USB VIDEO SWITCH: IMPLEMENTATION DETAILS

By

Ammar Hattab

Prof. Gabriel Taubin

Brown University

Abstract

This paper describes the implementation details of a low cost USB video switch device, which can combine the video from multiple UVC compliant USB cameras to emulate a real UVC webcam gadget want (capturing live video from different video capture source) which can be used with one USB cable, and when connected to the PC, it is recognized as a single USB Camera of a special type, and could be used in different applications like Skype..etc.

Introduction

There are many applications that require switching between several video sources (or combining them), it's especially important in live broadcasting systems where the production switcher is an essential element and also in the video conferencing applications [1].

The problem with current video switching systems is the high cost and complexity of the used equipment which had limited their usage; in broadcasting industry most vision mixers are targeted at the professional market [1]; many of them using giant switchers with perhaps over 50 inputs, while in video conferencing there are two kinds of systems: dedicated systems which have all the required components packaged into a single piece of equipment, and desktop systems which integrates everything into a single PC, and their complexity varies from the large group videoconferencing to the individual videoconferencing, and in many situations, the application requires a simple and low cost video switching solution.

USB cameras are now used everywhere which provides the user with a low-cost, convenient, and simple connectivity solution with ease of use and installation, with many high definition versions already available to a level that is sufficient for many applications, so finding a way to switch/combine the video from multiple USB cameras is very useful and has many applications.

Embedded Linux Development

There are many types of embedded systems with different tools, resources and development levels, one important type is the embedded Linux-based environment which is already running in many consumer electronic devices (ex: Android Phones).

The general established practice for working with embedded Linux devices is to develop and build software on a 'host' PC system (on which all of the general tools are available) and then copy the results to the BeagleBoard using its network connection, or by directly using its microSD card.

The kernel must be compiled for whatever target architecture it will run on, and generally, its compiled using a cross-compiler – a special version of GCC that is able to run on a fast desktop PC, creating binary code that will run on an ARM processor instead of the PC processor.

Linux kernel functionality could be extended using kernel modules (instead of having all functionality in one big kernel image); one type of module is the device driver, which allows the kernel to access hardware connected to the system [2].

We should distinguish between two types of drivers: Device drivers and Gadget drivers, device drivers communicates and manage the functionality of devices connected to Linux system ("host" role), and Gadget driver allows the Linux system to act in the "slave" role.

One important category of devices that could be connected to computer are USB devices, USB was designed to standardize the connection of peripherals to personal computers, and there are a wide range of USB devices intended for a wide range of purposes. There are USB device driver (host drivers) and also USB gadget driver (slave drivers).

The functionality of USB devices is defined by class codes, communicated to the USB host to affect the loading of suitable software driver modules for each connected device. This provides for adaptability and device independence of the host to support new devices from different manufacturers.

One of these classes is the USB Device Class Definition for Video Devices, or USB Video Class (**UVC**) which is a specification that defines video streaming functionality on the Universal Serial Bus. It allows for interconnectivity of webcams to computers without the need for proprietary device drivers. Microsoft Windows XP SP2, Linux and Mac OS X have UVC support built in and do not require extra device drivers, although they are often installed to add additional features, that is much like nearly all mass storage devices (USB flash disks, external SATA disk enclosures,) can be managed by a single driver because they conform to the USB Mass Storage specification; UVC compliant peripherals only need a generic driver. [3]

The UVC specification covers webcams, digital camcorders, analog video converters, analog and digital television tuners, and still-image cameras that support video streaming for both video input and output.

The UVC drivers use the Video4Linux 2 (V4L2) API in Linux which is a video capture application programming interface for Linux, supporting many USB webcams, TV tuners, and other devices. V4L2 is closely integrated with the Linux kernel. V4L2 has two important parts; the video capture interface which grabs video data from a tuner or camera device; and the video output interface allow applications to drive peripherals which can provide video images outside of the computer [4].

