

## How to Interface a MIPI® CSI-2 Image Sensor with EZ-USB® CX3™

Author: Manu Kumar, Savan Javia

Software Version: EZ-USB FX3 SDK1.3.3

Related Resources: [Click Here](#)

### More code examples? We heard you.

For a consolidated list of USB SuperSpeed Code Examples, visit <http://www.cypress.com/101781>.

The high bandwidth provided by USB 3.1 Gen 1 (earlier referred as USB 3.0) puts high demands on ICs that connect peripherals to USB. This application note focuses on a popular USB 3.1 Gen 1 application: a camera (MIPI CSI-2 image sensor interfaced with EZ-USB CX3) streaming uncompressed data into a PC. This application note also gives implementation details about the USB Video Class (UVC). Conforming to this class allows the camera device to operate using built-in PC drivers and Host applications, such as e-CAMView, Media Player Classic and VLC Media Player.

## Contents

1	Introduction.....	1	5.6	Configuring the Image Sensor.....	28
2	Interfacing an Image Sensor to CX3.....	3	5.7	Starting Video Streaming.....	29
2.1	MIPI CSI-2 Interface .....	3	5.8	Selecting and Switching Frame Settings .....	29
2.2	GPIF II Block.....	5	5.9	Setting Up DMA Buffers.....	29
3	Setting Up the DMA System .....	7	5.10	Handling DMA Buffers During Video Streaming.....	29
3.1	DMA Buffers .....	12	5.11	Resuming the Stream at the End of a Frame .....	30
4	USB Video Class (UVC) .....	13	5.12	Terminating the Video Stream .....	30
4.1	Enumeration Data.....	13	6	Hardware Setup .....	30
4.2	Operational Code.....	14	6.1	Testing With the CX3 RDK .....	30
4.3	USB Video Class Requirements .....	14	6.2	Designing Your Own Board .....	31
5	CX3 Firmware.....	22	7	UVC-Based Host Applications.....	31
5.1	Application Threads .....	24	8	Troubleshooting.....	31
5.2	Handling UVC Requests.....	24	9	Summary .....	32
5.3	Initialization .....	25	10	Related Resources.....	33
5.4	Enumeration .....	25			
5.5	Configuring the MIPI CSI-2 Controller .....	25			

## 1 Introduction

The high bandwidth provided by USB 3.1 Gen 1 puts heavy demands on ICs that connect peripherals to USB. A popular example is a camera streaming uncompressed data into a PC. As USB bandwidth has increased with version 3.1 Gen 1, so has the resolution of cameras that attach to PCs. To address high-bandwidth camera interface requirements, an alliance called Mobile Industry Processor Interface (MIPI) created a specification called Camera Serial Interface 2 (CSI-2).

This application note provides the implementation details for a MIPI CSI-2-to-USB 3.1 converter based on Cypress's EZ-USB CX3, a variant of EZ-USB FX3™ created specifically for this purpose.

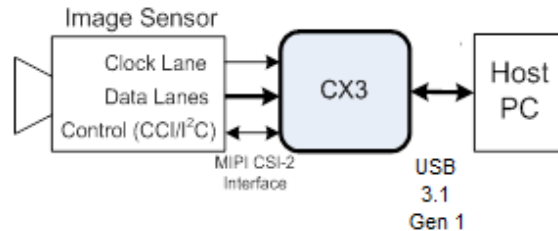
If you are new to Cypress EZ-USB product family, please read the application note “AN75705 - Getting Started with EZ-USB® FX3™” so that you can easily understand this application note.

CX3 converts data coming from the image sensor into a format compatible with USB Video Class (UVC). Conforming to this class allows the camera to operate using built-in OS drivers, making the camera compatible with Host applications, such as e-CAMView, Media Player Classic and VLC Media Player.

A derivative of FX3, CX3 differs from FX3 as follows:

- A MIPI CSI-2 controller with a MIPI CSI-2 receiver interface is added.
- USB 2.0 OTG and Charger Detection functionality are removed.
- Two clock references are needed: CLKIN for the core and REFCLK for the MIPI CSI-2 Controller.
- Only a 19.2-MHz oscillator is supported for CLKIN.

Figure 1. The Camera Application



Because CX3 is a specialized version of FX3, all FX3 development tools and most FX3 literature apply to CX3. An example of this compatibility is [AN75779](#), which describes another camera design (with a parallel interface) based on FX3. Much of the background material about the inner workings of CX3 is taken from that note because the data transfer architectures in both chips are identical.

This application note provides details intended to help you understand the available Cypress firmware project. If your design uses the same image sensor as the one described in this note, little or no code modification is necessary. For a different sensor, the note points out the code modules that require minor modification.

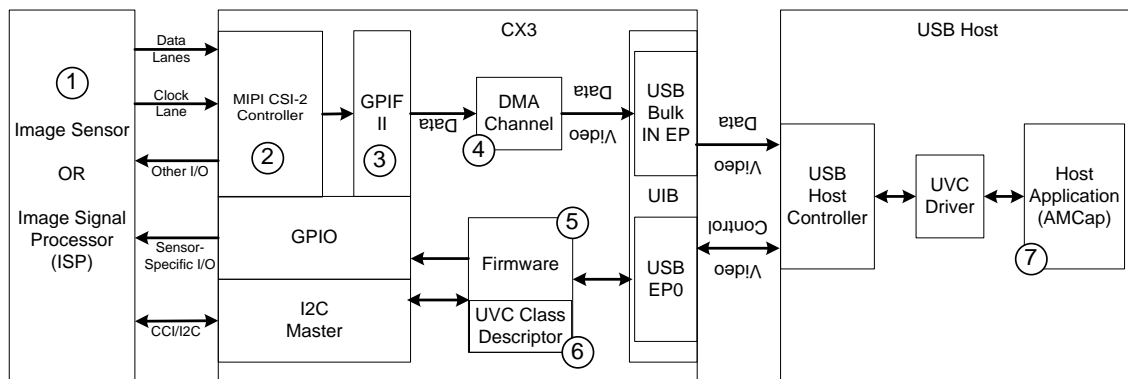
[Figure 1](#) illustrates the camera application. On the right side is a PC equipped with a SuperSpeed USB 3.1 port. On the left side is an image sensor with a MIPI CSI-2 interface that can support the following:

- One to four MIPI CSI-2 data lanes
- RAW8/10/12/14, YUV422 (CCIR/ITU 8/10-bit), RGB888/666/565, compressed formats like M-JPEG and user-defined 8-bit, 16-bit and 24-bit image formats

For example, the OV5640 sensor supports VGA 60 fps, HD (720p) 60 fps, Full HD (1080p) 30 fps, and 5 M-Pixel 15 fps. The following sections provide details on how the `cycx3_uvc_ov5640` example project (available with FX3/CX3 SDK installation) is designed to stream YUV formats with VGA, HD, and Full HD resolutions.

[Figure 2](#) has sub-blocks of the block diagram numbered. Tasks executed by each sub-block are described.

Figure 2. System Block Diagram



1. Connect the MIPI CSI-2 based image sensor to CX3 and configure it using the Camera Control Interface (CCI) bus.
2. Configure the MIPI CSI-2 controller in CX3 to read image data from the sensor, de-packetize it, and send it to the GPIF II Block. See [Section 2.1](#).
3. Configure the GPIF II block according to the image data format. See [Section 2.2](#).

4. Construct a DMA channel that moves the image data from the GPIF II Block to the USB Interface Block (UIB). In this application, header data must be added to the image sensor's video data to conform to the UVC specification. As such, the DMA is configured to enable the CPU to add the required header to the DMA Buffers. This channel must be designed so that maximum bandwidth can be used to stream video from the image sensor to the PC. See [Section 3](#).
5. The CX3 firmware initializes the hardware blocks of CX3 ([Section 5.3](#)), configures the image sensor and MIPI CSI-2 controller ([Section 5.5](#)), enumerates the device as a UVC camera ([Section 4.3.1](#)), handles UVC-specific requests ([Section 4.3.2](#)), translates video control settings (such as brightness) to the image sensor over the CCI (I<sup>2</sup>C) interface ([Section 5.5](#)), adds a UVC header to the video data stream ([Section 4.3.4](#)), and commits the video data with headers to USB ([Section 5.10](#)).
6. Provide the proper USB Descriptors so that the Host recognizes the peripheral as a device conforming to the UVC. See [Section 4](#).
7. Use a Host application (such as e-CAMView, Media Player Classic or VLC Media Player) that accesses the UVC driver to configure the image sensor over the UVC control interface and to receive video data over the UVC streaming interface. See [Section 7](#).

If the camera is plugged into a USB 2.0 port, the CX3 firmware uses the CCI bus to select reduced frame rate and frame size to accommodate the lower USB bandwidth. The Host can optionally use the Video Control interface to send brightness, contrast, hue, saturation, exposure, auto focus, and PTZ (pan, tilt, zoom) adjustments to the camera.

## 2 Interfacing an Image Sensor to CX3

To connect a MIPI CSI-2 image sensor to CX3 and to read data from it, you need to understand the CSI-2 interface and CX3's DMA capabilities.

### 2.1 MIPI CSI-2 Interface

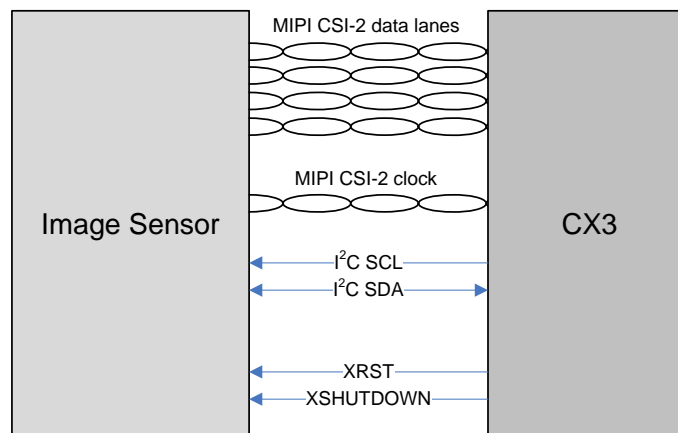
As camera applications become sophisticated, a larger demand is placed on higher-resolution image sensors. This demand pushes the limit on common parallel image sensor interfaces, which can be difficult to expand and require many interconnects. The Mobile Industry Processor Interface (MIPI) Alliance therefore designed the Camera Serial Interface 2 (CSI-2) standard to provide standard, robust, low-power, and high-speed serial interface that supports a wide range of imaging solutions.

The MIPI CSI-2 interface is a unidirectional differential serial interface with data and clock signals. There can be up to four data lanes each transferring data at up to 1 Gbps.

The control interface used to configure the image sensor is compatible with the I<sup>2</sup>C standard and is referred to as Camera Control Interface (CCI).

The MIPI CSI-2 controller in CX3 provides these two interfaces and some additional signals that image sensors commonly require.

Figure 3. CX3's Interface with the Image Sensor



The three additional signals shown in this interface are:

**XRST:** Image sensors usually require a reset signal from CX3 before configuration. This reset can be done using the XRST pin in CX3.

**XSHUTDOWN:** When the Host is not requesting a video stream from the image sensor, the sensor can be shut down or put in a standby mode to reduce power consumption. This is done using the shutdown pin in a sensor, which can be controlled using the CX3 XSHUTDOWN pin.

**MCLK:** The master (or reference) clock required by the image sensor to be supplied using an external clock oscillator.

Refer to the datasheet, [EZ-USB® CX3™ MIPI CSI-2 to SuperSpeed USB Bridge Controller](#), for the pin mapping of the CSI-2, CCI, and the three additional signals. In addition, refer to the [CX3 RDK Schematics](#) for a complete example.

The MIPI CSI-2 controller can be configured for one to four data lanes, different data formats (such as RAW, YUV, or MJPEG), and different camera resolutions. Refer to [Section 5.5](#) for details. This application uses a sensor configured to provide YUV image data on four data lanes.

When configured, the MIPI CSI-2 controller accepts serial image data from the image sensor and de-packetizes and converts it into parallel data to be sent over a parallel interface. This interface uses the following signals:

- FV: Frame Valid (indicates start and end of a frame)
- LV: Line Valid (indicates start and end of a line)
- PCLK: Pixel Clock
- Data: 8-, 16-, and 24-bit data bus for image data

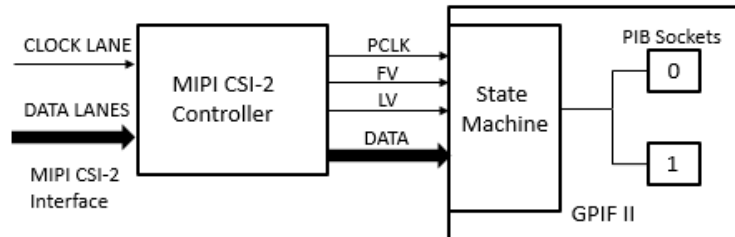
The timings of these signals are similar to the parallel interface described in Section 3.1 of [AN75779](#).

