

How to interface a MIPI CSI-2 image sensor with EZ-USB™ CX3

About this document

Scope and purpose

This application note gives implementation details on how to interface a MIPI CSI-2 image sensor with EZ-USB™ CX3.

Intended audience

This document is intended for anyone who uses the USB 3.2 Gen 1 application: a camera (MIPI CSI-2 image sensor interfaced with EZ-USB™ CX3) streaming uncompressed data into a PC.

Keypoints

- MIPI CSI-2 Sensor Integration with EZ-USB™ CX3
- USB Video Class (UVC) implementation
- Firmware, hardware, and tool support

About this product family

Product family

EZ-USB™ is a family of high-performance USB peripheral controllers designed to add USB 3.2 Gen 1 (5 Gbps) connectivity to a wide range of embedded systems, including those in industrial automation, medical imaging, machine vision, and consumer electronics.

Target applications

- [Healthcare](#)
- [Industrial automation](#)
- [Machine vision](#)
- [Robotics](#)
- [Industrial robots system solutions for Industry 4.0](#)
- [Presence sensing](#)

Table of contents

Table of contents

About this product family	1
Table of contents.....	2
1 Introduction	4
2 Interfacing an image sensor to CX3.....	6
2.1 MIPI CSI-2 interface	6
2.2 GPIF II block	7
2.2.1 DMA basics.....	8
2.2.2 Why two sockets are used.....	8
2.2.3 GPIF II state machine	9
3 Setting up the DMA system	11
3.1 DMA buffers.....	17
4 USB video class (UVC)	19
4.1 Enumeration data	19
4.2 Operational code.....	19
4.3 UVC requirements	20
4.3.1 USB descriptors for UVC.....	20
4.3.1.1 Video control (VC) interface.....	21
4.3.1.2 Video streaming (VS) interface	21
4.3.2 UVC-specific requests	23
4.3.2.1 Control requests: brightness, PTZ, hue, saturation, and others	27
4.3.2.2 Streaming requests: probe and commit control	27
4.3.2.3 Video data format: YUY2	27
4.3.2.4 UVC video data header	28
5 CX3 MIPI Configuration Tool	30
5.1 Launching the tool	30
5.2 Adding sensor details to CX3 Configuration tool	32
5.2.1 Mapping input data format to output video format based on parallel data width selection	35
5.3 Configuring CX3 MIPI receiver.....	38
6 CX3 firmware	41
6.1 Application threads.....	43
6.1.1 USB suspend event	43
6.2 DMA channel reset event.....	43
6.3 Handling UVC requests	44
6.4 Initialization.....	44
6.5 Enumeration.....	44
6.6 Configuring the MIPI CSI-2 controller	44
6.7 Configuring the image sensor	45
6.8 Starting video streaming.....	45
6.9 Selecting and switching frame settings.....	45
6.10 Setting up DMA buffers.....	46
6.11 Handling DMA buffers during video streaming	46
6.12 Resuming the stream at the end of a frame	46
6.13 Terminating the video stream	47
7 Hardware setup	48
7.1 Testing with the CX3 RDK.....	48

Table of contents

7.1.1	Kit procurement	48
7.2	Designing your own board	48
8	UVC-based host applications	49
9	Troubleshooting	50
10	Summary	52
11	Related resources	53
	Revision history.....	54
	Disclaimer.....	55

Introduction

1 Introduction

The high bandwidth provided by USB 3.2 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 3.2 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 3.2 Gen 1 converter based on Infineon EZ-USB™ CX3, a variant of EZ-USB™ FX3 created specifically for this purpose.

If you are new to EZ-USB™ product family, for more details, see the [AN75705 - Getting Started with EZ-USB™ FX3](#) application note.

CX3 provides the ability to add SuperSpeed USB connectivity, over USB Video Class (UVC), to image sensors supporting the MIPI CSI-2 interface. 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, Webcamoid, 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

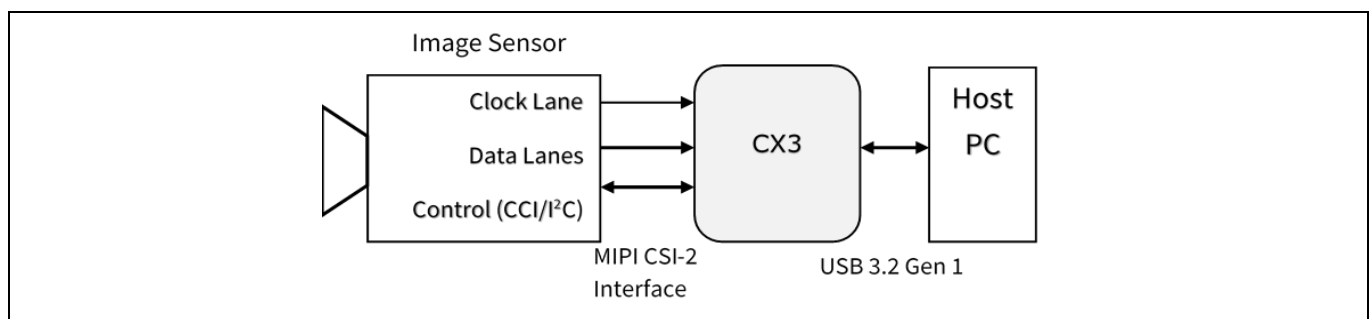


Figure 1 Camera application

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 Infineon 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 3.2 Gen 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

Introduction

- 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 the sub-blocks of the block diagram numbered. Tasks executed by each sub-block are described.

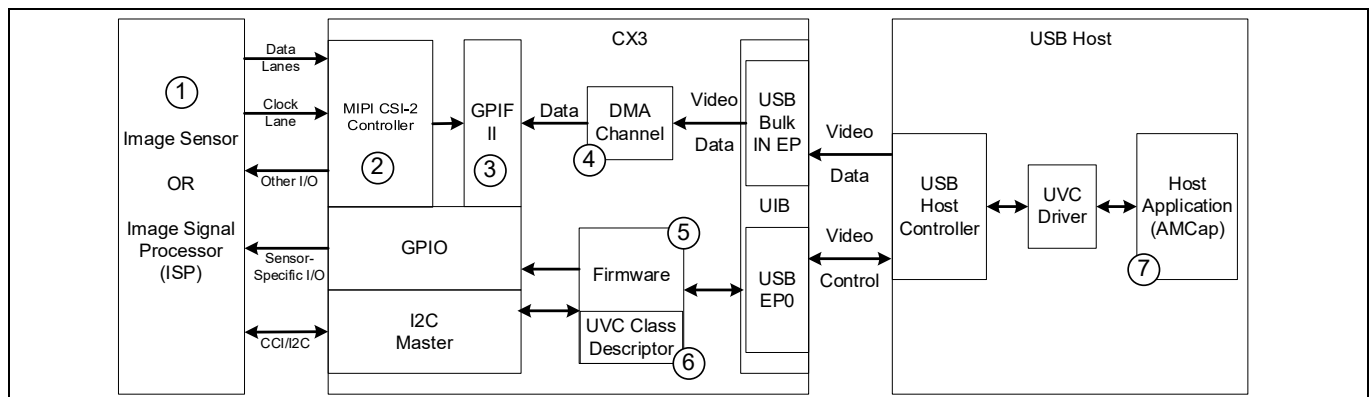


Figure 2 System block diagram

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 [MIPI CSI-2 interface](#)
3. Configure the GPIF II block according to the image data format. See [GPIF II block](#)
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 the maximum bandwidth can be used to stream video from the image sensor to the PC. See [Setting up the DMA system](#)
5. The CX3 firmware initializes the hardware blocks of CX3 (see [Initialization](#)), configures the image sensor (see [Configuring the image sensor](#)) and MIPI CSI-2 controller (see [Configuring the MIPI CSI-2 controller](#)), enumerates the device as a UVC camera (see [Enumeration](#)), handles UVC-specific requests (see [Handling UVC requests](#)), translates video control settings (such as brightness) to the image sensor over the CCI (I²C) interface, adds a UVC header to the video data stream (see [Handling DMA buffers during video streaming](#)), and commits the video data with headers to USB
6. Provide the proper USB descriptors so that the host recognizes the peripheral as a device conforming to the UVC. See [USB video class \(UVC\)](#)
7. Use a host application (such as e-CAMView, Webcamoid, 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 [UVC-based host applications](#)

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 (if available) 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.

Interfacing an image sensor to CX3

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 a 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 I²C 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.

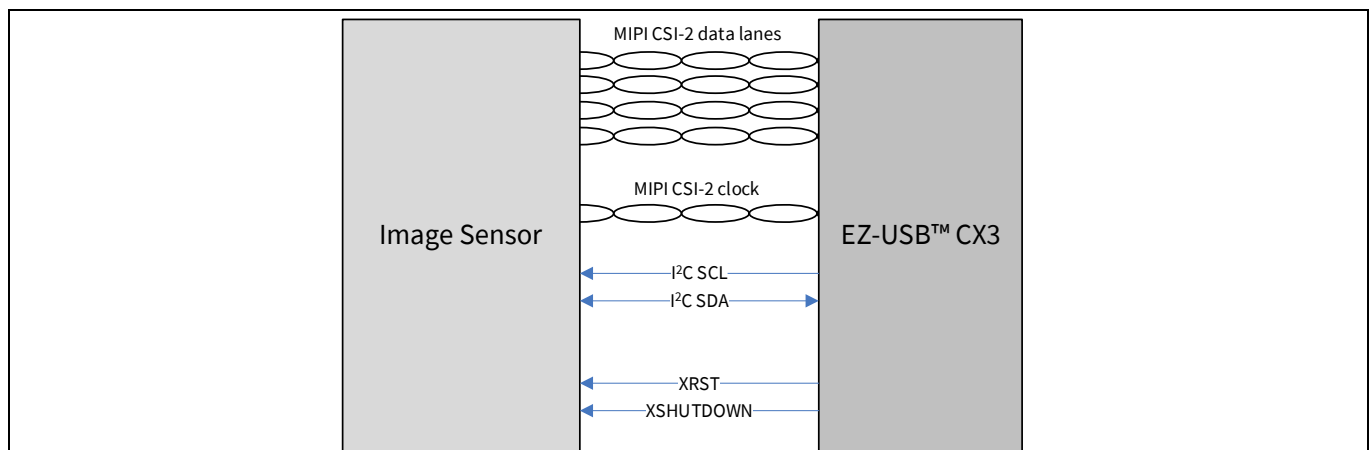


Figure 3 CX3's interface with the image sensor

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

Note: The **CyU3PMipicsiSetSensorControl** API in the FX3 SDK can be used to drive XRES and XSHUTDOWN signals to the sensor.

Interfacing an image sensor to CX3

MCLK: The master (or reference) clock required by the image sensor to be supplied using an external clock oscillator. CX3 provides MCLK only for testing the image sensor. This signal is not suitable to use in the final product. For production, use an external clock generator as the clock input for the sensor.

See the datasheet, [EZ-USB™ CX3 MIPI CSI2 to USB 5 Gbps Camera Controller](#), for the pin mapping of the CSI-2, CCI, and the three additional signals. In addition, see 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. See [CX3 firmware](#) 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 the “Image sensor interface” section 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.

The Camera Serial Interface (CSI) clock setting in the CX3 MIPI configuration utility is in DDR mode (data is sampled on both the clock edges). The following table lists the maximum supported CSI clock for each MIPI lane setting and the resulting throughput for each configuration.

Number of MIPI data lanes	Maximum allowed CSI clock frequency	Maximum data rate per MIPI lane	Maximum supported bit rate
1	500 MHz	1 Gbps	1 Gbps
2	500 MHz	1 Gbps	2 Gbps
3	400 MHz	800 Mbps	2.4 Gbps
4	300 MHz	600 Mbps	2.4 Gbps

Note: Calculate the maximum supported bit rate for CX3 (CSI clock x 2 x number of MIPI lanes) using the CSI clock considering zero blanking times.

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).

