

如何使用 EZ-USB® FX3™ 在 USB 视频类别（UVC）框架内实现图像传感器接口

作者：**Karnik Shah**

关联项目：有

软件版本：**FX3 SDK1.2.3**

相关应用笔记：**AN75705**

USB 3.0 提供的高带宽对于将外设连接至 USB 的各个 IC 提出很高的要求。本应用笔记重点介绍了一种 USB 3.0 的流行应用：摄像头（即连接到 EZ-USB® FX3™ 的图像传感器）将非压缩数据输送给 PC。应用笔记突出介绍了特定设计 FX3 的特性，以便能够最大化数据吞吐量而不影响接口的灵活性。本应用笔记也提供了有关 USB 视频类别（UVC）的实施细节。符合该类别的摄像设备能够使用内置的 PC 驱动程序和主机应用程序（如 AMCap 和 VLC 媒体播放器）进行运行。最后，本应用笔记还指出了如何使用 FX3 中灵活的图像传感器接口连接两个图像传感器，以便实现三维（3D）图像和移动跟踪应用。

目录

| | | | |
|---|----|-------------------------|----|
| 1. 简介 | 2 | 6. 硬件设置 | 34 |
| 2. USB 视频类别（UVC） | 3 | 6.1 硬件采购 | 34 |
| 2.1 枚举数据 | 3 | 6.2 FX3 DVK 板设置 | 34 |
| 2.2 工作代码 | 3 | 7. 基于 UVC 的主机应用 | 35 |
| 2.3 USB 视频类别要求 | 3 | 7.1 运行演示 | 35 |
| 3. GPIF II 图像传感器接口 | 9 | 8. 故障排除 | 36 |
| 3.1 图像传感器接口 | 9 | 9. 连接两个图像传感器 | 36 |
| 3.2 图像传感器接口的引脚映射情况 | 10 | 9.1 使用 UVC 传输交错图像 | 37 |
| 3.3 乒乓 DMA 缓冲区 | 10 | 9.2 固件修改检查表 | 38 |
| 3.4 设计策略 | 12 | 10. 总结 | 39 |
| 3.5 GPIF II 状态机 | 12 | 11. 关于作者 | 39 |
| 3.6 使用 GPIF II Designer 实现图像传感器接口 | 14 | 12. 文档修订记录 | 40 |
| 4. 设置 DMA 系统 | 23 | 全球销售和 design 支持 | 41 |
| 4.1 DMA 缓冲区的相关内容 | 26 | 产品 | 41 |
| 5. FX3 固件 | 27 | PSoC® 解决方案 | 41 |
| 5.1 应用线程 | 29 | | |
| 5.2 初始化 | 29 | | |
| 5.3 枚举 | 29 | | |
| 5.4 使用 I ² C 接口配置图像传感器 | 29 | | |
| 5.5 启动视频流 | 29 | | |
| 5.6 设置 DMA 缓冲区 | 29 | | |
| 5.7 在视频串流期间中处理 DMA 缓冲区 | 30 | | |
| 5.8 帧结束时进行清理 | 30 | | |
| 5.9 终止视频流 | 30 | | |
| 5.10 增加“调试”接口 | 30 | | |

1. 简介

USB 3.0 提供的高带宽对用于将外设连接至 USB 的各个 IC 提出很高的要求。一个流行的示例就是摄像头将非压缩数据输送到 PC 中。在本应用笔记中，通过使用赛普拉斯的 EZ-USB® FX3™ 芯片可以实现转换器的功能，该转换器的一端连接到图像传感器，另一端则通过 USB 3.0 连接至主机 PC。FX3 使用它的第二代通用可编程接口（GPIF II）来提供图像传感器接口，并通过其超速 USB 单元连接至 PC。FX3 固件将来自图像传感器的数据转换为符合 USB 视频类（UVC）的格式。符合此类别的摄像设备能够使用 OS 内置的驱动程序进行操作，使摄像头与主机应用（如 AMCap 和 VLC 媒体播放器）相互兼容。

图 1. 摄像头应用

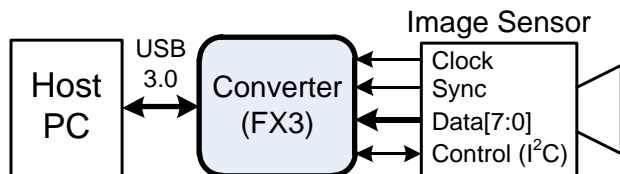
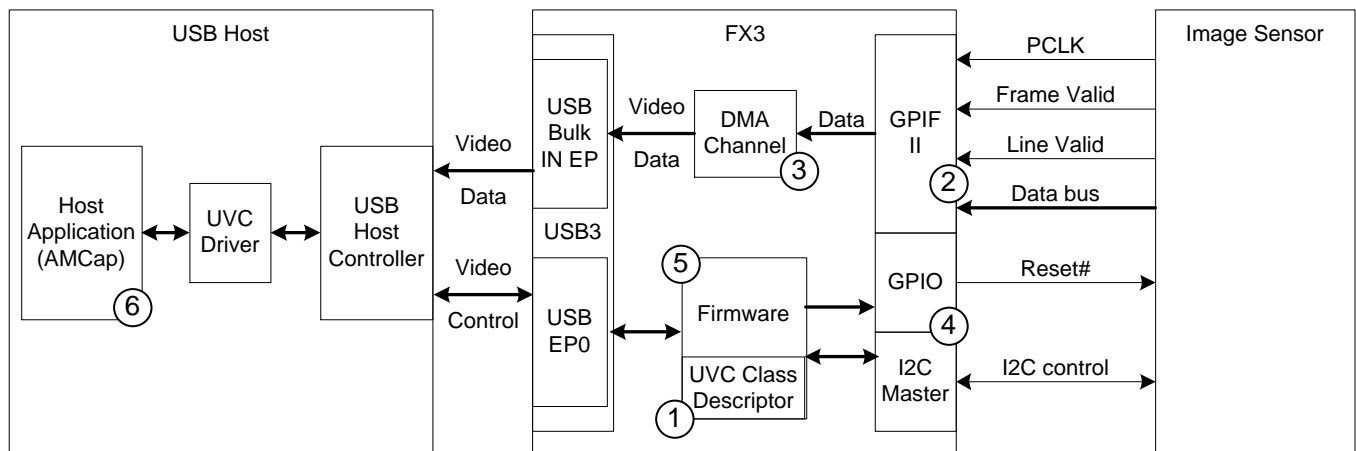


图 2. 系统框图



1. 提供正确的 USB 描述符，保证主机能够识别符合 UVC 的外围设备。有关详细信息，请参考第 2 节中的内容。
2. 实现连接到图像传感器的并向总线连接可通过使用 FX3 GPIF II 接口实现该操作。通过使用名称为 GPIF II Designer（GPIF II 设计程序）的赛普拉斯工具，可以在图形状态机编辑器中自定义设计波形。由于该接口是可编程的，因此稍作更改也能将其自定义为其他图像传感器，例如并向总线可配置为 16 位数据总线。有关详细信息，请参考第 3 节中的内容。

图 1 描述的是摄像头应用。左侧是一个带有超速 USB 3.0 端口的 PC，而右侧是一个具有下面各项特性的图像传感器：

- 8 位同步并行数据接口
- 每个像素为 16 位
- YUY2 颜色空间
- 分辨率为 1280 x 720 个像素（720p）
- 30 帧每秒
- 帧/行有效信号均为高电平有效
- 正向时钟边沿极性

即使应用笔记讨论了一个特定的图像传感器接口，上面所述的是多种图像传感器接口的通用特性，即使可能有轻微差异，如数据总线宽度和信号极性。基于 GPIF II 模块的可编程性质，这些变化很容易作出调节适应。此外，FX3 会使用其 I²C 接口实现控制总线，用作配置图像传感器。

图 2 显示的是系统框图更加详细的信息。为框图的主要子模块进行编号，这些任务由下述的各个子模块执行：

3. 构建一个 DMA，它会将来自 GPIF II 模块上图像传感器的数据转移到 USB 接口模块（UIB）内。在本应用中，必须将头数据添加到图像传感器内的视频数据中，以符合 UVC 规格的要求。因此应配置 DMA，以允许 CPU 将所需的头数据添加到 DMA 缓冲区内。该通道必须设计使最大带宽满足将视频从图像传感器输送到 PC 的要求。有关详细信息，请参考第 4 节中的内容。
4. 使用 FX3 I²C 主控来控制图像传感器的总线。通过赛普拉斯的标准库调用可对 I²C 和 GPIO 单元进行编程，有关内容在第 5.4 节中进行了介绍。

5. FX3 固件初始化 FX3 的硬件模块 (第 5.2 节)，枚举成为一个 UVC 摄像头 (第 2.3.1 节)，处理 UVC 特定请求 (第 2.3.2 节)，通过 I²C 接口将视频控制设置 (如亮度) 传输给图像传感器 (第 5.4 节)，将 UVC 头数据添加到视频数据流 (第 2.3.4 节)，将带有头数据的视频数据提交给 USB (第 5.7 节)，以及维持 GPIF II 状态机 (第 5.8 节)。
6. 主机应用 (如 AMCap 或 VLC 媒体播放器) 将存取 UVC 驱动程序，以通过 UVC 控制接口配置图像传感器，并通过 UVC 流接口接收视频数据。欲了解有关基于 UVC 的主机应用的详情，请参考第 7 节。

如果将摄像头插入 USB 2.0 端口，FX3 固件会使用 I²C 控制总线将帧率从 30 FPS 下降到 15 FPS，并将帧大小从 1280 x 720 个像素降低到 640 x 480 个像素，以匹配较低的 USB 带宽。PC 主机可以选用控制接口，将亮度、平移、倾斜和缩放等调整内容传输给摄像头。一般可同时实现平移、倾斜和缩放，并被称为 PTZ。

2. USB 视频类别 (UVC)

要符合 UVC，则需要下面两个 FX3 代码模块：

1. 枚举数据
2. 工作代码

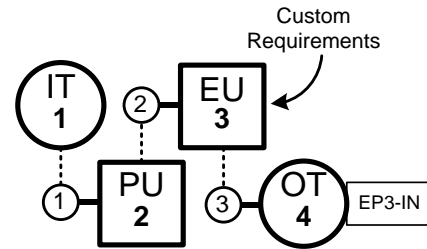
2.1 枚举数据

应用笔记附带的代码包括名称为 `cyfxuvcdscr.c` 的文件 (在第 2.3.1 节中说明)，该文件包含 UVC 的枚举数据。可从 usb.org 上获取用于定义 UVC 描述符的 USB 规范。本节介绍了高阶观点的描述符。UVC 器件具有以下四个逻辑元素：

1. 输入摄像头终端 (IT)
2. 输出端 (OT)
3. 处理单元 (PU)
4. 扩展单元 (EU)

这些元素在描述符中互相连接，如图 3 所示。通过将描述符中的各终端编号连接起来，可连接上述各元素。例如，输入 (摄像头) 终端描述符声明了其 ID 为 1；处理单元描述符的输入连接的 ID 为 1，所以逻辑上表示他连接至输入端。输出端描述符制定使用哪个 USB 端点，在本情况下是 BULK-IN 端点 3。

图 3. 摄像头架构的 UVC 图



描述符还包括了视频属性 (如宽度、高度、帧率、帧大小和位深度) 以及控制属性 (如亮度、曝光、增益、对比度和 PTZ) 等内容。

2.2 工作代码

主机枚举摄像头后，UVC 驱动程序会向摄像头发出一系列请求，以确定工作特征。该操作叫做能力请求阶段。请求阶段先于串流阶段，其中主机应用会启动串流视频。FX3 固件会响应到达 USB 控制端点 (EP0) 的请求。

例如，假设 UVC 器件表明它支持某个 USB 描述符中的亮度控制。在能力请求阶段，UVC 驱动程序查询器件，以发现相关的亮度参数。

当主机发出一个请求要求更改亮度值时，UVC 驱动程序将发出 ‘SET’ (设置) 控制请求，用以更改亮度值 (SET_CUR)

同样，当主机应用需要选择输送一个受支持的视频格式/帧率/帧大小时，它会发出输送请求。共有两种类别的请求：PROBE 和 COMMIT。PROBE 请求用于确定 UVC 器件是否准备好接受串流模式的切换。串流模式是图像格式、帧大小和帧率的组合。

2.3 USB 视频类别要求

本应用笔记的固件项目位于名称为 **USBVideoClass** 的文件夹内。本节介绍了示例项目如何满足 UVC 的要求。UVC 要求设备执行下列任务：

- 使用 UVC 特定的 USB 描述符进行枚举
- 处理 USB 描述符所记录的 UVC 控制和输送能力的 UVC 特定 SET/GET 请求
- 以符合 UVC 的颜色格式进行输送视频数据
- 添加一个符合 UVC 的标头到每个图形载荷内

可在 [UVC 规范](#) 中了解这些要求的详情。

2.3.1 UVC 的 USB 描述符

`cyfxuvcdscr.c` 文件含有 USB 描述符的表格。字节阵列 “CyFxUSBHSCfgDscr” (高速) 和 “CyFxUSBSSCfgDscr” (超速) 包含了 UVC 特定的描述符。这些描述符执行下面各子描述符树：

配置描述符

- 接口关联描述符
- 视频控制（VC）接口描述符
 - VC 接口标头描述符
 - 输入（摄像头）终端描述符
 - 处理单元描述符
 - 扩展单元描述符
 - 输出端描述符
 - VC 状态中断端点描述符
- 视频流（VS）接口描述符
 - VS 接口输入标头描述符
 - VS 格式描述符
 - VS 帧描述符
- BULK-IN 视频端点描述符

配置描述符是一个标准的 USB 描述符，它定义了 USB 设备在其子描述符中的功能。接口关联描述符用于向主机表明该设备是否属于标准 USB 类别。此处，该描述符使用了下面两个接口来上报符合 UVC 的设备：视频控制（VC）接口和视频流（VS）接口。因为 UVC 器件具有两个独立的接口，因此它是一个 USB 复合器件。

2.3.1.1 视频控制接口

VC 接口描述符及其子描述符将上报所有与控制接口有关的能力。示例包括亮度、对比度、色调、曝光和 PTZ 等控制。

VC 接口标头描述符是一个 UVC 特定的接口描述符，它表明了该 VC 接口属于哪一个 VS 接口。

输入（摄像头）终端描述符、处理单元描述符、扩展单元描述符以及输出端描述符均包含了各个位字段，这些位字段说明了相应终端或单元所支持的特性。

摄像头终端控制着机械（或等效数字）特性，比如传输视频流的设备的曝光和 PTZ。

处理单元控制着图像的各项属性，如正在流过它的视频的亮度、对比度以及色调。

与标准的 USB 供应商要求相似，扩展单元可以添加供应商指定的特性。在该设计中，扩展单元是空白的，但仍包含了该描述符，作为自定义功能的占位符。请注意，如果利用可改扩展单元，则标准主机应用将无法认出其特性，除非修改主机应用使其能识别这些特性。

输出端用于描述这些单元（IT、PU、EU）和主机之间的接口。VC 状态中断端点描述符是一个用于中断端点的标准 USB 描述符。该端点可用于传输 UVC 特定的状态信息。该端点的此功能属于本应用笔记范围外的内容。

UVC 规范已将这些功能分好了类，您可轻松安排实现该类别指定的控制请求。但实现上述功能是应用特定的。通过将相应的功能位设置为‘1’，可以在各自终端/单元描述符的位字段“bmControls”（`cyfxuvcdscr.c`）中记录所支持的控制功能。在枚举时，UVC 器件的驱动程序会轮询控制的详细信息。通过 EP0 请求实现轮询的细节。所有相似的请求（包括视频流请求）均由 `uvc.c` 文件中的 `UVCAppEP0Thread_Entry` 函数处理。

2.3.1.2 视频流接口

视频流接口描述符以及它的子描述符记录了各种帧格式（如非压缩、MPEG、H.264）、帧的分辨率（宽度、高度和位深度）以及帧率。根据所记录的值，主机应用可以选择通过改变所支持的帧格式、帧分辨率和帧率的组合来切换串流模式。

VS 接口的输入标头描述符指定了下面 VS 格式描述符的数量。

VS 格式描述符包含了图像的长宽比和颜色格式，如非压缩或压缩格式。

VS 帧描述符包含了图像的分辨率以及该分辨率的帧率。如果摄像头支持不同的分辨率，则多个 VS 帧描述符会遵循 VS 格式描述符。

BULK-IN 视频端点描述符是一个标准的 USB 端点描述符，它包含的有关批量端点可用于输送视频的信息。

本示例中使用单一的分辨率和帧率。其图像的特性包含在三个描述符中，如下面三个表格所示的内容（只有相关的字节偏移量显示）。

表 1. VS 格式描述符的值

| VS 格式描述符字节偏移量 | 特性 | 超速值 | 高速值 |
|---------------|-----|------|-----|
| 23-24 | 长宽比 | 16:9 | 4:3 |

表 2. VS 帧描述符的值

| VS 帧描述符字节偏移量 | 特性 | 超速值 | 高速值 |
|----------------|-----------------|--------------------------|-------------------------|
| 5-8 | 分辨率 (宽, 高) | 1280x720 | 640x480 |
| 17-20 | 图像的最大尺寸, 单位为字节 | 1280x720x2 (2 个字节/像素) | 640x480x2 (2 个字节/像素) |
| 21-24, 或 26-29 | 帧间隔, 单位为 100 ns | 0x51615 (30 FPS) | 0xA2C2A (15 FPS) |

请注意，多字节数值会先列出最低有效位（LSB）（低位优先），因此，例如帧率为 0x00051615，即 33.33 毫秒，或 30 FPS。

表 3. 探针/提交结构的值

| 探针/提交结构的字节偏移量 | 特性 | 超速值 | 高速值 |
|---------------|----------------|--------------------------|-------------------------|
| 2 | 格式索引 | 1 | 1 |
| 3 | 帧索引 | 1 | 1 |
| 4-7 | 帧间隔，单位为 100 ns | 0x51615 (30 FPS) | 0xA2C2A (15 FPS) |
| 18-21 | 图像的最大尺寸，单位为字节 | 1280x720x2 (2 个字节/像素) | 640x480x2 (2 个字节/像素) |

可以修改该设计，通过更改上述三个表格中的输入项来支持不同的图像分辨率，比如 1080 p。

2.3.2 UVC 特定的请求

UVC 规范使用了 USB 控制端点 EP0，使之能够向 UVC 器件进行控制和传输所请求的通信。这些请求用于发现并更改与视频相关的控制属性。UVC 规范将这些与视频相关的控制内容定义为性能。通过这些性能您可以更改图像属性或输送视频。性能（第一项内容）可以是视频控制属性（如亮度、对比度和色调），也可以是视频串流模式的属性（如颜色格式、帧大小和帧率）。通过 USB 配置描述符的 USB 特定部分，可以记录其性能。每个性能都带有其属性。各项性能的属性如下所述：

- 最小值
- 最大值
- 最小值和最大值之间值的数量。
- 默认值
- 当前值

SET 和 GET 是两类 UVC 特定请求。SET 用于更改属性的当前值，而 GET 则用于读取属性。

下面是 UVC 特定请求的列表：

- SET_CUR 是 SET 唯一请求的类型
- GET_CUR 用于读取当前值
- GET_MIN 用于读取支持的最小值。
- GET_MAX 用于读取支持的最大值。
- GET_RES 用于读取分辨率（步长值表示最小值和最大值之间受支持的数值）

- GET_DEF 用于读取默认值
- GET_LEN 用于读取属性大小，单位为字节
- GET_INFO 用于查询特定性能的状态/支持情况。

针对已给的属性，UVC 规范将上述请求定义为强制或可选的。例如，如果 SET_CUR 请求对于一种特殊性能是可选的，则通过 GET_INFO 请求可确定它是否存在。如果摄像头不支持性能的某个请求，在主机向摄像头发出请求时必须通过搁置控制端点来指出这一点。

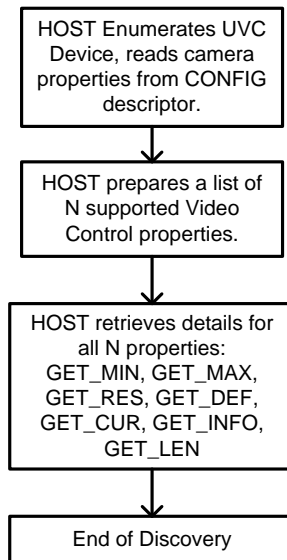
这些请求中的字节字段符合其目标性能。这些字节字段分层次，它具有与第 2.3.1 节中所述的 UVC 特定描述符相同的结构。第一层用以识别接口（视频控制还是视频流）

如果第一层确认该接口是视频控制，则第二层将确认终端或者单元，并且第三层确认该终端或单元的性能。例如，如果目标性能为亮度控制，则：

- 第一层用于视频控制
- 第二层用于处理单元
- 第三层用于亮度控制

如果第一层确认该接口为视频流，则第二层将为 PROBE（探针）或 COMMIT（提交）。这样便不存在第三层。当主机需要 UVC 器件启动输送或更改串流模式时，主机会先确定器件是否支持新的串流模式。要确定该项，主机会发出一系列的 SET 和 GET 请求，并将第二层设为 PROBE。器件可接受或拒绝对串流模式进行的更改。如果器件接受更改请求，则主机将通过发出 SET_CUR 请求并把第二层设为 COMMIT 进行确认。图 6 显示的是主机和器件间的互动。下面三个流程图显示的是主机与 UVC 器件间的互动情况。