The maximum throughput supported by CX3 is 2.4 Gbps, since the maximum GPIF II data bus width which can be used in CX3 is 24-bit and the maximum PCLK supported is 100 MHz.

## 2.2 GPIF II Block

The parallel output interface from the MIPI CSI-2 controller is connected to the GPIF II Block, which is a part of the Processor Interface Block (PIB).

Figure 4. GPIF II Block



The GPIF II Block uses a state machine to read data from this interface into two DMA sockets. To know how data transfers happen, an understanding of CX3's DMA capabilities is required.

### 2.2.1 DMA Basics

The DMA (Direct Memory Access) of CX3 allows fast and optionally uninterrupted data transfers between any two blocks. To understand how these data transfers happen, it is important to know the following terminology:

- Socket
- DMA Descriptor
- DMA Buffer

A **socket** is a point of connection between a peripheral hardware block and the CX3 RAM. Each peripheral hardware block on CX3 such as USB, GPIF, UART, and SPI has a fixed number of sockets associated with it. The number of independent data flows through a peripheral is equal to the number of its sockets. The socket implementation includes a set of registers that point to the active DMA Descriptor and enable or flag interrupts associated with the socket.

A **DMA Descriptor** is a set of registers allocated in the CX3 RAM. It holds information about the address and size of a DMA Buffer as well as pointers to the next DMA Descriptor. These pointers create DMA Descriptor chains.

A **DMA Buffer** is a section of RAM used for intermediate storage of data transferred through the CX3 device. DMA Buffers are allocated from the RAM by the CX3 firmware, and their addresses are stored as part of DMA Descriptors.

### 2.2.2 Why Two Sockets Are Used

Before understanding why two sockets are used, a more detailed understanding of sockets is required.

Sockets can directly signal each other through events or they can signal the CX3 CPU via interrupts. This signaling is configured by firmware. Consider, for example, a video data stream from the GPIF II block to the USB block. The GPIF socket can tell the USB socket that it has filled data in a DMA Buffer, and the USB socket can tell the GPIF socket that a DMA Buffer has been emptied. This implementation is called an *automatic DMA channel*. The automatic DMA channel implementation is typically used when the CX3 CPU does not have to modify any data in a data stream.

Alternatively, the GPIF socket can send an interrupt to the CX3 CPU to notify it that the GPIF socket has filled a DMA Buffer. The CX3 CPU can relay this information to the USB socket. The USB socket can send an interrupt to the CX3 CPU to notify it that the USB socket has emptied a DMA Buffer. Then the CX3 CPU can relay this information back to the GPIF socket. This implementation is called a *manual DMA channel*.

The manual DMA channel implementation is typically used when the CX3 CPU has to add, remove, or modify data in a data stream. The firmware example described in this application note uses the manual DMA channel implementation because the firmware needs to add a UVC video data header.

A socket that writes data to a DMA Buffer is called a *producer socket*. A socket that reads data from a DMA Buffer is called a *consumer socket*. A socket uses the values of the DMA Buffer address, DMA Buffer size, and DMA Descriptor chain stored in a DMA Descriptor for data management.

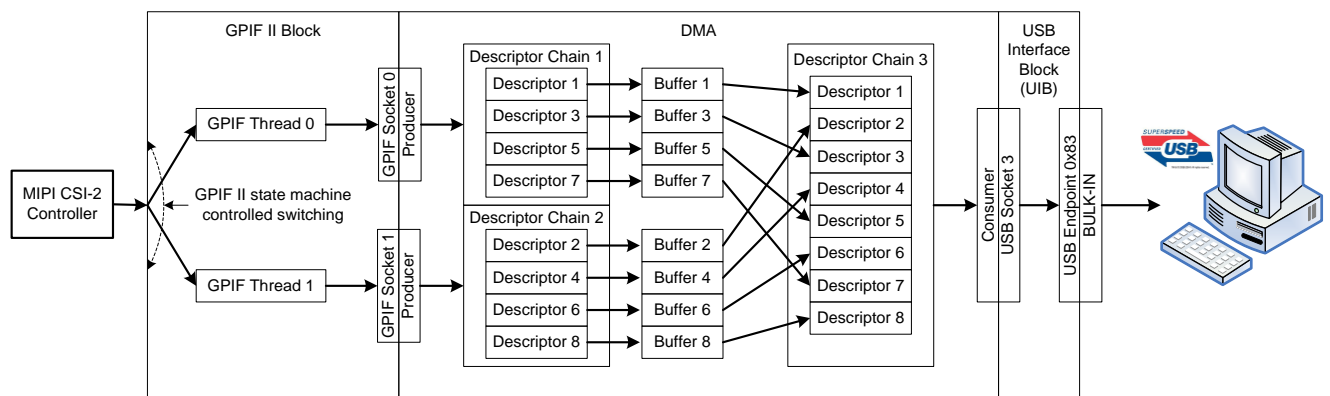
A socket takes a finite amount of time (up to a few microseconds) to switch from one DMA Descriptor to another after it fills or empties a DMA Buffer. The socket cannot transfer data while this switch is in progress. This latency is overcome in the GPIF II block using two GPIF threads.

A GPIF thread is a dedicated data path in the GPIF II block that connects the data pins to a socket. Only one GPIF thread can transfer data at a time.

The GPIF thread selection mechanism is like a MUX. Switching the active GPIF thread switches the active socket for the data transfer, thereby changing the DMA Buffer used for data transfers. This switch has a one-clock-cycle latency, making it essentially instantaneous. The GPIF II state machine implements this switch at a DMA Buffer boundary, thus masking the latency of the GPIF socket switching to a new DMA Descriptor. This allows the GPIF II block to take in data from the sensor without any loss when the DMA Buffer is full.

Figure 5 shows the sockets, DMA Descriptors, and DMA Buffer connections used in this application along with the data flow. Two GPIF threads are used to fill in alternate DMA Buffers. These GPIF threads use separate GPIF sockets (acting as producer sockets) and DMA Descriptor chains (Descriptor chain 1 and Descriptor chain 2). The USB socket (acting as a consumer socket) uses a third DMA Descriptor chain (Descriptor chain 3) to read the data out in the correct order. For more details on sockets, socket switching, and the associated delays, refer to [Section 3](#).

Figure 5. CX3 Data Transfer Architecture



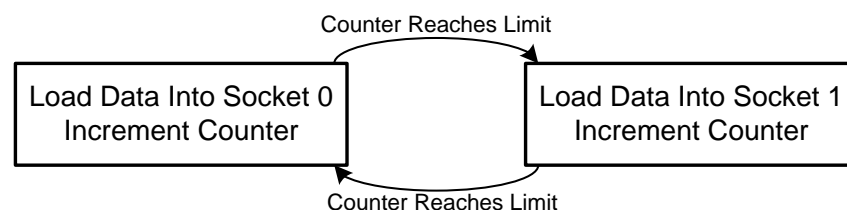
### 2.2.3 GPIF II State Machine

The GPIF II Block internally uses a state machine to read video data from the parallel output interface of the MIPI CSI-2 controller.

As described in the previous section ([Section 2.2.2](#)), the state machine loads video data into two sockets to prevent data loss when switching between DMA Descriptors (and in effect, DMA Buffers).

This is done by using a counter to keep track of the amount of data read into the socket and then switching to the other socket when the counter hits the limit. The counter's limit is set to the DMA Buffer size.

Figure 6. Data Transfer Into Two Sockets



The counter increments by one every clock cycle. Therefore, depending on the data bus width of the interface, the value of the counter limit would change. For this example, if the data bus width is 16 bits and the DMA Buffer size is 16 KB (that is, 16,384), two bytes are read every cycle and so the programmed limit should be  $(16384/2) - 1 = 8191$ .

In general, the DMA Buffer count limit is:

$$count = \left( \frac{producer\_buffer\_size(L)}{data\_bus\_width \text{ (in bytes)}} \right) - 1$$

#### A Note on Bus Width:

The GPIF II data bus width must be selected depending on the data format of the image sensor. In this application note, the sensor is configured to output 16-bit YUV422 data and therefore, the data bus is configured to 16 bits. Refer to the CyU3PMipicsiDataFormat\_t section of the [FX3 SDK API Guide](#) for more details on bus widths for each image format.

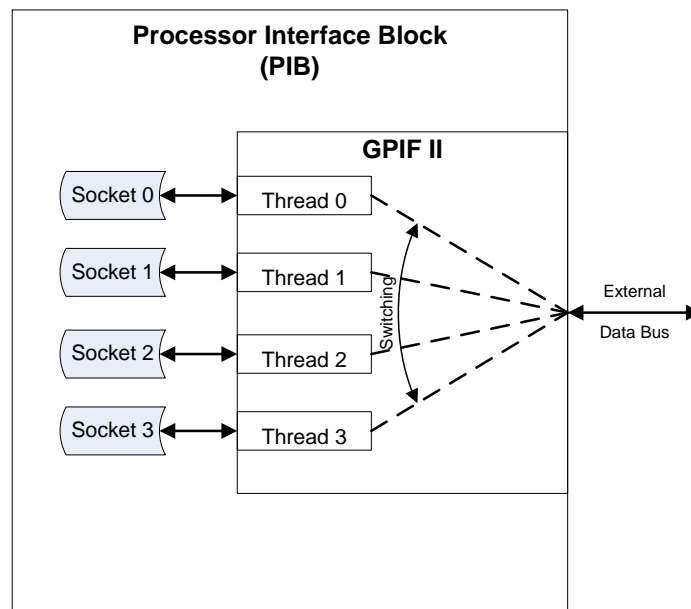
After a frame is transmitted, the state machine interrupts the CPU to indicate successful frame transfer completion, allowing it to add headers and perform other frame completion tasks.

The sections that follow explain the details of the DMA channel to stream data and the firmware that supports UVC.

### 3 Setting Up the DMA System

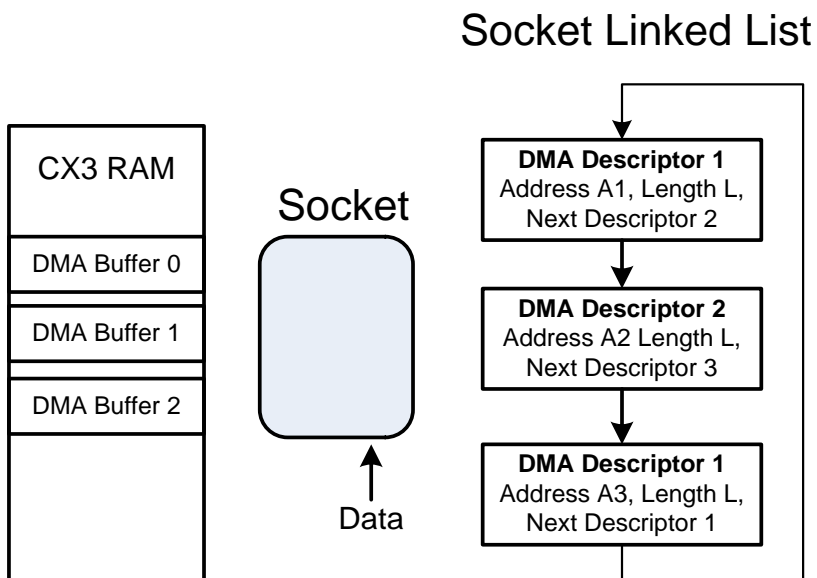
The GPIF II block can run up to 100 MHz with up to 24 bits of data (300 MBps). To transfer the data into internal DMA Buffers, GPIF II uses two GPIF threads connected to DMA producer sockets (explained in [Section 2.2.2](#)). Default mapping ([Figure 7](#)) of the sockets and GPIF threads is used for this application—Socket 0 is connected to GPIF thread 0, and Socket 1 is connected to GPIF thread 1. Note that only two of the available four threads are used in this case. The GPIF thread switching is accomplished in the GPIF II state machine described in the previous section.

Figure 7. Default GPIF II Socket/Thread Mapping



To understand DMA transfers, the concept of a socket, first introduced in [Section 2.2.1](#), is further explored in the following four figures. [Figure 8](#) shows the two main socket attributes, a linked list, and a data router.

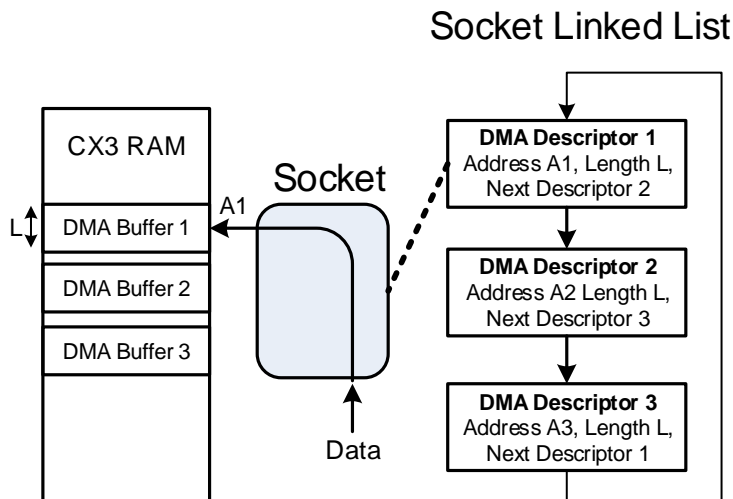
Figure 8. A Socket Routes Data According to a List of DMA Descriptors



The socket linked list is a set of data structures in main memory called DMA Descriptors. Each Descriptor specifies a DMA Buffer address ( $A_n$ ) and length ( $L$ ) as well as a pointer to the next DMA Descriptor. As the socket operates, it retrieves the DMA Descriptors one at a time, routing the data to the DMA Buffer specified by the Descriptor address and length. When  $L$  bytes have transferred, the socket retrieves the next Descriptor and continues transferring bytes to a different DMA Buffer.

