index of string*/
    0x00,                   /* 16 iSerialNumber  - index of string*/
    0x01                    /* 17 bNumConfigurations */
};

/* USB Configuration Descriptor */
static unsigned char cfgDesc[] = {

  0x09,                         /* 0  bLength */
  USB_DESCTYPE_CONFIGURATION, /* 1  bDescriptorType - Configuration*/
  0xAE,00,                      /* 2  wTotalLength */
  0x02,                         /* 4  bNumInterfaces */
  0x01,                         /* 5  bConfigurationValue */
  0x03,                         /* 6  iConfiguration - index of string */
  0x80,                         /* 7  bmAttributes - Bus powered */
  0xFA,                         /* 8  bMaxPower (in 2mA units) - 500mA */

  /* Interface Association Descriptor */
  0x08,                         /* 0 bLength */
  USB_DESCTYPE_INTERFACE_ASSOCIATION,  /* 1 bDescriptorType - Interface Association */
  0x00,                         /* 2 bFirstInterface - VideoControl i/f */
  0x02,                         /* 3 bInterfaceCount - 2 Interfaces */
  USB_CLASS_VIDEO,              /* 4 bFunctionClass - Video Class */
  USB_VIDEO_INTERFACE_COLLECTION,  /* 5 bFunctionSubClass - Video Interface Collection */
  0x00,                         /* 6 bFunctionProtocol - No protocol */
  0x02,                         /* 7 iFunction - index of string */

  /* Video Control (VC) Interface Descriptor */
  0x09,                         /* 0 bLength */
  USB_DESCTYPE_INTERFACE,       /* 1 bDescriptorType - Interface */
  0x00,                         /* 2 bInterfaceNumber - Interface 0 */
  0x00,                         /* 3 bAlternateSetting */
  0x01,                         /* 4 bNumEndpoints */
  USB_CLASS_VIDEO,              /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_CONTROL,            /* 6 bInterfaceSubClass - VideoControl Interface */
  0x00,                         /* 7 bInterfaceProtocol - No protocol */
  0x02,                         /* 8 iInterface - Index of string (same as iFunction of IAD) */

  /* Class-specific VC Interface Header Descriptor */
  0x0D,                         /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VC_HEADER,                /* 2 bDescriptorSubType - HEADER */
  0x10, 0x01,                   /* 3 bcdUVC - Video class revision 1.1 */
  0x28, 0x00,                   /* 5 wTotalLength - till output terminal */
  WORD_CHARS(100000000),        /* 7 dwClockFrequency - 100MHz (Deprecated) */
  0x01,                         /* 11 bInCollection - One Streaming Interface */
  0x01,                         /* 12 baInterfaceNr - Number of the Streaming interface */

  /* Input Terminal (Camera) Descriptor - Represents the CCD sensor (Simulated here in this demo) */
  0x12,                         /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VC_INPUT_TERMINAL,        /* 2 bDescriptorSubType - INPUT TERMINAL */
  0x01,                         /* 3 bTerminalID */
  0x01, 0x02,                   /* 4 wTerminalType - ITT_CAMERA type (CCD Sensor) */
  0x00,                         /* 6 bAssocTerminal - No association */
  0x00,                         /* 7 iTerminal - Unused */
  0x00, 0x00,                   /* 8 wObjectiveFocalLengthMin - No optical zoom supported */
  0x00, 0x00,                   /* 10 wObjectiveFocalLengthMax - No optical zoom supported*/
  0x00, 0x00,                   /* 12 wOcularFocalLength - No optical zoom supported */
  0x03,                         /* 14 bControlSize - 3 bytes */
  0x00, 0x00, 0x00,             /* 15 bmControls - No controls are supported */

  /* Output Terminal Descriptor */
  0x09,                         /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
```

```
  USB_VC_OUPUT_TERMINAL,      /* 2 bDescriptorSubType - OUTPUT TERMINAL  */
  0x02,                       /* 3 bTerminalID */
  0x01, 0x01,                 /* 4 wTerminalType - TT_STREAMING type */
  0x00,                       /* 6 bAssocTerminal - No association */
  0x01,                       /* 7 bSourceID - Source is Input terminal 1 */
  0x00,                       /* 8 iTerminal - Unused */