Interfacing an image sensor to CX3

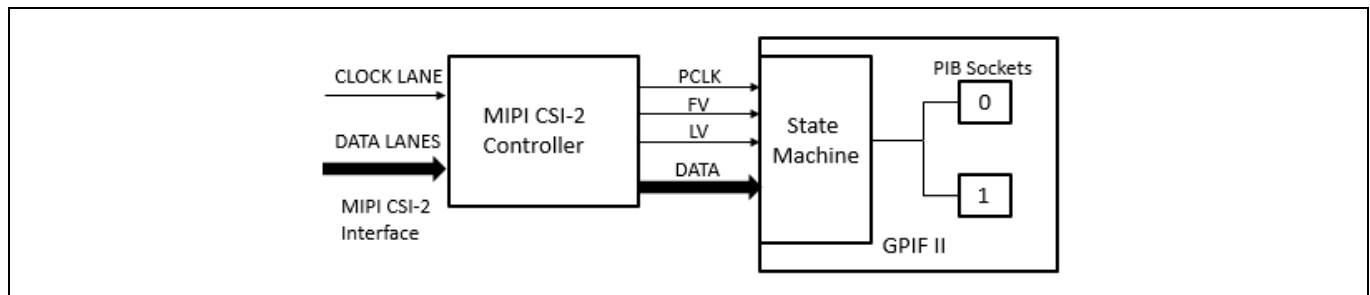


Figure 4 GPIF II block

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*.

Interfacing an image sensor to CX3

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, see [Setting up the DMA system](#).

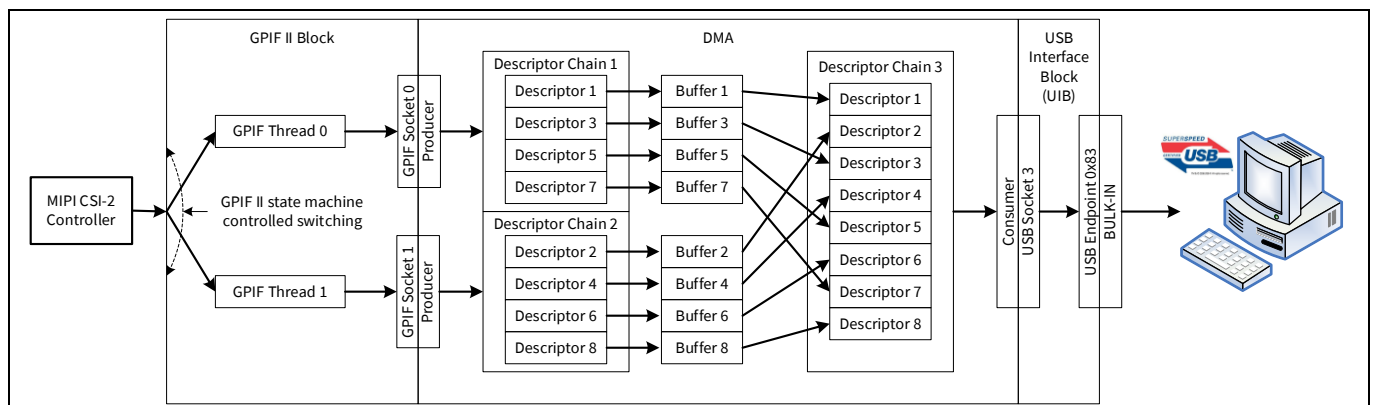


Figure 5 CX3 Data transfer architecture

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 (see [Why two sockets are used](#)), 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.

Interfacing an image sensor to CX3

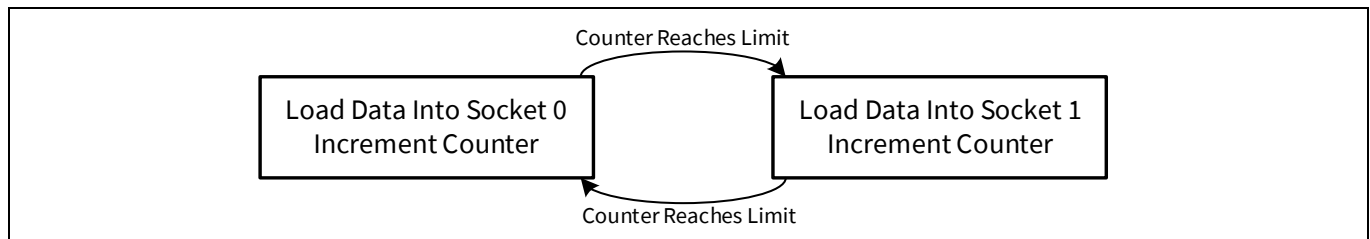


Figure 6 Data transfer Into two sockets

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:

$$count = \left(\frac{producer_buffer_size(L)}{data_bus_width\ (in\ bytes)} \right) - 1$$

A note on bus width:

The GPIF II data bus width must be selected depending on the data format of the image sensor. The resolution width (or line size) in bytes should be divisible by the GPIF II bus width. In this application note, the sensor is configured to output 16-bit YUV422 data and therefore, the data bus is configured to 16 bits. See 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.

A note on image line size:

The resolution width or line size in bytes should be a multiple of 4 bytes. If this condition is not met, the CSI-2 MIPI controller block of CX3 will add additional padding bytes to the line data.

Setting up the DMA system

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 the [Why two sockets are used](#) section). 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.

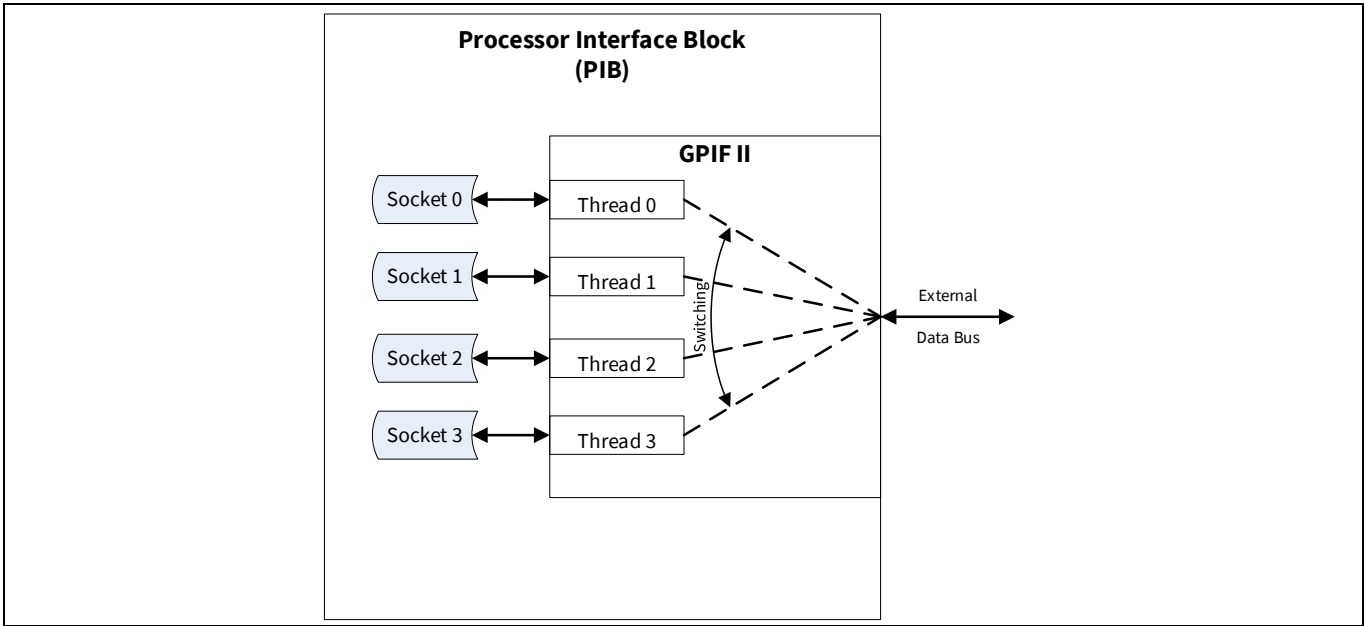


Figure 7 Default GPIF II socket/thread mapping

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

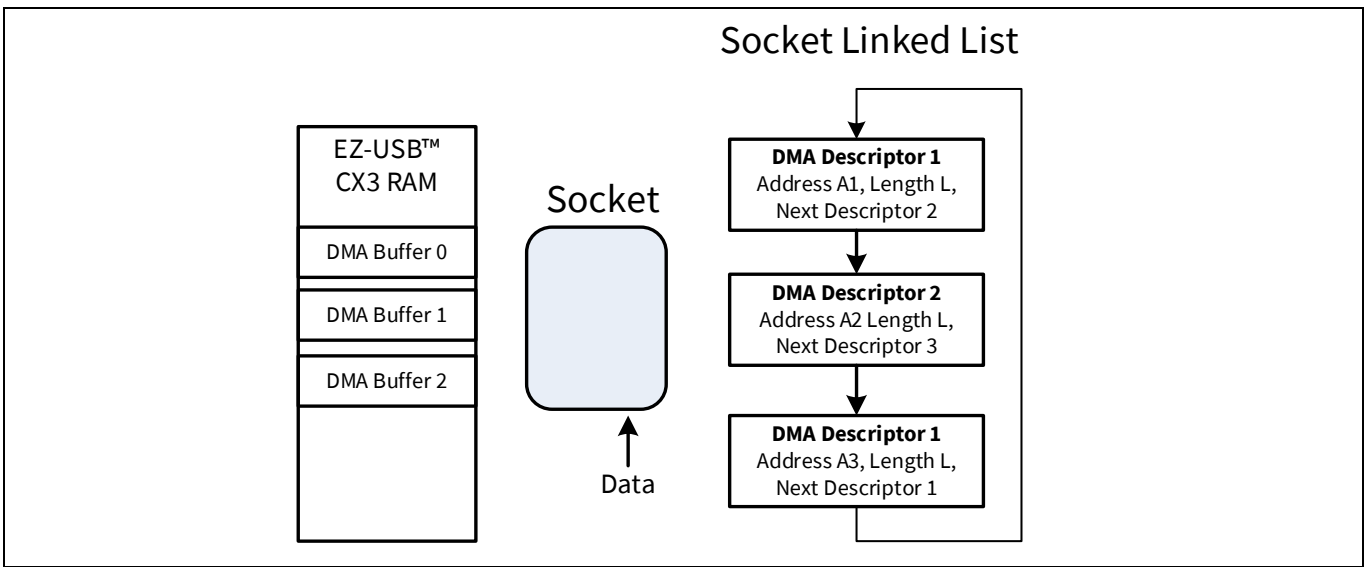


Figure 8 A socket routes data according to a list of DMA descriptors

How to interface a MIPI CSI-2 image sensor with EZ-USB™ CX3

Setting up the DMA system

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.

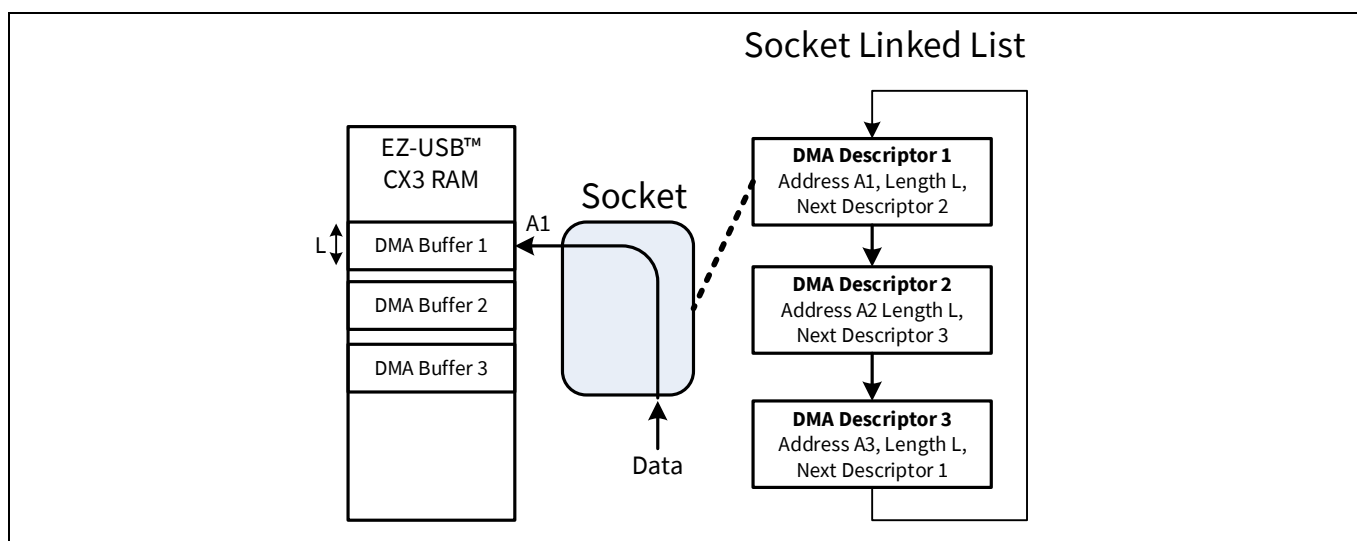


Figure 9 A socket operating with DMA descriptor 1

In [Figure 9](#), the socket has loaded DMA descriptor 1, which tells it to transfer bytes to RAM starting at A_1 until it has transferred L bytes, at which time it retrieves DMA descriptor 2 and continues with its address and length settings A_2 and L , respectively ([Figure 10](#)).