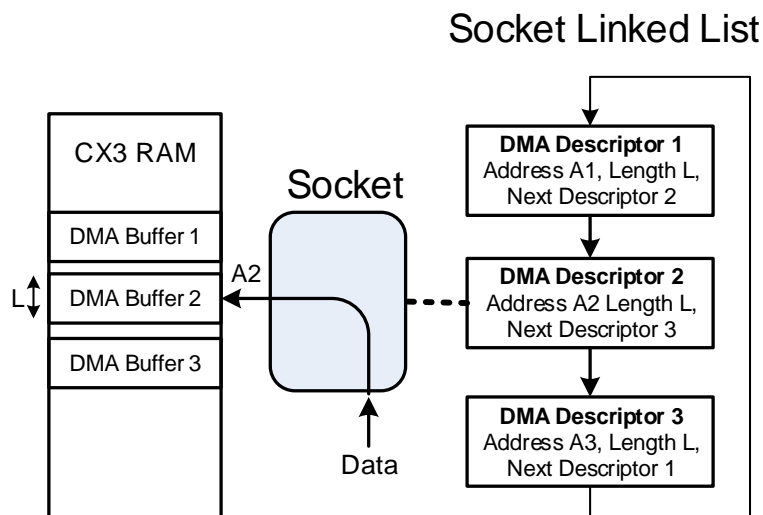
This structure makes a socket extremely versatile because any number of DMA Buffers can be created anywhere in memory and they can be automatically chained together. For example, the socket in [Figure 9](#) retrieves DMA Descriptors in a repeating loop.

Figure 9. A Socket Operating With DMA Descriptor 1



In Figure 9, the socket has loaded DMA Descriptor 1, which tells it to transfer bytes to RAM starting at A1 until it has transferred  $L$  bytes, at which time it retrieves DMA Descriptor 2 and continues with its address and length settings A2 and  $L$ , respectively (Figure 10).

Figure 10. A Socket Operating With DMA Descriptor 2



In Figure 11, the socket retrieves the third DMA Descriptor and transfers data starting at A3. When it has transferred  $L$  bytes, the sequence repeats with DMA Descriptor 1.

Figure 11. A Socket Operating With DMA Descriptor 3

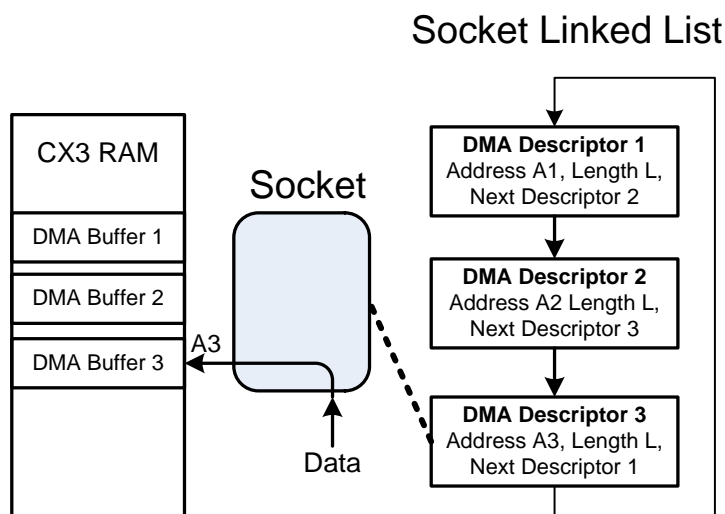
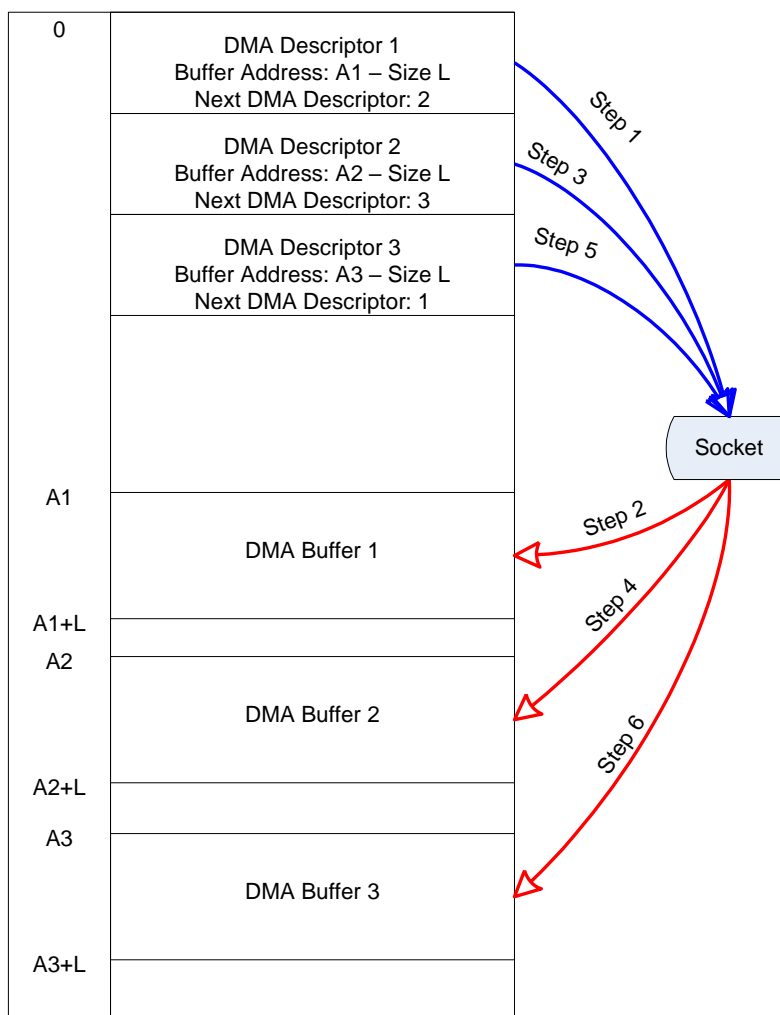


Figure 12 shows a DMA data transfer in more detail. This example uses three DMA Buffers of length  $L$  chained in a circular loop. CX3 memory addresses are on the left. The blue arrows show the socket loading the socket linked list Descriptors from memory. The red arrows show the resulting data paths. The following steps describe the socket sequence as data is moved to the internal DMA Buffers.

**Step 1:** Load DMA Descriptor 1 from memory into the socket. Get the DMA Buffer location (A1), DMA Buffer size ( $L$ ), and the next Descriptor (DMA Descriptor 2) information. Go to step 2.

**Step 2:** Transfer data to the DMA Buffer location starting at A1. After transferring DMA Buffer size  $L$  amount of data, go to step 3.

Figure 12. DMA Transfer Example



**Step 3:** Load DMA Descriptor 2 as pointed to by the current DMA Descriptor 1. Get the DMA Buffer location (A2), DMA Buffer size (L), and the next Descriptor (DMA Descriptor 3) information. Go to step 4.

**Step 4:** Transfer data to the DMA Buffer location starting at A2. After transferring DMA Buffer size L amount of data, go to step 5.

**Step 5:** Load DMA Descriptor 3 as pointed to by the current DMA Descriptor 2. Get the DMA Buffer location (A3), DMA Buffer size (L), and the next Descriptor (DMA Descriptor 1) information. Go to step 6.

**Step 6:** Transfer data to the DMA Buffer location starting at A3. After transferring DMA Buffer size L amount of data, go to step 1.

This simple scheme has an issue in the camera application. A socket takes time to retrieve the next DMA Descriptor from memory; typically 1  $\mu$ s. If this transfer pause occurs in the middle of a data transfer, video data is lost. To prevent this loss, the DMA Buffer size can be set as a multiple of the video line length. This makes the DMA Buffer switching pause coincide with the time that the video line is inactive (i.e., vertical blanking interval of the sensor). However, this approach lacks flexibility, if, for example, the video resolution is changed.

Setting the DMA Buffer size exactly equal to the line size is also not a good solution because it does not take advantage of the USB 3.1 Gen 1 maximum burst rate for BULK transfers. USB 3.1 Gen 1 allows a maximum of 16 bursts of 1024 bytes over BULK Endpoints. This is why the DMA Buffer size is set to 16 KB.

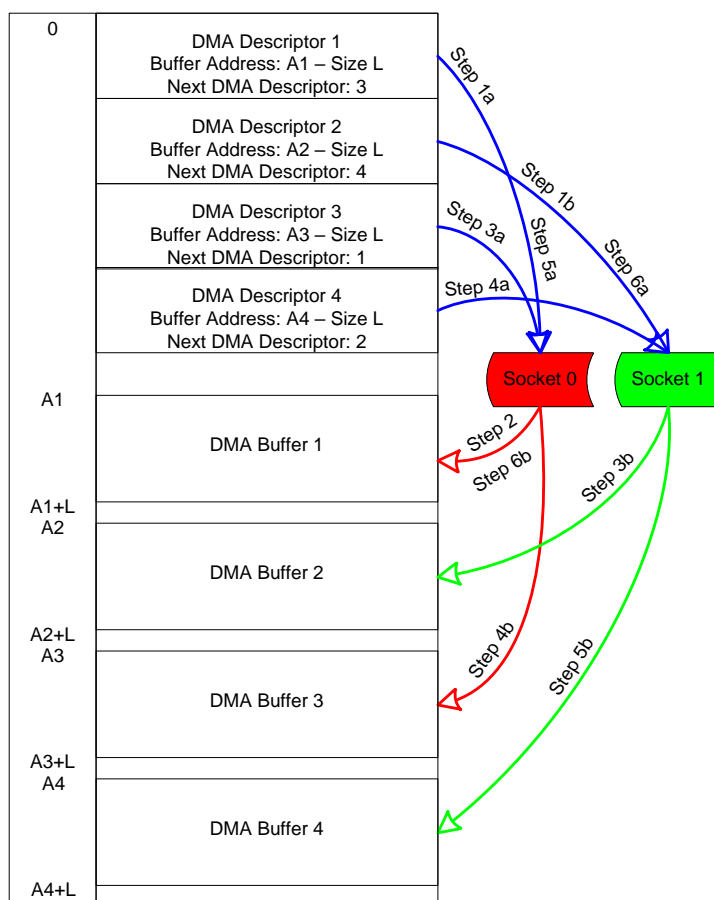
A better solution is to take advantage of the fact that sockets can be switched without latency—in one clock cycle. Therefore, it makes sense to use *two* sockets to store data into four interleaved DMA Buffers.

**Note** At least two buffers are required per socket to prevent data loss i.e., the socket can be writing to one buffer while the host is reading from the other.

**Note** The number of buffers per socket shown in Figure 5 (four per socket) and later figures (two per socket) differs; this is to simplify the illustration. However, the actual firmware sets up four buffers per socket.

Data transfer using dual sockets is described in Figure 13, again with numbered execution steps. Socket 0 and Socket 1 access to DMA Buffers is differentiated by red and green arrows (data paths for individual sockets), respectively. The “a” and “b” parts of each step occur simultaneously. This parallel operation of the hardware eliminates the DMA Descriptor retrieval dead time and allows the GPIF II to stream data continuously into internal memory. These steps correspond to the “Step” line in Figure 12.

Figure 13. Dual Sockets Yield Seamless Transfers



**Step 1:** At initialization of the sockets, Socket 0 and Socket 1 load the DMA Descriptor 1 and DMA Descriptor 2, respectively.

**Step 2:** As soon as the data is available, Socket 0 transfers the data to DMA Buffer 1. The transfer length is  $L$ . At the end of this transfer, go to step 3.

**Step 3:** The fixed function state machine in GPIF II switches the GPIF thread and, therefore, the socket for data transfer. Socket 1 starts to transfer data to DMA Buffer 2, and, at the same time, Socket 0 loads the DMA Descriptor 3. By the time Socket 1 finishes transferring  $L$  amount of data, Socket 0 is ready to transfer data into DMA Buffer 3.

**Step 4:** GPIF II now switches back to the original GPIF thread. Socket 0 now transfers the data of length  $L$  into DMA Buffer 3. At the same time, Socket 1 loads the DMA Descriptor 4, making it ready to transfer data to DMA Buffer 4. After Socket 0 finishes transferring the data of length  $L$ , go to step 5.

**Step 5:** GPIF II routes Socket 1 data into DMA Buffer 4. At the same time, Socket 0 loads DMA Descriptor 1 to prepare to transfer data into DMA Buffer 1. Notice that Step 5a is the same as Step 1a except that Socket 1 is not initializing but, rather, transferring data simultaneously.