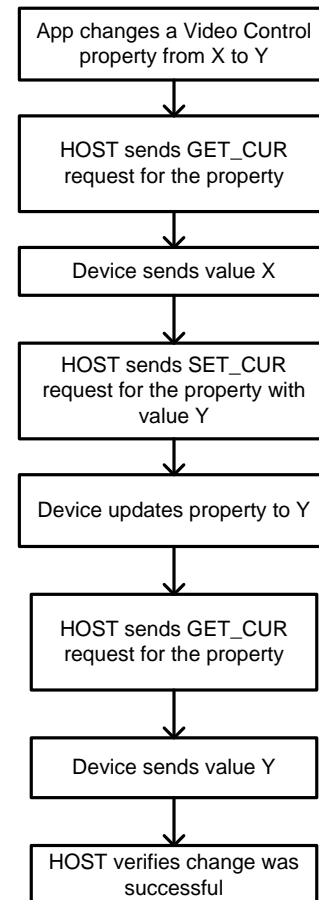
图 4. UVC 枚举以及探索流程



当 UVC 插入到 USB 时，主机将枚举它，并发现摄像头支持的属性的细节（图 4）。

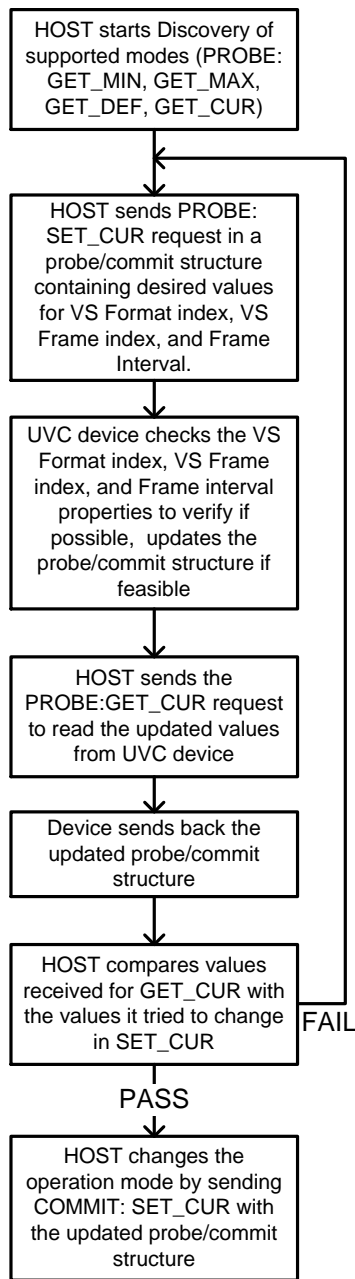
在视频运行期间，摄像头的操作者可能会在主机应用所提供的显示对话框中更改摄像头的属性（如亮度）。图 5 显示了此种交互情况。

图 5. 主机应用更改了摄像头的设置



进行输送前，主机应用将发出一些探针请求，用以发现可能的串流模式。确定了默认串流模式后，UVC 驱动程序将发出 COMMIT 请求。图 6 显示了该过程。在该端点结束，UVC 驱动程序可以从 UVC 器件输送视频。

图 6. 主机与摄像头之间进行预输送时的对话框



2.3.2.1 控制请求 — 亮度和 PTZ 控制

亮度和 PTZ 控制都是在相关项目中实现的。通过定义 **uvc.h** 文件中的 “UVC_PTZ_SUPPORT”，可以打开 PTZ。这些性能可能受图像传感器的支持，也可能不受它的支持。若不受支持，则必须设计特定硬件，以实现它们。在任何情况下，这些性能在所控制的 USB 侧的固件实现仍然相同。然而，图像传感器的实现会有区别。因此，只提供了

占位符函数实现 USB 侧的控制。你必须写入代码，以实现图像传感器特定的 PTZ。

注意：本应用笔记中所描述的所有功能均在 **uvc.c** 文件中实现，除非另有提及。该文件是存储在 **USBVideoClass** 文件夹中项目的一部分（该文件夹位于本应用笔记附带的源代码 zip 文件中）。

主机应用（通过 EP0）向处理单元发出视频控制请求，用以控制亮度。通过 **CyFxUVCAppInUSBSetupCB** 函数，可以处理所有设置请求。该函数可检测到主机是否已经发出了 UVC 特定的请求（控制或流），然后设置一个事件标志：

“CY_FX_UVC_VIDEO_CONTROL_REQUEST_EVENT” 或

“CY_FX_UVC_VIDEO_STREAM_REQUEST_EVENT”。此时，函数 **UVCAppEP0Thread_Entry**（EP0 应用线程）会处理该标志。

亮度控制请求将触发视频控制请求的事件标志。正在等待触发这种标志的 EP0 应用线程会对视频控制请求进行解码，并调用相应的函数。EP0 应用线程调用 **UVCHandleProcessingUnitRqts** 函数用以处理亮度请求。

更改 **UVCHandleProcessingUnitRqts** 函数，以实现处理与单元相关的控制项（亮度、对比度、色调等）。更改 **UVCHandleCameraTerminalRqts** 函数，以实现摄像头终端控制。更改 **UVCHandleExtensionUnitRqts** 函数，以实现所有供应商制定的请求。要支持这些控制，您必须设置 USB 描述符中的相应位。UVC 规范包括摄像头终端、处理单元和扩展单元 USB 描述符等详细信息。

2.3.2.2 串流传输请求 — 探针与提交控制

通过 **UVCHandleVideoStreamingRqts** 函数，可以处理流相关的请求。当 UVC 驱动程序需要从 UVC 器件输送视频时，首先要进行协商。在该阶段，驱动程序将发出一个 PROBE（探针）请求，比如：GET_MIN、GET_MAX、GET_RES 和 GET_DEF。作为响应，FX3 固件会返回一个 PROBE 结构。该结构包含了 USB 描述符的索引，包括视频格式和视频帧、帧速率、帧的最大尺寸以及负载大小等（即 UVC 驱动程序在每次转换中可获取的字节数量）。

对于 USB 2.0 或 USB 3.0 的连接，“CY_FX_UVC_PROBE_CTRL”的 switch 语句将处理流的协商阶段（在不同的模式下所支持的视频属性也不同）。请注意，因为在 USB 2.0 或 USB 3.0 连接中都支持同一个串流模式，所有 GET_MIN、GET_MAX、GET_DEF 和 GET_CUR 的报告值均相同。如果需要在多个串流模式，则这些值会不一样。

“CY_FX_UVC_COMMIT_CTRL”的 switch 语句处理了流阶段的开始部分。COMMIT 控制的 SET_CUR 请求表示主机将接着开始传输视频。因此，COMMIT 控制的 SET_CUR 会控制 “CY_FX_UVC_STREAM_EVENT” 事件，表示主

应用线程 **UVCAppThread_Entry** 启动 GPIF II 状态机以进行传输视频。

2.3.3 视频数据格式：YUY2

UVC 规范只支持一个视频数据的颜色格式子集。因此，您需要选择一个能输送符合 UVC 规范要求颜色格式的图像传感器。本应用笔记包含了一个被称为 YUY2 的非压缩颜色格式。大部分（而不是所有）图像传感器均支持这种格式。YUY2 颜色格式是 YUY 颜色格式中 4:2:2 的采样版本。亮度值 Y 是对所有像素进行采样的，而色度值 U 和 V 是仅针对偶像素进行采样的。这会创建“宏像素”，每个宏像素描述两个总共使用 4 字节的图像像素。请注意，所有其他字节都是 Y 值，而 U 和 V 值仅用于表示偶像素。

Y0、U0、Y1、V0（前两个像素）

Y2、U2、Y3、V2（接下来的两个像素）

Y4、U4、Y5、V4（接下来的两个像素）

请参考[维基百科](#)，了解有关颜色格式的详细信息。

注意：不支持 RGB 格式。尽管 UVC 规格不支持的单色图像，但通过将单色图像数据作为 Y 值发送，并将所有 U 和 V 值都设置为 0x80，仍可以将一个 8 位的单色图像作为 YUY2 格式显示。

2.3.4 UVC 视频数据标头

UVC 类别非压缩视频负载需要一个 12 字节的标头数据。标头数据描述了所传输图像数据的属性。例如，它包括一个“新帧”位，图像传感器控制器（FX3）可以为每个帧切换该位。FX3 代码还可以设置标头数据中的错误位，以表示在传输当前帧的过程中发生的错误。每个 USB 传输操作都需要 UVC 数据标头。请参考 UVC 规范，了解详细信息。

表 4 显示的是 UVC 视频标头数据的格式。

表 4. UVC 视频标头数据格式

| 字节偏移 | 字段名称 | 说明 |
|------|------|----------------------------------|
| 0 | HLF | 标头长度字段指定了标头的长度，单位为字节 |
| 1 | BFH | 位字段标头表示图像数据的类型、视频流的状态以及其他字段的当前情况 |
| 2-5 | PTS | 呈现时间戳显示的是本地器件时钟单元中源时钟的时间 |
| 6-11 | SCR | 源时钟参考指示系统时间时钟和 USB 帧开始（SOF）标志计数器 |

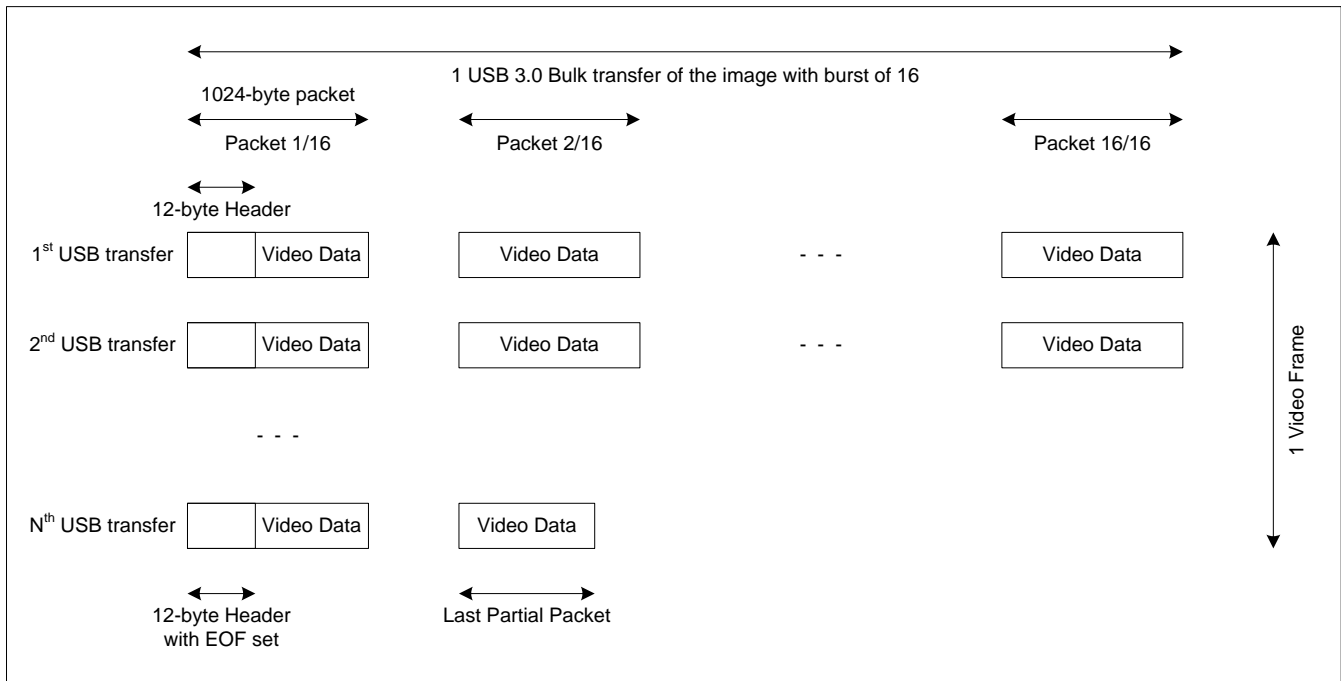
HLF 的值通常为 12。PTS 和 SCR 字段都是可选的。固件示例将这些字段内填充数据 0。位字段标头（BFH）保存变化值在帧结束处。表 5 显示的是 BFH 格式（BFH 是视频标头数据的一部分）。

表 5. 位字段标头（BFH）格式

| 位偏移 | 字段名称 | 说明 |
|-----|------|--|
| 0 | FID | 帧标识符位在每个图像帧的起始边界上进行切换，在图像帧的其余部分该位保持不变 |
| 1 | EOF | 帧结束位指示视频结束，仅在属于图像帧的最后一个 USB 传输操作中设置该位 |
| 2 | PTS | 存在时间戳位指示了 UVC 视频数据标头中 PTS 字段是否存在（1 表示存在） |
| 3 | SCR | 源时钟参考位指示了 UVC 视频数据标头中 SCR 字段是否存在（1 表示存在） |
| 4 | RES | 保留，将其设置为 0 |
| 5 | STI | 静态图像位指示视频取样是否属于静态图像 |
| 6 | ERR | 错误位表示在器件的传输过程中是否发生错误 |
| 7 | EOH | 标头结束位，如果被设置，表示 BFH 字段的结束 |

图 7 显示的是如何将这些标头数据添加到本应用的视频数据中。应针对所有 USB 批量传输操作添加 12 字节的标头。在这里，每个 USB 传输操作共有 16 个批量数据包。USB 3.0 的批量数据包的大小为 1024 个字节。

图 7. UVC 视频数据传输



3. GPIF II 图像传感器接口

FX3的GPIF II模块是一个很灵活的状态机，可以对它进行自定义，使之能够将FX3引脚连接至外部硬件（如图像传感器）。要设计状态机，需要了解该接口的要求以及FX3的DMA功能。

3.1 图像传感器接口

图 2 显示的是一个典型的图像传感器接口。图像传感器一般需要收到一个来自 FX3 控制器的复位信号。它可以通过使用 FX3 通用输入/输出（GPIO）得到处理。

图像传感器通常使用一个 I²C 接口，这样可以允许控制器对图像传感器的参数进行读取和写入操作。图像传感器还会使用串行外设接口（SPI）或通用异步接收器/发送器（UART）接口实现同样操作。FX3 的 I²C、SPI 和 UART 模块都具有该功能。本应用使用 I²C 来配置图像传感器。

为了传输图像，图像传感器提供下面各信号：

1. FV：帧有效（表示帧的开始和结束）
2. LV：行有效（表示行的开始和结束）
3. PCLK：像素时钟（即同步接口的时钟）
4. Data（数据）：图像数据的八个数据线

图 8 显示的是 FV、LV、PCLK 和 Data 信号的时序图。图像传感器生效 FV 信号，以表示帧的开始。然后逐行传输图像数据。在每行的传输操作中会激活 LV 信号，因为图像传感器会驱动 YUY2 格式的 8 位像素（该格式中每两个像素使用了 4 个字节）。在每个 PCLK 的上升沿上字节数据都会进入给 GPIF II 单元。

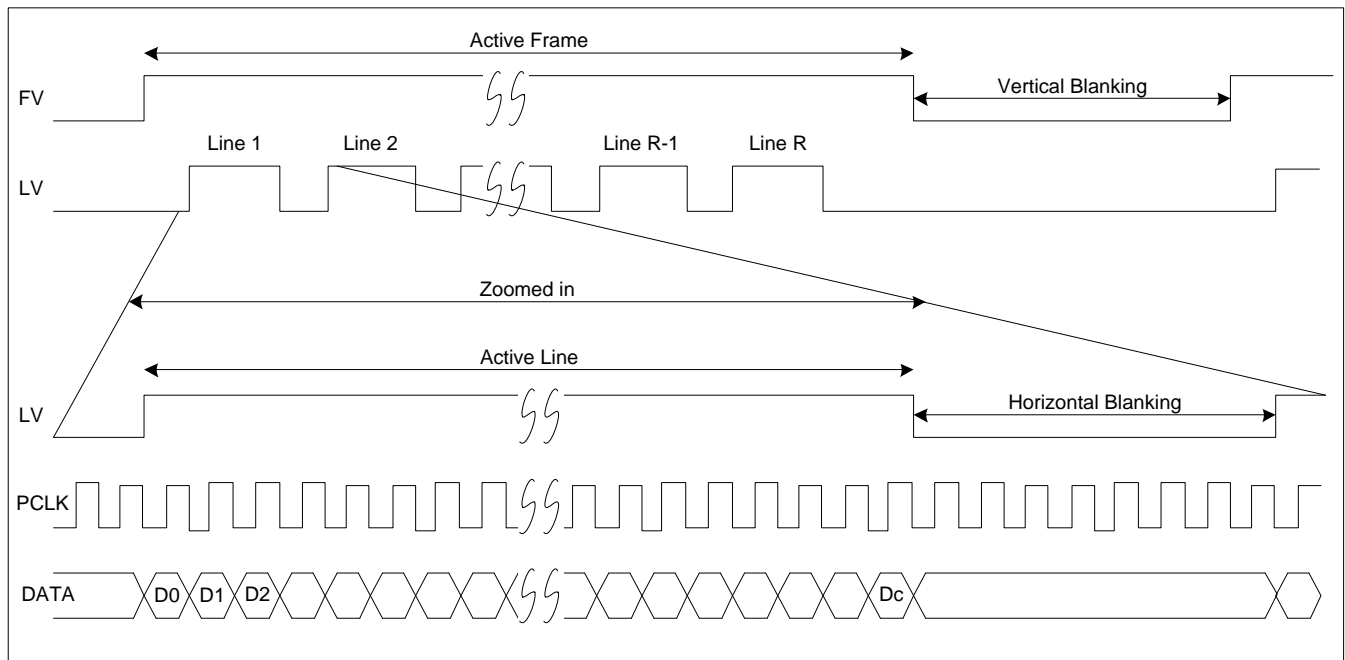
可将 FX3 GPIF II 总线配置为 8、16 或 32 位数据的总线。本应用使用了图像传感器中的 8 位总线。如果图像传感器提供的总线是非字节对齐的（比如 12 位总线），请使用下一个更大的大小来填充或清除未使用的位。

3.1.1 GPIF II 接口的要求

根据时序图（图 8），GPIF II 的状态机有下面各项要求：

- 仅在 LV 及 FV 信号激活时，GPIF II 模块才会从数据引脚传输数据。
- 图像传感器不提供流量控制。因此，该接口必须传输一整行视频而不能间断。在该设计中，每一行都有 1280 个像素，每个像素需要两个字节，所以每行要传输 2560 个字节。
- 每一帧结束时都要通知 CPU，使其更新相应标头位。

图 8. 图像传感器接口时序图



3.2 图像传感器接口的引脚映射情况

表 6 显示的是 GPIF II 到图像传感器的引脚映射详情。

表 6. 并联图像传感器接口的引脚映射情况

| EZ-USB FX3 引脚 | 具有 8 位数据总线的同步并联图像传感器接口 |
|---------------------------|------------------------|
| GPIO[28] | LV |
| GPIO[29] | FV |
| GPIO[0:7] | DQ[0:7] |
| GPIO[16] | PCLK |
| I ² C_GPIO[58] | I ² C_SCL |
| I ² C_GPIO[59] | I ² C_SDA |

如果图像传感器使用的是 UART 或 SPI 接口，并使用了 16 位数据总线，那么请采用表 7 中描述的引脚映射。

表 7. 图像传感器的其他引脚映射情况

| EZ-USB FX3 引脚 | 图像传感器接口 (额外引脚) |
|---------------|-----------------------|
| GPIO[8:15] | DQ[8:15] |
| GPIO[46] | GPIO/UART_RTS |
| GPIO[47] | GPIO/UART_CTS |
| GPIO[48] | GPIO/UART_TX |
| GPIO[49] | GPIO/UART_RX |
| GPIO[53] | GPIO/SPI_SCK/UART_RTS |
| GPIO[54] | GPIO/SPI_SSN/UART_CTS |
| GPIO[55] | GPIO/SPI_MISO/UART_TX |
| GPIO[56] | GPIO/SPI_MOSI/UART_RX |

注意：如需完整的 EZ-USB FX3 引脚映射，请参见 [EZ-USB FX3 超速 USB 控制器数据手册](#)。

3.3 乒乓 DMA 缓冲区

要了解 FX3 的数据输出和输入，需要明白下面各术语：

- 插座
- DMA 描述符
- DMA 缓冲区
- GPIF 线程

插座是外设硬件模块和 FX3 RAM 间的连接点。FX3 上的每个外设硬件模块 (如 USB、GPIF、UART 和 SPI) 具有与本身相连的固定插座数量。输出给外设的独立数据数量等于该外设上的插座的数量。插座的实现包括一组寄存器, 用于指向有效的 DMA 描述符, 并使能或标志与插座本身相连的中断。

DMA 描述符是一组位于 FX3 RAM 中的寄存器。它保存了 DMA 缓冲区的地址和尺寸数据, 以及指向下一个 DMA 描述符的指针。这些指针构建成了 DMA 描述符链。

DMA 缓冲区是 RAM 的一部分, 用于存储通过 FX3 器件传输的中间数据。通过 FX3 固件, 可将部分 RAM 空间作为 DMA 缓冲区使用。这些缓冲区的地址被存储为 DMA 描述符的一部分。

GPIF 线程是 GPIF II 模块内的专用数据路径, 用来将外部数据引脚同插座连接起来。

插座可通过各个事件来互相发出信号, 或者它们可通过中断向 FX3 CPU 发出信号。这些操作是由固件配置的。例如, 将数据流从 GPIF II 模块传输给 USB 模块。GPIF 插座可以通知 USB 插座它已经向 DMA 缓冲区填充了数据, 或者 USB 插座可以通知 GPIF 插座它已经清空了该 DMA 缓冲区。该操作被称为自动 DMA 通道。当 FX3 CPU 不用修改数据流中的任何数据时, 通常会使用自动 DMA 通道实现。