  /* Standard Interrupt Endpoint Descriptor */
  0x07,                       /* 0 bLength */
  USB_DESCTYPE_ENDPOINT,      /* 1 bDescriptorType */
  (VIDEO_STATUS_EP_NUM | 0x80), /* 2 bEndpointAddress - IN endpoint*/
  0x03,                       /* 3 bmAttributes - Interrupt transfer */
  0x40, 0x00,                 /* 4 wMaxPacketSize - 64 bytes */
  0x09,                       /* 6 bInterval - 2^(9-1) microframes = 32ms */

  /* Class-specific Interrupt Endpoint Descriptor */
  0x05,                       /* 0 bLength */
  USB_DESCTYPE_CS_ENDPOINT,   /* 1 bDescriptorType - Class-specific Endpoint */
  0x03,                       /* 2 bDescriptorSubType - Interrupt Endpoint */
  0x40, 0x00,                 /* 3 wMaxTransferSize - 64 bytes */

  /* Video Streaming Interface Descriptor */
  /* Zero-bandwidth Alternate Setting 0 */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE,     /* 1 bDescriptorType - Interface */
  0x01,                       /* 2 bInterfaceNumber - Interface 1 */
  0x00,                       /* 3 bAlternateSetting - 0 */
  0x00,                       /* 4 bNumEndpoints - No bandwidth used */
  USB_CLASS_VIDEO,            /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_STREAMING,        /* 6 bInterfaceSubClass - VideoStreaming Interface */
  0x00,                       /* 7 bInterfaceProtocol - No protocol */
  0x00,                       /* 8 iInterface - Unused */

  /* Class-specific VS Interface Input Header Descriptor */
  0x0E,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_INPUT_HEADER,        /* 2 bDescriptorSubType - INPUT HEADER */
  0x01,                       /* 3 bNumFormats - One format supported */
  0x47, 0x00,                 /* 4 wTotalLength - Size of class-specific VS descriptors */
  (VIDEO_DATA_EP_NUM | 0x80), /* 6 bEndpointAddress - Iso EP for video streaming */
  0x00,                       /* 7 bmInfo - No dynamic format change */
  0x02,                       /* 8 bTerminalLink - Denotes the Output Terminal */
  0x01,                       /* 9 bStillCaptureMethod - Method 1 supported */
  0x00,                       /* 10 bTriggerSupport - No Hardware Trigger */
  0x00,                       /* 11 bTriggerUsage */
  0x01,                       /* 12 bControlSize - 1 byte */
  0x00,                       /* 13 bmaControls - No Controls supported */

  /* Class-specific VS Format Descriptor */
  0x1B,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_FORMAT_UNCOMPRESSED, /* 2 bDescriptorSubType - FORMAT UNCOMPRESSED */
  0x01,                       /* 3 bFormatIndex */
  0x01,                       /* 4 bNumFrameDescriptors - 1 Frame descriptor followed */
  0x59,0x55,0x59,0x32,
  0x00,0x00,0x10,0x00,
  0x80,0x00,0x00,0xAA,
  0x00,0x38,0x9B,0x71,        /* 5 guidFormat - YUY2 Video format */
  BITS_PER_PIXEL,             /* 21 bBitsPerPixel - 16 bits */
  0x01,                       /* 22 bDefaultFrameIndex */
  0x00,                       /* 23 bAspectRatioX */
  0x00,                       /* 24 bAspectRatioY */
  0x00,                       /* 25 bmInterlaceFlags - No interlaced mode */
  0x00,                       /* 26 bCopyProtect - No restrictions on duplication */

  /* Class-specific VS Frame Descriptor */
  0x1E,                       /* 0 bLength */
  USB_DESCTYPE_CS_INTERFACE,  /* 1 bDescriptorType - Class-specific Interface */
  USB_VS_FRAME_UNCOMPRESSED,  /* 2 bDescriptorSubType */
  0x01,                       /* 3 bFrameIndex */
  0x01,                       /* 4 bmCapabilities - Still image capture method 1 */
  SHORT_CHARS(WIDTH),         /* 5 wWidth - 480 pixels */
  SHORT_CHARS(HEIGHT),        /* 7 wHeight - 320 pixels */
  WORD_CHARS(MIN_BIT_RATE),   /* 9 dwMinBitRate */
  WORD_CHARS(MAX_BIT_RATE),   /* 13 dwMaxBitRate */
```

```
  WORD_CHARS(MAX_FRAME_SIZE), /* 17 dwMaxVideoFrameBufSize */
  WORD_CHARS(FRAME_INTERVAL), /* 21 dwDefaultFrameInterval (in 100ns units) */
  0x01,                       /* 25 bFrameIntervalType */
  WORD_CHARS(FRAME_INTERVAL), /* 26 dwFrameInterval (in 100ns units) */