**Step 6:** GPIF II switches sockets again, and Socket 0 starts to transfer data of length L into DMA Buffer 1. It is assumed that by now, the DMA Buffer is empty, having been depleted by the UIB consumer socket. At the same time, Socket 1 loads the DMA Descriptor 2 and is ready to transfer data into DMA Buffer 2. The cycle now goes to Step 3 in the execution path.

GPIF II Sockets can transfer video data only if the consuming side (USB) empties and releases the DMA Buffers in time to receive the next chunk of video data from GPIF II. If the consumer is not fast enough, the sockets drop data because their DMA Buffer writes are ignored. As a result, the byte counters lose sync with the actual transfers, which can propagate to the next frame. Therefore, a cleanup mechanism is required if the frame is not transferred for a long time. This mechanism is described in [Section 5.1](#).

A frame transfer can end in one of four possible scenarios:

- Socket 0 has transferred a full DMA Buffer
- Socket 1 has transferred a full DMA Buffer
- Socket 0 has transferred a partial DMA Buffer
- Socket 1 has transferred a partial DMA Buffer

In the last two cases, the CPU needs to commit the partial DMA Buffer to the USB consumer.

This application note uses the project present in `SDK_INSTALL_PATH\firmware\cx3_examples\cycx3_uvc_ov5640` folder where `SDK_INSTALL_PATH` is the location of the [FX3 SDK](#) Installation. The default installation path for 32-bit systems is `C:\Program Files\Cypress\EZ-USB FX3 SDK\1.3`. For 64-bit systems, it is `C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3`.

The DMA channel is initialized using a function in the `cycx3_uvc.c` file called “CyCx3AppInit”. The DMA channel configuration details are customized in the “dmaCfg” structure in the same function. The DMA channel type is set to `MANUAL_MANY_TO_ONE`.

In addition, the USB Endpoint that streams data to the USB 3.1 Gen 1 Host is configured to enable a burst of 16 over the 1024-byte BULK Endpoint. This is set using the “endPointConfig” structure passed in the “CyU3PSetEpConfig” function, with the Endpoint constant set to “CX3\_EP\_BULK\_VIDEO”.

### 3.1 DMA Buffers

This section summarizes how CX3 DMA Buffers are created and used in this application. The integral parts of a DMA channel are described in [Section 2.2.1](#).

In this application, the GPIF II unit is the producer, and the USB unit is the consumer. This application uses the GPIF thread switching feature in the GPIF II block to avoid data drops.

When a producer socket loads a DMA Descriptor, it checks the associated DMA Buffer to see if it is ready for a write operation. The producer socket changes its state to *active* for writing data into CX3 RAM if it finds that the DMA Buffer is empty. The producer socket locks the DMA Buffer for write operations.

When a producer socket is finished writing to a DMA Buffer, it releases the lock so that the consumer socket can access the DMA Buffer. The action is called “Buffer wrap-up” or simply “wrap-up.” The DMA unit is then said to commit the DMA Buffer to the CX3 RAM. The producer socket is said to have produced a DMA Buffer. A DMA Buffer should be wrapped up only while the producer is not actively filling it.

If a DMA Buffer fills completely, as it does repeatedly during a frame, the wrap-up operation is automatic. The producer socket releases the lock on the DMA Buffer, commits it to the CX3 RAM, switches to an empty DMA Buffer, and continues to write the video data stream.

When a consumer socket loads a DMA Descriptor, it checks the associated DMA Buffer to see if it is ready for a read operation. The consumer socket changes its state to *active* for reading the data from CX3 RAM if it finds that the DMA Buffer is committed. The consumer socket locks the DMA Buffer for read operations.

- After the consumer socket has read all the data from the DMA Buffer, it releases the lock so that the producer socket can access the DMA Buffer. The consumer socket is said to have consumed the DMA Buffer.

- If the same DMA Descriptors are used by the producer and consumer sockets, the DMA Buffer full/empty status is communicated automatically between the producer and consumer sockets via the DMA Descriptors and intersocket events.
- In this application, because the CPU needs to add a 12-byte UVC header, the producer socket and the consumer socket need to load different sets of DMA Descriptors. The DMA Descriptors loaded by the producer socket will point to DMA Buffers that are at a 12-byte offset from the corresponding DMA Buffers that the DMA Descriptors loaded by the consumer socket point to.
- Due to different DMA Descriptors for producer and consumer sockets, the CPU must manage the communication of the DMA Buffer status between the producer and the consumer sockets. This is why the DMA channel implementation is called a “Manual DMA” channel.
- After a DMA Buffer is produced by the GPIF II block, the CPU is notified via an interrupt. The CPU then adds the header information and commits the DMA Buffer to the consumer (USB Interface Block).
- On the GPIF II side, the video data in DMA Buffers are automatically wrapped up and committed to the CX3 RAM for all but the last DMA Buffer in the frame.
- At the end of a frame, the final DMA Buffer is likely not filled completely. In this case, the CPU intervenes and manually wraps up the DMA Buffer and commits it to the CX3 RAM. This is called a “forced wrap-up.”

## 4 USB Video Class (UVC)

This section introduces you to the USB Video Class, which is a protocol on top of the Universal Serial Bus Specification to transport video data to a PC.

Conforming to the UVC class requires two CX3 code modules:

- Enumeration Data
- Operational Code

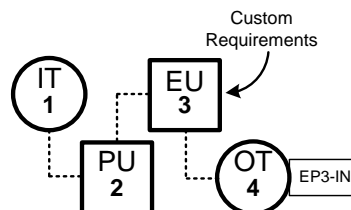
### 4.1 Enumeration Data

The `cycx3_uvc_ov5640` project of the SDK installation includes a file named `cycx3_uvcdescr.c` (explained in [Section 4.3.1](#)) that contains the UVC enumeration data. The USB specification, which defines the format for UVC Descriptors, is available at [usb.org](http://usb.org). This section gives a high-level view of the Descriptors. A UVC device has four logical elements and each element is defined by a descriptor:

- Input Camera Terminal (IT)
- Output Terminal (OT)
- Processing Unit (PU)
- Extension Unit (EU)

The elements connect in the Descriptors, as shown in [Figure 14](#). Connections are made between elements by associating terminal numbers in the Descriptors. For example, the Input (Camera) Terminal Descriptor declares its ID to be 1, and the Processing Unit Descriptor specifies its input connection to have the ID of 1, logically connecting it to the Input Terminal. The Output Terminal Descriptor specifies which USB Endpoint to use—in this case, BULK-IN Endpoint 3.

Figure 14. UVC Diagram of the Camera Architecture



The Descriptors also include video properties such as width, height, frame rate, frame size, and bit depth; control properties such as brightness, exposure, gain, contrast, and PTZ (pan, tilt, and zoom), among others.

## 4.2 Operational Code

After the Host enumerates the camera, the UVC driver sends a series of requests to the camera to determine operational characteristics. This is called the “capability request phase.” It precedes the streaming phase, in which the Host application starts streaming video.

For example, suppose a UVC device indicates that it supports brightness control in one of its USB Descriptors. During the capability request phase, the UVC driver queries the device to discover the relevant brightness parameters.

When a Host application makes a request to change the brightness value, the UVC driver issues a SET control request to change the brightness value (SET\_CUR). This is done over the USB control Endpoint (EP0).

Similarly, when the Host application chooses to stream a supported video format, frame rate, or frame size, it issues streaming requests. There are two types: PROBE and COMMIT. PROBE requests are used to determine if the UVC device is ready to accept changes to the streaming mode while COMMIT requests are used to make the changes. A streaming mode is a combination of image format, frame size, and frame rate.

## 4.3 USB Video Class Requirements

This section explains how the UVC requirements are satisfied by the example project. UVC requires a device to do the following:

- Enumerate with the UVC-specific USB Descriptors
- Handle SET/GET UVC-specific requests for the UVC control and stream capabilities reported in the USB Descriptors
- Stream video data in a UVC-conformant color format
- Add a UVC conformant header for every image payload

Details of these requirements are found in the [UVC specification](#).

### 4.3.1 USB Descriptors for UVC

The `cycx3_uvcdescr.c` file contains the USB Descriptor tables. The byte arrays “CyCx3USBHSCfgDscr” (Hi-Speed) and “CyCx3USBSSCfgDscr” (SuperSpeed) contain the UVC-specific Descriptors. These Descriptors implement the following tree of sub-Descriptors:

- Configuration Descriptor
  - Interface Association Descriptor
  - Video Control (VC) Interface Descriptor
    - VC Interface Header Descriptor
      - Input (Camera) Terminal Descriptor
      - Processing Unit Descriptor
      - Extension Unit Descriptor
      - Output Terminal Descriptor
    - VC Status Interrupt Endpoint Descriptor
  - Video Streaming (VS) Interface Descriptor
    - VS Interface Input Header Descriptor
      - VS Format Descriptor
    - VS Frame Descriptor
  - BULK-IN Video Endpoint Descriptor

The Configuration Descriptor is a standard USB Descriptor that defines the functionality of the USB device in its sub-Descriptors. The Interface Association Descriptor is used to indicate to the Host that the device conforms to a standard USB class. Here, this Descriptor reports a UVC-conformant device with two interfaces: Video Control (VC) interface and Video Streaming (VS) interface. Having two separate interfaces makes the UVC device a USB composite device.

## Video Control (VC) Interface

The VC Interface Descriptor and its sub-Descriptors report all of the control interface-related capabilities. Examples include brightness, contrast, hue, exposure, and PTZ controls.

The VC Interface Header Descriptor is a UVC-specific interface Descriptor that points to the VS interfaces to which this VC Interface belongs.

The Input (Camera) Terminal Descriptor, the Processing Unit Descriptor, the Extension Unit Descriptor, and the Output Terminal Descriptor contain bit fields that describe features supported by the respective terminal or unit.

The Camera Terminal controls mechanical (or equivalent digital) features, such as exposure and the PTZ of the device that transmits the video stream.

The Processing Unit controls image attributes, such as brightness, contrast, and hue of the video being streamed through it.

The Extension Unit allows vendor-specific features to be added, much like standard USB Vendor Requests. In this design, the Extension Unit is empty, but the Descriptor is included as a placeholder for custom features. Note that if the Extension Unit is utilized, the standard Host application will not see its features unless the Host application is modified to recognize them.

The Output Terminal is used to describe an interface between these units (IT, PU, EU) and the Host. The VC Status Interrupt Endpoint Descriptor is a standard USB Descriptor for an Interrupt Endpoint. This Endpoint can be used to communicate UVC-specific status information. The functionality of this Endpoint is outside the scope of this application note.

The UVC specification divides these functionalities so that you can easily structure the implementation of the class-specific control requests. However, the implementation of these functionalities is application-specific. The supported control capabilities are reported in the bit field “bmControls” (*cycx3\_uvcdescr.c*) of the respective terminal or unit descriptor by setting corresponding capability bits to ‘1’. The UVC device driver polls for details about the control on enumeration. The polling for details is carried out over EP0 requests. All such requests, including the video streaming requests, are handled by the *CyCx3AppUSBSetupCB* function in the *cycx3\_uvc.c* file.

## Video Streaming (VS) Interface

The VS Interface Descriptor and its sub-descriptors report the various frame formats (e.g., uncompressed, MPEG, H.264, and so on), frame resolutions (width, height, and bit depth), and frame rates. Based on the values reported, the Host application can choose to switch streaming modes by selecting supported combinations of frame formats, frame resolutions, and frame rates.

The VS Interface Input Header Descriptor specifies the number of VS Format Descriptors that follow.

The VS Format Descriptor contains the images’ aspect ratio and the color format, such as uncompressed or compressed.

The VS Frame Descriptor contains image resolution and all supported frame rates for that resolution. If the camera supports different resolutions, multiple VS Frame Descriptors follow the VS Format Descriptor.

The BULK-IN Video Endpoint Descriptor is a standard USB Endpoint Descriptor that contains information about the bulk Endpoint used for streaming video.

This example uses two frame descriptors to support two sets of resolutions and frame rates. Its image characteristics are contained in three Descriptors, as shown in the following three tables (only relevant byte offsets are shown).