或者, GPIF 插座可以向 FX3 CPU 发送一个中断, 来通知 GPIF 插座已经填充了 DMA 缓冲区。FX3 CPU 可将该信息传输给 USB 插座。USB 插座可向 FX3 CPU 发送一个中断, 来通知 USB 插座已经清空了 DMA 缓冲区。此时, FX3 CPU 可以将此信息反馈给 GPIF 插座。该操作被称为手动 DMA 通道。如果 FX3 CPU 需要添加、删除或修改数据流中的数据, 通常需要执行该手动 DMA 通道操作。本应

用笔记中的固件示例使用了手动 DMA 通道操作, 因为该固件要求添加 UVC 视频数据包头。

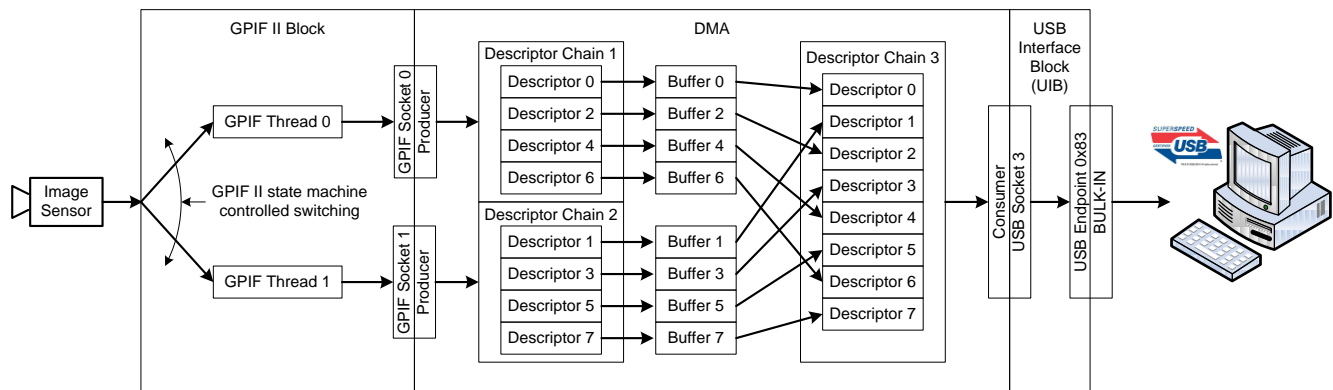
生产插座是指用于将数据写入 DMA 缓冲区内插座。消耗插座是指读取 DMA 缓冲区内数据的插座。插座使用存储在 DMA 描述符上的 DMA 缓冲区地址、DMA 缓冲区大小和 DMA 描述符链的值来管理数据 (第 4 节)。插座填充或清空 DMA 缓冲区后, 需要一个有限的时间量 (多达几微秒) 从一个 DMA 描述符转移到另一个描述符。转换过程中, 插座不能传输数据。对于没有流量控制的接口, 该延迟是一个问题。图像传感器接口便属于这类情况。

通过使用多个 GPIF 线程, 可以在 GPIF II 模块中处理这问题。GPIF II 模块实现四个 GPIF 线程。但每次只有一个 GPIF 线程能够传输数据。GPIF II 状态机必须选择一个有效的 GPIF 线程来传输数据。

GPIF 线程选择机制同 MUX 一样。GPIF II 状态机使用内部控制信号或外部信号选择有效的 GPIF 线程。在这示例中, 内部控制信号控制在各 GPIF 线程之间进行切换。切换有效的 GPIF 线程时会切换用于数据传输的有效插座, 从而修改用于数据传输的 DMA 缓冲区。GPIF 线程切换没有延迟。在 DMA 缓冲区边界上实现对 GPIF II 状态机的切换, 这样可以屏蔽在切换到新的 DMA 描述符时, GPIF 插座所要求的延迟。这样, 当 DMA 缓冲区被填充时, GPIF II 模块可以使用传感器中的数据。

图 9 显示的是本应用程序中使用的各个插座、DMA 描述符和 DMA 缓冲区连接, 以及数据流。使用两个 GPIF 线程填充备用的 DMA 缓冲区。这些 GPIF 线程使用了单独的 GPIF 插座 (作为生产插座使用) 和 DMA 描述符链 (描述符链 1 和描述符链 2)。USB 插座 (作为消耗插座使用) 使用了另一个 DMA 描述符链 (描述符链 3) 按正确顺序读取数据。

图 9. FX3 数据传输架构



3.4 设计策略

在编译 GPIF II 状态机的详细信息前，建议考虑基本的传输策略。

图 10. 在视频线内的数据传输

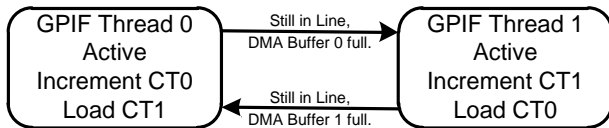
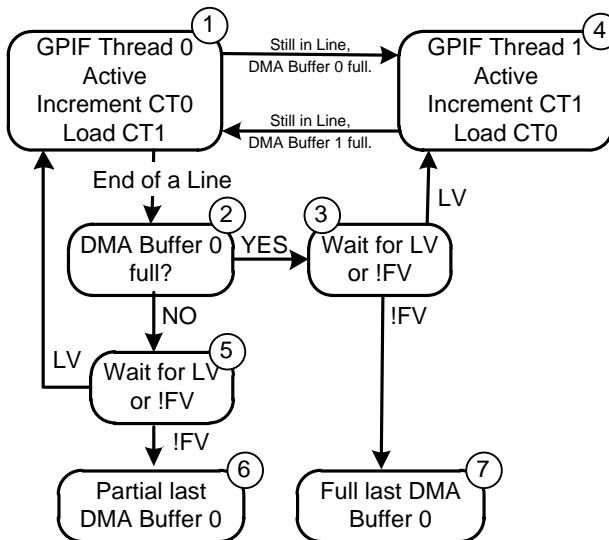


图 10 显示的是在活动的水平线期间，填满 DMA 缓冲区时基本 DMA 缓冲区的乒乓操作。GPIF II 状态机有三个独立的计数器。在这示例中，GPIF II 单元使用两个计数器 (CT0 和 CT1) 来计算写入到一个 DMA 缓冲区内的字节数量。当计数达到 DMA 缓冲区的限制时，将采取“DMA 缓冲区已满”分支，并且 GPIF II 会切换 GPIF 线程。由于各字节是在一个 GPIF 线程中进行传输的，因此需要初始化其他 GPIF 线程的计数器，以便在切换到下一个 GPIF 线程后，计数器可以计算字节。

如果图像传感器取消了激活 LV，则表示它已到达视频线的结尾。这时，状态机有几个选项，如图 11 所示，其中 LV 表示有效线信号和 FV 表示有效帧信号。各选项结果取决于 DMA 缓冲区是否填满和该帧是否完成。

图 11. 在视频线结尾上数据传输的决策



注意：由“线结尾”开始的映像决策树也是从状态 4 开始的；为了提高清晰度，在该框图中没有显示。

在每个线的结尾处 (LV=0)，状态机从状态 1 转换到状态 2，然后它会检查 DMA 缓冲区字节计数器，以此确定 DMA 缓冲区是否已满。如果 DMA 缓冲区已满，则状态机将转换到状态 3。在状态 3 中，如果 FV 为低电平，则表示已经传输完了全帧，并且进入状态 7。在状态 7 中，CPU 通过一

个中断 (表明完整的最后 DMA 缓冲) 被警惕。通过这信息，CPU 可以设置用于下一个帧的 GPIF II。

在状态 3 中，如果 FV=1，则表示图像传感器正在传输某一帧，并且它将要重新激活 LV=1，以指出下一个视频线。当 LV=1 时，状态机从状态 3 转换到状态 4，并会执行 GPIF 线程切换，该切换与从状态 1 转换到状态 4 的切换相同。因此，路径 1-4 和 1-2-3-4 都进行了一个 GPIF 线程切换。这些路径的区别在于第二个路径需要占用一个额外的周期，这是因为它在线结尾上有一个暂停。

如果 DMA 缓冲区未满，则状态机将从状态 2 转换到状态 5。状态 5 与状态 3 相似，它将等待该帧结束或 LV 重新激活 (开始下一个视频线)。如果 LV=1，则它将继续填充处于状态 1 的 DMA 缓冲区 0。如果 FV=0，则表示图像传感器已完成传送一个视频帧，而且 DMA 缓冲区 0 只被部份填充。在状态 6 中，状态机向 CPU 发送另一种中断，用以允许它通过 USB 发送短 DMA 缓冲区。

3.5 GPIF II 状态机

FX3 GPIF II 是一个可多达 256 种状态的可编程状态机。每个状态可以执行下列操作：

- 驱动多个控制线
- 发送或接收数据和/或地址
- 向内部 CPU 发送中断

状态转换取决于内部或外部信号，如 DMA 就绪信号和图像传感器的有效帧/线信号。

开始设计 GPIF II 状态机，可在图像传感器波形中，选择一点为状态机的起始位置。由正向 FV 跃变指出的帧起始位置是一个合逻辑的起始点。GPIF II 检测该沿会先等待 FV=0 (第一个状态)，然后再等待 FV=1 (第二个状态) 这两个过程。第二个状态亦初始化一个传输计数器去回应一个存储满了视频数据的 DMA 缓冲区。状态机将测试该计数器的值，如果达到极限值，它会切换 GPIF 线程 (DMA 缓冲区)。当 DMA 缓冲区已满时，则表示已经达到了计数器的极限值。

状态机使用两个 GPIF II 内部计数器来计算 DMA 缓冲区字节：GPIF II 地址计数器 ADDR 和数据计数器 DATA。每当 GPIF II 状态机切换 GPIF 线程时，它将为其他 GPIF 线程启动合适的计数器。由于进行加载计数器的限制值需要一个时钟周期，因此该值比终端计数值小 1。

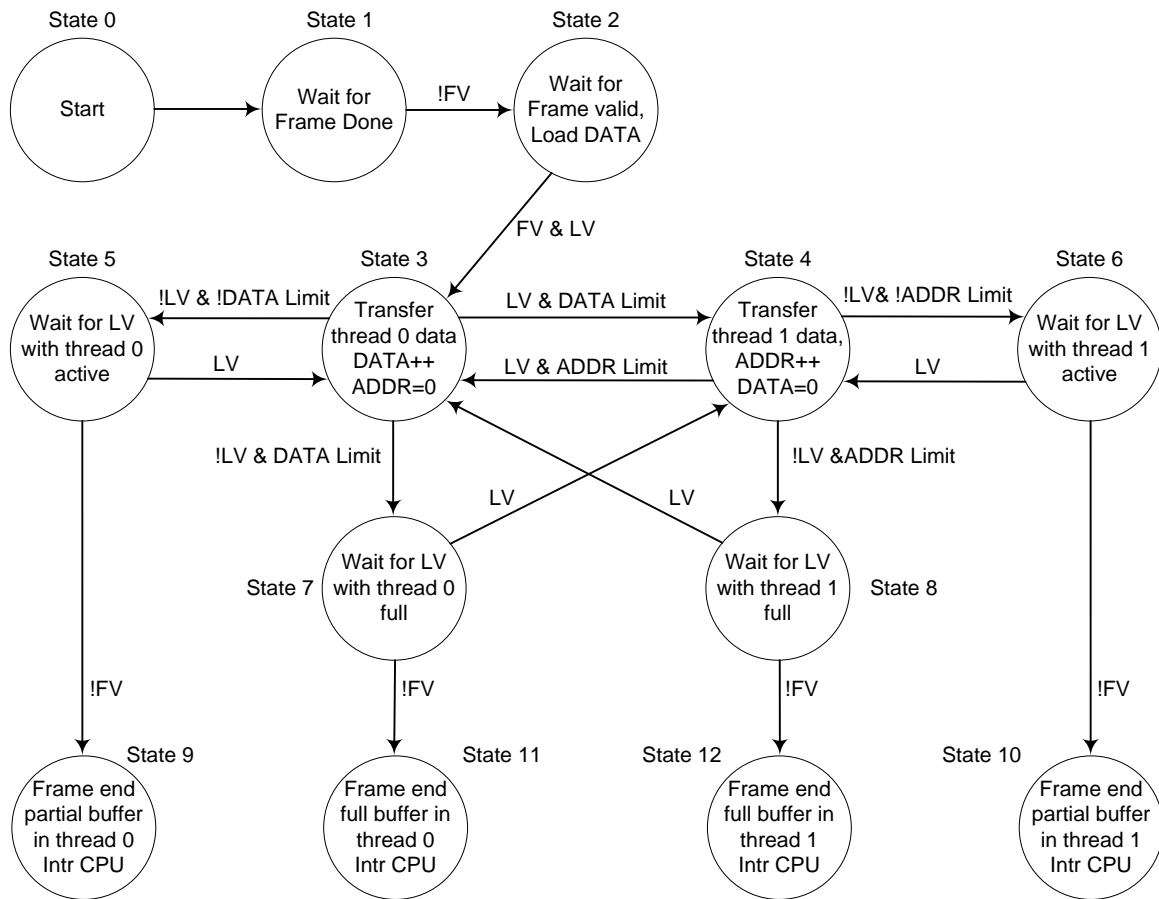
每经过一个时钟周期，传输计数器的值都会加 1。因此，根据接口的数据总线宽度，计数器的限定值会不一样。在该示例中，数据总线的宽度为 8 位，DMA 缓冲区大小为 16,368 字节 (如章节 5.6 中所述)，因此编程限定值为 16,367。一般情况下，DMA 缓冲区的计数限制值为：

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

所以对于 16 位的接口, DMA 缓冲区大小为 8184 个 16 位字, 编程限制值为 8183。对于 32 位的接口, DMA 缓冲区大小为 4092 个 32 位字, 编程极限值为 4091。

图 12 显示的是 GPIF II 状态机的详细信息。两个 DMA 缓冲区字节计数器分别为 GPIF II DATA 和 ADDR 计数器。这些计数器同图 11 中显示的 CT0 和 CT1 相对应。

图 12.GPIF II 状态机的框图



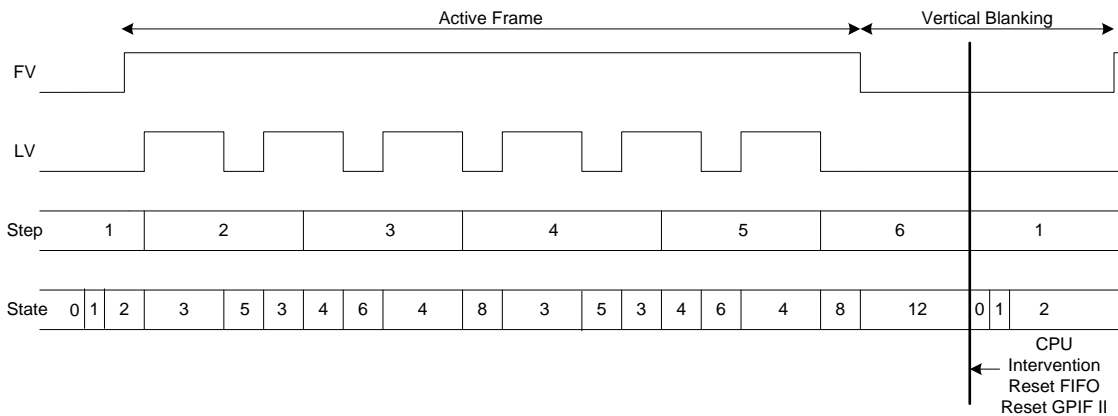
在每一帧的结尾处, CPU 收到四个可能中断请求中的一个, 指出 DMA 缓冲区的编号和已满状态 (状态 9-12)。可以使用这些请求执行回调函数, 以使能 CPU 处理某些任务, 具体如下所示:

1. 如果帧结尾处有一个 DMA 缓冲区 (状态 9 和 10) 未, 则 USB 传输提供最后的 DMA 缓冲区。如果在帧结尾处 DMA 缓冲区为满, 则 GPIF II 会自动将它传送到 USB 传输 (状态 11 和 12) 中。

2. 等待消耗插座 (USB) 传送最后的 DMA 缓冲区数据; 然后将该通道和 GPIF II 状态机复位到状态 0, 以预备下一个帧。
3. 处理所有特殊应用任务, 以指出帧的前进。UVC 需要通过切换 12 字节插座中的位来表明帧改变事件。

图 13 将图像传感器波形连接到 GPIF II 状态, 显示一小部分水平线。“Step”线处理 DMA 系统, 具体情况在章节 4 和图 45 中进行介绍。

图 13. 图像传感器接口、数据路径执行以及状态机的关联



3.6 使用 GPIF II Designer 实现图像传感器接口

本节介绍的是如何使用 GPIF II Designer 来设计图传感器接口。有关参考材料，请访问附带源中压缩文件内的“**GPIF II Designer/ImageSensorInterface.cydsn**”目录路径，以查找整个项目。

设计过程包括下面三个步骤：

1. 使用 GPIF II Designer 创建项目
2. 选择接口定义
3. 在画布上绘制状态机

3.6.1 创建项目

启动 GPIF II Designer。GUI 出现，如图 14 所示。按照下图所示的详细内容进行操作。每一个图都包含了多个子步骤。介绍每一步的详细信息，请参考《**GPIF II Designer 用户指南**》中的内容。

图 14. 启动 GPIF II Designer

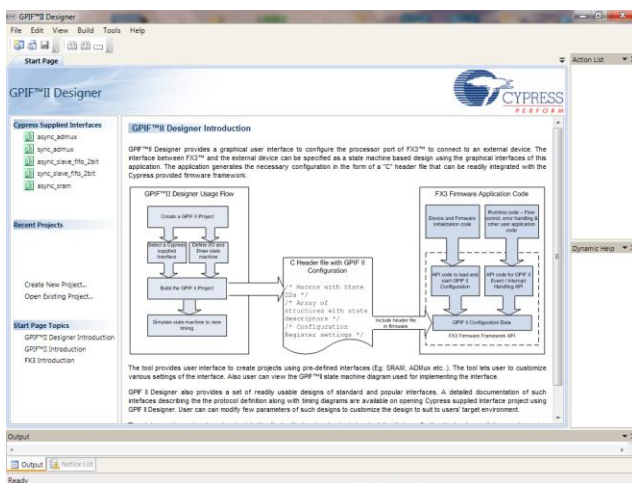


图 15. 打开 File 菜单并选择 New Project 项

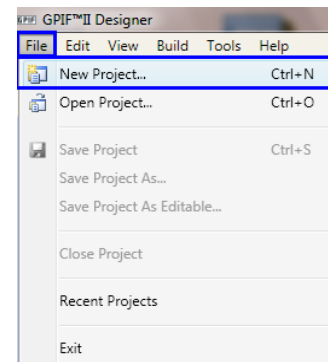
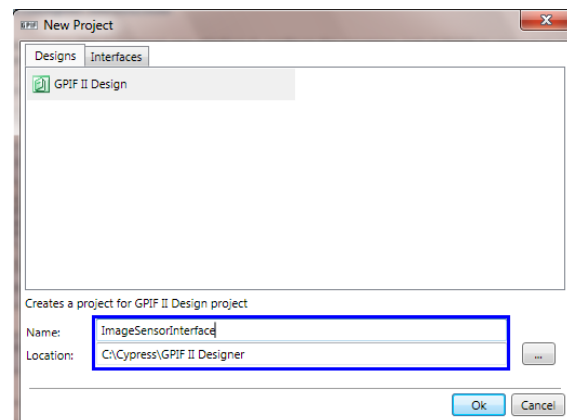


图 16. 输入项目名称和位置



现在完成了项目创建操作，GPIF II Designer 被打开，通过它可以访问接口定义和状态机选项卡。在接口定义选项卡中，FX3 位于左侧，图像传感器（标签为“应用处理器”）位于右侧。接下来进行设置这两者之间的接口。

3.6.2 定义接口

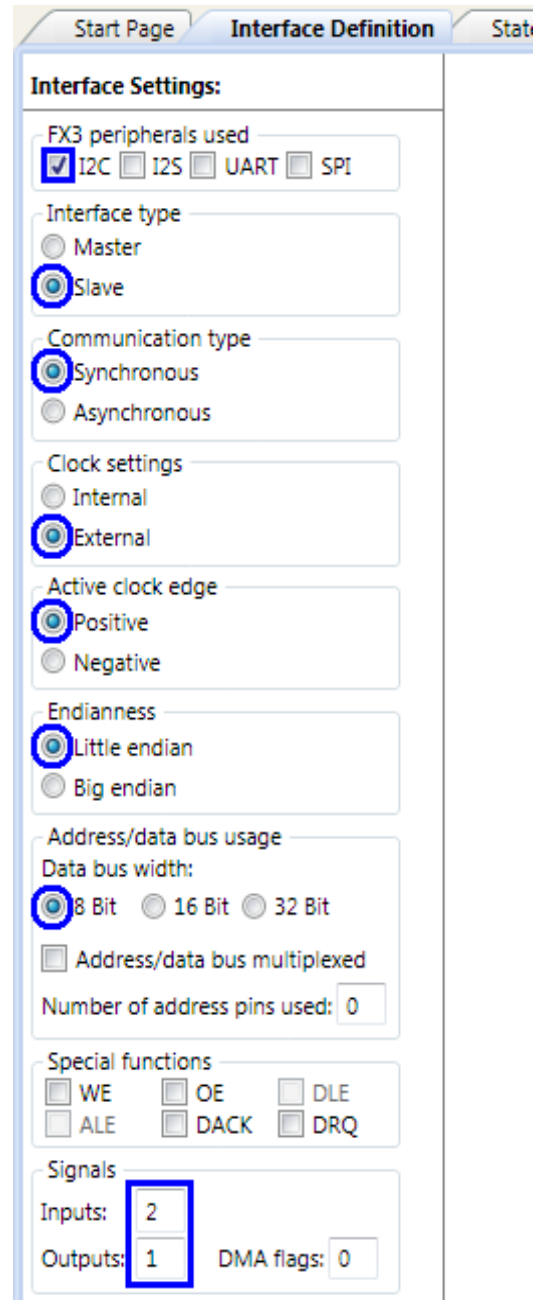
在该项目中，图像传感器具有 8 位数据总线与 FX3 器件相连。它将 GPIO 28 用于 LV 信号，GPIO 29 用于 FV 信号，另外 GPIO 22 为低电平有效输入的传感器复位（表 6）。