V4L2 in the drivers (in the kernel space) exposes its functionality to the user-space applications using ioctl() interface to perform the various video functions (whether it's a capture device or output device), this gave the developers much flexibility to write their applications; among the important ioctl are vidioc_querycap to check for driver capabilities, vidioc_reqbufs to establish a set of buffers, vidioc_querybuf to query the status of a buffer, mmap to map buffers into its address space, vidioc_qbuf to enqueue a buffer which means putting it into the driver's incoming queue, and vidioc_dqbuf to dequeue a filled (capturing) or displayed (output) buffer from the driver's outgoing queue.[5]

BeagleBoard:

In order to remove the barriers from embedded Linux-based development a small group of volunteers (some of them from Texas Instruments) made BeagleBoard which is a small, (relatively) low-cost, durable (hard to 'brick') ARM-based development platform that couples an open source hardware design that anyone can look at with open source software. [2]

BeagleBoard is based upon the ARM Cortex-A8 System-on-Chip (SoC) processor. This means that BeagleBoard uses software compiled using the ARM Instruction Set Architecture ('ARM arch'), and Linux has been available for the ARM architecture for many years now

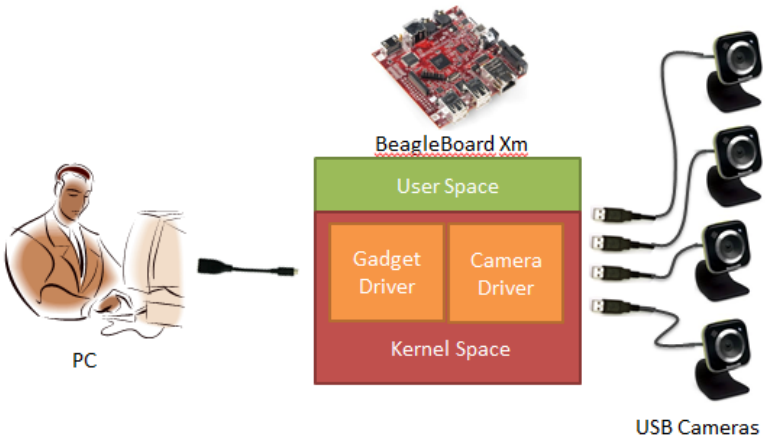
BeagleBoard Xm version has a high-speed USB 2.0 OTG port with four-port high-speed USB 2.0 hub, which make it very suitable for the USB video switch application.

Development Environment

In order to overcome many of the development challenges we chose to use TI Sitara Linux SDK for BeagleBoard Xm, which provides an easy set up and quick out-of-box experience, the SDK is also free, and does not require any run-time royalties to Texas Instruments, the SDK contains the Linux kernel for BeagleBoard Xm, which make it easy to modify and compile the kernel and then to deploy it to the BeagleBoard, it also includes a special IDE called Code Composer Studio™ IDE v5 for Linux development, which make it easy to write user-space applications, to compile them, to copy the result to BeagleBoard using drag and drop and to debug them. [6]