Table 1. VS Format Descriptor Values

VS Format Descriptor Byte Offset	Characteristic	SuperSpeed Value	Hi-Speed Value
23-24	Width-to-Height ratio	16:9	4:3

Table 2. First VS Frame Descriptor Values

VS Frame Descriptor Byte Offset	Characteristic	SuperSpeed Value	Hi-Speed Value
5-8	Resolution (W,H)	0x780, 0x438 (1920 × 1080)	0x140, 0xF0 (320 × 240)

17-20	Maximum image size in bytes	0x3F4800 (1920 × 1080 × 2)	0x25800 (320 × 240 × 2)
21-24, also 26-29	Frame Interval in 100-ns units	0x51615 (30 fps)	0x1B207 (90 fps)

Table 3. Second VS Frame Descriptor Values

VS Frame Descriptor Byte Offset	Characteristic	SuperSpeed Value	Hi-Speed Value
5-8	Resolution (W,H)	0x500, 0x2D0 (1280 × 720 )	0x280 ,0x1E0 (640 × 480)
17-20	Maximum image size in bytes	0x1C2000 (1280 × 720 × 2)	0x96000 (640 × 480× 2)
21-24, also 26-29	Frame Interval in 100-ns units	0x28B0A (60 fps)	0x28B0A (60 fps)

Note that multiple-byte values are stored and sent LSB first (i.e., little-endian). So, for example, the first SuperSpeed frame rate is 30fps and the frame interval is:

$$FrameInterval = \frac{1}{30fps \times 100ns} = 333333 = 0x51615$$

This is stored as 0x15, 0x16, 0x05, and 0x00 in the descriptor. This design can be adapted to support different image resolutions by modifying the entries in these three tables.

#### 4.3.2 UVC-Specific Requests

The UVC specification uses USB control Endpoint EP0 to communicate control and streaming requests to the UVC device. These requests are used to discover and change the attributes of the video-related controls. The UVC specification defines these video-related controls as capabilities. These capabilities allow you to change image properties or to stream video.

A capability can be a video control property, such as brightness, contrast, and hue, or video stream mode properties such as the color format, frame size, and frame rate. Capabilities are reported via the UVC-specific section of the USB Configuration Descriptor. Each of the capabilities has attributes. The attributes of a capability are as follows:

- Minimum value
- Maximum value
- Number of values between the minimum and the maximum
- Default value
- Current value

SET and GET are the two types of UVC-specific requests. SET is used to change the current value of an attribute, while GET is used to read an attribute.

Here is a list of UVC-specific requests:

- SET\_CUR is the only type of SET request.
- GET\_CUR reads the current value.
- GET\_MIN reads the minimum supported value.
- GET\_MAX reads the maximum supported value.
- GET\_RES reads the resolution (step value to indicate the supported values between min and max).
- GET\_DEF reads the default value.
- GET\_LEN reads the size of the attribute in bytes.
- GET\_INFO queries the status or support for a specific capability.

The UVC specification defines these requests as either mandatory or optional for a given capability. For example, if the SET\_CUR request is optional for a particular capability, its presence is determined through the GET\_INFO request. If the camera does not support a certain request for a capability, it must indicate this by stalling the control Endpoint when the request is issued from the Host to the camera.

There are byte fields in these requests that qualify their target capability. These byte fields have a hierarchy, which follows the same structure as the UVC-specific Descriptors described in [Section 4.3.1](#). The first level identifies the interface (video control or video streaming).

If the first level identifies the interface as video control, the second level identifies the terminal or unit, and the third level identifies the capability of that terminal or unit. For example, if the target capability is the brightness control, then:

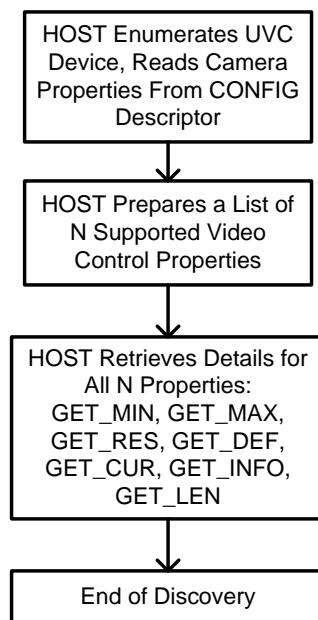
- First level = video control
- Second level = processing unit
- Third level = brightness control

If the first level identifies the interface as video streaming, the second level would be PROBE or COMMIT. There is no third level.

When the Host wants the UVC device to start streaming or to change the streaming mode, the Host first determines if the device supports the new streaming mode. To determine this, the Host sends a series of SET and GET requests with the second level set to PROBE. The device either accepts or rejects the change to the streaming mode. If the device accepts the change request, the Host confirms it by sending the SET\_CUR request with the second level set to COMMIT.

The following three flowcharts show how the Host interacts with a UVC device.

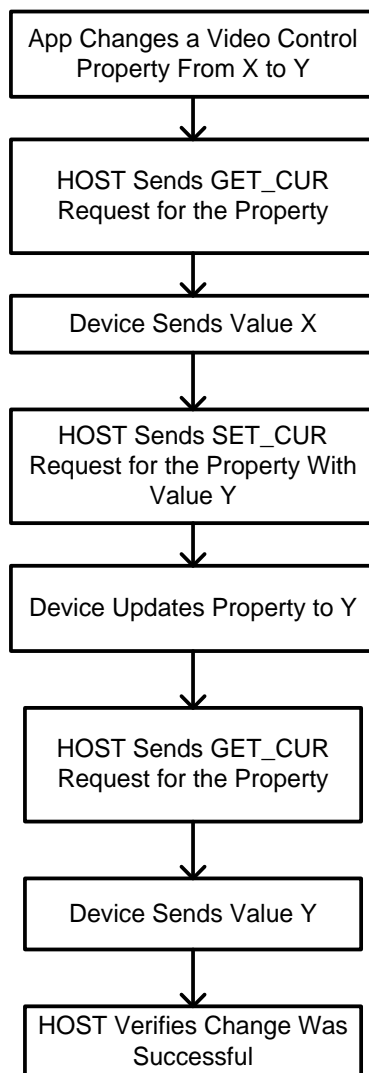
Figure 15. UVC Enumeration and Discovery Flow



When the UVC device is plugged into USB, the Host enumerates it and discovers details about the properties supported by the camera ([Figure 15](#)).

During a video operation, a camera operator may change a camera property, such as brightness, in a display dialog presented by the Host application. [Figure 16](#) shows this interaction.

Figure 16. Host Application Changes a Camera Setting



Before starting to stream, the Host application issues a set of probe requests to discover the possible streaming modes. After the default streaming mode is decided, the Host issues a COMMIT request with the Frame and Format IDs of the desired resolution and frame rate. This process is shown in [Figure 17](#). The structure sent by the host along with the COMMIT request is shown in [Table 4](#).

At this point, the UVC driver is ready to stream video from the UVC device.

Figure 17. Host-Camera Prestreaming Dialogue

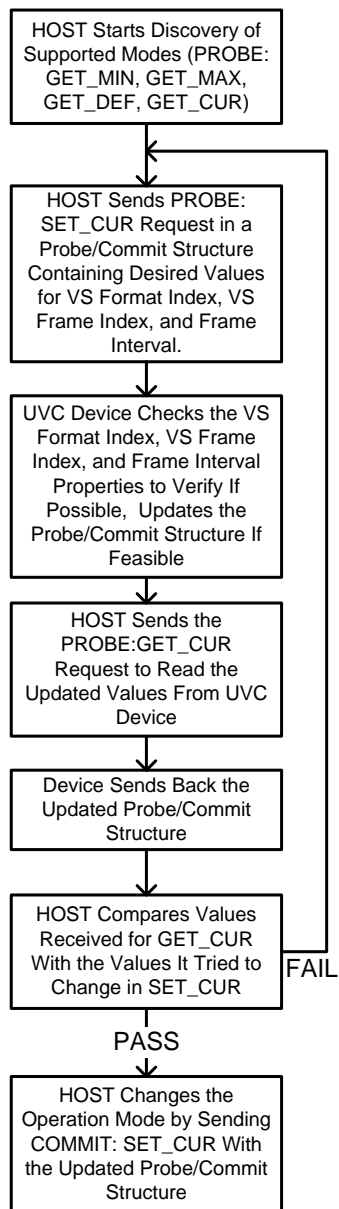


Table 4. Commit Structure Values Sent by the Host Before a 1080p @ 30fps Stream Is Started

Probe/Commit Structure Byte Offset	Characteristic	Value
2	Format Index	1
3	Frame Index	1
4-7	Frame Interval in 100-ns units	0x51615 (30 fps)
18-21	Maximum image size in bytes	0x3F4800 (1920 × 1080 × 2)

### Control Requests: Brightness, PTZ, Hue, Saturation, and Others

The image sensor may support controlling image features such as brightness, hue, and saturation, and external hardware might be added to add PTZ (pan, tilt, and zoom) support.

To support this in the CX3 firmware, the Video Control Descriptor must be modified to indicate support of the specific control. For example, to support Brightness control, bit 0 of the `bmControls` field in the Processing Unit Descriptor should be set. For more details refer to the UVC spec. referred in [Section 4.1](#)

The Host can now send requests directed towards the specific terminal. The different types of requests that the Host can send are explained in [Section 4.3.2](#). These requests are to be handled in the `CyCx3AppUSBSetupCB` function.

Upon receipt of a request, the firmware can send requests to the image sensor or its associated hardware. The implementation is application-specific.

### Streaming Requests: Probe and Commit Control

The `CyCx3AppUSBSetupCB` function handles streaming-related requests. When the UVC driver needs to stream video from a UVC device, the first step is negotiation. In that phase, the driver sends PROBE requests such as `GET_MIN`, `GET_MAX`, `GET_RES`, and `GET_DEF`. In response, the CX3 firmware returns a PROBE structure. The structure contains the USB Descriptor indices of video format and video frame, frame rate, maximum frame size, and payload size (the number of bytes that the UVC driver can fetch in one transfer).

In the example, the `GET_CUR` request is handled, which then checks for the current active connection and sends the appropriate probe structure.

Following the `GET_CUR` handling, `SET_CUR` requests are handled. These requests are sent at the start of the streaming phase.

The `SET_CUR` request for COMMIT control indicates that the Host will start streaming video after the request completes. Depending on the frame descriptor index that the Host sends (see [Table 4](#) for details of the data structure sent by the Host), the image sensor is configured to send video data with the requested resolution and frame rate and then the appropriate MIPI CSI-2 interface parameters are set. This then starts the video stream.

### Video Data Format: YUY2

The UVC specification supports only a subset of color formats for video data. Therefore, you should choose an image sensor that streams images in a color format that conforms to the UVC specification. This application note covers an uncompressed color format called YUY2, which is supported by most, but not all, image sensors. The YUY2 color format is a 4:2:2 down-sampled version of the YUV color format. Luminance values Y are sampled for every pixel, but chrominance values U and V are sampled only for even pixels. This creates “macro pixels”, each of which describes two image pixels using a total of four bytes. Notice that every other byte is a Y value, and the U and V values represent only even pixels:

Y0, U0, Y1, V0 (first two pixels)

Y2, U2, Y3, V2 (next two pixels)

Y4, U4, Y5, V4 (next two pixels)

Refer to [Wikipedia](#) for additional information on color formats.

**Note:** The RGB format is not supported in the UVC spec. However, for Windows, Microsoft provides an extension to the UVC spec to support a variety of other formats including RGB888 and RGB565. They are specified by including the appropriate GUID in the Video Streaming format descriptor. Refer to [Microsoft web page on Media Types](#) for more details.

**Note:** Although a monochrome image is not supported as a part of the UVC specification, an 8-bit monochrome image can be represented in the YUY2 format by using the 8-bit RAW data as the Y value and setting all the U and V values to 0x80. This should be handled by the Image sensor / ISP.

### UVC Video Data Header

The UVC class requires a 12-byte header for uncompressed video payloads. The header describes the properties of the image data being transferred. For example, it contains a “new frame” bit that the image sensor controller (CX3) toggles every frame. The CX3 code also can set an error bit in the header to indicate a problem in streaming the current frame. This UVC data header is required for every USB transfer. Refer to the UVC specification for additional details.

Table 5 shows the format of the UVC video data header.