该图像传感器还可通过使用连接到 FX3 的 I²C 对图像传感器寄存器进行访问，例如，为 720p 模式配置传感器。“接口设置”列显示的是各需要的选项。

另外，在下一个阶段中，可以使用指示输入信号来创建转换公式。图 17 显示的是可选的接口设置。

1. 在“Signals”（信号）项中，为 LV 和 FV 选择两个输入。
2. 在“Signals”（信号）项中，为 nSensor_Reset 选择一个输出。
3. 在“FX3 peripherals used”（使用的 FX3 外设）项中，选择 I²C。这样可以激活 FX3 I²C 主器件。
4. 在“Interface type”（接口类型）项中，选择“Slave”（从器件）。由于图像传感器提供时钟，并驱动数据总线，因此 GPIF II 作为从器件使用。
5. 在“Communication type”（通信类型）项中，选择“Synchronous”（同步）。这样反映了在图像传感器中存的在同步时钟。
6. 在“Clock settings”（时钟设置）项中，选择“External”（外部）。图像传感器将为 GPIF II 提供它的 PCLK 信号。
7. 在“Active clock edge”（活动时钟沿）项中，选择“Positive”（上升沿）。图像传感器在它的上升沿上启用数据转换。
8. 在“Endianness”（字节顺序）项中，选择“Little endian”（低位优先）。字节顺序指出数据总线中比一个字节的宽度更大的字节顺序。对于 8 位的接口，该设置不受影响。
9. 在“Address/data bus usage”（地址/数据总线的使用情况）项中，选择“8 bit”（8 位）。图像传感器提供了一个 8 位的数据总线。

图 17. 接口设置选项

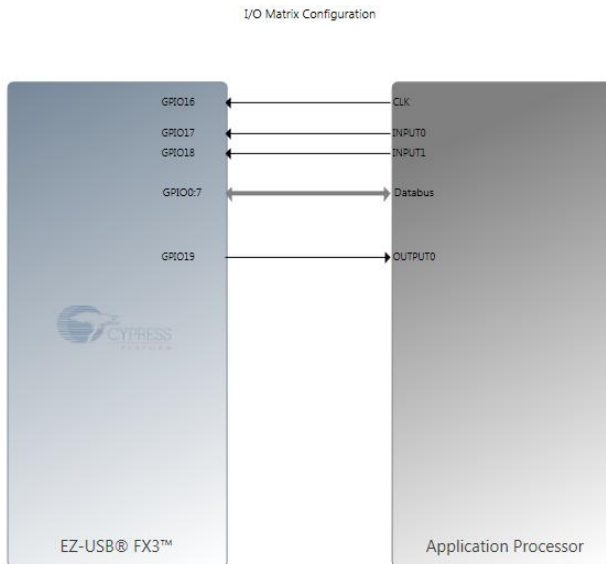


The screenshot shows the 'Interface Definition' tab in the Cypress IDE. The 'Interface Settings' panel is active, displaying various configuration options for the FX3 peripheral. The settings are as follows:

- FX3 peripherals used:** I2C (checked), I2S, UART, SPI.
- Interface type:** Master, Slave (selected).
- Communication type:** Synchronous (selected), Asynchronous.
- Clock settings:** Internal, External (selected).
- Active clock edge:** Positive (selected), Negative.
- Endianness:** Little endian (selected), Big endian.
- Address/data bus usage:** Data bus width: 8 Bit (selected), 16 Bit, 32 Bit. Address/data bus multiplexed (unchecked).
- Number of address pins used:** 0.
- Special functions:** WE, OE, DLE, ALE, DACK, DRQ (all unchecked).
- Signals:** Inputs: 2 (highlighted in a blue box), Outputs: 1, DMA flags: 0.

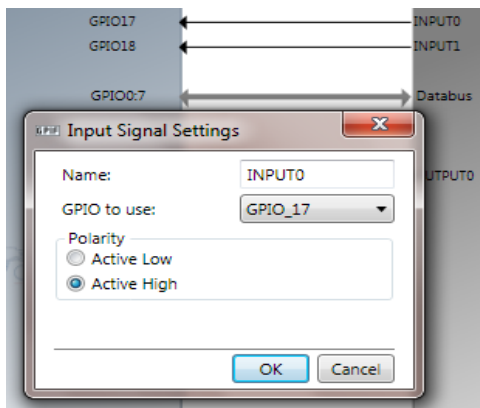
“I/O Matrix configuration”（I/O 矩阵配置）现在的情况如图 18 所示。接下来需要修改输入和输出信号的属性，包括信号名称、引脚映像（例如，哪个 GPIO 作为输入或输出）、信号极性以及输出信号的初始值。

图 18. 当前的框图



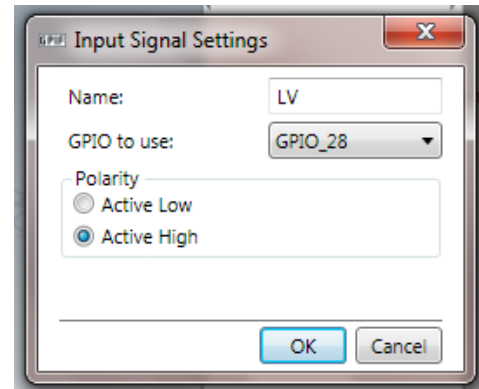
在应用处理器区中双击 **INPUT0** 标签，这样可以打开输入信号的属性，如图 19 所示。

图 19. INPUT0 的默认属性



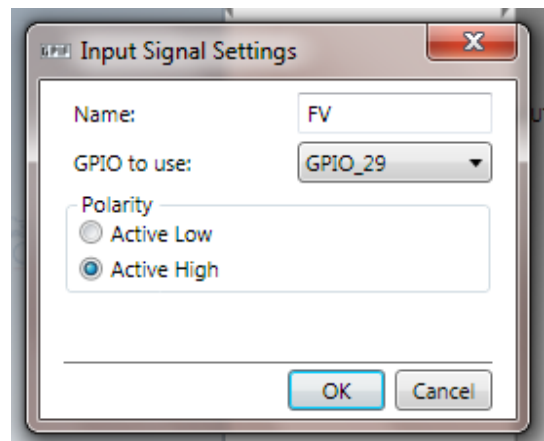
将信号名称修改为“LV”，并将“GPIO to use:”（被使用的 GPIO）设置为 **GPIO_28**（表 6）。将该极性保持为“Active High”（高电平有效）。现在这些属性的情况如图 20 所示。单击“OK”。

图 20. 已编辑的 INPUT0 属性



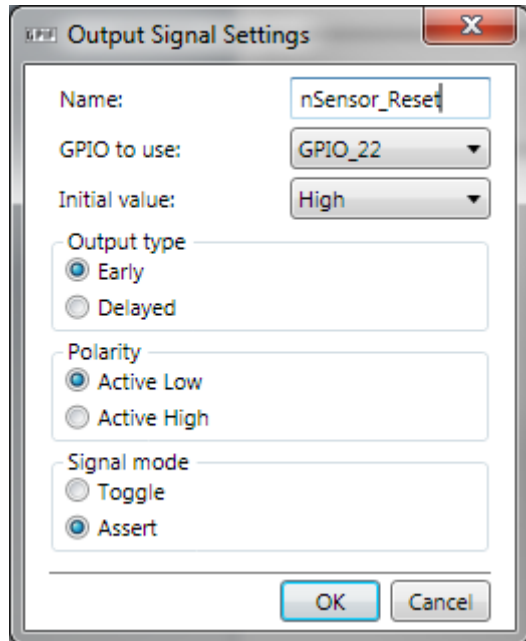
然后，双击 **INPUT1** 标签，将信号名称修改为 **FV**，并将 GPIO 设置为 **GPIO_29**，然后保持该极性为高电平有效（图 21）。

图 21. 编辑 INPUT1 属性



双击 **OUTPUT0** 标签并根据图 22 修改各设置内容。

图 22. 已编辑的 OUTPUT0 属性



这样便完成了接口设置。设置各属性（如信号名称）时，GPIF II 设计的所有其他内容会根据该修改而不断更新。例如，该框图内包含了信号名称，而不是通用的输入和输出名称。另外，当您使用状态机设计工具时，可用的信号选项（如 LV 和 FV 信号）会自动显示为下拉列表中的各选项。

3.6.3 绘制状态机

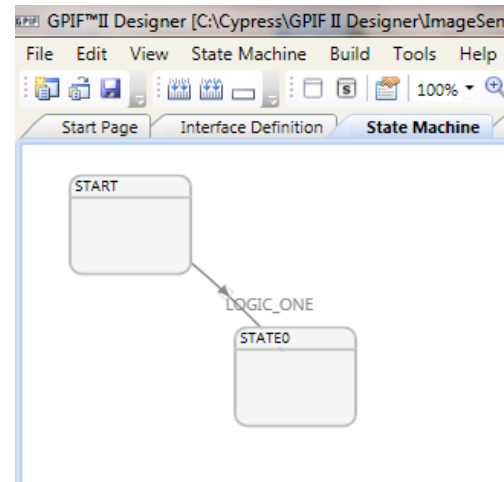
通过点击 **State Machine**（状态机）选项卡可以打开状态机设计图。状态机设计包括下面三个步骤：

1. 创建状态
2. 在各状态下添加操作
3. 使用转换条件创建状态间的转换

绘制 GPIF II 状态机

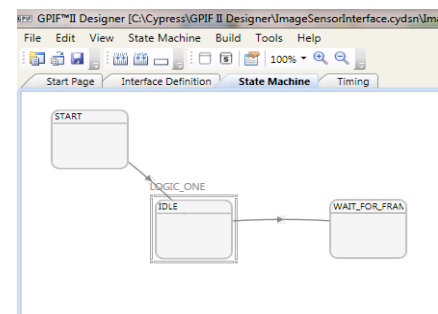
通过点击 **State Machine**（状态机）选项卡打开设计图。未编辑的设计图有两种状态：**START** 和 **STATE0**。无条件转换（**LOGIC_ONE**）将连接各状态（图 23）。

图 23. 初始状态机画布



1. 通过双击 **STATE0** 框内部将 **STATE0** 的名称改为 **IDLE**，然后编辑名称文本框。通过“重复操作直到发生下一个转换为止”项可以确定进入该状态时，仅发生一次操作还是在该状态内每个时钟上均发生。对于没有操作的状态，如这状态，则检验栏状态不可用。
2. 通过右击画布中的空白位置并选择“Add State”（添加状态）项，可以添加一个新的状态。在新状态内部双击，并将它的名称改为 **WAIT_FOR_FRAME_START**。
3. 创建从 **IDLE** 状态转换到 **WAIT_FOR_FRAME_START** 状态的转换。将光标放置在 **IDLE** 状态框的中心位置。该光标将变为+形状，用以指出转换输入。将鼠标拖放到 **WAIT_FOR_FRAME_START** 状态的中心位置。在状态框内出现一个小正方形，它显示的是其中心。如果在状态框中心任何位置中放开鼠标，那么不会创建转换。重新尝试。请注意，这状态切换还没有任何条件。

图 24. 步骤 1 到 3 的结果



4. 通过双击转换线，编辑从 **IDLE** 转为 **WAIT_FOR_FRAME_START** 的转换公式（图 25）。请注意，LV 和 FV 显示作为公式应项目并对对应已重新命名的框图信号。要想选择 **FV 低**，请使用按键和信号选择编译该公式。点击 **Not** 按键，在公式输入窗口内将显示“!”符号。然后选择 **FV** 信号并点击“Add”（添

加）按键（或双击 FV 输入），以创建最后的“!FV”公式。也可以通过在公式输入窗口中键入“!FV”来直接输入该公式。这时，该转换如图 26 所示。

图 25. 双击转换线后将显示该窗口

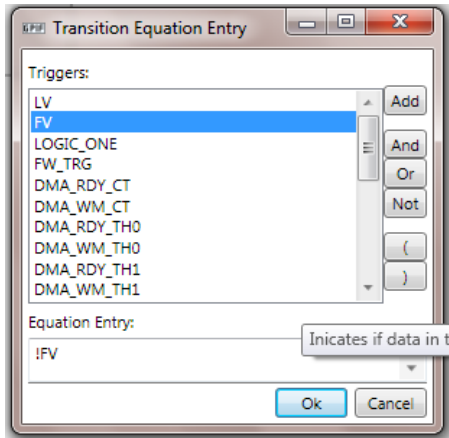
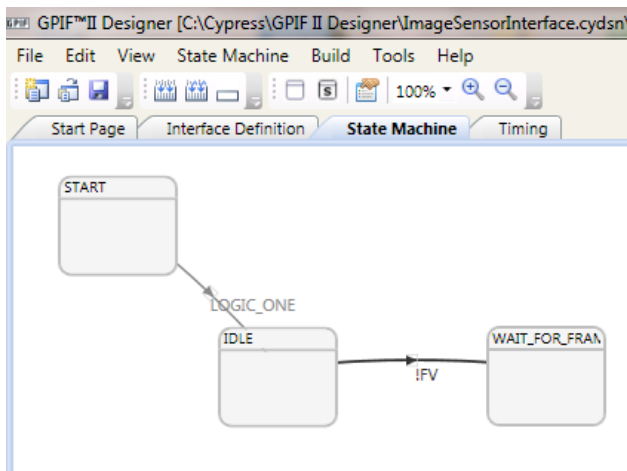
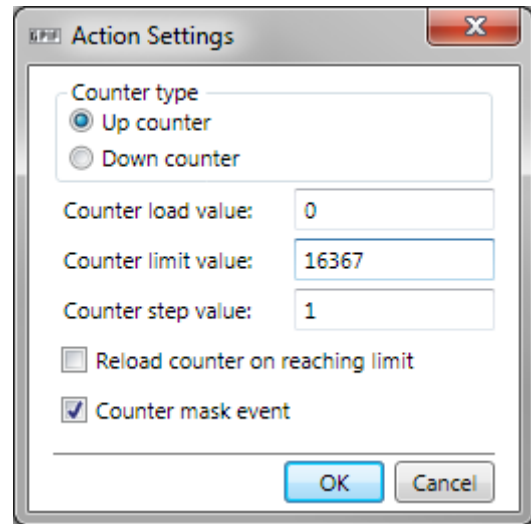


图 26. 已编辑的转换



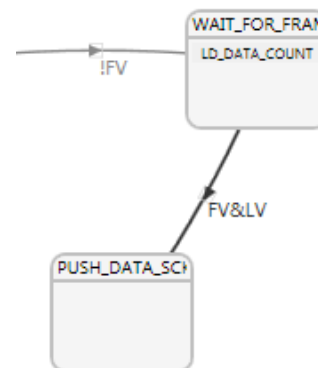
5. 在 WAIT_FOR_FRAME_START 状态下，我们要初始化两字节计数器：因为我们必须在计数器递增的状态前初始化计数器。我们重新回顾一下，本设计的 DATA 计数器使用的是插座 0 缓冲区，而 ADDR 计数器使用的是插座 1 缓冲区。GPIF II Designer 右上窗口的“Action List”（操作列表）窗口内显示的是各操作选项。要想将某个操作添加到状态中，需要将它名称拖拉到状态框中的操作列表内。将 LD_DATA_COUNT 和 LD_ADDR_COUNT 操作拖拉到 WAIT_FOR_FRAME_START 状态框内。
6. 要想编辑各操作属性，请双击状态框中的操作名称。然后编辑两个操作的属性，如图 27 所示。如果勾选“Counter mask event”（计数器屏蔽事件）检验栏，当计数器达到它的限定值时可以禁用其中断请求。

图 27. LD_DATA/ADDR_COUNT 操作设置



7. 添加一个名称为 PUSH_DATA_SCK0 的新状态（将图像传感器数据推入插座 0 内）。该状态每个时钟将一个图像传感器字节传输到 GPIF II 接口上，再路由到插座 0。
8. 创建一个从 WAIT_FOR_FRAME_START 状态到 PUSH_DATA_SCK0 状态的转换。
9. 编辑该转换公式输入创建公式 FV&LV，以便对应 FV 和 LV 激活时。结果状态转换如图 28 所示。

图 28. PUSH_DATA_SCK0 和它的转换条件 FV&LV

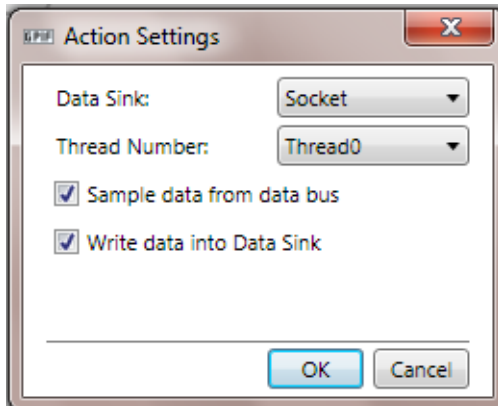


10. 将 COUNT_DATA 操作添加到 PUSH_DATA_SCK0 状态内。这样，DATA（插座 0）计数器的值将在每个 PCLK 的上升沿上递增。
11. 将 IN_DATA 操作添加到 PUSH_DATA_SCK0 状态内。使用该操作可以从数据总线上读取数据，并写入到 DMA 缓冲区内。
12. 将 LD_ADDR_COUNT 操作添加到 PUSH_DATA_SCK0 状态内，以重载 ADDR 计数器。

如上所述，计数器加载是在递增计数器的状态中完成的。ADDR 计数器计算传输到 SCK1 内的字节。

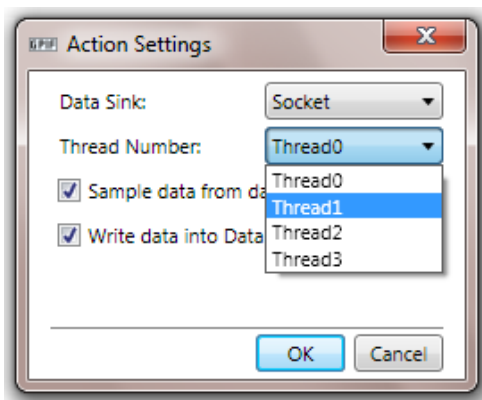
- 在 PUSH_DATA_SCK0 状态中，编辑操作 IN_DATA 的属性，如图 29 所示。

图 29. PUSH_DATA_SCK00 操作设置



- 添加名称为 PUSH_DATA_SCK1 的新状态。将 COUNT_ADDR、IN_DATA 和 LD_DATA_COUNT 操作添加到该状态内。
- 在 PUSH_DATA_SCK1 状态中，设置 IN_DATA 操作的属性配置，以使用 Thread1，如图 30 所示。

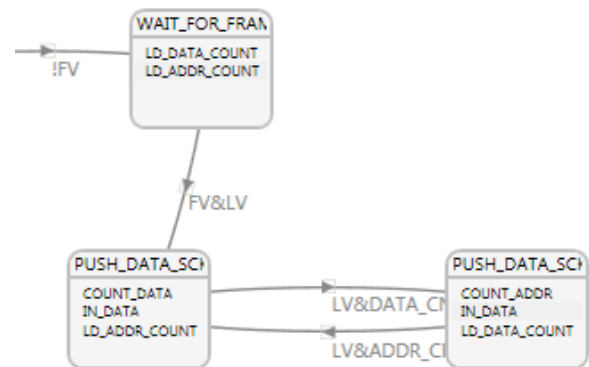
图 30. PUSH_DATA_SCK1 IN_DATA 操作



- 创建从 PUSH_DATA_SCK0 状态到 PUSH_DATA_SCK1 状态的转换。使用公式“LV and DATA_CNT_HIT”编辑该转换的公式输入。
- 创建从 PUSH_DATA_SCK1 状态到 PUSH_DATA_SCK0 状态的转换。反转方向。使用公式“LV and ADDR_CNT_HIT”编辑该转换的公式输入。

这些转换都是在线有效期间於 DMA 缓冲区边界，各 GPIF 线程之间进行的切换。图 31 显示这部分状态机。

图 31.图 10 中的乒乓操作

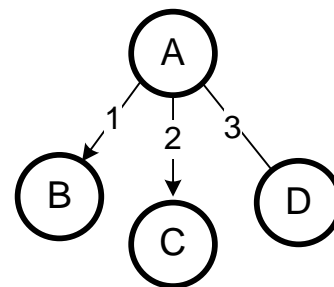


- 将名称为“LINE_END_SCK0”的新状态添加到 PUSH_DATA_SCK0 状态的左侧。
- 将名称为“LINE_END_SCK1”的新状态添加到 PUSH_DATA_SCK1 状态的右侧。

当图像传感器切换到下一个视频线时，而激活 LV 信号被取消，则会进入这两个状态。由于不同的 GPIF 线程可轮流执行同一个操作，因此插座 0 和插座 1 侧都需要这些状态。

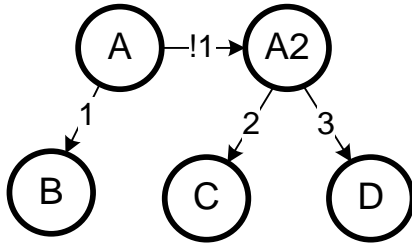
PUSH_DATA 状态需要三个可能的退出转换，但 GPIF II 硬件最多只能实现两个可能的转换。通过创建一个虚拟状态处理三转换要求，该状态不进行任何操作，它只是在两个状态之间分配三个退出条件。为了演示该操作，图 32 显示的是一个状态 A 根据条件 1、2 或 3 转换到状态 B、C 或 D。