System Components

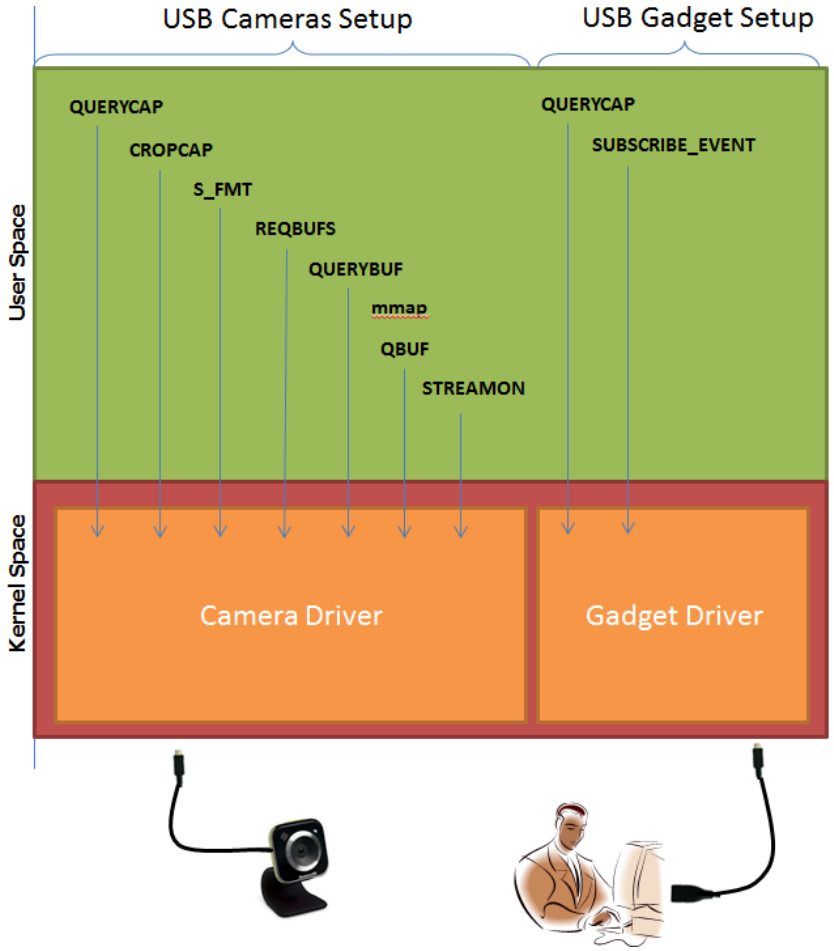
The basic system components are the BeagleBoard Xm running the application connected to PC using a USB cable from the USB OTG port in the BeagleBoard, and four Microsoft LifeCam HD-5000 USB cameras connected to the host USB port in the BeagleBoard, when connected to the PC, the BeagleBoard is recognized as a single USB Camera of a special type, and could be used in different applications like Skype, the user space application talks with the two drivers; the USB gadget driver (to send the video to the PC) and the USB camera driver (to receive video from the USB cameras) using ioctl system calls, the two drivers are kernel modules that should be loaded before running the user space application, the UVC webcam gadget kernel driver is located under drivers/usb/gadget directory and combilable as g_webcam.ko kernel module.



The Application

The two USB drivers (Gadget and device drivers) in the kernel space expose their functionality to user space using ioctl system calls, so video application in the user space (whether it's for capturing or for outputting) will do several ioctl to the driver in order to perform any V4L2 functionality, in our case the same user space application will do both capturing and outputting functions at the same time, where it will be capturing images from the USB cameras connected to BeagleBoard using the device driver, and it will be outputting combined images.

The application will do a basic setup which will allocate and map the video buffers in the drivers to the user space, and then it will start an infinite loop that includes taking images from the video buffers of the four USB cameras driver and combine them, then outputting the combined image to the video buffer of the gadget driver, the following diagram shows the setup process in both user space and kernel space:



The setup process starts by enumerating the available inputs (Camera driver side) and outputs (Gadget driver side) and choose between them, then it has to come to an agreement with the driver on an actual video format supported by the hardware, the V4L2 API describes image formats using a specific structure that contains the image width, height, pixelformat, bytesperline, interlacing field and colorspace, which all together describe a buffer of video data in a reasonably complete manner, the first step is to query the

supported formats using the VIDIOC_ENUM_FMT ioctl, then when the application wants to change the hardware's format for real, it could do a VIDIOC_S_FMT call, where the driver should first ensure that the hardware is idle before changing that format, in our case and for simplicity we have used YUV uncompressed video format, with 720p frame resolution in the UVC gadget driver side, and to align the USB cameras we have used the same YUV video format with 360p frame resolution to be able to combine all the four images in one image.