Table 5. UVC Video Data Header Format

Byte Offset	Field Name	Description
0	HLF	Header Length Field specifies the length of the header in bytes
1	BFH	Bit Field Header indicates type of the image data, status of the video stream and presence or absence of other fields
2-5	PTS	Presentation Time Stamp indicates the source clock time in native device clock units
6-11	SCR	Source Clock Reference indicates system time clock and USB Start-Of-Frame (SOF) token counter

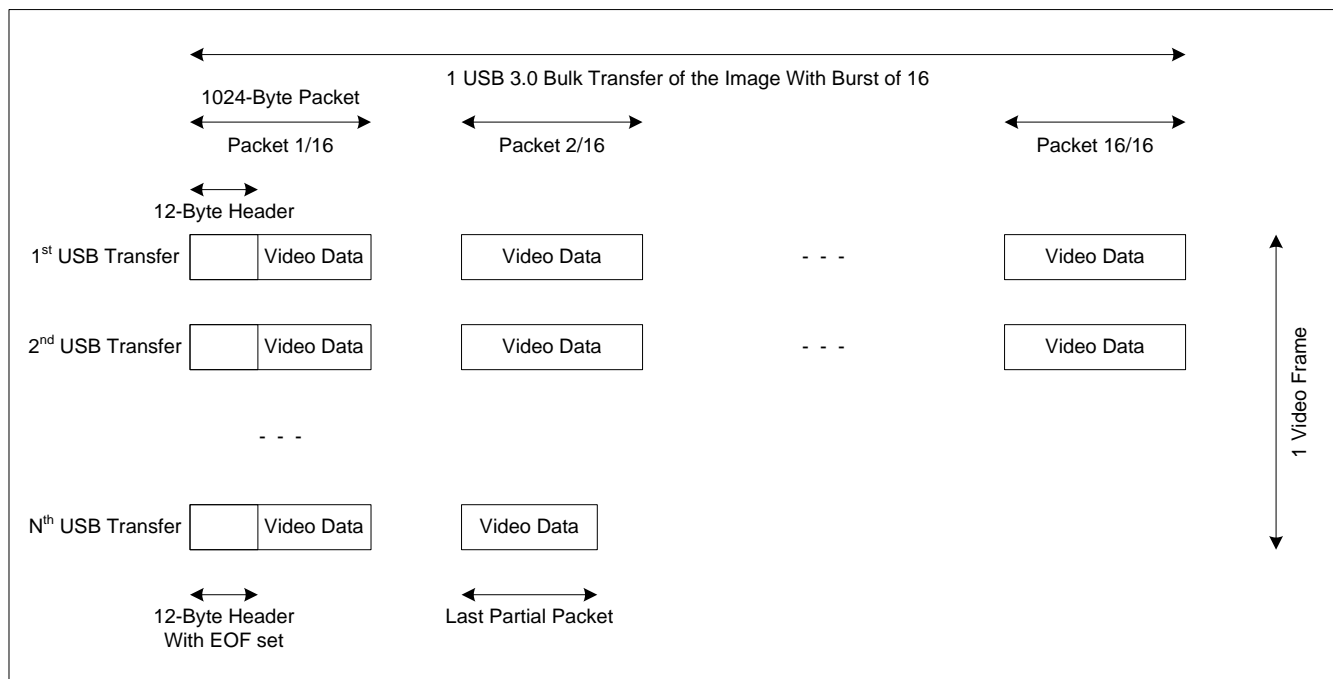
The value for the HLF is always 12. The PTS and the SCR fields are optional. The firmware example populates zeros in these fields. The Bit Field Header (BFH) keeps changing value at the end of a frame. Table 6 shows the format of the BFH that is a part of the UVC video data header.

Table 6. Bit Field Header (BFH) Format

Bit Offset	Field Name	Description
0	FID	Frame Identifier bit toggles at each image frame start boundary and stays constant for the rest of the image frame
1	EOF	End of Frame bit indicates the end of a video and is set only in the last USB transfer belonging to an image frame
2	PTS	Presentation Time Stamp bit indicates the presence of a PTS field in the UVC video data header (1=present)
3	SCR	Source Clock Reference bit indicates the presence of an SCR field in the UVC video data header (1=present)
4	RES	Reserved, set to 0
5	STI	Still Image bit indicates if the video sample belongs to a still image
6	ERR	Error bit indicates an error in the device streaming
7	EOH	End of Header bit, when set, indicates the end of the BFH fields

Figure 18 shows how these headers are added to the video data in this application. The 12-byte header is added for every USB bulk transfer. Here, each USB transfer has a total of 16 bulk packets. The USB 3.1 Gen 1 bulk packet size is 1024 bytes.

Figure 18. UVC Video Data Transfers



## 5 CX3 Firmware

This application note uses the `cycx3_uvc_ov5640` example project that is available with the installation of the FX3/CX3 SDK. The firmware first initializes the CX3 CPU and configures its I/O. Then, it makes a function call (`CyU3PKernelEntry`) to start the ThreadX RTOS. The RTOS initializes itself, creates a few internal threads, and then calls `CyFxApplicationDefine` to let the user create application-specific threads.

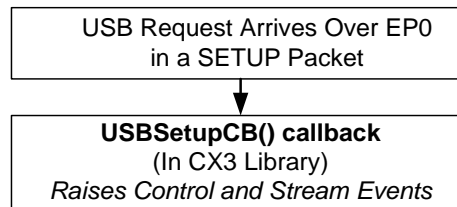
This example creates two application threads: `uvcAppThread` and `uvcMipiErrorThread`. The RTOS allocates resources to run these application threads and schedule the execution of the application thread functions, `CyCx3UvcAppThread_Entry` and `CyCx3UvcMipiErrorThread`, respectively. [Figure 19](#) shows the basic program structure.

The example firmware described in this application note implements only basic image streaming functionality with the OV5640 image sensor. If you're using another sensor, or if additional functionality in OV5640 is required, the `cycx3_uvc.c` file must be modified accordingly. [Table 7](#) summarizes the code modules and the functions implemented in each module.

Table 7. Example Project Files

File	Description
<i>cyfctx.c</i>	No changes needed. Use this file as provided with the project. It contains the variables that the RTOS and the CX3 API library use for memory mapping and the functions that the CX3 API library uses for memory management.
<i>cycx3_uvc.c</i>	<p>Main source file for the UVC application. Changes are needed when modifying the code to support controls such as brightness, PTZ, etc., and when modifying to add support for different video streaming modes.</p> <p>Contains the following functions:</p> <ul style="list-style-type: none"> <li><i>main</i>: Initializes the CX3 device, sets up caches, configures the CX3 I/Os, and starts the RTOS kernel.</li> <li><i>CyFxApplicationDefine</i>: Defines the two application threads that are executed by the RTOS.</li> <li><i>CyCx3AppThread_Entry</i>: This function is executed by the first application thread. It calls the initialization functions for CX3's internal blocks, enumerates the device, and then handles the device suspend and frame completion tasks.</li> <li><i>CyCx3AppMipiErrorThread</i>: This function is executed by the second application thread. It waits for events that indicate errors in the MIPI CSI-2 controller and then reads them using <i>CyU3PMipicsiGetErrors</i>.</li> <li><i>CyCx3AppDebugInit</i>: Initializes CX3's UART block for printing debug messages</li> <li><i>CyCx3AppInit</i>: Initializes CX3's GPIO block, Processor Block or PIB (GPIF II is a part of PIB), I2C/CC1 interface, MIPI CSI-2 Rx interface and the sensor (sets configuration to 1080p 30fps in SuperSpeed mode). It also initializes the USB block for enumeration, and endpoint configuration memory for USB transfers and creates DMA channel configuration for data transfers from two GPIF II sockets to the USB sockets.</li> <li><i>CyCx3AppGpifCB</i>: Handles CPU interrupts generated from the GPIF II state machine</li> <li><i>CyCx3AppDmaCallback</i>: Keeps track of the outgoing video data from CX3 to the Host. It adds the header to the image data and can signal the first application thread to indicate frame completion.</li> <li><i>CyCx3AppUSBSetupCB</i>: Handles all control requests sent by the Host, sets events indicating that UVC-specific requests have been received from the Host, and detects when streaming is stopped. It also handles the UVC class-specific probe and commit control requests that are required to start and stop video streaming.</li> <li><i>CyCx3AppUSBEventCB</i>: Handles USB events such as suspend, cable disconnect, reset, and resume.</li> <li><i>CyCx3AppErrorHandler</i>: Error handler function. This is a placeholder function for you to implement error handling if necessary.</li> <li><i>CyCx3AppAddHeader</i>: Adds a UVC header to the video data during active streaming.</li> </ul>
<i>cycx3_uvcdscr.c</i>	Contains the USB enumeration Descriptors for the UVC application. This file needs to be changed if the frame rate, image resolution, bit depth, or supported video controls need to be changed. The UVC specification has all the required details.
<i>cycx3_uvc.h</i>	<p>Contains switches to modify the application behavior to turn ON/OFF the debug prints for frame counts and the MIPI error thread.</p> <p>Contains constants that are used in common by the <i>cycx3_uvc.c</i> and <i>cycx3_uvcdscr.c</i> files.</p>
<i>cyfx_gcc_startup.s</i>	<p>This assembly source file contains the CX3 CPU startup code. It has functions that set up the stacks and interrupt vectors.</p> <p>No changes needed.</p>

Figure 19. High level Camera Project Structure


**UVC.C**

**CyCx3UvcAppInUSBSetupCB()**  
*Handles Control & Stream events*  
**CyCx3GpifCB()**  
*Handles Frame Termination*  
**CyCx3UvcAppDmaCallback()**  
*Handles DMA Transfers*

## 5.1 Application Threads

Two application threads enable concurrent functionality.

The `uvcAppThread` (application thread) initializes the CX3 peripherals and handles two internal events: *USB Suspend* and *DMA Channel Reset*.

The `uvcMipiErrorThread` monitors MIPI CSI-2 errors and returns the error count.

### 5.1.1 USB Suspend Event

When CX3 is put to suspend by the Host, the USB Suspend Event (`CX3_USB_SUSP_EVENT_FLAG`) is triggered, which does the following:

1. Disables the clocks of the MIPI CSI-2 controller and puts the interface into a low-power mode
2. Puts the image sensor to sleep
3. Places the CX3 core in a low-power suspend mode with USB bus activity as the wakeup source

When some activity is detected on the USB bus, the device wakes up. After wakeup, the image sensor is powered up and the MIPI CSI-2 controller is reactivated.

### 5.1.2 DMA Channel Reset Event

If frame transfer takes too long to complete, the stale data is flushed out and the video stream is restarted. This is done using the `CX3_DMA_RESET_EVENT` flag, which is set when a 500-ms timer (used as a watchdog) fires.

This timer is reset whenever a full frame is sent to the Host and fires only when a frame is stuck in the DMA Buffers for a long time. The video stream is also restarted if committing the DMA Buffer to the Host fails (which can result in the DMA channel going out of sequence).

If the `CX3_ERROR_THREAD_ENABLE` preprocessor macro is enabled, another thread – `uvcMipiErrorThread` – is created to monitor CSI-2 errors.

This thread's entry function (`CyCx3UvcMipiErrorThread`) periodically calls `CyU3PMipicsiGetErrors` to get error counts from the MIPI CSI-2 controller. Error counters can be used for debugging if a blank screen is observed (see [Section 8](#) for more details on how to troubleshoot with MIPI CSI-2 error counters).

## 5.2 Handling UVC Requests

The CX3 firmware handles UVC-specific control requests (`SET_CUR`, `GET_CUR`, `GET_MIN`, and `GET_MAX` described in [Section 4.3.2](#)) over the Control Endpoint (EP0). Class-specific control requests are handled by the `CyCx3AppUSBSetupCB` (CB=Callback) function. Whenever one of these control requests is received by CX3, this function decodes the received setup data and handles it.

For probe control requests directed to the Video Streaming interface (see [Section 4.3.2.2](#)), it sends the corresponding probe control structure to the Host. If a commit control request is made, video streaming is started.

All requests (except the Get Status request) directed to the Video Control interface (see [Section 4.3.2.1](#)) are stalled (the USB Host receives the STALL handshake) as no Video Control feature is supported in this example. If your example supports brightness, exposure, PTZ, or any other controls, you must handle them here.

### 5.3 Initialization

The `CyCx3UvcAppThread_Entry` function calls `CyCx3AppDebugInit` to initialize the UART debugging capability and `CyCx3ApplInit` to initialize the rest of the required blocks, DMA channels, and USB Endpoints.

### 5.4 Enumeration

In the `CyCx3ApplInit` function, a call to the `CyU3PUsbSetDesc` function enumerates CX3 as a UVC device. UVC Descriptors are defined in the `cycx3_uvcdscr.c` file. These Descriptors are defined for an image sensor sending 16 bits per pixel using the uncompressed YUY2 format, with 2 resolutions: 1920 x 1080 pixels at 30 fps and 1280 x 720 pixels at 60 fps. Refer to [Section 4.3.1](#) if you need to change these settings.

### 5.5 Configuring the MIPI CSI-2 Controller

The `CyCx3ApplInit` function first initializes the I<sup>2</sup>C, GPIO and PIB blocks that interface with the MIPI CSI-2 controller.

After the DMA channels are created, the GPIF II state machine is loaded by calling `CyU3PMipicsiGpifLoad`, which takes in the data bus width and Buffer size as parameters. The Buffer size to be passed as the parameter will be the DMA Buffer size as seen from the Producer Socket. So, the value loaded will be

`dma_buffer_size – (dma_header_size + dma_footer_size)`

The bus width selected should match with the type of image data being sent over CSI-2. Bus widths required for each supported image data type are defined in the [SDK API Guide](#).

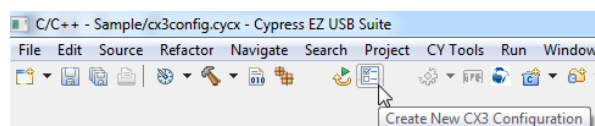
The state machine is then started and kept paused until image streaming is requested by the USB Host.

After the state machine is loaded and started, the MIPI CSI-2 controller can be initialized. This is done by calling `CyU3PMipicsiInit`, which will place the image sensor on reset by asserting the XRES pin. To bring the sensor out of reset, the XRES pin is de-asserted using the `CyU3PMipicsiSetSensorControl` function.

The MIPI CSI-2 controller is then configured (by the `CyU3PMipicsiSetIntfParams` function). This function takes in a structure of type `CyU3PMipicsiCfg_t` which contains the configuration parameters.