图 32. 状态 A 要求退出条件 1、2 和 3



要使它跟 GPIF II 兼容，需要添加虚拟状态 A2，如图 33 所示。

图 33. 添加虚拟状态 A2，是为了与 GPIF II 相兼容



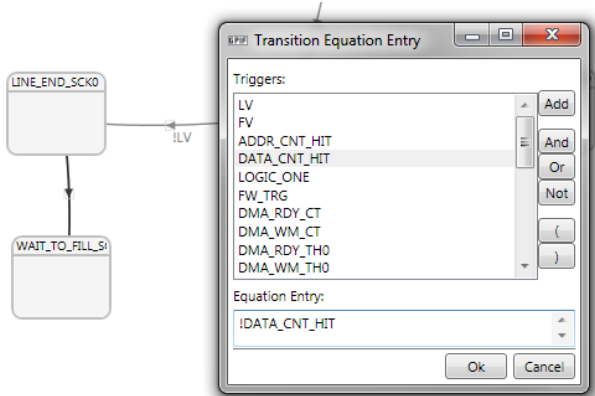
创建 LINE_END 为虚拟状态。

20. 创建从 PUSH_DATA_SCK0 状态到 LINE_END_SCK0 状态的转换 (该转换使用转换公式 “(not LV)”)。
21. 创建从 PUSH_DATA_SCK1 状态到 LINE_END_SCK1 状态的转换 (该转换使用转换公式 “(not LV)”)。
22. 在 LINE_END_SCK0 状态下，创建新状态 “WAIT_TO_FILL_SCK0”。
23. 在 LINE_END_SCK1 状态下创建新状态 “WAIT_TO_FILL_SCK1”。

当 DMA 缓冲区未满却取消激活有效线时，会进入这两个状态。

24. 创建从 LINE_END_SCK0 状态到 WAIT_TO_FILL_SCK0 状态的转换 (该转换使用转换公式 “(not DATA_CNT_HIT)”)，如图 34 所示。

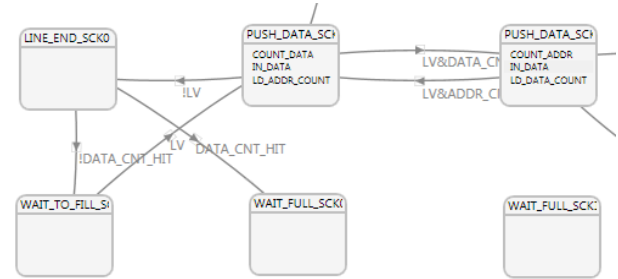
图 34. DATA_CNT_HIT 表示计数器已达到它的编程极限



25. 创建从 “WAIT_TO_FILL_SCK0” 状态到 “PUSH_DATA_SCK0” 状态的回转 (该回转使用公式 “LV”)。一旦该线有效，数据传输将恢复到同一个插座内。
26. 创建从 “LINE_END_SCK1” 状态到 “WAIT_TO_FILL_SCK1” 状态的转换 (该转换使用转换公式 “(not ADDR_CNT_HIT)”)。

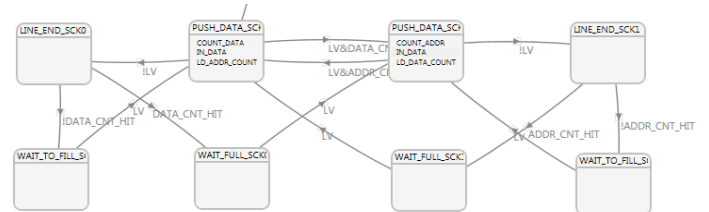
27. 创建从 “WAIT_TO_FILL_SCK1” 状态到 “PUSH_DATA_SCK1” 状态的回转 (该回转使用公式 “LV”)。
28. 在 “PUSH_DATA_SCK0” 状态下添加新状态 “WAIT_FULL_SCK0_NEXT_SCK1”。
29. 在 “PUSH_DATA_SCK1” 状态面添加新状态 “WAIT_FULL_SCK1_NEXT_SCK0”。
30. 在这两个状态期间 (WAIT_FULL_)，图像传感器在 DMA 缓冲区边界上切换各线。因此，下一个字节传输必须使用当前未激活的 GPIF 线程。
31. 创建从 “LINE_END_SCK0” 状态到 “WAIT_FULL_SCK0_NEXT_SCK1” 状态的转换 (该转换使用公式 “DATA_CNT_HIT”)。图 35 显示的是该状态机的一部分。

图 35. 添加的 WAIT_FULL 状态



32. 创建从 “LINE_END_SCK1” 状态到 “WAIT_FULL_SCK1_NEXT_SCK0” 状态的转换 (该转换使用公式 “ADDR_CNT_HIT”)。
33. 创建从 “WAIT_FULL_SCK0_NEXT_SCK1” 状态到 “PUSH_DATA_SCK1” 状态的转换 (该转换使用公式 “LV”)。请注意，这样操作会在框图中创建交叉链接。
34. 创建从 “WAIT_FULL_SCK1_NEXT_SCK0” 状态到 “PUSH_DATA_SCK0” 状态的转换 (该转换使用公式 “LV”)。图 36 显示的是该状态机的一部分。

图 36. DMA 缓冲区充满时进行等待



35. 在 “WAIT_TO_FILL_SCK0” 状态下创建新状态 “PARTIAL_BUF_IN_SCK0”。

36. 在 “WAIT_TO_FILL_SCK1” 状态下创建新状态 “PARTIAL_BUF_IN_SCK1”。

PARTIAL_BUF_ 状态表示帧结束，在这里最后 DMA 缓冲区未充满。当 CPU 响应 GPIF II 生成的中断请求时，它要求手动传送未充满的 DMA 缓冲区。

37. 在 “WAIT_FULL_SCK0_NEXT_SCK1” 状态下添加新状态 “FULL_BUF_IN_SCK0”。

38. 在 “WAIT_FULL_SCK1_NEXT_SCK0” 状态下添加新状态 “FULL_BUF_IN_SCK1”。

FULL_BUF_ 的状态表示，DMA 缓冲区中的最后字节是帧数据的结束（该缓冲区与相应的 GPIF 线程相关联）。由于 GPIF II 硬件负责传送已满的 DMA 缓冲区，因此任何其他操作都是应用特定的。

39. 创建从 “WAIT_TO_FILL_SCK0” 状态到 “PARTIAL_BUF_IN_SCK0” 状态的转换（该转换使用公式 “not FV”）。
40. 创建从 “WAIT_TO_FILL_SCK1” 状态到 “PARTIAL_BUF_IN_SCK1” 状态的转换（该转换使用公式 “not FV”）。
41. 创建从 “WAIT_FULL_SCK0_NEXT_SCK1” 状态到 “FULL_BUF_IN_SCK0” 状态的转换（该转换使用公式 “not FV”）。
42. 创建从 “WAIT_FULL_SCK1_NEXT_SCK0” 状态到 “FULL_BUF_IN_SCK1” 状态的转换（该转换使用公式 “not FV”）。
43. 在 “PARTIAL_BUF_IN_SCK0”、
“PARTIAL_BUF_IN_SCK1”、

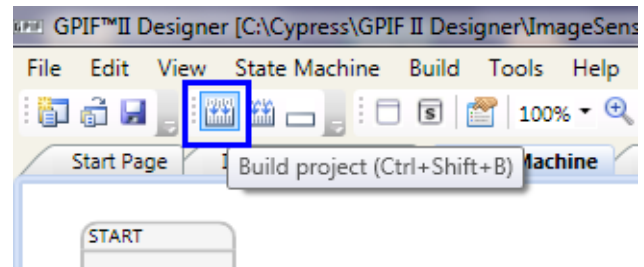
“FULL_BUF_IN_SCK0” 和
“FULL_BUF_IN_SCK1” 状态中添加操作
“Intr_CPU”。

图 38 显示的是最后状态机。与图 12 相比，主要区别在于该图添加了 PUSH_DATA 状态，这样可以适应任何状态中最多两个转换。

44. 通过选择 “File-Save Project As” 保存该项目。

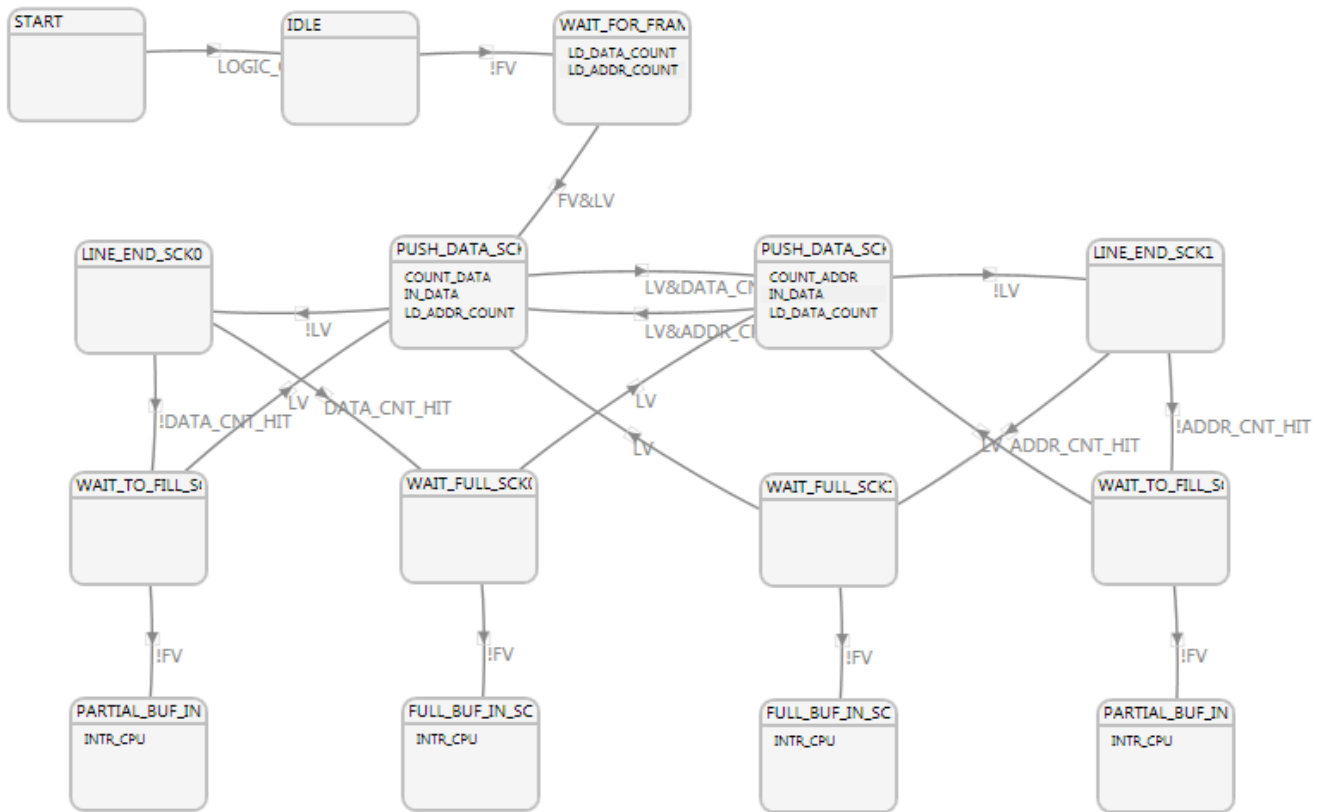
45. 使用图 37 中加亮显示的 Build 图标编译该项目。项目输出窗口指出的是编译的状态。

图 37. BUILD 按键



46. 检查项目的输出。在项目目录中，该输出显示为一个名称为 **cyfxgpif2config.h** 的头文件。如果检查该头文件，您会发现 GPIF Designer II 已经创建了各 GPIF II 内部设置阵列。FX3 固件将使用这些设置来配置 GPIF II 并定义其状态机。请勿直接编辑该文件，而应该通过 GPIF II Designer 进行操作。

图 38. 最后状态机的框图



3.6.4 编辑 GPIF II 接口的详细内容

本章节介绍的是如何修改接口设置（若需要）。例如，如果图像传感器/ASIC 具有 16 位宽的数据总线，那么您需要修改 GPIF II 接口，这样才能适应该数据总线。请按照下列步骤进行操作：

- ### 3.6.4 编辑 GPIF II 接口的详细内容
- 本章节介绍的是如何修改接口设置（若需要）。例如，如果图像传感器/ASIC 具有 16 位宽的数据总线，那么您需要修改 GPIF II 接口，这样才能适应该数据总线。请按照下列步骤进行操作：
1. 打开 GPIF II Designer 中的 **ImageSensorInterface.cyfx** 项目。（不能直接编译该项目）。
 2. 依次选择 File->Save Project As。
 3. 在出现的对话框内合适的位置，使用合适的名称保存该项目。
 4. 关闭当前打开的项目（File->Close Project）。
 5. 打开步骤 3 所保存的项目。
 6. 在 **Interface Definition**（接口定义）选项卡中选择 **16 Bit** 选项，以设置 **Address/Data Bus Usage**（地址/数据总线的使用情况）。
 7. 打开 **State Machine**（状态机）选项卡。
 8. 在状态机画布中，双击 **WAIT_FOR_FRAME_START** 状态中的 **LD_DATA_COUNT** 操作。将计数器的限定值改为 8183。
 9. 对 **LD_ADDR_COUNT** 操作进行相同的操作。
 10. 保存项目。
 11. 编译项目。
 12. 在步骤 3 所选择的位置内，将生成的新 **cyfxgpif2config.h** 头文件复制到固件的项目目录中。如果有 **cyfxgpif2config.h** 文件，它将被覆盖。在附件源的 zip 文件中寻找固件项目目录 **USBVideoClass**。
-
- 注意：**如果您将 GPIF II 总线的宽度改为 32 位，请确保固件中 **iomatrix** 配置的 **isDQ32Bit** 参数被设置为 **CyTrue**。
-
- 下一节将介绍用于数据流和支持 UVC 的固件的 DMA 通道。

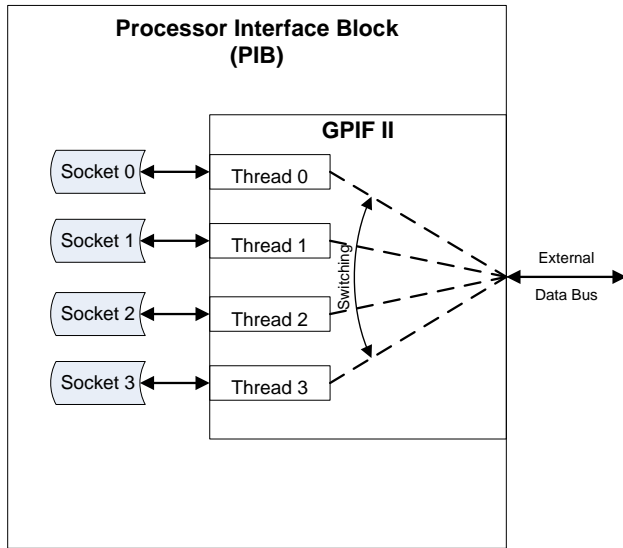
注意: 如果您将 GPIF II 总线的宽度改为 32 位, 请确保固件中 iomatrix 配置的 isDQ32Bit 参数被设置为 CyTrue。

下一节将介绍用于数据流和支持 UVC 的固件的 DMA 通道。

4. 设置 DMA 系统

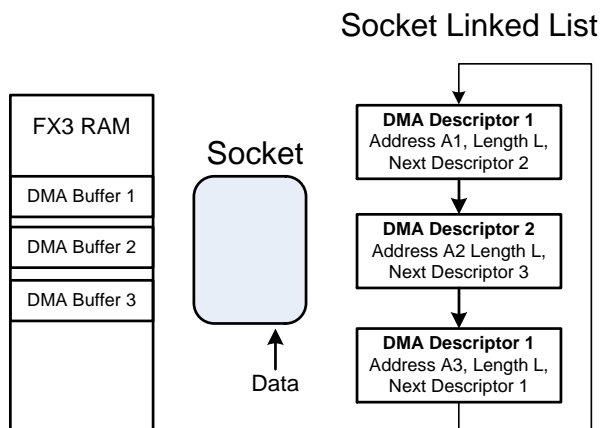
GPIF II 模块 (处理器接口模块 (PIB) 的一部分) 的工作频率可达 100 MHz, 而且数据总线宽度为 32 位 (400 MBps)。为了将数据传输到内部 DMA 缓冲区内, GPIF II 使用多个 GPIF 线程连接到 DMA 生产插座 (如 [章节 3.3](#) 中所述)。本应用使用了四个 GPIF 线程中的两个。它使用插座和 GPIF 线程的默认映射 ([图 39](#)) — 插座 0 与 GPIF 线程 0 相连, 插座 1 与 GPIF 线程 1 相连。在上一章节中设计的 GPIF II 状态机内实现 GPIF 线程切换。

图 39. GPIF II 插座/线程的默认映像



为了理解 DMA 传输, 下面四个图中继续使用了插座的概念。[图 40](#) 显示的是两个主插座属性、一个链接列表和一个数据路由器。

图 40. 插座根据 DMA 描述符列表路由数据

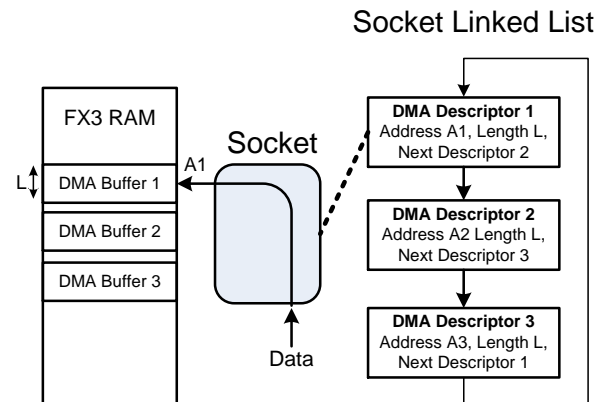


插座链接列表是主存储器中的一组数据结构, 这些结构又被称为 DMA 描述符。每个描述符指定了 DMA 缓冲区的地址

和长度, 以及指向下个 DMA 描述符的指针。插座运行时, 每次仅检索各 DMA 描述符中的一个描述符, 这样可以将数据路由到描述符地址和长度所指定的 DMA 缓冲区。传输 L 个字节后, 该插座会检索下一个描述符, 并继续将各字节传输到另一个 DMA 缓冲区内。

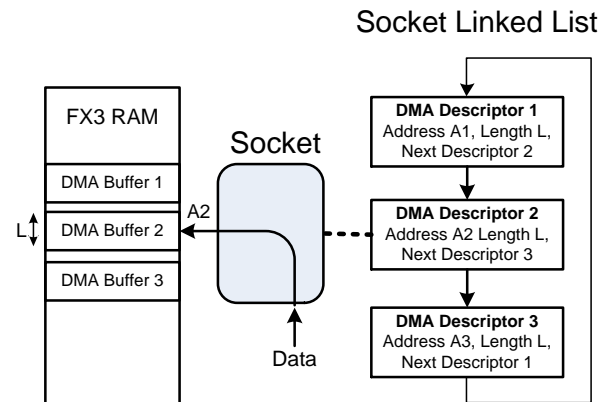
这结构使插座变得非常灵活, 因为可以在存储器中的任何位置上创建任何 DMA 缓冲区数量, 而且这些缓冲区可以自动被链接在一起。例如, [图 40](#) 中的插座以重复循环检索 DMA 描述符。

图 41. 使用 DMA 描述符 1 运行的插座



在 [图 41](#) 中, 该插座加载了 DMA 描述符 1, 并且传输操作开始于 A1 地址的字节, 直到传输 L 个字节结束。这时, 它将检索 DMA 描述符 2 并对其地址和长度设置 (A2 和 L) 进行相同的操作 ([图 42](#))。

图 42. 使用 DMA 描述符 2 运行的插座



在 [图 43](#) 中, 该插座检索第三个 DMA 描述符, 并传输从 A3 地址开始的数据。传输 L 个字节后, 将使用 DMA 描述符 1 重复该序列。

图 43. 使用 DMA 描述符 3 运行的插座

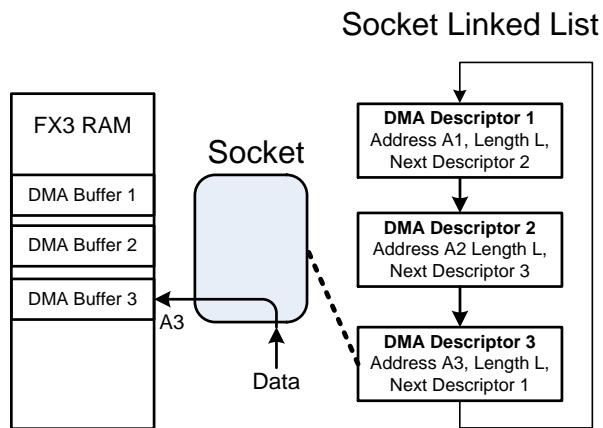
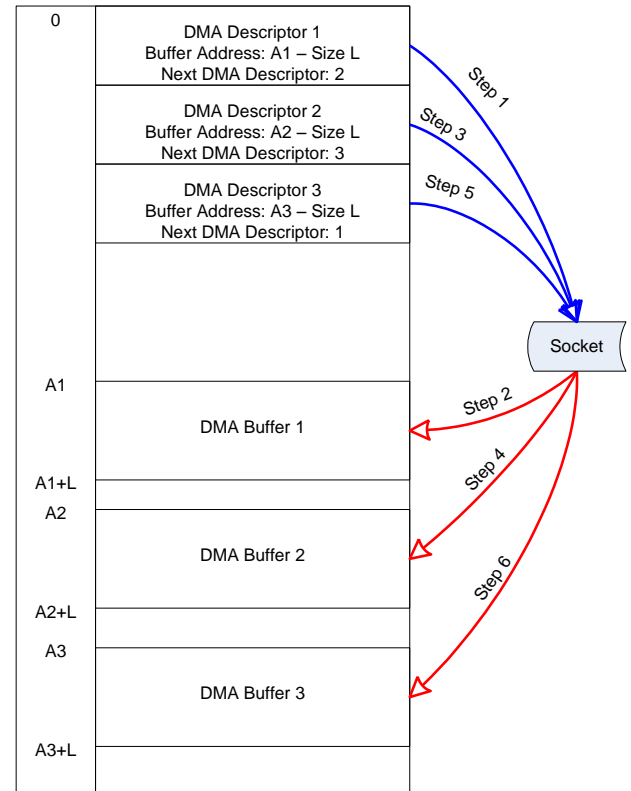