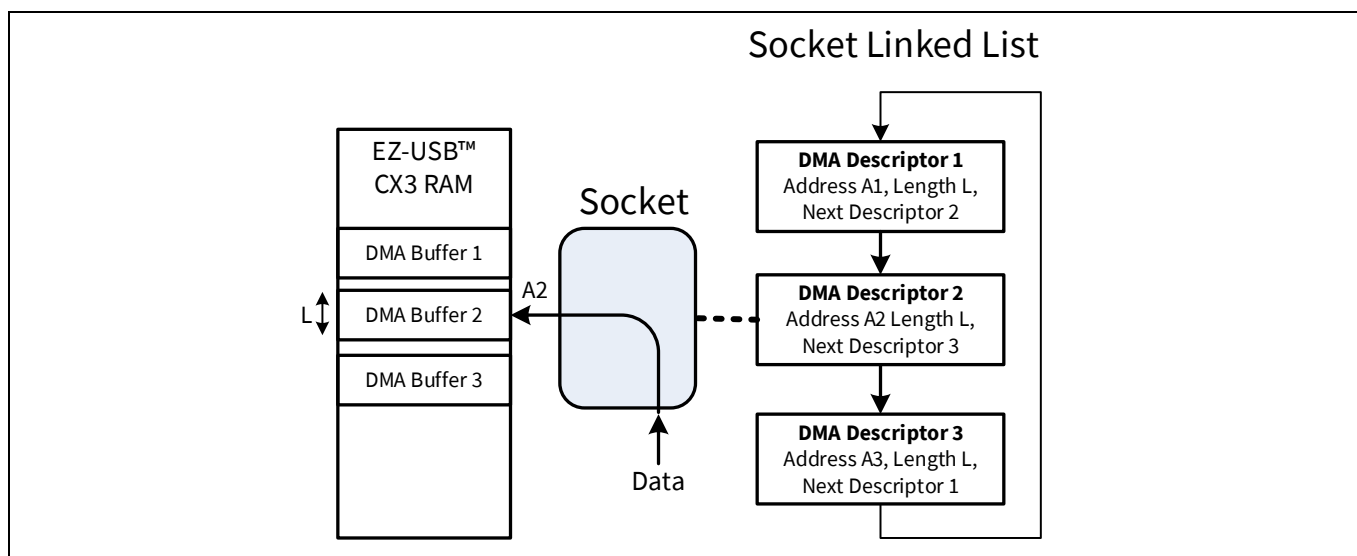


Figure 10 A socket operating with DMA descriptor 2

Setting up the DMA system

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.

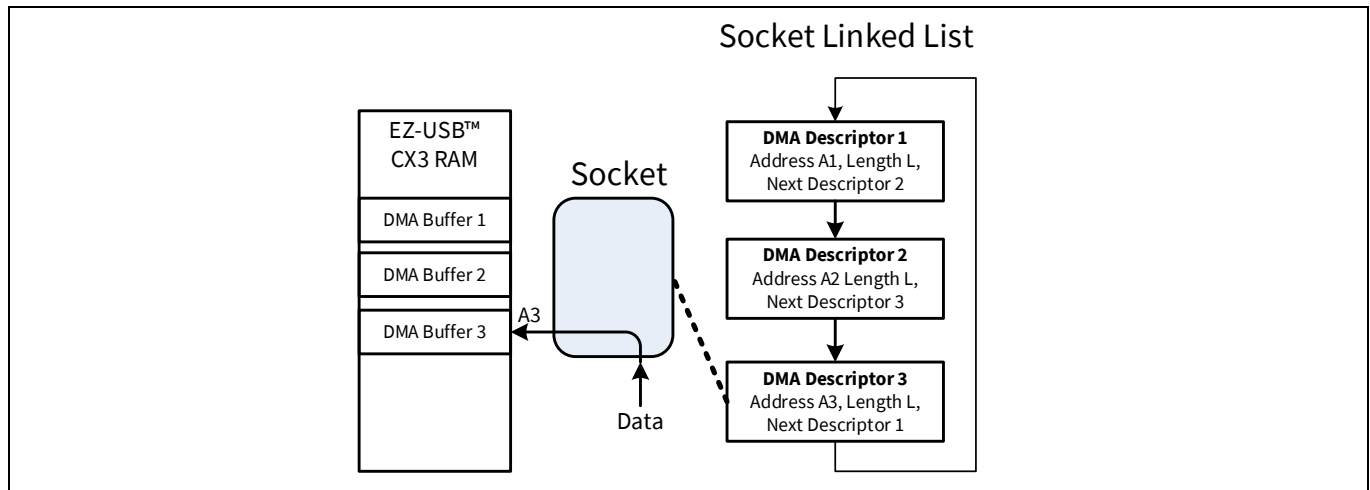


Figure 11 A socket operating with DMA descriptor 3

[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.

Setting up the DMA system

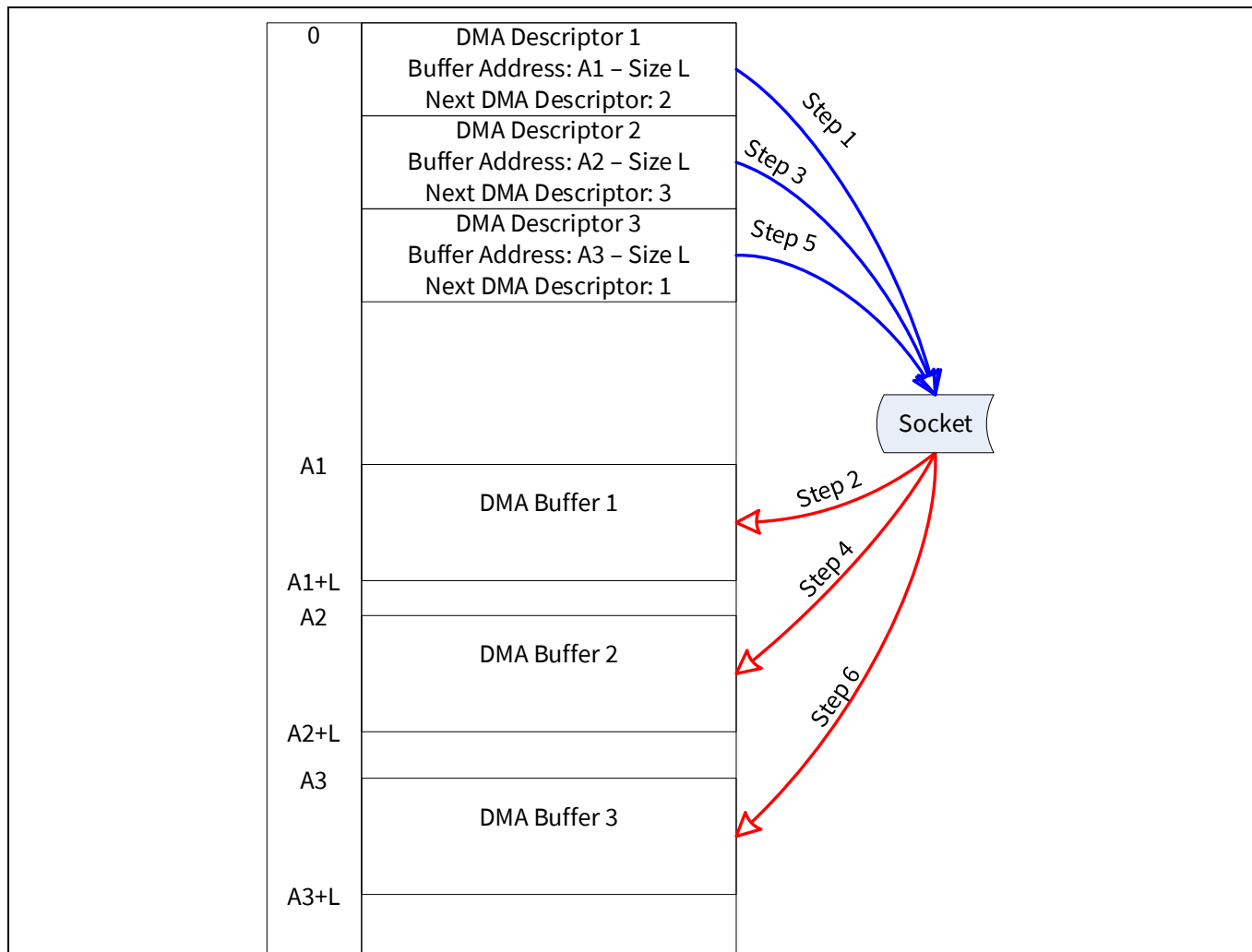


Figure 12 DMA transfer example

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 μ 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., horizontal/line blanking interval of the sensor). However, this approach lacks flexibility, if, for example, the video resolution is changed.

How to interface a MIPI CSI-2 image sensor with EZ-USB™ CX3

Setting up the DMA system

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 3.2 Gen 1 maximum burst rate for BULK transfers. USB 3.2 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](#).

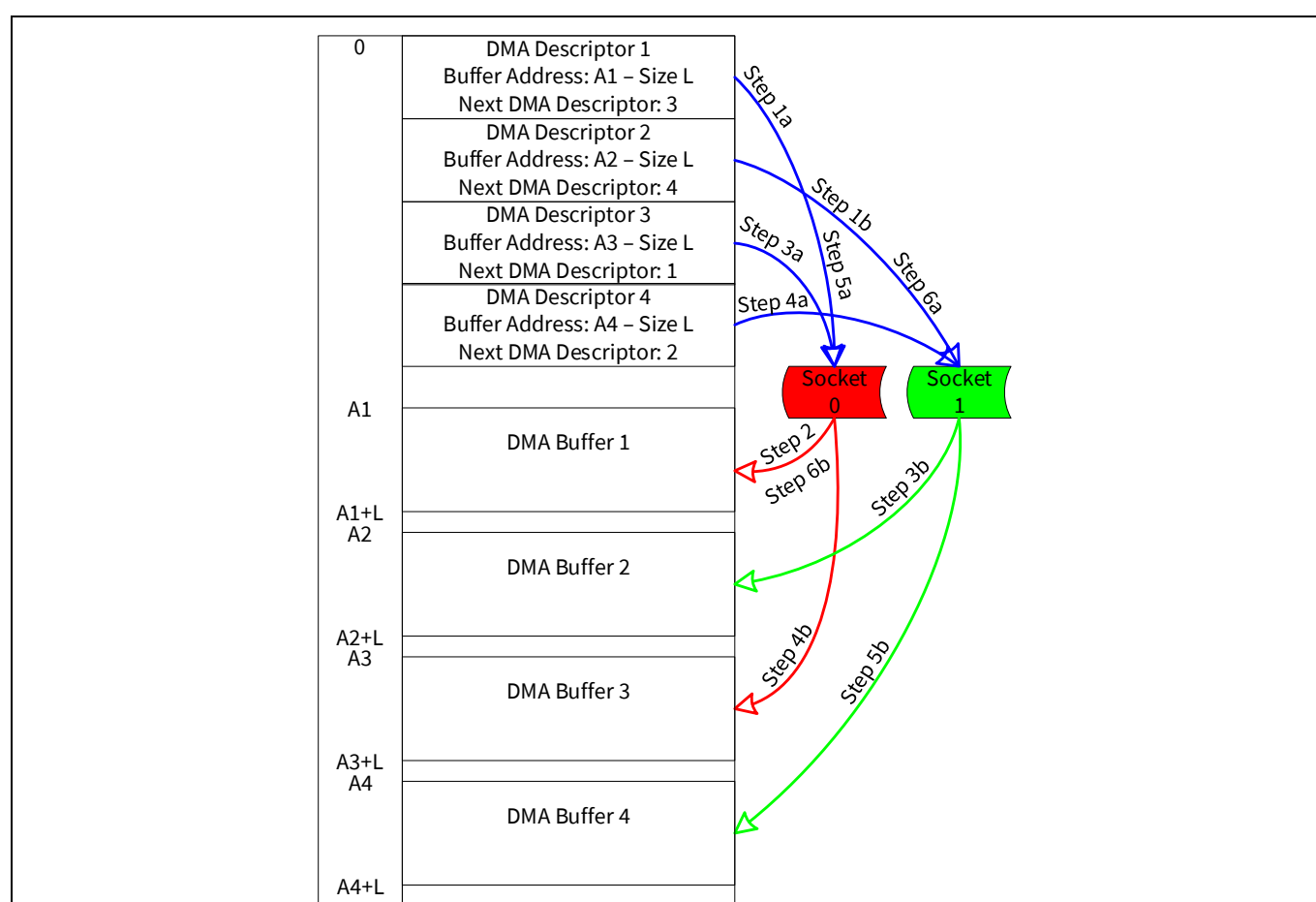


Figure 13 Dual sockets yield seamless transfers

Setting up the DMA system

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 the [Application threads](#) section.