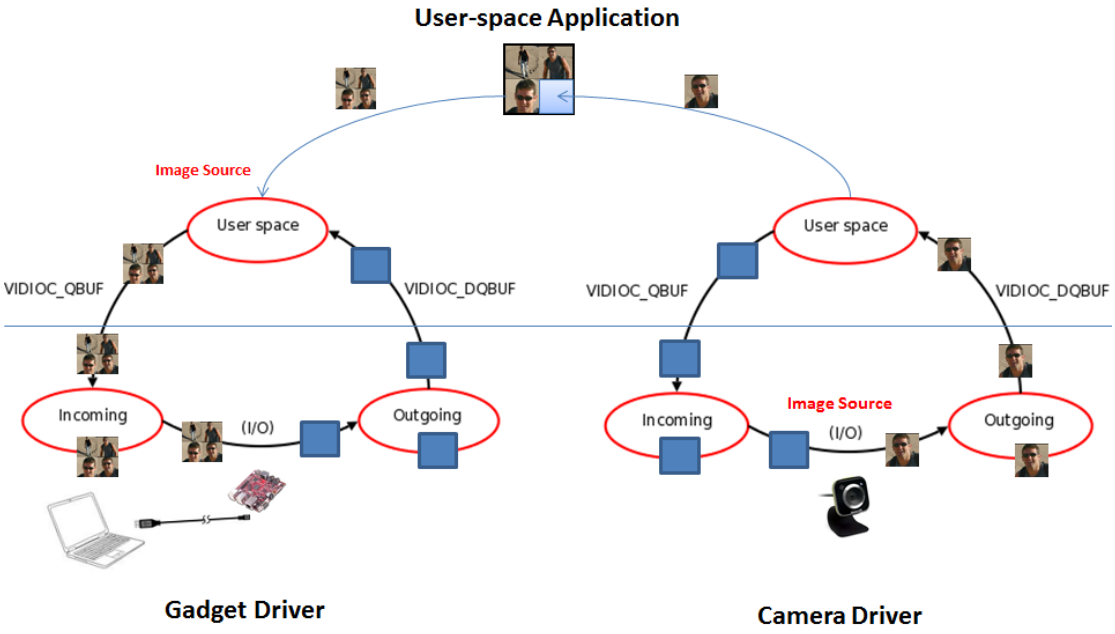
Then it should check for driver capabilities (using vidioc_querycap call). Once a streaming application has performed its basic setup, it will turn to the task of organizing its I/O buffers. the first step is to establish a set of buffers (using vidioc_reqbufs call), V4L2 framework supports a number of IO methods and to achieve zero-memcpy of video frames as they pass from V4L2 video-capture domain to UVC video-output domain, it is important to select appropriate IO methods for working on videobuffers, normally the supported IO methods are IO_METHOD_USERPTR and IO_METHOD_MMAP, in the case of Memory-mapped buffers the application will typically step through each allocated buffer (using vidioc_querybuf call) and map it into its address space (using mmap call)

Once finished establishing the buffers and mapping them, and before the application can start streaming I/O, it must put at least one buffer into the driver's incoming queue (this is called enqueuing a buffer using vidioc_qbuf call); for an output device, those buffers should also be filled with valid frame data.

Then there is the matter of telling the device to transfer image data to (or from) the user-space buffer. This buffer will not be contiguous in physical memory - it will, instead, be broken up into a large number of separate 4096-byte pages (on most architectures). If your device transfers images in smaller pieces (a USB camera, for example), direct DMA to user space may be easier to set up.

The only remaining step is to actually tell the device to start performing streaming I/O. The Video4Linux2 driver method for this task is vidioc_streamon, the call to vidioc_streamon() should start the device after checking that type makes sense. The driver can, if need be, require that a certain number of buffers be in the incoming queue before streaming can be started.