Cypress provides a **CX3 Configuration Tool** (screen shots shown in Figures 20 to 23) for easy generation of a CX3 project. This tool can generate the complete CX3 firmware project (including UVC/Vendor Class descriptors, handling of UVC requests, video streaming DMA engine, and the configuration of MIPI CSI-2 controller) based on user inputs. The procedure to access the tool and operate it are covered in [CX3 Application Software / USB Driver: Frequently Asked Questions – KBA91298](#) and CX3 Configuration Tool Help available in Cypress EZ-USB Suite (Eclipse) at *Help > Help Contents > Cypress EZ-USB Guides > EZ-USB Suite Guide > CX3 Configuration Utility*.

Figure 20. CX3 Configuration Tool Shortcut



**CX3 Configuration**

☒ Create New Mipi Receiver Configuration Project

Sensor Selection

☒ Create a Configuration with Basic Settings

☐ Select a Pre Defined Configuration

☐ Select an User defined Configuration

Project

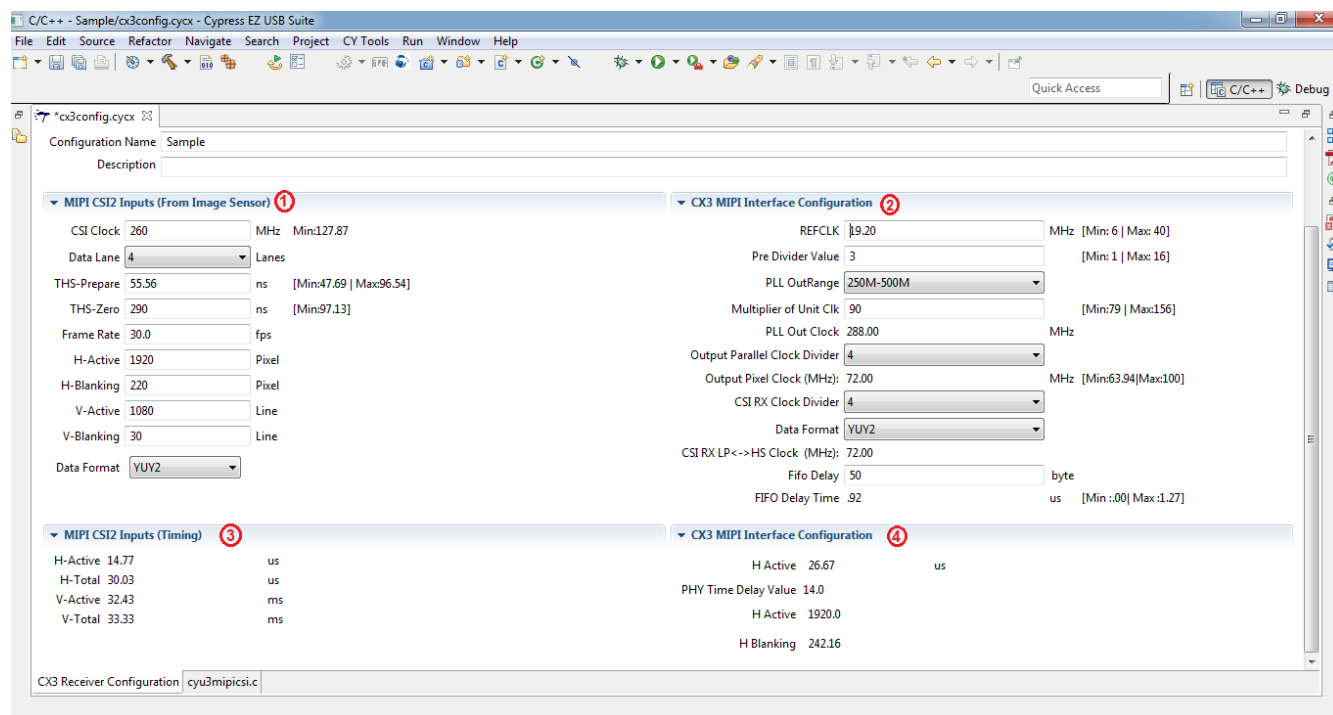
☒ Create New Project

☐ Add Only Mipi Receiver Configuration File

Add to Existing Project

[illegible]

Figure 23. CX3 MIPI Receiver Configuration Tool



### 5.5.1 CX3 MIPI Receiver Configuration

CX3 configuration tool can be used to generate a complete CX3 project or just the MIPI receiver configuration (which can be added to an existing project). A brief overview of fields in the MIPI Receiver Configuration tab (Figure 23) is provided in this section.

**MIPI CSI2 Inputs:** Refer to '1' marked in Figure 23. These fields are the CSI-2 input parameters. For example, Figure 23 shows a CSI-2 sensor interface streaming YUY2 Full HD (1080p) at a 260-MHz CSI clock with H-Blanking of 220 pixels and V-Blanking of 30 lines. It is necessary that these values match the actual CSI-2 interface parameters because calculations based on these inputs are used to validate the MIPI Controller configuration parameters.

All the parameters (including the data format) in this section are used for calculation purpose only. The data format provided in this section is used to get bits per pixel of the format being streamed.

Currently, the tool does not use THS-Prepare and THS-Zero value fields. To modify the THS-Prepare and Zero values, the SDK supports an API 'CyU3PMipicsciSetPhyTimeDelay'. See the FX3 SDK [API guide](#) for details on this API. A PHY Time Delay of 9 (2<sup>nd</sup> argument of above API) works with most sensors/ISPs.

**MIPI CSI2 Input Timings:** Refer to '3' marked in Figure 23. These are the timings calculated based on CSI-2 Input parameters.

**CX3 MIPI Interface Configuration:** Refer to '2' marked in Figure 23. The parameters in this section are used for configuring the three clocks (PLL Clock, CSI RX LP <-> HS clock, Parallel Output Pixel Clock) used within the MIPI CSI-2 controller. For details on these clocks, refer to Section 1.7 of [CX3 TRM](#).

The Output Pixel Clock (PCLK), Data format and Fifo Delay are the important parameters which decide the successful functioning of the MIPI controller in CX3.

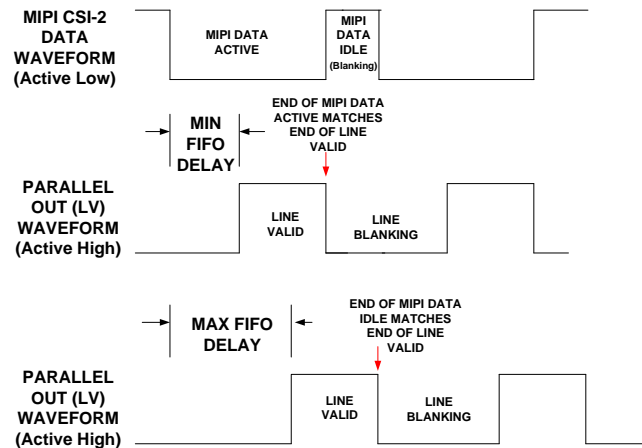
The Output Pixel Clock set should be within the range specified by the tool.

Data format decides the data bus width of the Parallel Output Interface. See Section 1.6 of CX3 TRM for the list of data formats supported and the Parallel Out data bus width supported by each format.

Fifo Delay parameter, when set, adds a delay to the Parallel Output at the beginning of each line. Fifo Delay is necessary when the Output Parallel Pixel Clock set is higher than the 'minimum' Output Parallel Pixel PCLK value suggested by the tool. The Figure 23 depicts the concept of Fifo delay. Though it must be noted that the fifo delay min and max values are not calculated through simple subtractions of MIPI H-Active time and Output Parallel H-Active time. There is a line buffer available within the MIPI controller, which needs to be accounted for while calculating the Fifo Delay time. The tool takes care of this adjustment.

**Note:** The current MIPI Line data should be sent out completely over the parallel interface before the next MIPI line data arrives. If this rule is violated, vertical splits can be observed in the video frame. So Fifo Delay should be carefully chosen.

Figure 23. Fifo Delay Concept



For configuring any other sensor to stream any other format, create a fresh project in the [FX3 SDK](#) and select the input video format when you see the sensor configuration page (Shown in Figure 22). The RGB format can be streamed in the same manner as the YUV format depending on the number of bytes in a pixel. To get more details on streaming the RGB format, refer other example projects in the [FX3 SDK](#). For streaming the MJPEG format at different resolutions and interfacing a CX3 device with an ISP (image signal processor), refer the source code of the CX3-based [Ascella RDK](#). Note that you need to purchase the Ascella - Cypress® CX3™ THine® ISP 13MP reference design kit (RDK) to get the firmware sources of the CX3-based Ascella RDK. Step-by-step guidelines are provided [here](#) to help you with this kit.

If you are looking for a MJPEG firmware example with the OV5640 sensor, you should sign NDA with OmniVision and create a [Technical Support](#) case.

## 5.6 Configuring the Image Sensor

After the MIPI CSI-2 interface is configured, the image sensor needs to be configured appropriately. This is done near the end of the CyCx3ApplInit function. Cypress provides a precompiled library that has functions to initialize and configure the OV5640 image sensor.

For other image sensors, the *cyu3imagesensor.c* file has a few helper functions that can be used to configure the image sensor.

The image sensor is configured using CX3's I<sup>2</sup>C master block. SensorWrite2B, SensorWrite, SensorRead2B, and SensorRead functions in the *cyu3imagesensor.c* file can be used to write and read image sensor configuration over I<sup>2</sup>C.

The SensorWrite2B and SensorWrite functions call the CyU3PI2cTransmitBytes standard API to write data to the image sensor. The SensorRead2B and SensorRead functions call the CyU3PI2cReceiveBytes standard API to read data from the image sensor. For more details on these APIs, refer to the [FX3 SDK API Guide](#).

## 5.7 Starting Video Streaming

A USB Host application such as VLC Player, e-CAMView, Media Player Classic or VirtualDub sits on top of the UVC driver to set the USB interface and the USB alternate setting combination to one that streams video (usually Interface 0 Alternate setting 1), and to send a PROBE/COMMIT control. This is an indication by the Host that it will shortly begin to stream video data. On a stream event, the USB Host application starts requesting image data from the CX3; the CX3 should then start sending the image data from the image sensor to the USB 3.1 GEN 1 Host.

When the CX3 receives a stream event, the USB event callback (CyCx3AppUSBSetupCB) configures the image sensor and the MIPI CSI-2 interface depending on the frame index received. Configuration parameters will depend on the resolution and frame rate requested by the Host; this is explained in the next section ([Section 5.8](#)).

Because the GPIF II state machine does not need to run until the Host requests image data, it is kept paused. When the stream event arrives, the CyCx3UvcStart function is called. This resumes the state machine (via CyU3PGpifSMControl), wakes up the MIPI CSI-2 interface (via CyU3PMipicsiWakeup), and powers up the image sensor.

**Note** The [SDK API Guide](#) contains detailed information about MIPI CSI-2- and GPIF II-related functions used in the preceding and following sections.

## 5.8 Selecting and Switching Frame Settings

A camera can support multiple resolutions and frame rates. Each supported resolution/frame-rate setting is placed in its own Video Streaming Frame Descriptor as described in [Section 4.3.1.2](#).

When a particular resolution or frame-rate setting is chosen in the USB Host application, the USB Host sends a SET\_CUR commit request to the Video Streaming interface. The fourth byte in the received structure indicates the frame Descriptor to be selected.

In the provided example project, if the device is operating in SuperSpeed, index 0 supports a 720p@60 fps video stream and index 1 supports a 1080p@30 fps video stream. When the USB Host wants to stream 1080p video, it sends a SET\_CUR request with the frame index set to '1'.

The CyCx3UvcAppUSBSetupCB function reads this index and, depending on what it received, configures the image sensor and the MIPI CSI-2 controller to stream video with the requested settings.

## 5.9 Setting Up DMA Buffers

The UVC specification requires adding a 12-byte header to each USB transfer (meaning each 16-KB DMA Buffer in this application). However, the CX3 architecture requires that any DMA Buffer associated with a DMA Descriptor must have a size that is a multiple of 16 bytes.

Reserving 12 bytes in a DMA Buffer for the CX3 CPU to fill in would place the DMA Buffer boundary at a nonmultiple of 16 bytes. Therefore, the DMA Buffer size should be 16,384 minus 16, not 12. This makes the DMA Buffer size, not including the 12-byte header that the CX3 firmware adds, 16,384-16=16,368 bytes. The DMA Buffer is ordered as the 12-byte header, the 16,368 video bytes, and finally four unused bytes at the end of the DMA Buffer. This creates a DMA Buffer that can utilize the maximum USB BULK burst size of 16 x 1024 byte packets, or 16,384 bytes.

## 5.10 Handling DMA Buffers During Video Streaming

The CyCx3Applnit function creates a manual DMA channel with callback notification for producer and consumer events.

The consumer notification is used to track the amount of data read by the Host. After the Host has read the entire frame, the tracking variables are reset and the GPIF II state machine is restarted. The producer event notification is used to append the 12-byte header and commit the data to the Host.