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 “CyCx3ApplInit”. 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 3.2 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”.

Setting up the DMA system

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 the [DMA basics](#) section.

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”

Note: *If the DMA buffer size in bytes is chosen such that it is not a multiple of the UVC frame size in bytes, the last buffer of the frame will result in a partially filled DMA buffer. This partially filled DMA buffer can be used to recognize the end-of-frame condition to add corresponding UVC headers while streaming the image data.*

Setting up the DMA system

Note: *The resulting partial DMA buffer size should be a multiple of 4, because the `CyU3PDmaSocketSetWrapUp` API can commit data only in multiples of 4.*

USB video class (UVC)

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_uvcdscr.c` (explained in the [USB descriptors for UVC](#) section) that contains the UVC enumeration data. The USB specification, which defines the format for UVC descriptors, is available at 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.

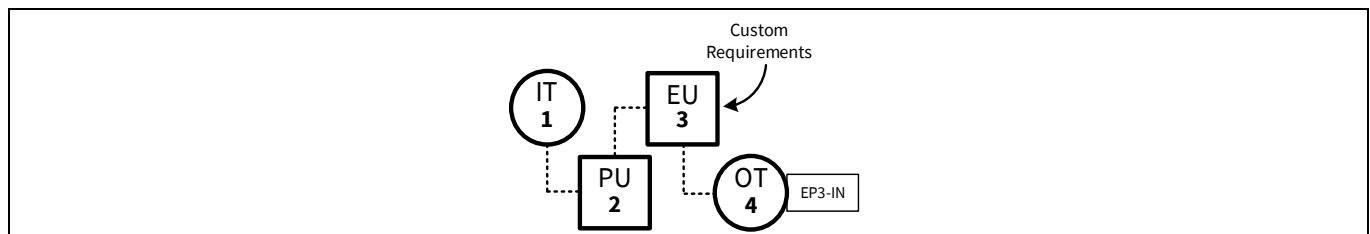


Figure 14 UVC diagram of the camera architecture

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.

USB video class (UVC)

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 UVC 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_uvcdscr.c* file contains the USB descriptor tables. The byte arrays “CyCx3USBHSSConfigDscr” (Hi-Speed) and “CyCx3USBSSConfigDscr” (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.

USB video class (UVC)

4.3.1.1 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_uvcdesc.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.

4.3.1.2 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.

USB video class (UVC)

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 30 fps 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.

USB video class (UVC)

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 the [USB descriptors for UVC](#) section. 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.

USB video class (UVC)

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.

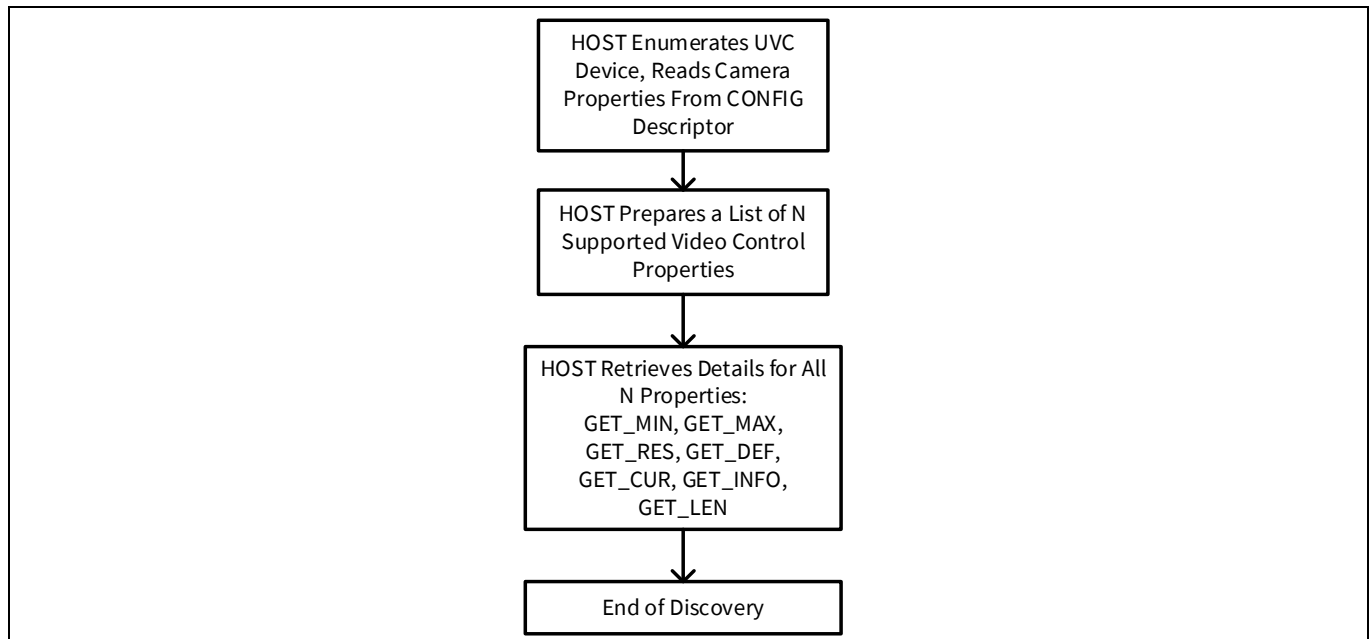


Figure 15 UVC enumeration and discovery flow

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.

USB video class (UVC)

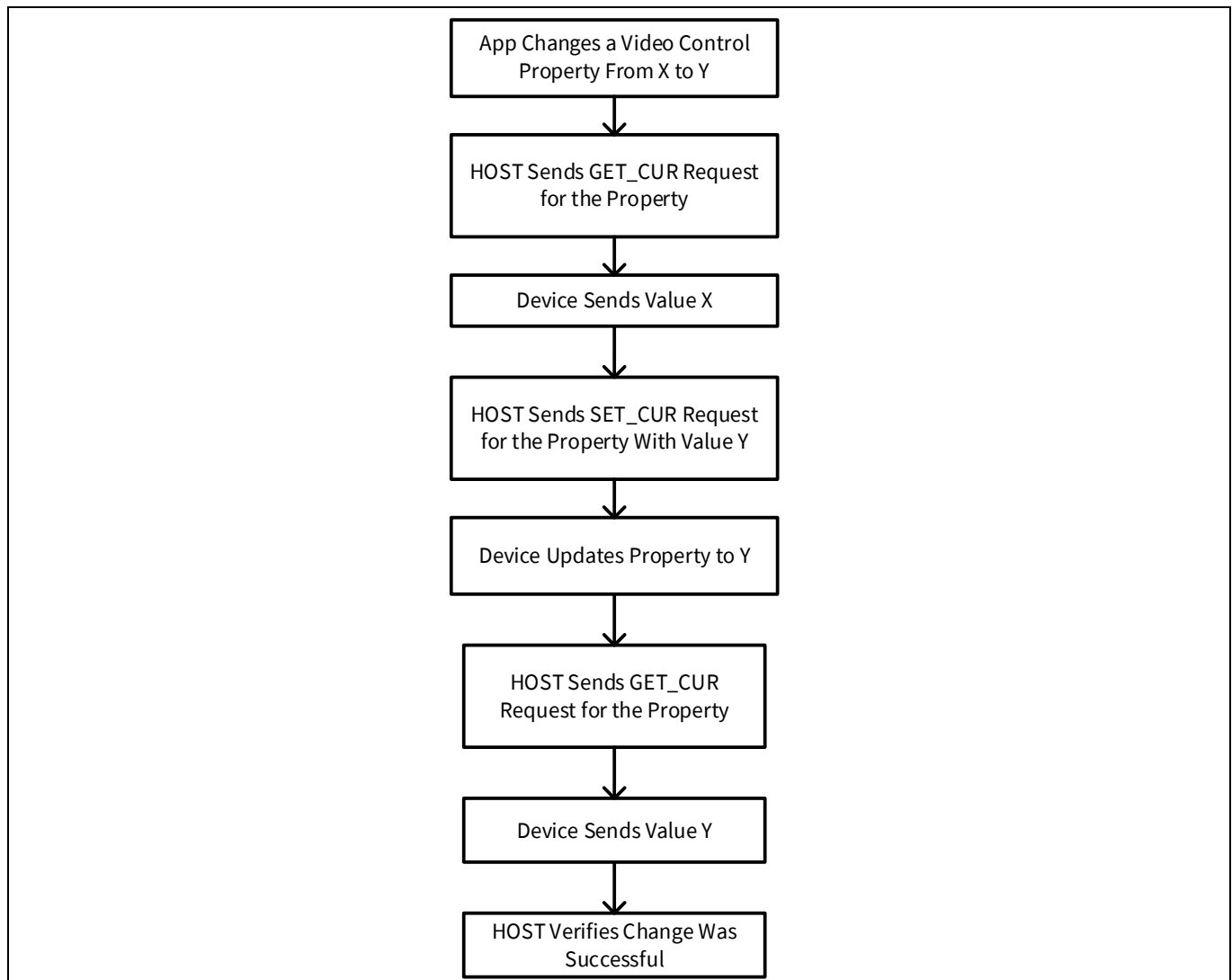


Figure 16 Host application changes a camera setting

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.