  /* Video Streaming Interface Descriptor */
  /* Alternate Setting 1 */
  0x09,                       /* 0 bLength */
  USB_DESCTYPE_INTERFACE,     /* 1 bDescriptorType - Interface */
  0x01,                       /* 2 bInterfaceNumber - Interface 1 */
  0x01,                       /* 3 bAlternateSetting -  1 */
  0x01,                       /* 4 bNumEndpoints */
  USB_CLASS_VIDEO,            /* 5 bInterfaceClass - Video Class */
  USB_VIDEO_STREAMING,        /* 6 bInterfaceSubClass - VideoStreaming Interface */
  0x00,                       /* 7 bInterfaceProtocol - No protocol */
  0x00,                       /* 8 iInterface - Unused */

  /* Standard VS Isochronous Video Data Endpoint Descriptor */
  0x07,                       /* 0 bLength */
  USB_DESCTYPE_ENDPOINT,      /* 1 bDescriptorType */
  (VIDEO_DATA_EP_NUM | 0x80), /* 2 bEndpointAddress - IN Endpoint */
  0x05,                       /* 3 bmAttributes - Isochronous EP (Asynchronous) */
  0x00, 0x04,                 /* 4 wMaxPacketSize 1x 1024 bytes*/
  0x01,                       /* 6 bInterval */

};

unsafe{
  /* String table - unsafe as accessed via shared memory */
  static char * unsafe stringDescriptors[]=
  {
    "\x09\x04",             /* Language ID string (US English) */
    "XMOS",                 /* iManufacturer */
    "XMOS USB Video Device",/* iProduct */
    "Config",               /* iConfiguration string */
  };
}

/* Endpoint 0 handles both std USB requests and Video class-specific requests */
void Endpoint0(chanend chan_ep0_out, chanend chan_ep0_in)
{
    USB_SetupPacket_t sp;

    unsigned bmRequestType;
    XUD_BusSpeed_t usbBusSpeed;

    XUD_ep ep0_out = XUD_InitEp(chan_ep0_out, XUD_EPTYPE_CTL | XUD_STATUS_ENABLE);
    XUD_ep ep0_in = XUD_InitEp(chan_ep0_in, XUD_EPTYPE_CTL | XUD_STATUS_ENABLE);

    UVC_InitProbeCommitData();

    while(1)
    {
       /* Returns XUD_RES_OKAY on success */
       XUD_Result_t result = USB_GetSetupPacket(ep0_out, ep0_in, sp);

       if(result == XUD_RES_OKAY)
       {
           /* Set result to ERR, we expect it to get set to OKAY if a request is handled */
           result = XUD_RES_ERR;

           /* Stick bmRequest type back together for an easier parse... */
           bmRequestType = (sp.bmRequestType.Direction<<7) |
                           (sp.bmRequestType.Type<<5) |
                           (sp.bmRequestType.Recipient);

           if ((bmRequestType == USB_BMREQ_H2D_STANDARD_DEV) &&
               (sp.bRequest == USB_SET_ADDRESS))
           {
             // Host has set device address, value contained in sp.wValue
           }

           switch(bmRequestType)
           {
               /* Direction: Device-to-host and Host-to-device
```

```
                 * Type: Class
                 * Recipient: Interface / Endpoint
                 */
                case USB_BMREQ_H2D_CLASS_INT:
                case USB_BMREQ_D2H_CLASS_INT:
                case USB_BMREQ_H2D_CLASS_EP:
                case USB_BMREQ_D2H_CLASS_EP:

                    /* Inspect for VideoControl Class interface number or
                     * VideoStreaming Class interface number or EP number;
                     * If an Entity is addressed, the High byte has to be checked
                     * for Entity ID */
                    if(sp.wIndex == 0 || sp.wIndex == 1 || sp.wIndex == (VIDEO_DATA_EP_NUM | 0x80))
                    {
                        /* Returns  XUD_RES_OKAY if handled,
                         *          XUD_RES_ERR if not handled,
                         *          XUD_RES_RST for bus reset */
                        result = UVC_InterfaceClassRequests(ep0_out, ep0_in, sp);
                    }
                    break;
            }
        } /* if ends */