图 44 显示的是 DMA 数据传输的详细内容。本示例使用了三个长度为 L 的 DMA 缓冲区，这些缓冲区通过链接方式形成一个圆形循环。FX3 存储器地址位于左边。蓝色箭头表示该插座从存储器加载插座链接列表的描述符。红色箭头显示的是结果数据的路径。下面各步骤描述的是在数据转移到内部 DMA 缓冲区时，需要对插座进行的操作序列。

步骤 1：将存储器中的 DMA 描述符 1 加载到插座内。获取有关 DMA 缓冲区位置（A1）、DMA 缓冲区大小（L）和下一个描述符（DMA 描述符 2）的信息。转到步骤 2。

步骤 2：将数据传输到开始于 A1 地址的 DMA 缓冲区位置。传输完 DMA 缓冲区大小 L 的数据量后，转到步骤 3。

图 44. DMA 传输示例



步骤 3：加载当前 DMA 描述符 1 所指向的 DMA 描述符 2。获取有关 DMA 缓冲区的位置（A2）、DMA 缓冲区的大小（L）以及下一个描述符（DMA 描述符 3）的信息。转到步骤 4。

步骤 4：将数据传输到开始地址为 A2 的 DMA 缓冲区所在的位置。传输完数量大小为 DMA 缓冲区 L 的数据后，转到步骤 5。

步骤 5：加载当前 DMA 描述符 2 所指向的 DMA 描述符 3。获取有关 DMA 缓冲区的位置（A3）、DMA 缓冲区的大小（L）以及下一个描述符（DMA 描述符 1）的信息。转到步骤 6。

步骤 6：将数据传输到开始地址为 A3 的 DMA 缓冲区所在的位置。在传输完数量大小为 DMA 缓冲区 L 的数据后，转到步骤 1。

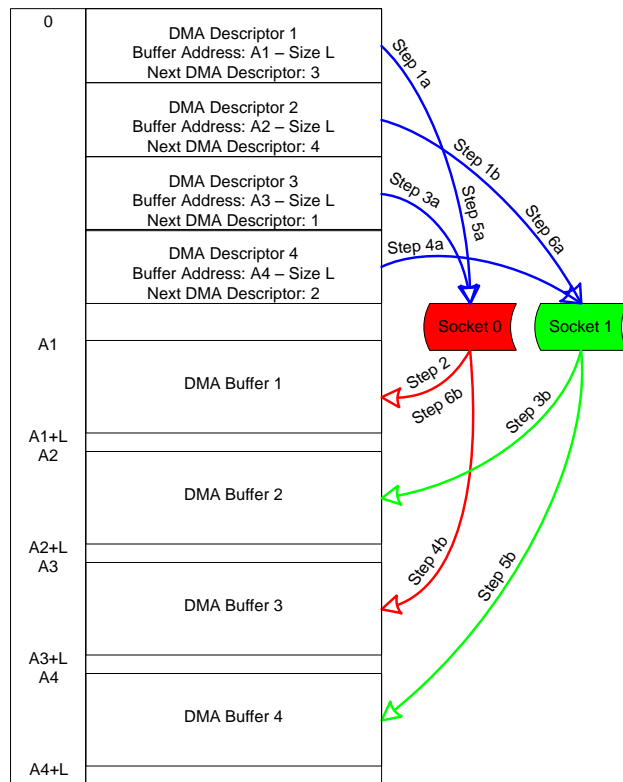
该简单方案在摄像头应用中存在问题。插座检索存储器中下一个 DMA 描述符所需要的时间通常为 1 微秒。如果在视频线中间暂停了该传输，将丢失视频数据。为了避免发生这种情况，可以将 DMA 缓冲区大小设置为的视频线长度的倍数。这样，DMA 缓冲区切换造成的暂停会与视频线无效事件（LV=0）同步发生。然而，在某些情况（如视频分辨率发生改变时），该方法缺少灵活性。

设置 DMA 缓冲区大小等于线大小也不是一个好方案，因为这样 BULK 传输将不会利用 USB 3.0 的最大突发速率。

USB 3.0 允许各 BULK 端点最大可达 16 个突发 x 1024 个字节。这就是将 DMA 缓冲区大小设置为 16 KB 的原因。

利用插座在一个时钟周期内进行切换而不需要延迟的特性，该方案更具有优势。因此，建议使用两个插座将数据存储在四个交错的 DMA 缓冲区内。图 45 显示的是使用了双插座的数据传输以及执行的各步骤。DMA 缓冲区的插座 0 和插座 1 访问分别使用红色和绿色箭头（数据路径的单独插座）区分。每一步骤中的 ‘a’ 和 ‘b’ 部分均同时发生。通过硬件的此并行操作可以消除 DMA 描述符的检索死时间并允许 GPIF II 连续将数据流动到内部存储器内。这些步骤同图 13 中的 “Step” 线相对应。

图 45. 使用双插座的无缝传输



步骤 1: 启动各插座时，插座 0 和插座 1 分别加载 DMA 描述符 1 和 DMA 描述符 2。

步骤 2: 当数据可用时，插座 0 将数据传输到 DMA 缓冲区 1 内，该传输的长度为 L。传输结束后，请转到步骤 3。

步骤 3: GPIF II 切换 GPIF 线程，从而切换进行数据传输的插座。插座 1 开始将数据传输到 DMA 缓冲区 2 内，同时插座 0 将数据加载到 DMA 描述符 3 内。在插座 1 完成传输量为 L 的数据时，插座 0 会准备好将数据传输到 DMA 缓冲区 3 内。

步骤 4: GPIF II 现在切换回到原始的 GPIF 线程。这时，插座 0 将长度为 L 的数据传输到 DMA 缓冲区 3 内。同时，插

座 1 将数据传输到 DMA 描述符 4 内，以便准备好将数据传输到 DMA 缓冲区 4 内。插座 0 完成传输量为 L 的数据后，请转到步骤 5。

步骤 5: GPIF II 将插座 1 中的数据布线到 DMA 缓冲区 4 内。同时，插座 0 通过加载 DMA 描述符 1 准备好将数据传输到 DMA 缓冲区 1 内。请注意，步骤 5 与步骤 1a 相同，但它的插座 1 未被启动，另外数据传输是同步进行的。

步骤 6: GPIF II 再次切换插座后，插座 0 开始将长度为 L 的数据传输到 DMA 缓冲区 1 内。这时，由于 UIB 消耗插座已经用尽，因此 DMA 缓冲区为空。同时，插座 1 将数据加载到 DMA 描述符 2 并准备将数据传输到 DMA 缓冲区 2 内。现在周期将转到执行路径的步骤 3。

只有消费端 (USB) 为空并及时释放了 DMA 缓冲区 (这样可以接受 GPIF II 中的下个视频数据块) 时，插座才能传输视频数据。如果消费端的速度不够快，插座将丢失数据因为 DMA 缓冲区的写操作被忽略。这样，字节计数器将不再与实际传输同步，而且此现象将传播到下一帧。因此每一帧的结尾处都需要一个清除机制。该机制在清除章节中进行了介绍。

根据图 12 所示的流程图，一个帧传输会以下面的四个可能状态结束：

- 插座 0 已经传输已满的 DMA 缓冲区
- 插座 1 已经传输已满的 DMA 缓冲区
- 插座 0 已经传输未了的 DMA 缓冲区
- 插座 1 已经传输未了的 DMA 缓冲区

对于未了的 DMA 缓冲区，CPU 需要将该缓冲区传送到 USB 消费者。

通过使用 `uvc.c` 文件中名称为 `CyFxFxUVCAppInit` 的函数可以启动 DMA 通道。在 `CyFxFxUVCAppInit` 函数中的 “`dmaMultiConfig`” 结构中对 DMA 通道的详细配置内容进行了自定义。它的类型被设置为 `MANUAL_MANY_TO_ONE`。

另外，将数据流动到 USB 3.0 主机的 USB 端点被配置为使能 16 突发 x 1024 字节的 BULK 端点。通过使用传送到 “`CyU3PSetEpConfig`” 函数中的 “`endPointConfig`” 结构 (端点常量设置为 “`CY_FX_EP_BULK_VIDEO`”) 将可实现该操作。

4.1 DMA 缓冲区的相关内容

本节介绍了如何创建 FX3 DMA 缓冲区，并将其用于该应用。

- DMA 通道的组成部分在 [章节 3.3](#) 中进行了介绍。
- 在这应用中，GPIF II 单元作为产生端，而 FX3 USB 单元作为消费端。
- 这应用使用了 GPIF II 模块中的 GPIF 线程切换性能以避免丢失数据。
- 当生产插座加载一个 DMA 描述符时，它将检查相关的 DMA 缓冲区以确定该缓冲区是否能够进行写操作。生产插座将其状态修改为“活动”。如果 DMA 缓冲区为空，它会把数据写入到 FX3 RAM 内。生产插座锁定 DMA 缓冲区，以用于进行写操作。
- 当生产插座完成写入 DMA 缓冲区后，它会释放锁定，以便消费插座访问 DMA 缓冲区。该操作称为“缓冲包装”或“包装”。然后，DMA 单元会将 DMA 缓冲区传送给 FX3 RAM。生产插座已经提供了一个 DMA 缓冲区。缓冲区只会在生产端不在填充时被包装。因而，状态框中才存在 FV=0 测试。
- 如果 DMA 缓冲区被完全填满，并且在帧期间重复填充操作，则包装操作会自动进行。生产插座释放 DMA 缓冲区上的锁定，将其传送到 FX3 RAM 内，切换到空 DMA 缓冲区，然后继续写入视频数据流。
- 当消耗插座加载了一个 DMA 描述符时，它将检查相关的 DMA 缓冲区以确定该缓冲区是否能够进行读操作。如果 DMA 缓冲区已被托付，消耗插座会将它的状态修改为“活动”去读取 FX3 RAM 中的数据。消耗插座锁定 DMA 缓冲区，以执行读操作。
- 当消耗插座已经读取了 DMA 缓冲区中所有的数据后，它会释放锁定以便使生产插座访问 DMA 缓冲区。消耗插座已经消耗了 DMA 缓冲区。
- 如果生产插座和消耗插座使用了相同的 DMA 描述符，那么 DMA 缓冲区的满/空状态将通过 DMA 描述符和插座间的事件在生产插座和消耗插座之间自动通信。
- 在本应用中，由于 CPU 需要添加一个 12 字节的 UVC 标头，因此生产插座和消耗插座需要加载不同 DMA 描述符组。生产插座加载的 DMA 描述符指向的是 DMA 缓冲区，这些缓冲区位于离消耗插座所加载的 DMA 描述符指向的相应 DMA 缓冲区 12 字节偏移的位置上。
- 由于生产插座和消耗插座使用了不同的 DMA 描述符，因此 CPU 必须管理生产插座和消耗插座之间 DMA 缓冲区状态的通信。这就是 DMA 通道实现被称为“手动 DMA”通道的原因。
- GPIF II 模块生产 DMA 缓冲区后，将通过中断通知 CPU。然后，CPU 添加标头信息并将 DMA 缓冲区传送到消费端（USB 接口模块）。
- 在 GPIF II 侧，DMA 缓冲区中的视频数据被自动包装，并被传送到 FX3 RAM（帧中的最后 DMA 缓冲区除外）。不需要 CPU 的干预。
- 在帧结尾处，最后的 DMA 缓冲区很大机会没有被完全充满。这时，CPU 会手动包装 DMA 缓冲区，并将该缓冲区传送到 FX3 RAM。这种操作称为“强制包装”。

5. FX3 固件

本应用笔记的示例固件位于源压缩文件中的 **USBVideoClass** 文件夹下面。请参考 [AN75705 — “EZ-USB FX3 入门”](#)，了解如何将固件项目导入到 Eclipse 工作区内。该示例固件可进行编译及枚举，但用户必须通过使用 **sensor.c** 和 **sensor.h** 文件为特定的图像传感器定制固件，以便可以从图像传感器流动视频。这些文件被作为示例

表 8. 示例项目文件

| 文件 | 说明 |
|----------------------------|--|
| sensor.c | <ul style="list-style-type: none"> 定义 SensorWrite2B、SensorWrite、SensorRead2B 和 SensorRead 函数，以 I²C 接口对图像传感器配置进行读写操作。这些函数假定图像传感器的 I²C 总线上的寄存器地址的宽度是 16 位的。 定义 SensorReset 函数可以控制图像传感器的复位线，另外通过使用 SensorInit 函数可以检测 I²C 连接并将图像传感器配置为默认的数据串流模式（使用 SensorScaling_HD720p_30fps 函数） 定义 SensorScaling_VGA 和 SensorScaling_HD720p_30fps 函数可以将图像传感器配置为所需要的数据串流模式。这些函数是占位符函数，因此必须使用特定于图像传感器的配置指令填充它们。 定义 SensorGetBrightness 和 SensorSetBrightness 函数可以对亮度值进行读写操作，该亮度值位于图像传感器的亮度控制寄存器中。这些函数是占位符函数，因此必须使用特定于图像传感器的配置指令填充它们。 |
| sensor.h | <ul style="list-style-type: none"> 包含用于图像传感器的常量（它的 I²C 从器件地址和 GPIO 复位编号）。在该文件内，您必须定义图像传感器的 I²C 从地址。 包含定义在 sensor.c 中的所有函数的声明。 |
| camera_ptzcontrol.c | <ul style="list-style-type: none"> 定义用于读和写摄像机的 PTZ 值的函数。这些函数是占位符函数，因此必须使用特定于图像传感器配置的指令填充它们。 取消注释 uvc.h 中的 “#define UVC_PTZ_SUPPORT” 线，以使能 PTZ 控制代码的占位符。 |
| camera_ptzcontrol.h | <ul style="list-style-type: none"> 包含用于 PTZ 控制实现的常量 包含了定义在 camera_ptzcontrol.c 文件中的所有函数的声明。 |
| cyfctx.c | <ul style="list-style-type: none"> 无需改动。使用与本应用笔记相关的项目提供的文件。 它包含了各种变量和函数。RTOS 和 FX3 API 库将这些变量用于存储器映射，而 FX3 API 库将这些函数用于存储器管理。 |
| cyfxgpif2config.h | <ul style="list-style-type: none"> 头文件由 GPIF II Designer 工具生成。无需更改。如果需要修改接口，则 GPIF II Designer 工具会生成一个新的头文件。它将取代旧的头文件。 包含传送到 uvc.c 文件中的 API 调用的结构和常量，以启动和运行 GPIF II 状态机。 |
| uvc.c | <ul style="list-style-type: none"> UVC 应用的主源文件。修改代码以支持其他控制功能（亮度和 PTZ 控制以外）或添加各种视频数据串流模式的支持时，需要进修改该文件。 它包括下面各函数： <ul style="list-style-type: none"> Main: 启动 FX3 器件、设置缓存、配置 FX3 I/O 和启动 RTOS 内核。 CyFxApplicationDefine: 定义 RTOS 执行的两个应用线程 UVCAppThread_Entry: 第一个应用线程函数，它调用 FX3 内部模块的启动函数，枚举器件，然后处理视频数据传输 UVCAppEP0Thread_Entry: 第二个应用线程函数，它等待指出已收到特定于 UVC 请求的事件并调用相应的函数进行处理这些请求 UVCHandleProcessingUnitRqts: 处理 VC 请求，用于处理单元性能 UVCHandleCameraTerminalRqts: 处理 VC 请求，用于输入终端性能 UVCHandleExtensionUnitRqts: 处理 VC 请求，用于扩展单元性能 UVCHandleInterfaceCtrlRqts: 处理普通的 VC 请求（不用于任何终端或单元） UVCHandleVideoStreamingRqts: 处理 VS 请求作修改数据串流模式 CyFxUVCAppInDebugInit: 初始化用于打印调试信息的 FX3 UART 模块 CyFxUVCAppInI2CInit: 初始化图像传感器配置的 FX3 I²C 模块 |

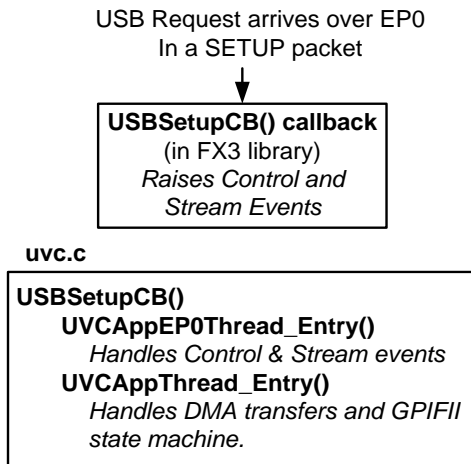
项目中的占位符，您必须在该位置上实现代码以配置图像传感器。

如果使用 Aptina 传感器板，如本应用笔记的 [章节 6](#) 所述的内容，并且签署了 Aptina NDA，赛普拉斯将提供 **sensor.c** 和 **sensor.h** 文件，用以同该特定传感器相应。通过本项目所提供的预编译加载图像可以尝试 **FX3-Aptina** 而不需要使用 NDA（[章节 7](#)）。表 8 汇总了各种代码模块以及用于实现每个模块的各个函数。

| 文件 | 说明 |
|---------------------------|---|
| | <ul style="list-style-type: none"> • CyFxCvAppInit: 初始化 FX3 GPIO 模块、处理器模块（GPIF II 为处理器模块的一部分）、传感器（配置为 720p 30fps）、枚举的 USB 模块、USB 传输的端点配置存储器以及用于将数据从 GPIF II 传输到 USB 的 DMA 通道配置 • CyFxCvAppGpifInit: 初始化 GPIF II 状态机并启动它 • CyFxCvGpifCB: 处理由 GPIF II 状态机生成的 CPU 中断 • CyFxCvAppCommitEOF: 将 GPIF II 状态机提供的部分 DMA 缓冲区调配给 FX3 RAM • CyFxCvAppInDmaCallback: 跟踪从 FX3 到主机的输出视频数据 • CyFxCvAppInUSBSetupCB: 处理所有由主机发送的控制请求，设置的事件表示已接收到由主机发送的 UVC 特定请求，并检测串流停止的时间 • CyFxCvAppInUSBEventCB: 处理各个 USB 事件，如暂停、线缆断开连接、复位和恢复等事件 • CyFxCvAppInAbortHandler: 当发生任何错误或检测到串流停止请求时，将停止流视频数据 • CyFxCvAppErrorHandler: 错误处理函数。这是一个占位符函数，它允许您执行错误处理（若需要） • CyFxCvAddHeader: 在有效串流期间将 UVC 标头添加到视频数据中 |
| cyfxuvcdscr.c | <ul style="list-style-type: none"> • 包含 UVC 应用的 USB 枚举描述符。如果需要修改帧速率、图像分辨率、位深度或支持的视频控制，则需要修改该文件。UVC 规范包含了所有需要的详细信息。 |
| uvc.h | <ul style="list-style-type: none"> • 包含各个用于修改应用行为的开关，以便打开/关闭 PTZ 支持、调试接口以及帧计数的调试打印。 • 包含的各常量通常用于 uvc.c、cyfxuvcdscr.c、camera_ptzcontrol.c 以及 sensor.c 文件。 |
| cyfx_gcc_startup.s | <ul style="list-style-type: none"> • 该汇编源文件包含了 FX3 CPU 启动代码。它具有用于设置堆栈和中断向量的函数。 • 无需改动。 |

首先，固件将初始化 FX3 CPU 并配置它的 I/O。然后，它将调用函数（**CyU3PKernelEntry**），以启动 ThreadX 实时操作系统（RTOS）。固件创建了两个应用线程：**uvcAppThread** 和 **uvcAppEP0Thread**。RTOS 将调配资源执行这些应用线程，并处理执行应用线程函数的顺序，即：分别执行 **UVCAppThread_Entry** 和 **UVCAppEP0Thread_Entry**。图 46 显示的是基本的程序结构。

图 46. 相机项目结构



5.1 应用线程

两个应用线程实现并行功能。**UVCAppThread** (应用) 线程用于管理视频数据流。它在启动流前等待流事件，在启动流后会调配 DMA 缓冲区，并在每一帧后或停止流时清除 FIFO。

固件将处理通过 EP0 的 UVC 特定控制请求 (SET_CUR、GET_CUR、GET_MIN 和 GET_MAX)，如亮度、PTZ 和 PROBE/COMMIT 控制。特殊类别的控制请求由应用线程中的 **CyFxFxUVCAppInUSBSetupCB** (CB=Callback) 函数处理。每当 FX3 接收到这些控制请求的中的某一个时，该函数将引发相应的事件，并立即释放主应用线程以便执行它的其他并行任务。然后，EP0 线程将触发这些事件，以服务类别特定的请求。

5.2 初始化

UVCAppThread_Entry 将调用 **CyFxFxUVCAppInDebugInit** 来初始化 UART 的调试能力，调用 **CyFxFxUVCAppInI2CInit** 来初始化 FX3 的 I²C 模块，调用 **CyFxFxUVCAppInInit** 来初始化其余所需模块、DMA 通道和 USB 端点。