USB video class (UVC)

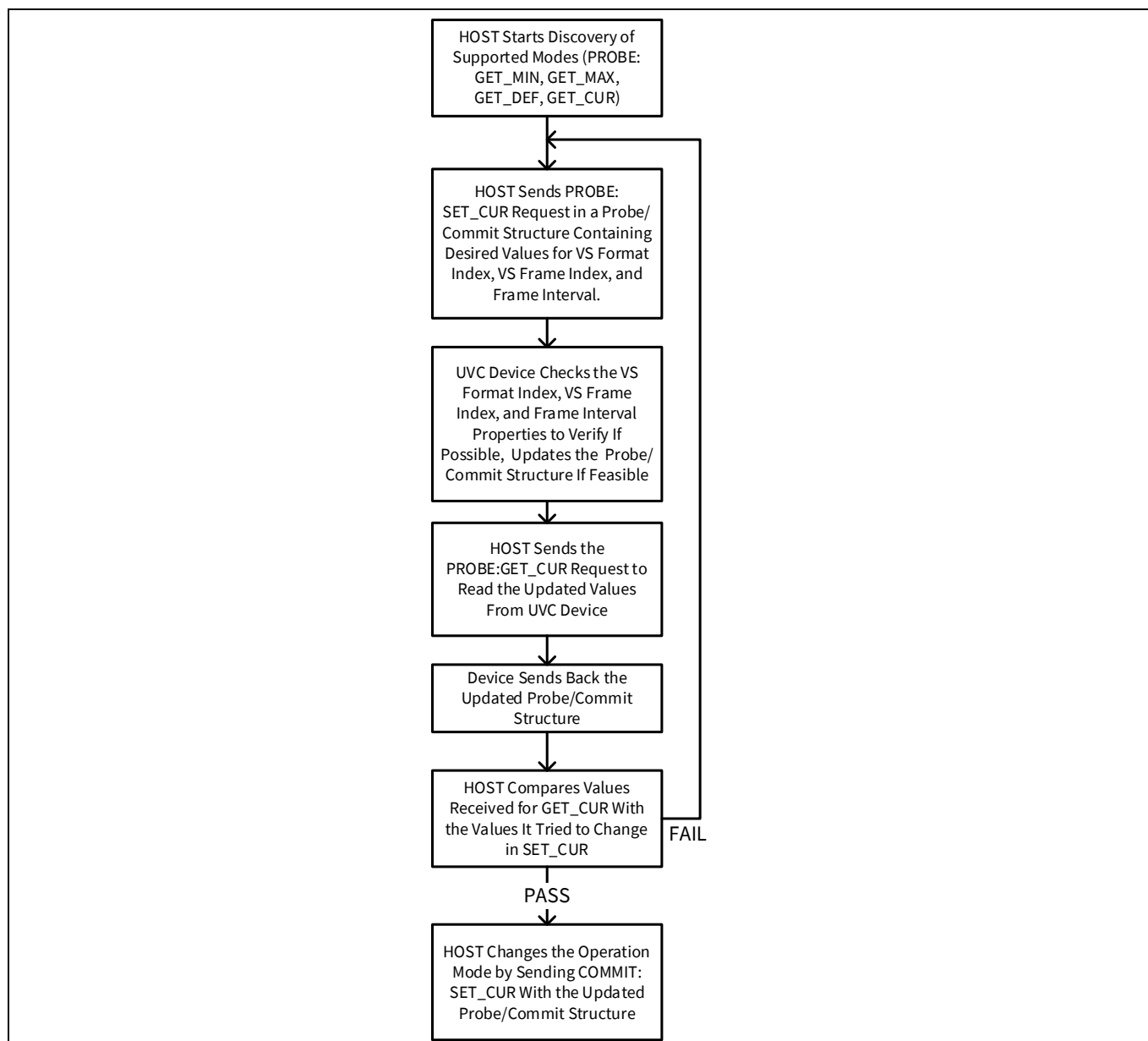


Figure 17 Host-camera prestreaming dialog

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)

USB video class (UVC)

4.3.2.1 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, see the UVC spec. referred in the [Enumeration data](#) section.

The host can now send requests directed towards the specific terminal. The different types of requests that the host can send are explained in the [UVC-specific requests](#) section. 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.

4.3.2.2 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.

4.3.2.3 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)

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

USB video class (UVC)

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. See [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.

4.3.2.4 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. See 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

USB video class (UVC)

Bit offset	Field name	Description
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 3.2 Gen 1 bulk packet size is 1024 bytes.

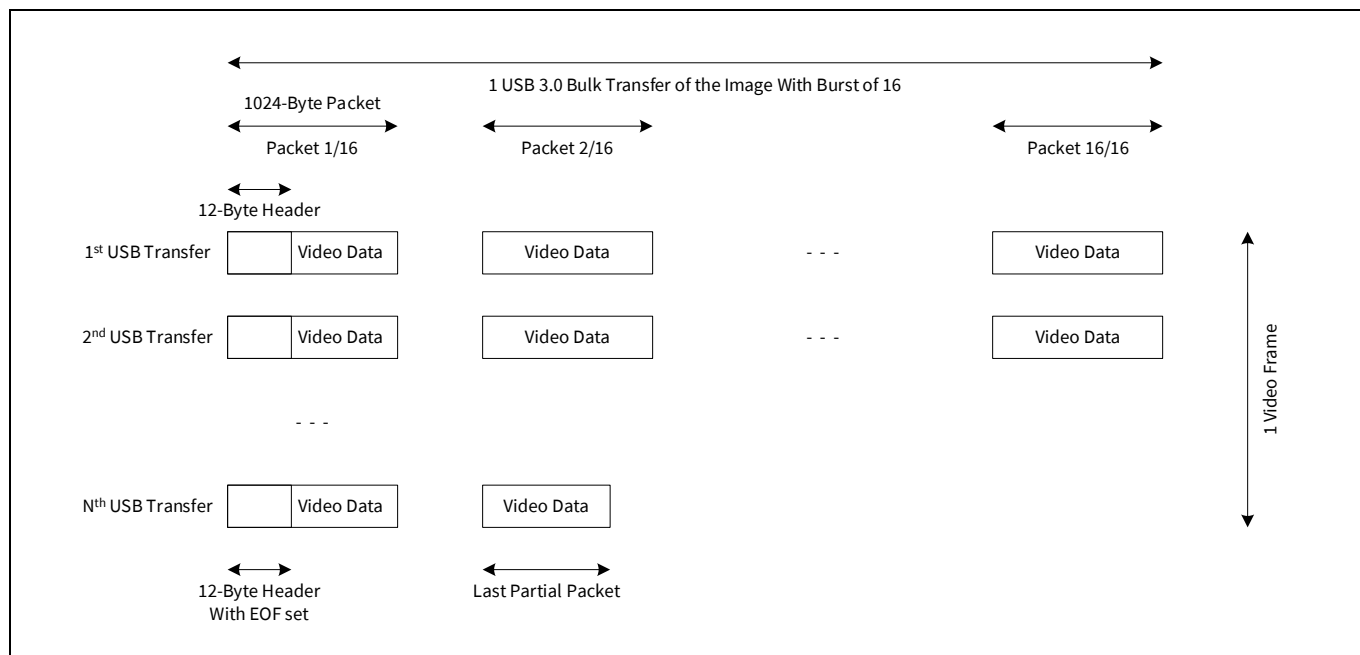


Figure 18 UVC video data transfer

CX3 MIPI Configuration Tool

5 CX3 MIPI Configuration Tool

Infineon provides a CX3 Configuration Tool for quick generation of a CX3 project. This tool is included with EZ-USB™ Suite IDE as part of the FX3 SDK. The tool is used to configure CX3's MIPI CSI-2 controller settings based on the input image sensor parameters. The tool can also generate a complete CX3 firmware project including UVC Class descriptors, handling of UVC requests, and video streaming DMA configuration on user inputs.

This section provides more details on the CX3 MIPI Configuration tool version 2.0 (included with FX3 SDK version 1.3.5).

5.1 Launching the tool

Open EZ-USB™ Suite and click on **Create New CX3 Configuration** button (see [Figure 19](#)).

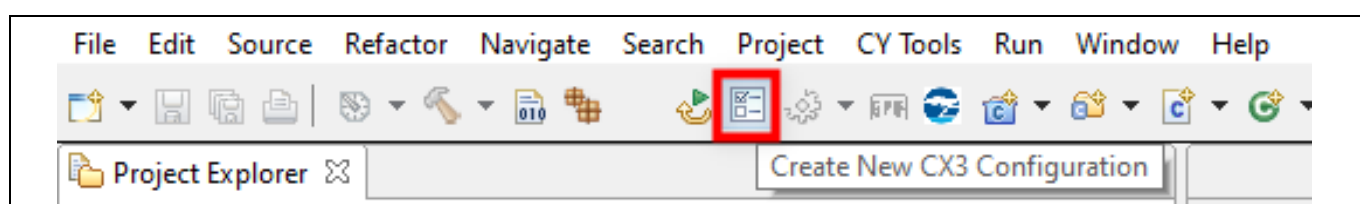


Figure 19 CX3 Configuration Tool shortcut

The **CX3 Configuration Tool v2.0** has the following options to create a new CX3 Configuration (see [Figure 20](#)):

Table 7 CX3 Configuration Tool v2.0

Option	Description
Create New MIPI Receiver Configuration Project	Select this option to create a CX3 Project.
Create a Configuration with Basic Settings	Select this option to create a CX3 Project with the basic settings.
Select a Pre Defined Configuration	Select this option to get a headstart in creating the CX3 project for any supported sensor.
Select an User Defined Configuration	You can create a sensor configuration and add to the CX3 MIPI Configuration tool using the Save as user defined configuration option in the tool.
Add Only MIPI Receiver Configuration File	Use this option if you already have a CX3 project, and only need the CX3 MIPI Receiver configuration file to be generated.

CX3 MIPI Configuration Tool

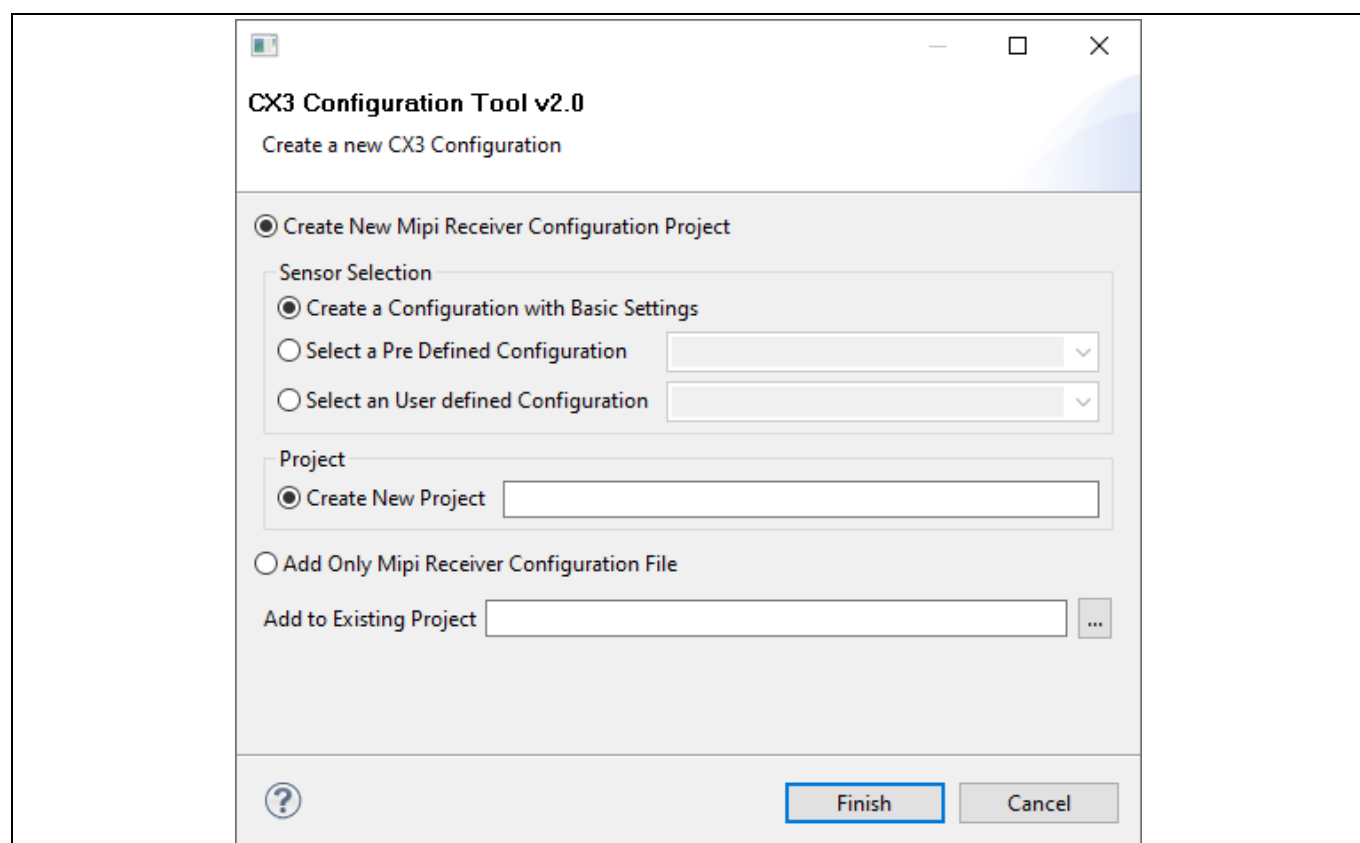


Figure 20 CX3 Configuration Tool start page



5.2 Adding sensor details to CX3 Configuration tool

The **Image Sensor Configuration** tab of the CX3 Configuration tool captures all the information related to the sensor required by the CX3’s MIPI Receiver. **Figure 21** shows various elements in this tab.