In the DMA callback, the firmware checks for a produced DMA Buffer via CyU3PDmaMultiChannelGetBuffer. A DMA Buffer becomes available to the CX3 CPU when the GPIF II producer DMA Buffer is committed or if it is forcefully wrapped up by the CX3 CPU. During an active frame period, the image sensor is streaming data and GPIF II produces full DMA Buffers. At this time, the CX3 CPU has to commit 16,380 bytes of data to USB.

At the end of a frame, usually the last DMA Buffer is partially filled. In this case, the firmware must forcefully wrap up the DMA Buffer on the producer side to trigger a producer event and then commit the DMA Buffer to USB with the appropriate byte count. The forceful wrap-up of the DMA Buffer (produced by GPIF II) using the CyU3PDmaMultiChannelSetWrapUp call is executed in the GPIF II callback function, CyCx3GpifCB. This callback function is triggered when GPIF II sets a CPU interrupt, which is raised when one frame ends.

**Note:** The UVC header carries information about the frame identifier and an end-of-frame marker. At the end of a frame, the firmware sets bit 1 and toggles bit 0 of the second UVC header byte (see `CyCx3UvcAddHeader`). In addition, the “hitFV” variable is set to indicate that frame capturing from the image sensor has ended.

The `glDmaDone` variable keeps track of DMA Buffers to ensure that all data has been drained from the CX3 FIFO.

### 5.11 Resuming the Stream at the End of a Frame

At the end of a frame, the GPIF II state machine generates a CPU interrupt, which starts the chain of events described above. When the last DMA Buffer of the frame is drained out by the USB Host, the `glDmaDone` variable becomes zero and the following actions are performed:

- Reset the 500-ms DMA Channel Reset Timer, which acts as a watchdog to flush out stale image data.
- Call the `CyU3PGpifSMSwitch` function to restart the GPIF II state machine at one of its START states. If the last Buffer of the frame was read by Socket 0, the GPIF II state machine is restarted at `ALPHA_CX3_START_SCK1` to start reading the next frame to Socket 1, and vice versa.

In addition, the UVC specification requires the Frame ID bit in the header to be toggled every frame. This is done in the `CyCx3UvcAddHeader` function just before the last Buffer is committed to the USB Host.

### 5.12 Terminating the Video Stream

There are three ways image streaming can be terminated:

- Camera disconnected from the Host
- USB Host program closes
- USB Host issues a reset or suspend request to CX3.

Because the video stream can be terminated when there is residual data in the FIFOs, proper cleanup is required.

When a video stream is terminated, the firmware resets streaming-related variables, resets the DMA channel, and pauses the GPIF II state machine. It also powers down the camera and the MIPI CSI-2 interface. These are done in the `CyCx3AppStop` function.

When the USB Host application closes, it issues a clear feature request on a Windows platform or a Set Interface with alternate setting = 0 request on a Mac platform. Streaming stops when this request is received. This request is handled in the `CyCx3AppUSBSetupCB` function when the target is `CY_U3P_USB_TARGET_ENDPT` and the request is `CY_U3P_USB_SC_CLEAR_FEATURE`.

After a video stream is terminated, the image sensor is powered down and the CX3 core is suspended using `CyU3PSysEnterSuspendMode`. With an activity on the USB bus, the CX3 core wakes up and powers up the image sensor to restart the video stream.

The image sensor can be powered down either by using CCI (I<sup>2</sup>C) commands or by using the XSHUTDOWN pin of CX3. This pin can be controlled using the `CyU3PMipicisiSetSensorControl` function with `CY_U3P_CSI_IO_XSHUTDOWN` as the first parameter.

## 6 Hardware Setup

### 6.1 Testing With the CX3 RDK

The current project has been tested on a setup that includes the CX3 Reference Design Kit (RDK), which includes an OmniVision OV5640 image sensor. Details about the kit are available on the Cypress website at [www.cypress.com/cx3](http://www.cypress.com/cx3) under the **Kits** tab.

#### 6.1.1 Kit Procurement

1. Buy the RDK from e-Con Systems at <http://www.e-consystems.com/CX3-Reference-Design-Kit.asp>
2. Cypress provides a binary library with the CX3 SDK that can perform basic operations on the OV5640 sensor. If additional functionality needs to be supported, sign an NDA with OmniVision (email your request to [usb3@cypress.com](mailto:usb3@cypress.com) for an expedited response). If you have an NDA, send it to [usb3@cypress.com](mailto:usb3@cypress.com). Cypress will then provide the OmniVision-specific source files after NDA verification.
3. Use a USB 3.1 GEN 1 Host-enabled computer to evaluate SuperSpeed performance.

The CX3 RDK comes with a Quick Start Guide, a Hardware User Manual, and a Firmware Build Manual to help you start using the RDK.

## 6.2 Designing Your Own Board

When designing your own board, a few guidelines should be followed to ensure that the board functions well. The [FX3/FX3S Hardware Design Guidelines](#) application note discusses recommended practices for FX3/FX3S hardware design, which are also applicable to CX3.

For MIPI CSI-2 signals, additional routing guidelines should be followed:

- The length of the traces on the board should not exceed 100 mm.
- Transmission-line impedance ( $Z_0$ ) for the tracks of differential pair should be  $100\ \Omega \pm 10\%$ .
- The intralane (between P and N lines of a pair) length mismatch should be  $< 0.5\ \text{mm}$ .
- The interlane (between two MIPI CSI lane signal pairs) length mismatch should be  $< 1.5\ \text{mm}$ .
- The space between the P and N signal tracks should be twice the width of the tracks.

## 7 UVC-Based Host Applications

Various Host applications allow you to display and capture video from a UVC device. The [Media Player Classic](#) is a popular choice for Windows OS. Two additional Windows apps are [VirtualDub](#) (an open-source application) and [e-CAMView](#).

Linux systems can use the V4L2 driver and VLC Media Player to stream video. The VLC Media Player is available on the web.

Mac platforms can use FaceTime, iChat, Photo Booth, and Debut Video Capture software to create an interface with the UVC device to stream video.

## 8 Troubleshooting

If you face any problem with the device or the firmware, the first step is to get more data and narrow down the source of the problem. To better achieve this, UART and JTAG debugging can be enabled.

The “CX3\_DEBUG\_ENABLED” switch in the `cycx3_uvc.h` file can be enabled to print a variety of counters, error messages and various other debug data. Connect the UART port on your board (or the CX3 RDK) to a PC via a UART cable or a USB-to-UART bridge. Open Hyperterminal, Tera Term, or another utility that gives access to the COM port on the PC. Set the UART configuration as follows before starting transfers: 115,200 baud, no parity, 1 stop bit, no flow control, and 8-bit data. This should be sufficient to capture debug prints.

In addition, a JTAG debugger can be used to debug the firmware in steps. Refer to Section 12.2.2.3 “Executing and Debugging” of the Programmer’s Manual (available at Start > All Programs > Cypress > EZ-USB FX3 SDK) for details on debugging via JTAG.

Some FAQs on CX3 firmware, hardware and application software have been documented in [KBA91297](#), [KBA91295](#) and [KBA91298](#).

A few common problems, their causes, and solutions to those problems are listed.

### **Problem: The device does not enumerate on the PC.**

**Cause 1:** An API in your firmware may have encountered an error condition. This will cause the error handler to be called and the firmware may stall or fail to enumerate.

**Solution 1:** Use a JTAG debugger or a UART/RS232 cable and inspect the return status of all calls to see if any of them failed. Then, use the API Guide to understand why the call failed.

**Cause 2:** The USB signal integrity in the board is poor.

**Solution 2:** Read the Design Guidelines application note and ensure a proper board design.

**Cause 3:** Appropriate drivers are not installed.

**Solution 3:** Try uninstalling the device and reinstalling it in the Device Manager. You can also change the VID/PID pair to force a new device installation.

**Cause 4:** Descriptors may have incorrect values.

**Solution 4:** A typical source of errors is the length field. Make sure that the `wTotalLength` field in the configuration descriptor has the correct overall length. Also, verify similar fields for class-specific Video Control and Video Streaming Descriptors.

**Problem: The device enumerates but the USB Host application (like e-CAMView) shows a black screen.**

**Cause 1:** Inspect the `glDMATxCount` variable in the `CyCx3UvcAppThread_Entry` function and verify that it increments. If it doesn't, there is probably a problem with the interface between CX3 and the image sensor.

**Solution 1:** Verify that the image sensor is connected properly to the CX3. Then, verify that sensor initialization, MIPI CSI-2 controller configuration, and GPIF II bus-width selection are done correctly.

**Cause 2:** If you see incrementing prints of `glDMATxCount`, the image data that is being sent out needs to be verified. First, check the total amount of data being sent out per frame. To know this, find the length of the data packets that have the end-of-frame bit set in the header. (The second byte of the header is either 0x8E or 0x8F for the end-of-frame transfer). For other data packets, the length of data transferred will be the DMA Buffer size minus the footer size.

The total image data transferred in a frame (not including the UVC header) should be  $\text{width} \times \text{height} \times 2$ .

**Solution 2:** If the total image size is less (or more) than what is required, the image sensor is probably sending less (or more) data than required. Check the image sensor and MIPI CSI-2 controller configuration.

**Problem: Video can be streamed in a Hi-Speed connection but not in a SuperSpeed connection**

**Cause:** This can be because of poor signal integrity on SSTX/SSRX lines. Verify that the SuperSpeed traces are following the guidelines described in [AN70707](#). You should also ensure that only certified cables are used.

Additionally, the `CX3_ERROR_THREAD_ENABLE` switch enables logging MIPI CSI-2 errors which can be used to see if the CSI-2 interface incurred an error. Refer to the [SDK API Guide](#) for more information on the types of errors.

If the problems still persist, create a Cypress [technical support case](#).

## 9 Summary

This application note describes how an image sensor with a MIPI CSI-2 interface and conforming to the USB Video Class can be implemented using Cypress's EZ-USB CX3. Specifically, it shows:

- How the Host application and driver interact with a UVC device
- How the UVC device manages UVC-specific requests
- How to configure the MIPI CSI-2 interface to receive data from typical image sensors
- How to display video streams and change camera properties in a Host application
- How to add a USB interface to the UVC device for debugging purposes
- How to find Host applications available on different platforms, including an open-source Host application project
- How to troubleshoot and debug the CX3 firmware, if required.

## 10 Related Resources

- [AN75705 - Getting Started with EZ-USB® FX3™](#)
- [AN70707 - EZ-USB® FX3™/FX3S™ Hardware Design Guidelines and Schematic Checklist](#)
- [AN75779 - How to Implement an Image Sensor Interface Using EZ-USB® FX3™ in a USB Video Class \(UVC\) Framework](#)
- [Designing FX3™/CX3-Based USB Type-C Products - KBA218460](#)
- [Ascella - Cypress® CX3™ THine® ISP 13MP reference design kit \(RDK\)](#)
- [Denebola – USB 3.0 UVC Reference Design Kit \(RDK\)](#)
- [Video: Cypress EZ-USB CX3 Introduction](#)
- [Video: Introduction to EZ-USB CX3 Solution for HD Video](#)

## Document History

Document Title: AN90369 - How to Interface a MIPI® CSI-2 Image Sensor with EZ-USB® CX3™

Document Number: 001-90369

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4336592	KUMR	04/08/2014	New Application Note.
*A	4560876	KUMR	11/04/2014	Edited the document based on internal review.
*B	4616110	KUMR	01/07/2015	Updated the document title
*C	4871745	NIKL	08/04/2015	Updated the AN to reflect the changes in the CX3 Configuration tool Template update
*D	5704174	SAVJ	05/10/2017	Updated logo, copyright, API and file names in the document. Added related resource section and more details on video formats supported by CX3 in section 5.5.1.

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at [Cypress Locations](#).

## Products

ARM® Cortex® Microcontrollers	<a href="http://cypress.com/arm">cypress.com/arm</a>
Automotive	<a href="http://cypress.com/automotive">cypress.com/automotive</a>
Clocks & Buffers	<a href="http://cypress.com/clocks">cypress.com/clocks</a>
Interface	<a href="http://cypress.com/interface">cypress.com/interface</a>
Internet of Things	<a href="http://cypress.com/iot">cypress.com/iot</a>
Memory	<a href="http://cypress.com/memory">cypress.com/memory</a>
Microcontrollers	<a href="http://cypress.com/mcu">cypress.com/mcu</a>
PSoC	<a href="http://cypress.com/psoc">cypress.com/psoc</a>
Power Management ICs	<a href="http://cypress.com/pmic">cypress.com/pmic</a>
Touch Sensing	<a href="http://cypress.com/touch">cypress.com/touch</a>
USB Controllers	<a href="http://cypress.com/usb">cypress.com/usb</a>
Wireless Connectivity	<a href="http://cypress.com/wireless">cypress.com/wireless</a>

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

## PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6](#)

## Cypress Developer Community

[Forums](#) | [WICED IOT Forums](#) | [Projects](#) | [Videos](#) | [Blogs](#) | [Training](#) | [Components](#)

## Technical Support

[cypress.com/support](http://cypress.com/support)



Cypress Semiconductor  
198 Champion Court  
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2014-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spanion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spanion, the Spanion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.