5.3 枚举

在 **CyFxFxUVCAppInInit** 函数中，调用了 **CyU3PUsbSetDesc** 函数，以确保 FX3 被枚举为一个 UVC 器件。在 **cyfxuvcdscr.c** 文件中定义了 UVC 描述符。这些描述符的定义用于以非压缩的 YUY2 格式传输每像素为 16 位的图像传感器，其设置为 1280 x 720 像素、30 FPS。若需要修改这些设置，请参考第 2.3.1 节中的内容。

5.4 使用 I²C 接口配置图像传感器

使用 FX3 的 I²C 主控模块来配置图像传感器。使用 **SensorWrite2B**、**SensorWrite**、**SensorRead2B** 和 **SensorRead** 函数 (定义在 **sensor.c** 文件中)，通过 I²C 接口对图像传感器配置进行读写操作。**SensorWrite2B** 和 **SensorWrite** 函数将调用标准的 API **CyU3PI2cTransmitBytes**，向图像传感器写入数据。

SensorRead2B 和 **SensorRead** 将调用标准的 API **CyU3PI2cReceiveBytes**，读取图像传感器内的数据。有关更多这些 API 的详细信息，请查阅 **FX3 SDK API 指南** 中的内容。

5.5 启动视频流

USB 主机应用 (如 VLC 播放器、AMCap 或 VirtualDub) 位于 UVC 驱动器之上，以便显示视频、将 USB 接口和 USB 备用设置组合到一个流视频 (通常为接口 0 备用设置 1)，并发送 PROBE/COMMIT 控制。该主机指示符表示快要启动一个视频数据流。发生流事件时，USB 主机应用会向 FX3 请求图像数据；FX3 应将图像数据从图像传感器发送给 USB 3.0 主机。在固件中，**UVCAppThread_Entry** 函数是一个无限循环。当没有流事件时，主应用线程将在该循环下等待，直到发生流事件为止。

注意：如果没有流事件，FX3 不需要传输任何数据。所以，不必要初始化 GPIF II 状态机来传输数据。否则，主机应用从 DMA 缓冲区读取数据前，DMA 缓冲区会被填充，并且 FX3 将传送一个坏帧。因此，只有发生了流事件时，才能初始化 GPIF II 状态机。

当 FX3 接收到流事件时，主应用线程将调用 **CyFxFxUVCAppGpifInit** 函数来启动 GPIF II 状态机。在该函数中，固件使用 **CyU3PGpifLoad** 函数将 GPIF II 参数 (寄存器设置和波形数据) 加载到 FX3 存储器内。然后，固件通过 **CyU3PGpifSMStart** 函数启动 GPIF II 状态机。将定义在 **cyfxgpif2config.h** 文件中的 **CyFxFxGpifConfig** 结构作为参数，并将其传递到 **CyU3PGpifLoad** 函数内。将启动状态名和启动条件作为参数使用，并将其传递到 **CyU3PGpifSMStart** 函数内。在 **cyfxgpif2config.h** 文件中定义了启动状态 (**START**) 和启动条件 (**ALPHA_START**)。**cyfxgpif2config.h** 文件由 GPIF II Designer 工具生成，如第 3.6 节中所述。

注意：FX3 SDK API 指南包含 GPIF II 关联函数的详细信息，如 **CyU3PGpifLoad** 和 **CyU3PGpifSMStart**。

5.6 设置 DMA 缓冲区

UVC 规范要求每个 USB 传输 (在本应用中即为每个 16 KB DMA 传输) 增加一个大小为 12 字节的标头。但是，FX3 架构要求所有与 DMA 描述符相关联的 DMA 缓冲区的大小必须为 16 字节的倍数。

由于在 DMA 缓冲区中保留了供 FX3 CPU 填充的 12 字节，因此 DMA 缓冲区大小不再是 16 字节的倍数。所以，DMA 缓冲区大小应为 16,384 减去 16，而不是减去 12。DMA 大小 (不包括 FX3 固件所添加的 12 字节标头) 为 16,384-16=16,368 字节。DMA 缓冲区的结构如下：12 字节的标头，16,368 视频字节，以及在 DMA 缓冲区终端的 4 个未使用字节。这样，DMA 缓冲区的实现最大空间为 16 x 1024 (16,384) 字节的 USB BULK 突发。

5.7 在视频串流期间中处理 DMA 缓冲区

CyFxCVAppInInit 函数为消耗事件创建了一个具有回调通知的 DMA 手动通道。该通知用于跟踪主机读取的数据量。视频帧传输结束后，将复位 DMA 通道，作为清除过程的一部分。只有传输完所有 DMA 缓冲区中的数据时，才可安全地复位 DMA 通道。如果复位了 DMA 通道，而管道中仍存在有效的图像帧数据，该数据将丢失。所以，该通知对固件的正常操作起着重要作用。

固件在流数据时处理 DMA 缓冲区。在主应用线程中，FX3 固件使用 **CyU3PDmaMultiChannelGetBuffer** 函数来检查 DMA 缓冲区。当 GPIF II 生成的 DMA 缓冲区被调配，或被 FX3 CPU 强制打包时，则 FX3 CPU 可使用它。在有效的帧期间，图像传感器使数据发生转移，并且 GPIF II 将提供填满的 DMA 缓冲区。此时，FX3 CPU 需要将 16,380 字节数据提交给 USB。

通常，帧结束时，最后的 DMA 缓冲区未饱和。在这种情况下，固件必须在生产端上强制打包 DMA 缓冲区，再触发一个生产事件，然后将 DMA 缓冲区中合适的字节数调配给 USB。强制打包 DMA 缓冲区 (GPIF II 生成) 由 GPIF II 回调函数 **callback function CyU3PDmaMultiChannelSetWrapUp** 执行。当 GPIF II 设置一个 CPU 中断时，将触发 **CyFxCVpifCB** 回调函数。取消置位帧有效信号时，将生成该中断，如 GPIF II 状态图所示。此时，通过设置 **hitFV** 变量可表示从图像传感器捕捉的帧已经结束。

UVC 标头包含了有关帧标识符和帧结束标志的信息。在帧的结束部分，FX3 固件设置第二个 UVC 标头字节的位 1，并切换它的位 0 (请查看 “**CyFxCVAddHeader**”)。

prodCount 和 **consCount** 变量分别跟踪每个 DMA 缓冲区的生产和消耗事件。这些变量有助于跟踪 DMA 缓冲区，从而能够确保已经从 FX3 FIFO 读取了所有数据。完成通过 USB 实现的帧传输时，固件可以使用这些变量来复位通道。

5.8 帧结束时进行清理

某帧结束时，GPIF II 状态机将生成 CPU 中断，从而启动前面介绍的事件串链。固件将使用 **UVCAppThread_Entry** 函数中的循环，从而等待 USB 端取得所有 DMA 缓冲区数据。设置 **hitFV** (由 GPIF 回调函数设置) 后，最后 DMA 缓冲区被调配给 USB，“**prodCount**” 的值等于 “**consCount**” 的值，并且固件执行如下操作：

- 复位 DMA 通道 (复位其 FIFO)
- 切换 UVC 标头的 FID 位
- 调用 **CyU3PGpifSMSwitch** 函数，GPIF II 状态机重启为 START 状态
- 等待 FV 再次转为高电平

5.9 终止视频流

共有三种方法可以终止图像流：断开主机与照相机的连接、关闭 USB 主机程序，或 USB 主机向 FX3 发送复位或暂停请求。所有这些操作都会触发 **CY_FX_UVC_STREAM_ABORT_EVENT** 事件 (请参考 **CyFxCVAppInAbortHdlr** 函数)。在 FX3 FIFO 中没有数据时，并非总会发生这操作。换句话说，需要作适当清除。固件会复位与流相关的变量，并复位 **UVCAppThread_Entry** 环中的 DMA 通道。由于不需要任何流，所以不会调用 **CyU3PGpifSMSwitch** 函数。固件这时会等待发生下一个流事件。

当关闭应用时，它将发送 Windows 平台上的发送清理请求，或发送 Mac 平台上备用设置为 0 的设置界面请求。接收到该请求时，流会停止。在

CY_U3P_USB_TARGET_ENDPT 及

CY_U3P_USB_SC_CLEAR_FEATURE 请求的 **switch**

case 语句下，**CyFxCVAppInUSBSetupCB** 函数将处理该请求。

5.10 增加“调试”接口

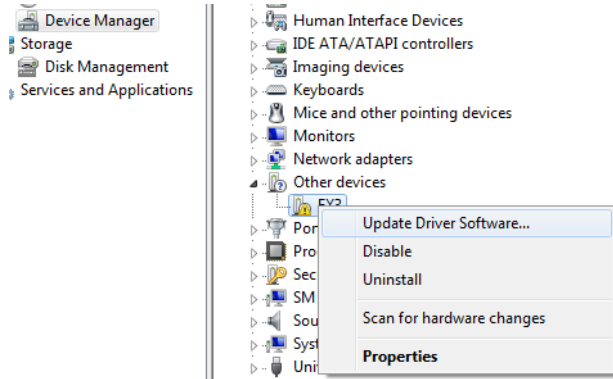
本部分介绍了如何向照相机应用增加第三个 USB 接口用于调试或其他自定义目的。

uvc.h 文件中的 **#define USB_DEBUG_INTERFACE** **switch** 语句使能了 **cyfxuvcdscr.c**、**uvc.h** 和 **uvc.c** 文件中的代码。示例提供了通过 I²C 对图像传感器寄存器进行的读和写操作。

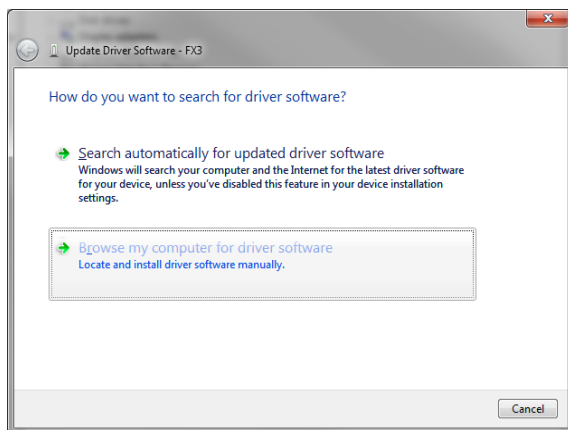
5.10.1 调试接口的详情

指定两个 BULK 端点用于该接口。EP 4 OUT 被配置为调试指令 BULK 端点，EP 4 IN 则被配置为调试响应 BULK 端点。当运行使能了调试接口的固件时，FX3 在枚举过程中报告了三个接口。前两个接口是 UVC 控制和流接口，第三个接口是调试接口。需要将第三个接口绑定到 FX3 SDK 的 **CyUSB3.sys** 驱动程序。您可以根据下面介绍的内容安装驱动程序。(示例使用的是 64 位 Windows 7 系统。XP 系统用户需要在 “.inf” 文件中添加 VID/PID，然后使用修改好的 “.inf” 文件在该接口上安装 **cyusb3.sys** 驱动程序。)

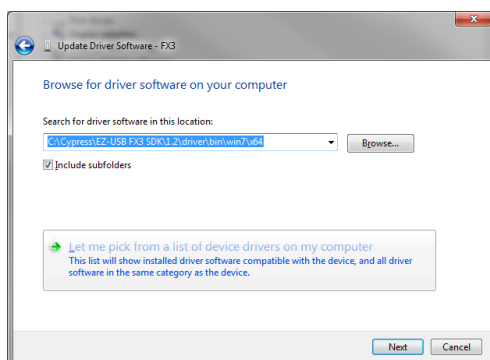
1. 打开器件管理器，右键点击“Other devices”下的 FX3（或同等选项），然后选中“Update Driver Software...”项



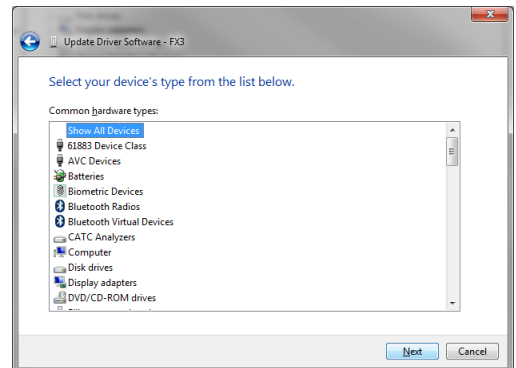
2. 在接下来显示的屏幕上，选中“Browse my computer for driver software”项



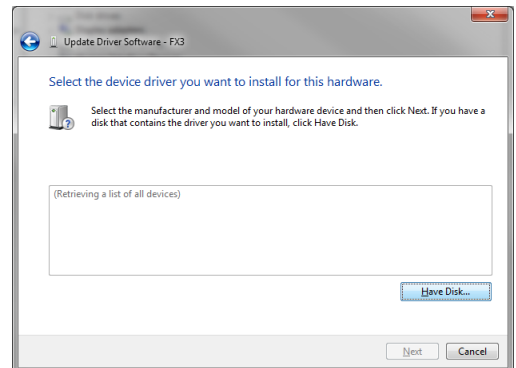
3. 然后选择“Let me pick from a list of device drivers on my computer”



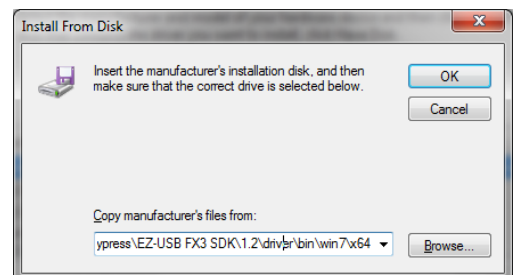
4. 点击“Next”



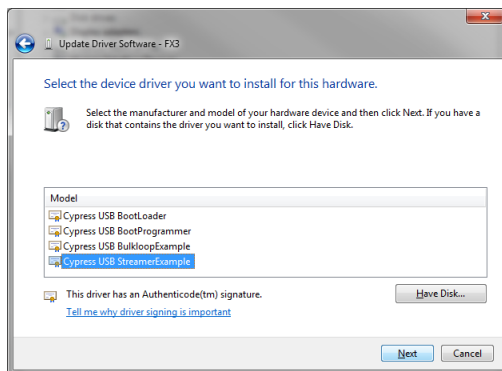
5. 点击“Have Disk...”，选中驱动程序



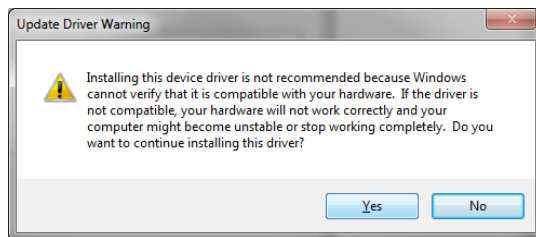
6. 浏览<SDK installation>\<version>\driver\bin\<OS>\x86 或\x64 文件夹中的 cyusb3.inf 文件，然后点击“OK”



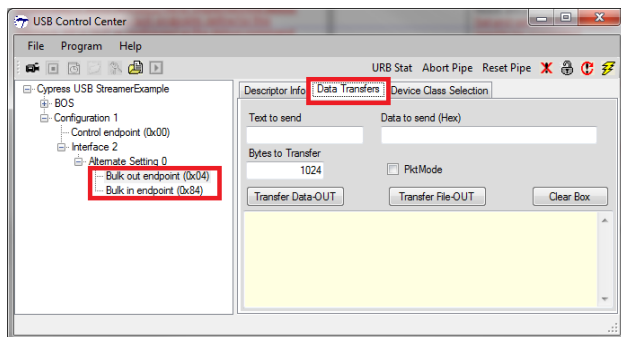
7. 选中一个操作系统版本，并通过点击“Next”进行安装



8. 点击警报对话框中的“Yes”（若出现）



在第三个接口上安装驱动程序后，器件将显示在赛普拉斯 USB 控制中心应用中。您可以在这里将 FX3 固件作为供应商特定的器件进行访问。当前实现的调试接口允许使用 EP4-OUT 指令和 EP4-IN 响应端点对图像传感器寄存器进行读和写操作。通过 Control Center 中的 <FX3 device name> → Configuration 1 → Interface 2 → Alternate Setting 0，访问这些端点，如下图所示。使用“Data Transfer”选项卡来发送一个指令或发送指令后接收到响应。

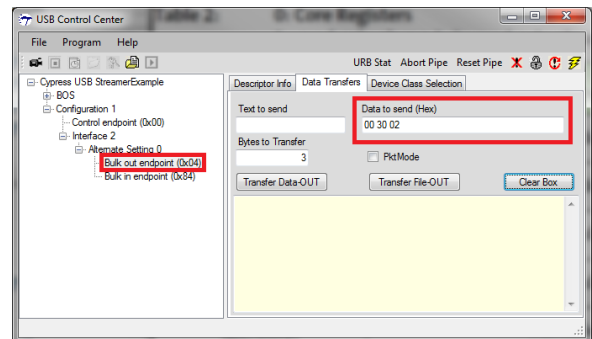


5.10.2 使用调试接口

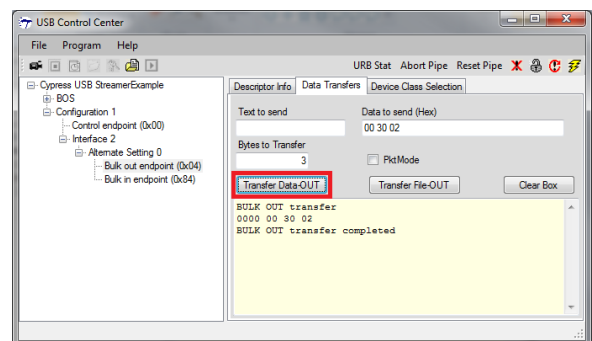
在调试接口中执行的四个指令分别是：单一读取、连续读取、单一写入和连续写入。在这里并没有错误检查，所以在输入指令时需要特别小心。您可以执行错误检查，以便确保功能正确。I²C 寄存器是 16 位的带宽，并且使用了 16 位进行寻址。

单一读取：

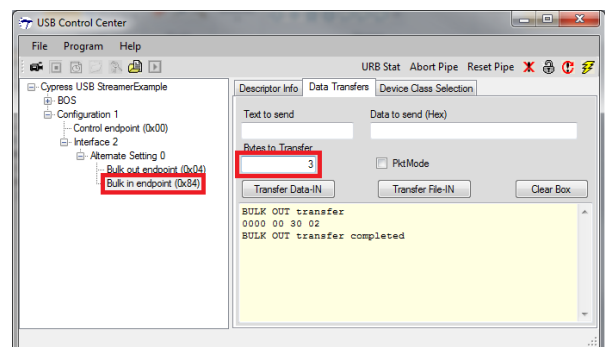
1. 在“Data transfers”选项卡下，选中指令端点并以十六进制的格式输入指令。单一读取的指令格式为 0x00 <寄存器地址高字节> <寄存器地址低字节>。下图显示的是寄存器地址 0x3002 的读指令。向十六进制的数框中输入指令时，请勿使用空格。例如，点击十六进制数据字段并输入“003002”。



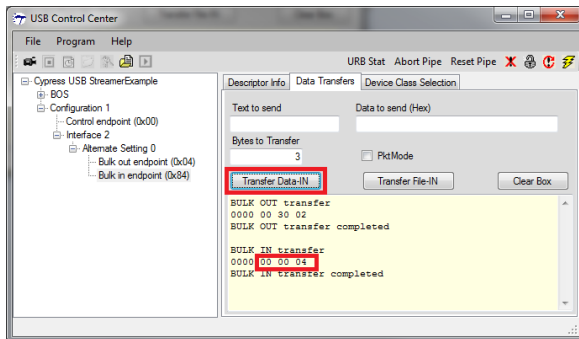
2. 点击“Transfer Data-Out”来发送指令。



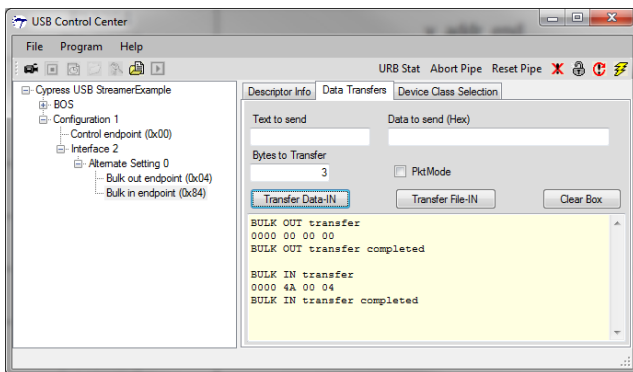
3. 选中响应端点并将“传输字节”字段设置为“3”，这样可以读取单一读取指令的响应。



4. 点击“Transfer Data-IN”，接收响应

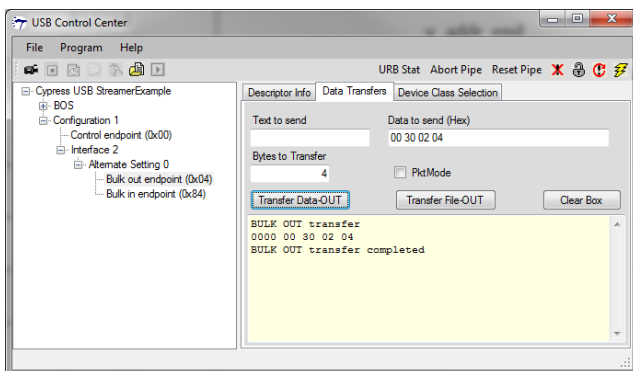


5. 响应的第一字节表示状态。状态为 0 时，表示指令已经通过；如果为其它状态值，则表示指令失败。后面各字节表示读回的寄存器值为 0x0004。该示例显示的是一个失败的传输，其中，状态字节为非 0 值，其余字节的值是前一个传输的过期值。