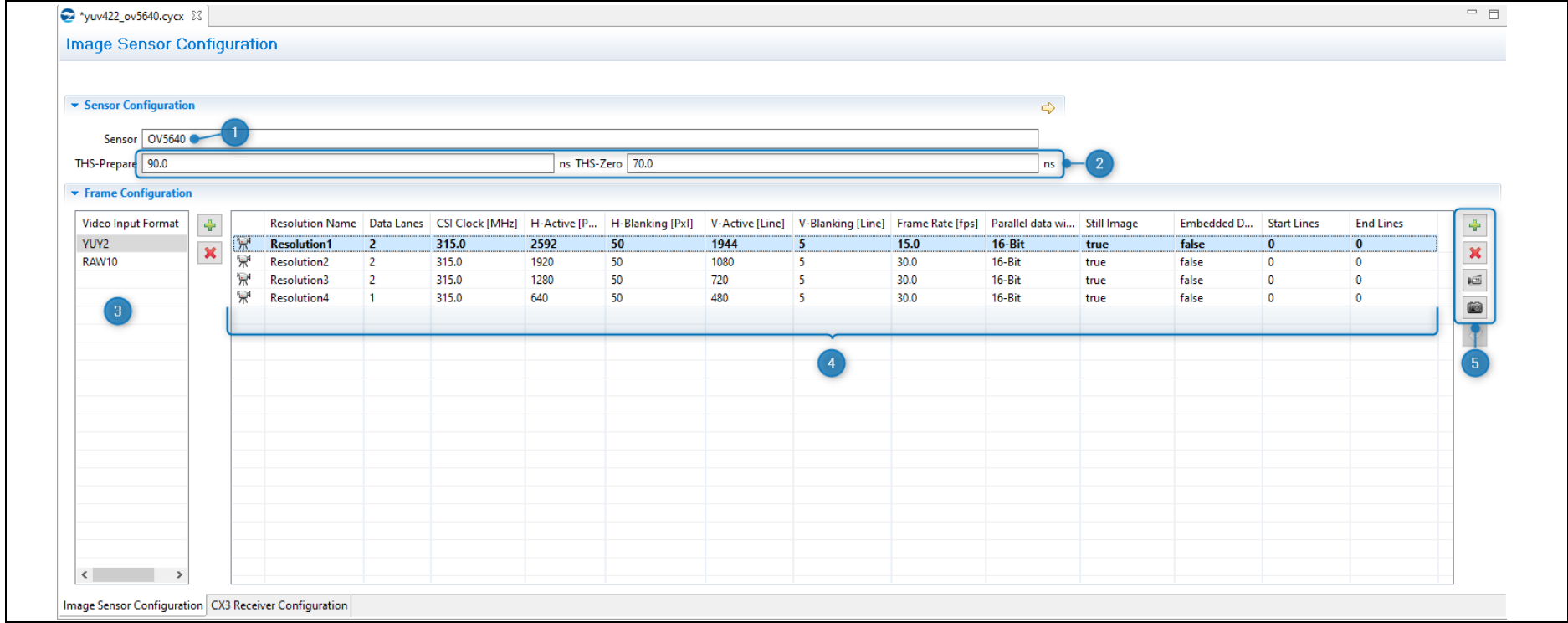


Figure 21 Sensor Configuration page of CX3 Configuration tool

CX3 MIPI Configuration Tool

The **Image Sensor Configuration** tab contains the following elements:

1. **Sensor name:** Enter the sensor name. This string will be used by the tool in place of the sensor name in the generated firmware
2. **THS-Prepare and THS-Zero:** Enter the THS-Prepare and THS-Zero values of the sensor in nano-seconds (ns) in these fields¹. The tool uses these values to configure the THS-Settle or the PHY Time Delay parameter of CX3's MIPI CSI-2 Receiver
3. **Video Input formats:** Choose from the input video formats list for the CX3 configuration

CX3 tool supports the following video formats:

Table 8 Video formats

Input video format	Details
YUY2	16-bits per pixel. Pixels from sensor arranged as P [15:0]. Arranged as Y1,U1,Y2, and V1
UYVY	16-bits per pixel. Pixels from sensor arranged as P [15:0]. Arranged as U1,Y1,V1, and Y2
YVYU	16-bits per pixel. Pixels from sensor arranged as P [15:0]. Arranged as Y1,V1,Y2, and U1
YUV422_10	10-bits per pixel. Pixels from sensor arranged as P [9:0]
RAW8	8-bits per pixel
RAW10	10-bits per pixel
RAW12	12-bits per pixel
RAW14	14-bits per pixel
RGB565_MODE0	24-bits per pixel Pixels from sensor arranged as P {2'b0, R[4:0], 3'b0, G[5:0], 2'b0, B[4:0], 1'b0}
RGB565_MODE1	24-bits per pixel Pixels from sensor arranged as P {3'b0, R[4:0], 2'b0, G[5:0], 3'b0, B[4:0]}
RGB565	16-bits per pixel. Pixels from sensor arranged as P {R[4:0], G[5:0], B[4:0]}
RGB666_MODE_0	24-bits per pixel Pixels from sensor arranged as P {2'b0, R[5:0], 2'b0, G[5:0], 2'b0, B[5:0]}
RGB666_MODE_1	24-bits per pixel Pixels from sensor arranged as P {6'b0, R[5:0], G[5:0], B[5:0]}
RGB24	24-bits per pixel Pixels from sensor arranged as P {R[7:0], G[7:0], B[7:0]}
MJPEG	Compressed video format
YUY2_SWAP	YUY2 with byte swap in pixel order. Arranged as U1,Y1,V1,and Y2
UYVY_SWAP	UYVY with byte swap in pixel order. Arranged as U1,Y1,V1, and Y2

¹ See the sensor datasheet or contact the sensor vendor for this information. THS-Prepare and THS-Zero are standard parameters defined in the MIPI D-PHY specification.

CX3 MIPI Configuration Tool

4. [Table 9](#) lists the sensor parameters.

Table 9 Sensor parameters





Parameter	Details
Data Lanes	Number of MIPI data lanes for the resolution
CSI Clock (MHz)	MIPI Clock frequency For example, if the data rate per lane is given as 600 Mbps, the CSI clock required is 300 MHz. This is because the MIPI clock is considered to be in DDR mode during the data transmission
H-Active	Resolution Horizontal Width
H-Blanking	Blanking pixels for each line
V-Active	Resolution Vertical Height
V-Blanking	Blanking lines per frame
Frame Rate	Number of frames per second (fps)
Parallel data width	The parallel bus width. CX3 supports 8, 16, and 24-bit parallel bus. It also supports packing and zero-padding pixel data. See Mapping input data format to output video format based on parallel data width selection for more details
Still Image	Enable still capture support
Embedded data	Reserve space for embedded data from sensor
Start lines	Number of embedded data lines at start of the frame
End lines	Number of embedded data lines at end of the frame

Note: Obtain the image sensor configuration settings from the image sensor vendor.

Note: The resolution width or line size (in bytes) should be a multiple of 4 bytes. If this condition is not met, the CSI-2 MIPI controller block of CX3 will add additional padding bytes to the line data.

5. Buttons

Table 10 Buttons

Buttons	Details
	Add a video format or a video resolution
	Delete a video format or a video resolution
	Select the current resolution as the optimum video resolution. This will be set as the optimum resolution in the UVC Format descriptor
	Select the current resolution as the optimum still resolution. This will be set as the optimum resolution in the UVC Still Capture descriptor

CX3 MIPI Configuration Tool

5.2.1 Mapping input data format to output video format based on parallel data width selection

CX3's CSI-2 controller provides an option to pack multiple pixels or zero-pad each pixel on its parallel output. Packing of pixels is useful to make maximum use of CX3's 24-bit parallel bus width. Padding is used to append zeros to the pixel data to match the bus width per pixel.

For example:

Consider an input video resolution of 1920×1080 (Hactive \times Vactive) with input video format RAW8. Each pixel is 8 bits in size. This results in an input frame size of Hactive \times Vactive \times 8 bits.

RAW8 is mapped to YUY2 output format (16-bits per pixel) by the CX3 tool so that the image output can be viewed on any standard UVC application.

The following are the options available for parallel bus width on CX3 tool for this format:

If you select...	Output
8-bit	CX3 outputs 8-bits per clock The Hactive should be adjusted as $Hactive \times 0.5$ to match the input frame size from sensor and output frame size from CX3 because the output pixel size is 16 bits (YUY2). The resulting resolution is $(1920/2) \times 1080 \times 16$
16-bit pack	CX3 outputs two pixels of 8-bits each per clock. The Hactive should be adjusted as $Hactive \times 0.5$ to match the input frame size from sensor and output frame size from CX3 because the output pixel size is 16 bits (YUY2). The resulting resolution is $(1920/2) \times 1080 \times 16$
16-bit zero pad	CX3 outputs one pixel of 16-bits size per clock (8-bits of pixel data and 8 bits of zero). In this case, resolution adjustment is not done because the output and input pixel size are 16 bits (YUY2). The resulting resolution is $1920 \times 1080 \times 16$
24-bit pack	CX3 outputs three pixels of 8-bits each per clock. The Hactive should be adjusted as $Hactive \times 0.5$ to match the input frame size from sensor and output frame size from CX3 because the output pixel size is 16 bits (YUY2). The resulting resolution is $(1920 / 2) \times 1080 \times 16$
24-bit zero pad	CX3 outputs one pixel of 24-bits size per clock (8-bits of pixel data and 16 bits of zero). The Hactive should be adjusted as $Hactive \times (24/16)$ to match the input and output frame size Because the output pixel size is 16-bits (YUY2). The resulting resolution is $(1920 \times 1.5) \times 1080 \times 16$

The mapping between various input video formats and parallel data width settings is given in [Table 11](#).

Note: *You do not need to take any action for this mapping. The CX3 Configuration Tool automatically adjusts the output video resolution based on the input video format and parallel bus width settings.*

Table 11 Mapping between various input video formats and parallel data width settings

Input Video format	Input bits per pixel	Parallel data width	Resulting bits per pixel	Input frame size	Output video format (video format in UVC descriptor)	Output bits per pixel	Output frame size	Hactive adjustment to match input and output frame size	Example: Output resolution for input of 1920 x 1080
RAW8	8	8-bit	8	Hactive × Vactive × 8	YUY2	16	Hactive × Vactive × 16	Hactive × 0.5	960 × 1080
		16-bit pack	8	Hactive × Vactive × 8				Hactive × 0.5	960 × 1080
		16-bit zero pad	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		24-bit pack	8	Hactive × Vactive × 8				Hactive × 0.5	960 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
RAW10	10	16-bit pack	10	Hactive × Vactive × 10				Hactive × 0.625	1200 × 1080
		16-bit zero pad	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		24-bit pack	10	Hactive × Vactive × 10				Hactive × 0.625	1200 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
RAW12	12	16-bit pack	12	Hactive × Vactive × 12				Hactive × 0.75	1440 × 1080
		16-bit zero pad	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		24-bit pack	12	Hactive × Vactive × 12				Hactive × 0.75	1440 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
RAW14	14	16-bit pack	14	Hactive × Vactive × 14				Hactive × 0.875	1680 × 1080
		16-bit zero pad	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		24-bit pack	14	Hactive × Vactive × 14				Hactive × 0.875	1680 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
RGB 565 mode 0	24	24-bit	24	Hactive × Vactive × 24	RGB565	16	Hactive × Vactive × 16	Hactive × 1.5	2880 × 1080
RGB 565 mode 1		24-bit	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
RGB 565	16	16-bit	16	Hactive × Vactive × 16				Hactive	1920 × 1080
RGB 666 mode 0	24	24-bit	24	Hactive × Vactive × 24	RGB888	24	Hactive × Vactive × 24	Hactive	1920 × 1080
RGB 666 mode 1		24-bit	24	Hactive × Vactive × 24				Hactive	1920 × 1080
RGB 24		24-bit	24	Hactive × Vactive × 24				Hactive	1920 × 1080
UYVY	16	16-bit	16	Hactive × Vactive × 16	UYVY	16	Hactive × Vactive × 16	Hactive	1920 × 1080

Input Video format	Input bits per pixel	Parallel data width	Resulting bits per pixel	Input frame size	Output video format (video format in UVC descriptor)	Output bits per pixel	Ouput frame size	Hactive adjustment to match input and output frame size	Example: Output resolution for input of 1920 x 1080
YUY2	16	24-bit zero pad	24	Hactive × Vactive × 24	YUY2			Hactive × 1.5	2880 × 1080
		24-bit pack	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		16-bit	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
		24-bit pack	16	Hactive × Vactive × 16				Hactive	1920 × 1080
		16-bit	16	Hactive × Vactive × 16				Hactive	1920 × 1080
YVYU	16	24-bit pack	16	Hactive × Vactive × 16	YVYU			Hactive	1920 × 1080
		24-bit zero pad	24	Hactive × Vactive × 24				Hactive × 1.5	2880 × 1080
		16-bit pad	16	Hactive × Vactive × 16				Hactive	1920 × 1080
YUV422_10	10	16-bit pad	16	Hactive × Vactive × 16	YUY2			Hactive	1920 × 1080
MJPEG	16	16-bit	16	Hactive × Vactive × 16	MJPEG			Hactive	1920 × 1080
		24-bit	16	Hactive × Vactive × 16				Hactive	1920 × 1080

Note: If you want to change this mapping or directly passthrough some formats (for example, RAW8) instead of reporting as YUY2, you can modify the `cycx3_uvcscr.c` file and use your custom UVC host application to display the received pixels.