        /* If we haven't handled the request about then do standard enumeration requests */
        if(result == XUD_RES_ERR )
        {
            /* Returns  XUD_RES_OKAY if handled okay,
             *          XUD_RES_ERR if request was not handled (STALLed),
             *          XUD_RES_RST for USB Reset */
            unsafe{
            result = USB_StandardRequests(ep0_out, ep0_in, devDesc,
                        sizeof(devDesc), cfgDesc, sizeof(cfgDesc),
                        null, 0, null, 0, stringDescriptors, sizeof(stringDescriptors)/sizeof(stringDescriptors
                           ↪ [0]),
                        sp, usbBusSpeed);
             }
        }

        /* USB bus reset detected, reset EP and get new bus speed */
        if(result == XUD_RES_RST)
        {
            usbBusSpeed = XUD_ResetEndpoint(ep0_out, ep0_in);
        }
    }
}

/* Buffer to hold Video data in YUYV format */
unsigned int gVideoBuffer[3][PAYLOAD_SIZE / 4];

/* Function to handle all endpoints of the Video class excluding control endpoint0 */
void VideoEndpointsHandler(chanend c_epint_in, chanend c_episo_in)
{
    XUD_Result_t result;
    int frame = 0x0C;
    int pts, tmrValue = 0;
    timer presentationTimer;

    int sofCounts = 0, frameCounts = 0;
    unsigned int index = 0;
    unsigned int i_index = 0;
    int split = (MAX_FRAME_SIZE / 6);
    int i_split = (MAX_FRAME_SIZE / 6);

    /* Initialize all endpoints */
    XUD_ep epint_in = XUD_InitEp(c_epint_in, XUD_EPTYPE_INT);
    XUD_ep episo_in = XUD_InitEp(c_episo_in, XUD_EPTYPE_ISO);

    /* Just to keep compiler happy */
    epint_in = epint_in;
    /* XUD will NAK if the endpoint is not ready to communicate with XUD */

    /* Fill video buffers with different color data */
    for(int i = 0; i < (PAYLOAD_SIZE/4); i++) {
        /* Set RED color */
        gVideoBuffer[0][i] = 0x7010D010;
```

```
        /* Set GREEN color */
        gVideoBuffer[2][i] = 0x00000000;
        /* Set BLUE color */
        gVideoBuffer[1][i] = 0xDC206020;
    }

    while(1)
    {
        int expectedPixels =  MAX_FRAME_SIZE;
        presentationTimer :> pts;

        /* Fill the buffers with payload header */
        for(int i=0; i<3; i++)
        {
            /* Make the Payload header */
            (gVideoBuffer[i], unsigned char[])[0] = PAYLOAD_HEADER_LENGTH;
            (gVideoBuffer[i], unsigned char[])[1] = frame;
            /* Set dwPresentationTime */
            (gVideoBuffer[i], unsigned short[])[1] = pts;
            (gVideoBuffer[i], unsigned short[])[2] = pts>>16;
            /* Set scrSourceClock */
            (gVideoBuffer[i], unsigned short[])[3] = pts;
            (gVideoBuffer[i], unsigned short[])[4] = pts>>16;
            (gVideoBuffer[i], unsigned short[])[5] = (sofCounts>>3) & 2047;
        }

        /* Just to simulate the motion in the video frames */
        i_split = (i_split - ((WIDTH)*8));
        if(i_split <= 0) {
            i_split = MAX_FRAME_SIZE / 6;
            i_index = (i_index + 1) % 3;
        }
        presentationTimer :> tmrValue;

        /* Let the frames scroll */
        index = i_index;
        split = i_split;

        /* Transmits single frame */
        while(expectedPixels > 0)
        {
            if(expectedPixels < (PAYLOAD_SIZE - PAYLOAD_HEADER_LENGTH)) {
                /* Payload transfer */
                result = XUD_SetBuffer(episo_in, (gVideoBuffer[index], unsigned char[]), expectedPixels+
                  ↪ PAYLOAD_HEADER_LENGTH);
            } else {
                /* Payload transfer */
                result = XUD_SetBuffer(episo_in, (gVideoBuffer[index], unsigned char[]), 1024);
            }
            /* Note down the SOF counts */
            sofCounts++;

            expectedPixels -= ((PAYLOAD_SIZE)- PAYLOAD_HEADER_LENGTH);

            if(expectedPixels <= (MAX_FRAME_SIZE - split)) {
                index = (index + 1) % 3;
                split += (MAX_FRAME_SIZE / 6);
            }
        }
        frame = frame ^ 1;  /* Toggle FID bit */
        frameCounts++;
    }
}
```

its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.