Once setup process is finished and streaming I/O starts, the systems goes into an infinite loop illustrated in the following diagram:



In the Camera driver side an empty buffer is available in the incoming queue (by the use of a VIDIOC_QBUF ioctl) the driver will grab it and have the USB camera perform the requested I/O transfer (capture the image), and then move the buffer to the outgoing queue. Eventually the application will want to claim buffers in the outgoing queue, returning them to the user-space state by using VIDIOC_DQBUF ioctl, so the driver will remove the first buffer from the outgoing queue, and return it to the user space, normally, if the outgoing queue is empty, this call should block until a buffer becomes available. V4L2 drivers are expected to handle non-blocking I/O, though, so if the video device has been opened with O_NONBLOCK, the driver should return -EAGAIN in the empty-queue case.

The user-space application will take the images from the four cameras and combine them in a single image and make it ready to be transferred to the UVC gadget side (to the PC).

So in the UVC webcam gadget side, the user-space application will fill an empty buffer (that was requested from the gadget driver outgoing queue using VIDIOC_DQBUF ioctl) with the previous combined image, and then the user-space application will send it to the UVC gadget driver incoming queue using VIDIOC_QBUF ioctl, the driver will grab the filled buffer from its incoming queue, have the UVC device perform the requested I/O transfer (to actually output the image through the USB cable to the PC), and then move the empty buffer to the outgoing queue to be used again.

When the application is done it should generate a call to vidioc_streamoff(), which must stop the device. The driver should also remove all buffers from both the incoming and outgoing queues, leaving them all in the user-space state.

Implementation Challenges

The most challenging part of embedded Linux development is choosing the Linux kernel version and choosing development environment and tools, we need a specific kernel version that contains the latest UVC drivers (gadget and device drivers) and also customized to work on BeagleBoard, it took a long time to try different kernel version (trying to compile, upload, and test them). For the development environment we wanted to have an easy way to modify the kernel/user space code, also to compile and test it on BeagleBoard, and especially we wanted an easy way for debugging as it is a very important part of the development in this area. And eventually we found the best development environment was TI Sitara SDK which includes the latest kernel source that contains the latest UVC drivers.

Another challenging issue was to make the UVC gadget driver actually working on BeagleBoard, because it's a newly developed driver and area, and to make it actually work on BeagleBoard required a lot of testing and debugging effort, one of the problems that we found is that Direct Memory Access (DMA) is not working correctly in BeagleBoard, then we discovered that it's a known issue, so to make the UVC driver working we have to disable the DMA in BeagleBoard kernel configurations.

Conclusion

In this paper we provided the implementation details of a low cost USB video switch built using BeagleBoard Xm, combining the images of four USB cameras and using the new USB webcam gadget driver in Linux, the implementation provides a low-cost, convenient, and simple solution to combine video of high definition USB cameras. In the future this implementation could be expanded to include switching for voice as well, and eventually it could be evolved into a full video conferencing solution.

References

[1] Luff, John: "Production switchers". *Broadcast Engineering*, November 1, 2002. Retrieved online on 7/3/2013 from http://broadcastengineering.com/products/production-switchers-1

[2] Masters, Jon: “An introduction to Embedded Linux, BeagleBoard & its Linux kernel port”, *Linux User & Developer*. Retrieved online on 7/3/2013 from http://www.linuxuser.co.uk/features/an-introduction-to-embedded-linux-beagleboard-its-linux-kernel-port/1

[3] Pinchart, Laurent: “Linux UVC driver and tools”. Retrieved online on 7/3/2013 from http://www.ideasonboard.org/uvc/

[4] Bill Dirks, Michael H. Schimek, Hans Verkuil, Martin Rubli, Andy Walls, Mauro Carvalho Chehab: “Video for Linux Two API Specification”. Retrieved online on 7/3/2013 from http://www.linuxtv.org/downloads/legacy/video4linux/API/V4L2_API/spec-single/v4l2.html

[5] *The Video4Linux2 API: an introduction*, retrieved online on 7/3/2013 from http://lwn.net/Articles/203924/

[6] Linux EZ Software Development Kit (EZSDK) for Sitara™ ARM® Processors. Retrieved online on 7/3/2013 from http://www.ti.com/tool/linuxezsdk-sitara