CX3 MIPI Configuration Tool

5.3 Configuring CX3 MIPI receiver

Use the CX3 configuration tool to generate a complete CX3 project or just the MIPI receiver configuration (that can be added to an existing project).

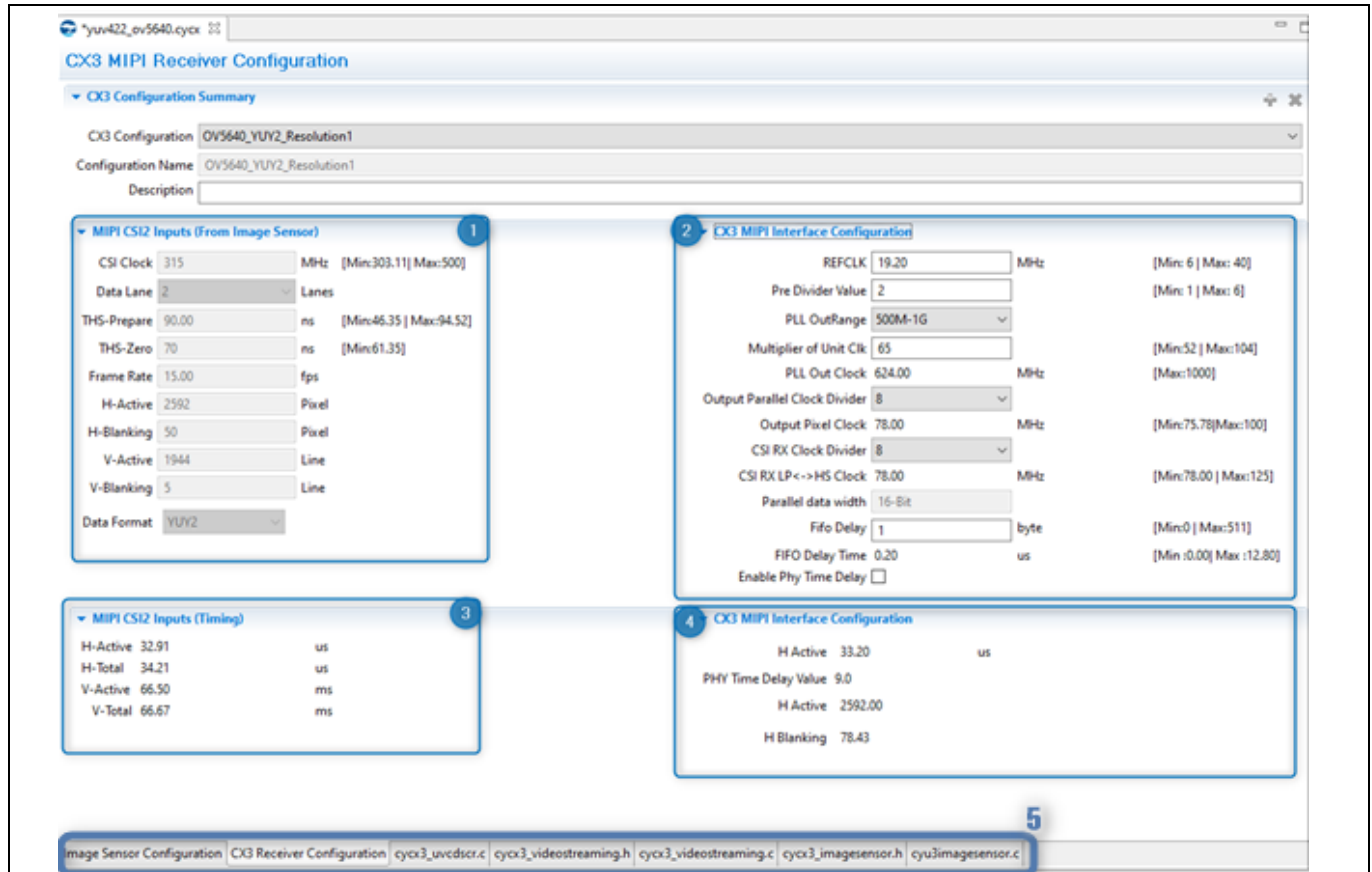


Figure 22 CX3 MIPI Receiver Configuration tool

The following are the fields in the MIPI Receiver Configuration tab (Figure 22):

1. **MIPI CSI2 Inputs (From Image Sensor):** The MIPI CSI2 Input tab parameters (such as CSI Clock, Data Lane, THS-Prepare, THS-Zero, Frame Rate, H-Active, H-Blanking, V-Active, V-Blanking, and Data Format) are auto-populated from the Image Sensor Configuration tab. These are the input parameters for the CSI-2 receiver interface.

For example, Figure 22 shows a CSI-2 sensor interface streaming YUY2 2592 x 1944 at a 315 MHz CSI clock with H-Blanking of 50 pixels and V-Blanking of 5 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.

CX3 MIPI Configuration Tool

2. **CX3 MIPI Interface Configuration:** Use CX3 MIPI Interface Configuration tab parameters (such as REFCLK, Pre Divider Value, PLL OutRange, Multiplier of Unit Clk, PLL Out Clock, Output Parallel Clock Divider, Output Pixel Clock, CSI RX Clock Divider, CSI RX LP <-> HS Clock, Parallel data width, Fifo Delay, FIFO Delay Time, and a Phy Time Delay check box) to configure the three clocks (PLL clock, CSI RX LP <-> HS clock, and parallel output pixel clock) that are used within the MIPI CSI-2 controller:

For more details on three clocks, see the following:

- “MIPI CSI-2 Block Clocks” section of the [CX3 TRM](#)
- [Analysis of CX3 clocking parameters – KBA226758](#)

The Output Pixel Clock (should be within the range specified by the tool), Data format (decides the data bus width of the parallel output interface), and FIFO Delay are the important parameters that decide the successful functioning of the MIPI controller in CX3.

See the “CX3 MIPI CSI-2 Stream Formats” section of [CX3 TRM](#) for the supported data formats and the parallel out data bus width supported by each format.

After setting the FIFO Delay parameter, it 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. [Figure 23](#) shows the FIFO delay concept. However, 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. Therefore, choose the FIFO delay carefully.

The tool uses THS-Prepare and THS-Zero value fields to calculate the PHY Time delay parameter of the CSI-2 MIPI Controller. Select the “Enable Phy Time Delay” checkbox in the CX3 MIPI Interface Configuration tab to enable or disable the API call.

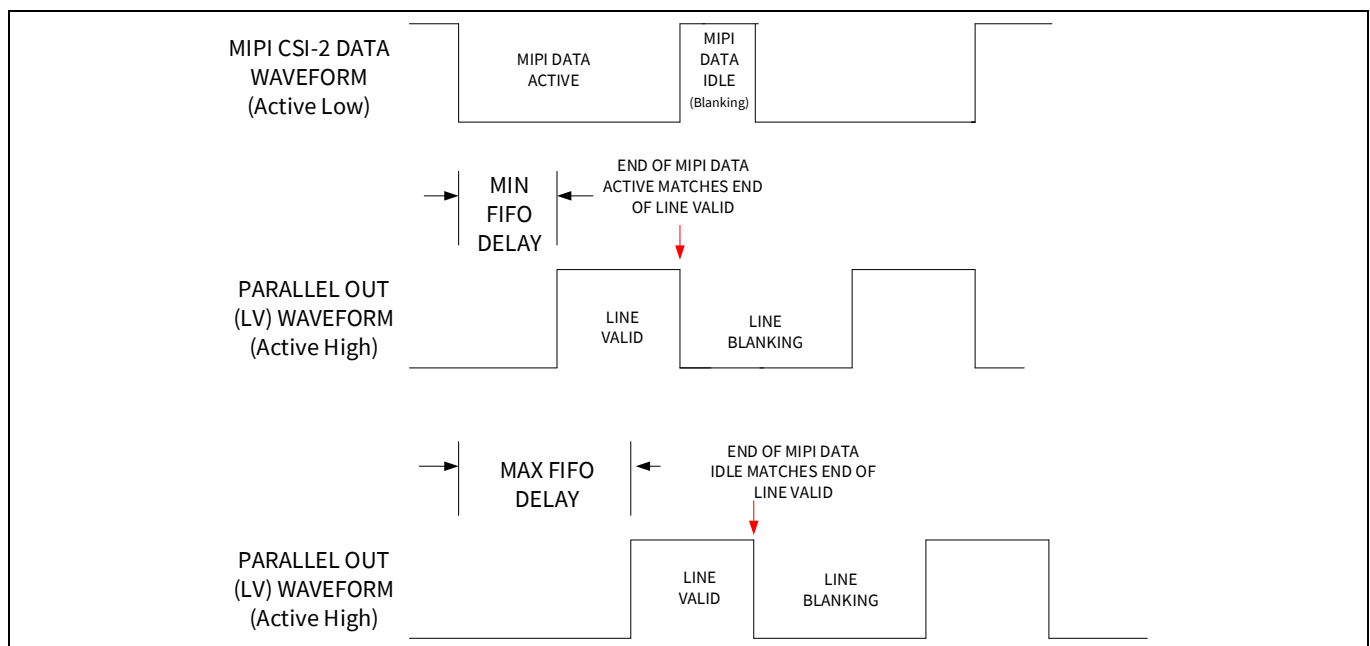


Figure 23 FIFO delay concept

CX3 MIPI Configuration Tool

3. **MIPI CSI2 Inputs (Timings):** These are the timings (such as **H-Active**, **H-Total**, **V-Active**, and **V-Total**) calculated based on CSI-2 Input parameters
4. **CX3 MIPI Interface Configuration:** These are the timings (such as **H Active**, **PHY Time Delay Value**, **H Active**, and **H-Blanking**) calculated based on MIPI CSI-2 interface configuration
5. **Save Modified files:** These files capture the source code modification based on changes done in the GUI. Click on each individual file tab and save it using (Ctrl+S)

*Note: This step is **mandatory** to ensure that the modifications are reflected in the project's original files.*

To configure 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 similarly to the YUV format, depending on the number of bytes in a pixel. For more details on streaming the RGB format, see other example projects in the [FX3 SDK](#). To stream the MJPEG format at different resolutions and to interface a CX3 device with an ISP (image signal processor), see the source code of the CX3-based [THEIA-CAM](#) from Thine Solutions. Note that you need to purchase this Thine ISP 13MP reference design kit (RDK) to get the firmware sources for the CX3-based THEIA-CAM.

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

CX3 firmware

6 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 24](#) 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 are using another sensor, or if additional functionality in OV5640 is required, the `cycx3_uvc.c` file must be modified accordingly. [Table 12](#) summarizes the code modules and the functions implemented in each module.

Table 12 Example project files

File	Description
<code>cyfctx.c</code>	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.
<code>cycx3_uvc.c</code>	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. Contains the following functions: <ul style="list-style-type: none"> • <i>main</i>: Initializes the CX3 device, sets up caches, configures the CX3 I/Os, and starts the RTOS kernel. • <i>CyFxApplicationDefine</i>: Defines the two application threads that are executed by the RTOS. • <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. • <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 <code>CyU3PMipicsiGetErrors</code>. • <i>CyCx3AppDebugInit</i>: Initializes CX3's UART block for printing debug messages • <i>CyCx3AppInit</i>: Initializes CX3's GPIO block, processor block or PIB (GPIF II is a part of PIB), I2C/CCI 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. • <i>CyCx3AppGpifCB</i>: Handles CPU interrupts generated from the GPIF II state machine

CX3 firmware

File	Description
	<ul style="list-style-type: none"> • <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. • <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. • <i>CyCx3AppUSBEventCB</i>: Handles USB events such as suspend, cable disconnect, reset, and resume. • <i>CyCx3AppErrorHandler</i>: Error handler function. This is a placeholder function for you to implement error handling if necessary. • <i>CyCx3AppAddHeader</i>: Adds a UVC header to the video data during active streaming.
<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>	Contains switches to modify the application behavior to turn ON/OFF the debug prints for frame counts and the MIPI error thread. Contains constants that are used in common by the <i>cycx3_uvc.c</i> and <i>cycx3_uvcdscr.c</i> files.
<i>cyfx_gcc_startup.s</i>	This assembly source file contains the CX3 CPU startup code. It has functions that set up the stacks and interrupt vectors. No changes needed.
<i>cyu3imagesensor.c</i>	Contains Image sensor configuration routines, structures, I2C register address and values
<i>cyu3imagesensor.h</i>	Contains declarations for the image sensor-related functions.
<i>cycx3_videostreaming.c</i>	Contains routines for image sensor video resolution and data structure for MIPI configuration
<i>cycx3_videostreaming.h</i>	Contains MIPI-related macros, UVC macros, and declarations for video resolutions
<i>cycx3gpifwaveform.h</i>	Contains data structure for GPIF configuration

CX3 firmware

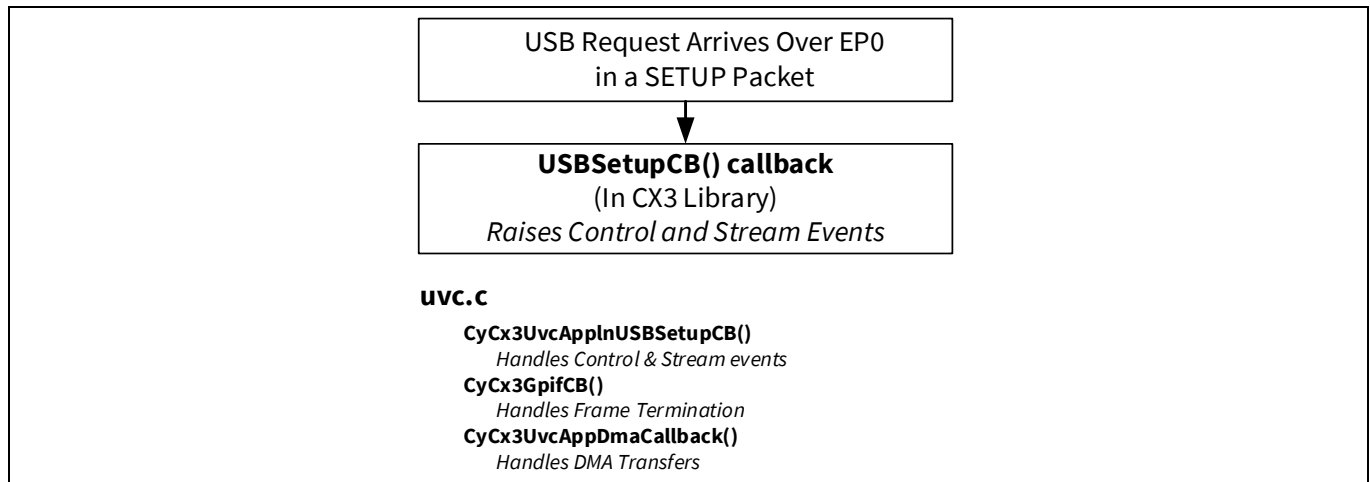


Figure 24 High-level camera project structure

6.1 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.

6.1.1 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.

6.2 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 the Troubleshooting section for more details on how to troubleshoot with MIPI CSI-2 error counters).

CX3 firmware

6.3 Handling UVC requests

The CX3 firmware handles UVC-specific control requests (SET_CUR, GET_CUR, GET_MIN, and GET_MAX described in the [UVC-specific requests](#)) 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 [Streaming requests: probe and commit control](#)), 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 [Control requests: brightness, PTZ, hue, saturation, and others](#)) 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.

6.4 Initialization

The CyCx3UvcAppThread_Entry function calls CyCx3AppDebugInit to initialize the UART debugging capability and CyCx3AppInit to initialize the rest of the required blocks, DMA channels, and USB endpoints.

6.5 Enumeration

In the CyCx3AppInit 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. See [USB descriptors for UVC](#) if you need to change these settings.

6.6 Configuring the MIPI CSI-2 controller

The CyCx3AppInit function first initializes the I²C, 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 that contains the configuration parameters. [CX3 MIPI Configuration tool generates](#) this structure.

6.7 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 `CyCx3AppInit` function. Infineon 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 I²C master block. `SensorWrite2B`, `SensorWrite`, `SensorRead2B`, and `SensorRead` functions in the `cyu3imagesensor.c` file can be used to write and read image sensor configuration over I²C.

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, see the [FX3 SDK API Guide](#).

6.8 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 3.2 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 [Selecting and switching frame settings](#) section.

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 `CyU3PMipiccsiWakeup`), and powers up the image sensor.

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

6.9 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 the [Video streaming \(VS\) interface](#) section.

When a particular resolution or frame-rate setting is chosen in the USB host application, the USB host sends a SET_CUR commit control 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.

CX3 firmware

6.10 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 non-multiple 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 16×1024 byte packets, or 16,384 bytes.

6.11 Handling DMA buffers during video streaming

The `CyCx3AppInit` 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.

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.

6.12 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. After the last DMA buffer of the frame is drained out by the USB host, the `glDmaDone` variable becomes zero.

The CX3 firmware then asserts a firmware trigger action using the `CyU3PGpifControlSWInput` API to restart its state machine to handle the next incoming frame.

In older versions of the CX3 firmware, the `CyU3PGpifSMSwitch` function is used 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.

6.13 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 (I²C) commands or by using the XSHUTDOWN pin of CX3. This pin can be controlled using the `CyU3PMipicsiSetSensorControl` function with `CY_U3P_CSI_IO_XSHUTDOWN` as the first parameter.

Hardware setup

7 Hardware setup

7.1 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 [Infineon website](#) under the **Kits** tab.

7.1.1 Kit procurement

1. Buy the RDK from [e-Con Systems](#)
2. Infineon 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 and contact [Tech Support](#). Infineon will then provide the OmniVision-specific source files after NDA verification
3. Use a USB 3.2 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.

7.2 Designing your own board

When designing your own board, a few guidelines should be followed to ensure that the board functions well. See [FX3/FX3S hardware design guidelines](#) application note to find out recommended practices for FX3/FX3S hardware design which are also applicable to CX3. Also, see [CX3 hardware FAQs KBA](#) for more details.

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

UVC-based host applications

8 UVC-based host applications

Various host applications allow you to display and capture video from a UVC device. The Windows camera application and [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 [Webcamoid](#) to stream video.

Mac platforms can use [Webcamoid](#), Photo Booth, and Debut Video Capture software to create an interface with the UVC device to stream video.

Troubleshooting

9 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. See 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 have been documented in the following KBAs:

- [KBA233853 - EZ-USB™ CX3 troubleshooting guide](#)
- [KBA91297 - CX3 Firmware: Frequently Asked Questions](#)
- [KBA91295 - CX3 Hardware: Frequently Asked Questions](#)
- [KBA91298 - CX3 Application Software / USB Driver: Frequently Asked Questions](#)

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. See [Designing your own board](#) for more details.
- **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:** USB 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.

Troubleshooting

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. See the [SDK API guide](#) for more information on the types of errors.

If the problems still persist, see similar cases in [Infineon Developer Community](#).

Summary

10 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 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 find host applications available on different platforms, including an open-source host application project
- How to troubleshoot and debug the CX3 firmware, if required

Related resources

11 Related resources

Infineon provides a wealth of data at www.infineon.com to help you to select the right USB device for the design, and to help you to integrate the device quickly and effectively into the design. For a comprehensive resources list, see the [CX3](#) webpage.

- Overview: [USB Portfolio](#)
- USB 3.0 Product Selectors: [FX3](#), [FX3S](#), [CX3](#), [HX3](#), and [SX3](#)
- Application notes: Infineon offers a large number of USB application notes covering a broad range of topics, from basic to advanced level. Recommended application notes for getting started with CX3 are:
 - [AN75705](#) - Getting Started with EZ-USB™ FX3
 - [AN75779](#) - How to Implement an Image Sensor Interface with EZ-USB™ FX3 in a USB Video Class (UVC) Framework
 - [AN76405](#) - EZ-USB™ FX3 Boot Options
 - [AN70707](#) - EZ-USB™ FX3/FX3S Hardware Design Guidelines and Schematic Checklist
 - [AN86947](#) - Optimizing USB 3.0 Throughput with EZ-USB™ FX3
 - [AN231295](#) - Getting started with EZ-USB™ SX3
- Code Examples:
 - [USB Super-Speed](#)
- Technical Reference Manual (TRM):
 - [EZ-USB™ CX3 Technical Reference Manual](#)
- Knowledge Base Articles:
 - [Analysis of CX3 Video Timing Parameters - KBA226779](#)
 - [Analysis of CX3 Clocking Parameters - KBA226758](#)
 - [CX3 Firmware: Frequently Asked Questions - KBA91297](#)
 - [CX3 Hardware: Frequently Asked Questions - KBA91295](#)
 - [CX3 Application Software / USB Driver: Frequently Asked Questions - KBA91298](#)
 - [EZ-USB™ CX3: Interfacing with the onsemi AP1302 ISP and AR1335 image sensor - KBA237926](#)
 - [EZ-USB™ CX3: Interfacing with the onsemi AR0234CS sensor - KBA236855](#)
 - [EZ-USB™ CX3 based USB3 Vision solution - KBA236151](#)
 - [Knowledge Base - Cypress Semiconductor Cage Code - KBA89258](#)
 - [Designing Type-C products based on EZ-USB™ FX3 and CX3 - KBA218460](#)
 - [Using Cypress CX3 Based USB 3 Camera Kits with Raspberry Pi 4 - KBA229301](#)
- Development Kits:
 - [EZ-USB™ CX3 THEIA-CAM - 13MP PDAF UVC Camera Solution](#)
 - [Denebola - USB 3.0 UVC Reference Design Kit \(RDK\)](#)
- Models:
 - [CX3 Device OrCad Schematic Symbol](#)
 - [CYUSB306x - IBIS](#)

Revision history

Revision history

Document revision	Date	Description of changes
**	2014-04-08	Initial release
*A	2014-11-04	Edited the document based on internal review.
*B	2015-01-07	Updated the document title
*C	2015-08-04	<ul style="list-style-type: none">Updated the AN to reflect the changes in the CX3 Configuration toolTemplate update
*D	2017-05-10	<ul style="list-style-type: none">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
*E	2022-04-01	Migrated to Infineon template
*F	2023-07-28	<ul style="list-style-type: none">Added details on CX3 Configuration Tool v2.0Updated related resources section
*G	2025-11-10	<ul style="list-style-type: none">Updated Figure 22 in Section 5.3Section 6: Updated files in Table 12Template update

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-11-10

Published by

Infineon Technologies AG
81726 Munich, Germany

© 2025 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

001-90369 Rev. *G

Important Notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability.

Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury. However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer.

Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG).

If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited. For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit <https://www.infineon.com>