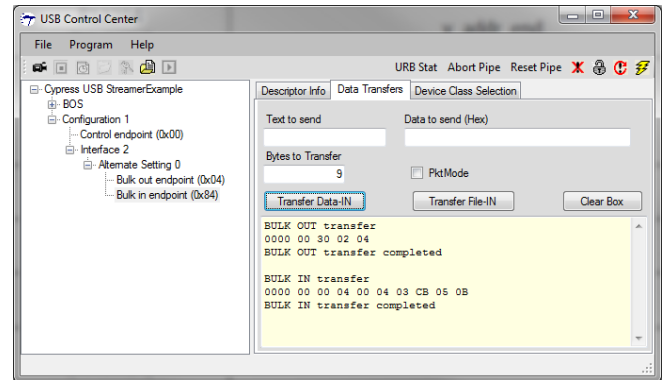


连续读取：

1. 在“Data transfers”选项卡下，选中指令端点并以十六进制的格式输入指令。连续读取的指令格式为 0x00 <寄存器地址高字节> <寄存器地址低字节> <N>。下图显示的是对开始地址为 0x3002 开始的四个（N=4）寄存器进行的读指令。

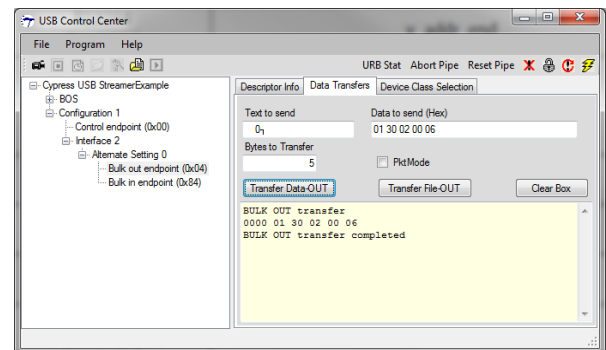


2. 在这种情况下，响应的“传输字节”为 $(N*2+1) = 9$ 。下图显示的是 FX3 读取的值。

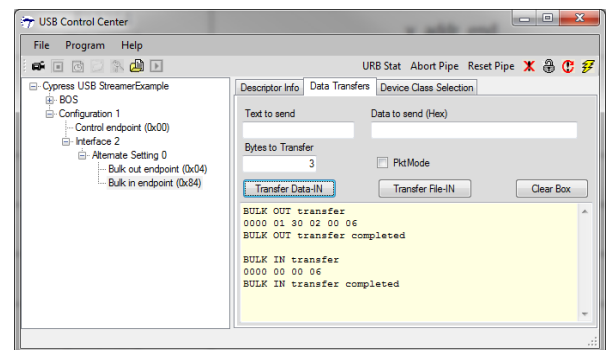


单一写入：

1. 在“Data transfers”选项卡中，选中指令端点并以十六进制的格式输入指令。单一写入的指令格式为 0x01 <寄存器地址高字节> <寄存器地址低字节> <寄存器值高字节> <寄存器值低字节>。下图显示的是向寄存器地址为 0x3002 的空间内写入 0x0006 值的写指令。

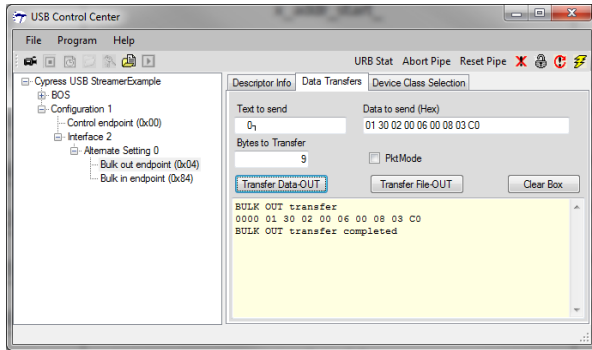


2. 单一写入的响应包含三个字节：<状态> <寄存器值高字节> <寄存器值低字节>。写入后将读取这些寄存器的值，因此您在指令中可以看到发送的同样值。

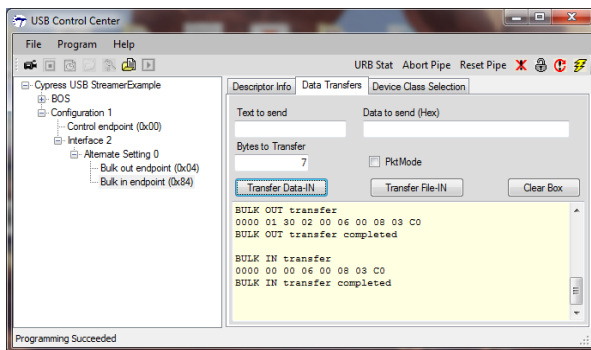


连续写入:

1. 在 “Data transfers” 选项卡下, 选中指令端点并以十六进制的格式输入指令。连续写入的指令格式为 0x01 <寄存器地址高字节> <寄存器地址低字节> (<寄存器值高字节> <寄存器值低字节>) * N 次数, 对各个寄存器进行 N 次写入操作。下图显示的是分别将 0x0006、0x0008 和 0x03C0 值连续写入寄存器 0x3002、0x3004 和 0x3006 (N=3) 内。



2. 响应格式为<状态> (<寄存器值高字节> <寄存器值低字节>) * N 值。
例如, 传输的总字节数为 $(2*N+1) = 7$ 。



6. 硬件设置

当前项目已经通过了测试, 其测试安装包括 FX3 DVK、Aptina 图像传感器 MT9M114 以及互联板。在赛普拉斯网站上的 **EZ-USB® FX3™ HD 720p 照相机套件** 中显示了如何得到这些组件的有关信息, 并总结如下:

6.1 硬件采购

1. 签署跟 Aptina 公司的保密协议 (向 fx3@cypress.com 邮箱发送邮件请求, 这样可以快速得到回复)。如果您已经签署了保密协议, 请将保密协议发到 fx3@cypress.com 邮箱内。验证保密协议后, 赛普拉斯将提供项目所需的特定于 Aptina 的 **sensor.c** 和 **sensor.h** 源文件。

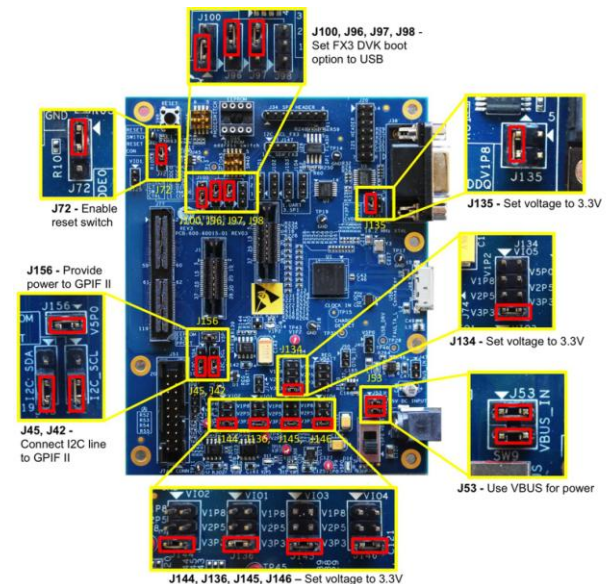
2. 从 **Aptina** 经销商那里购买 Aptina MT9M114 图像传感器接头板。
3. 购买 **EZ-USB FX3 开发套件 (CYUSB3KIT-001)**。
4. 请通过邮箱 fx3@cypress.com 联系赛普拉斯, 以便得到互联板。
5. 使用 USB 3.0 主机使能的电脑进行评估 SuperSpeed 的性能。

6.2 FX3 DVK 板设置

按照下面各步骤准备要测试视频应用的电路板:

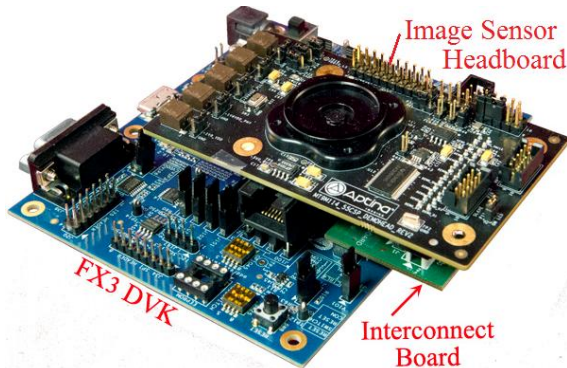
1. 按照图 47 中所示的内容配置 FX3 DVK 板上的跳线器。
请勿加载图中高亮显示以外的跳线器。

图 47. FX3 DVK 跳线器



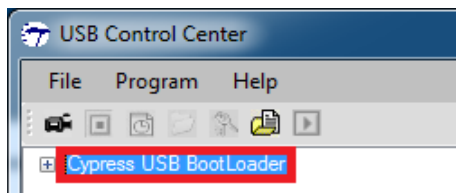
2. 将互联板插入 FX3 DVK J77。互联板上的连接器类型是唯一的, 另外插座是有方向性的, 所以只有正确地放置了它们的方向才算连接成功。
3. 将图像传感器模块连接到互联板。图 48 显示的三个电路板的装配。

图 48. 三个电路板的 720P 照相机装配



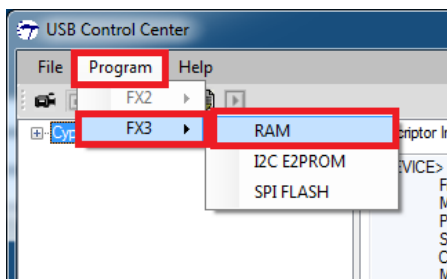
4. 使用了 DVK 提供的 USB 3.0 线缆将 FX3 DVK 电路板插到 USB 电脑上。
5. 使用 FX3 SDK 提供的 USB 控制中心应用将固件加载到电路板内。详细内容，请查阅 AN75705 — EZ-USB FX3 入门。以下是简要说明：
 - a. 启动 USB 控制中心。将其插入到 FX3 EV 电路板时，它被识别为 USB Bootloader（图 49）。

图 49. FX3 枚举为 Bootloader



- b. 依次选择“Program>FX3>RAM”，然后转到本应用笔记的附件所提供的 cyfxuvc.img 文件（图 50）。

图 50. 将代码加载到 FX3 RAM 内



7. 基于 UVC 的主机应用

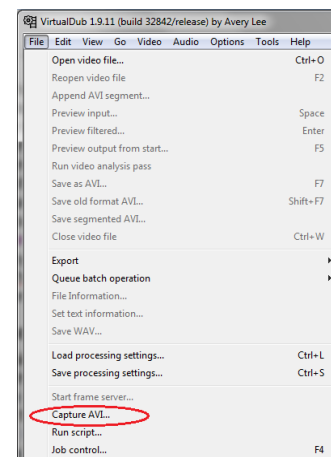
各种主机应用允许您使用 UVC 器件来显示并捕捉视频。VLC 媒体播放器很受欢迎。另一种广泛使用的 Windows 应用是 AMCAP。我们推荐 AMCap 的版本为 8.0，这是因为

执行数据流操作时，它的性能较稳定；其它更高版本的 AMCap，会使流嘈杂缓慢。另外两个 Windows 应用分别为 VirtualDub（开源应用）和 Debut Video Capture 软件。Linux 系统可以使用 V4L2 驱动程序和 VLC 媒体播放器进行视频流操作。可以在网上下载 VLC 媒体播放器。Mac 平台可以使用 FaceTime、iChat、Photo Booth 和 Debut Video Capture 软件创建同 UVC 器件相连的接口，以便执行视频流操作。

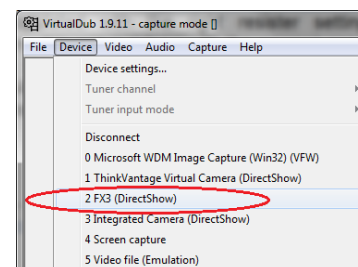
7.1 运行演示

图 48 中显示的 720P 套件的预编译代码映射文件位于 <http://www.cypress.com/?docID=41913> 网站上。按照下面各步骤运行该代码：

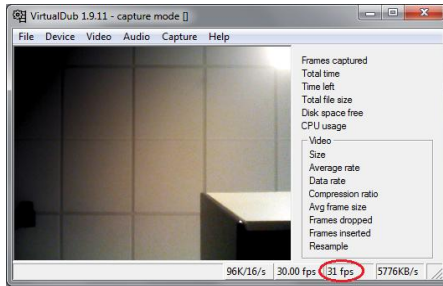
1. 将预编译固件镜像文件加载到 FX3 UVC 结构内，如第 6.1.2 节所述。
2. 此时，装置被重新枚举为 UVC 器件。操作系统将安装 UVC 驱动程序；此外不要求任何其它驱动程序。
3. 打开主机应用（例如，VirtualDub）。
4. 选择 Choose File → Capture AVI



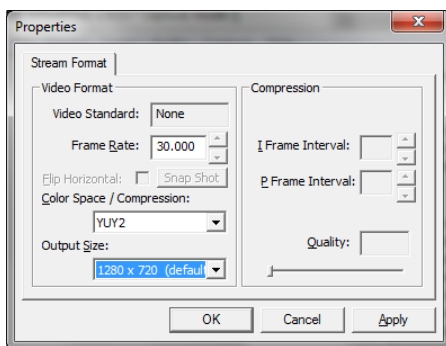
5. 选择 Device → FX3 (Direct Show)，这样会开始串流图像



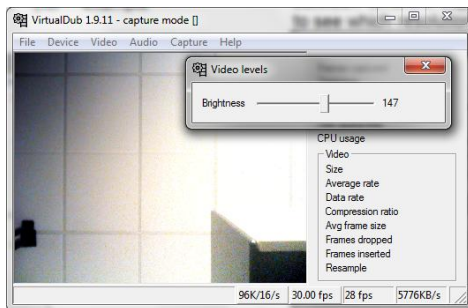
6. 右下方显示的是实际的帧率



7. 选择 Video → Capture Pin，可以指定受支持的分辨率并查看当前有效的分辨率。



8. 选择 Video → Levels 可修改亮度（通过修改滑条位置来修改数值）或其它受支持的控制指令。在 Video → Capture Filter，可找到其它控制指令。



8. 故障排除

如果出现黑屏，请按照下面各步骤进行调试：

1. 在 `uvc.h` 文件中存在“`DEBUG_PRINT_FRAME_COUNT`”开关。使能开关，以查看 FX3 是否有串流图像。该开关用于使能 UART 打印帧数量。短接 FX3 DVK 上跳线器 J101、J102、J103 和 J104 的引脚 1 和 2。这样可以将 UART 引脚连接到 FX3 DVK 上的 RS232 端口。使用 UART 线缆或 USB-UART 桥接器将 FX3 DVK 上的 UART 端口连接到 PC。打开能够访问 PC 上 COM 端口的 Hyperterminal、Tera Term 或其它工具。启动传

输前，请按照下面各项内容对 UART 进行配置：
115,200 波特、无奇偶校验位、1 个停止位、无流控以及 8 个数据位。这样应足以捕获调试打印。如果您在 PC 终端程序中没找到增量帧计数器，则 FX3 和图像传感器（GPIF 或传感器控制/初始化）之间的接口可能出现问题。

2. 如果您看到了增量帧计数器的打印，需要验证被发送出去的图像数据。USB 追踪可以显示每帧传输的数据量。
3. 为了检查被发送出去的每帧数据的总量，需要查找数据包。该数据包具有标头上设置的结束帧位。（表头的第二字节为 0x8E 或 0x8F，用以传输结束帧）。在一个帧上传传的图像数据（不包含 UVC 标头）的总量应为：宽度*高度*2。
4. 如果当前不是从 USB 追踪的数量，则图像传感器的设置或 GPIF 接口可能存在问题。
5. 如果图像数据的总量正确，并且主机应用程序仍未显示任何图像，请更换电脑主机。
6. 如果问题仍然存在，请创建赛普拉斯技术支持案例。

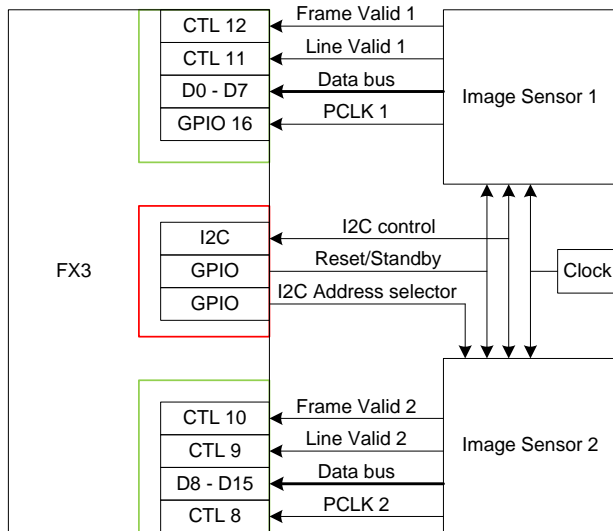
9. 连接两个图像传感器

主机应用程序（如：3-D 图像或移动跟踪）需要 FX3 同时传输两个图像传感器中的视频数据。如果传感器不同，则必须在图像传感器模块和 FX3 之间插入 FPGA，以便调整格式，将其并合成单一的视频数据通道。这种对两个不同的图像传感器进行的设置超出了本应用手册的范围。

一种常用的方法是使用相同的图像传感器。在本节中详细介绍了这种方法。

图 51 显示的是连接的详细信息。绿色模块在 GPIF II 里面，红色模块属于 FX3 低带宽外设（I²C/GPIO）的一部分。需要同步化两个传感器，以便获得相同的帧时序，并使 GPIF II 在 16 位数据总线上同步输入每个 8 位数据流。

图 51. 连接到 FX3 的两个相同的图像传感器



在图 51 中假设两个图像传感器的情况如下：

1. 每个图像传感器的总线宽度均为 8 位，因此 GPIF II 总线宽度为 16 位。
2. 两个图像传感器是同步的。因此，这两个图像传感器使用相同的时钟、LV 与 FV 转换和像素时序。也就是说这两个图像传感器的帧可以被精确地重叠。某些图像传感器具有外部触发输入，可以用来同步两个视频流。其他图像传感器可以使用不同的同步方法。可应用的传感器数据手册将对其进行了详细介绍。
3. 对于配置，这两个图像传感器使用 I²C。应用手册中使用的图像传感器需要 I²C 控制寄存器在完全相同的时间内进行写操作，以便实现各图像传感器模块之间的同步。使用 FX3 GPIO 引脚控制一个的传感器中的 I²C 地址来完成该操作。FX3 使用 I²C 写同时配置两个图像传感器。使用可配置的地址引脚切换到图像传感器上的不同 I²C 地址，FX3 可以读取各个传感器。
4. 应通过同一个 FX3 GPIO 输出引脚驱动每个传感器的复位信号。同样，每个传感器的备用引脚（如果存在的话）应共享另一个 FX3 GPIO 输出引脚。
5. 禁用图像传感器的自动配置。例如，在两种传感器中自动曝光，自动增益和自动白平衡等特性都被关闭。关断它们可以确保在图像传感器上进行集成和处理任何图像同步进行。这样可使两个传感器的帧同时从图像传感器输出。

如连接图所示，将帧有效 2、行有效 2 和 PCLK 2 信号连接到 FX3，但是 GPIF II 模块不使用它们，因为认为图像传感器是同步的。这些信号被连接到 FX3，因此在调试和开发过程中，可以通过监控这些信号来检查各个图像传感器间的同步精度。

9.1 使用 UVC 传输交错图像

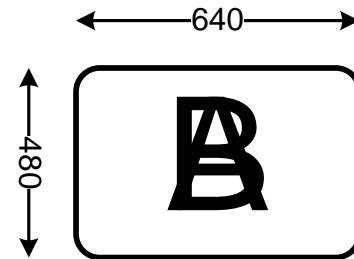
UVC 规范不能辨别所发送的图像中使用了多少图像传感器。它只能辨别一个图像参数组：帧的宽度和高度，以及每帧中总字节的数量。为了符合多个图像传感器的要求，UVC 驱动程序必须读取已修改的描述符，以便执行并通过内部一致性检查。如果这些一致性检查失败，UVC 驱动程序不能将图像数据传送给主机应用，导致应用失败。需要修改描述符，以便 UVC 驱动程序将额外帧识别为单一图像传感器。

下面两个示例说明了如何实现该操作。

9.1.1 示例 1：两个 640×480 的单色传感器

两个图像传感器可提供 640 x 480 单色（每像素 1 字节）数据。FX3 在每帧中同时接收到两个完整的图像，如图 52 所示。

图 52. FX3 从两个图像传感器接收到的数据



描述符继续报告 640 x 480 图像的尺寸。通过将 A 图像放置在 Y 数据中，并将 B 图像放置在 U 和 V 数据中，这样可以调节双倍数据的尺寸。

可通过下面公式计算每帧的字节数：

$$\text{每帧的字节数} = \text{每像素的字节数} \times \text{图像传感器数} \times \text{分辨率}$$

其中：

$$\text{分辨率} = \text{宽度（像素）} \times \text{高度（像素）}$$

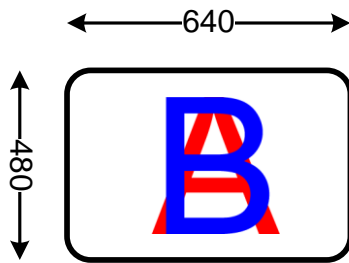
例如：

$$\text{每帧的字节数} = 1 \times 2 \times 640 \times 480 = 614,400 \text{ 字节}$$

9.1.2 示例 2：两个 640 x 480 的彩色传感器

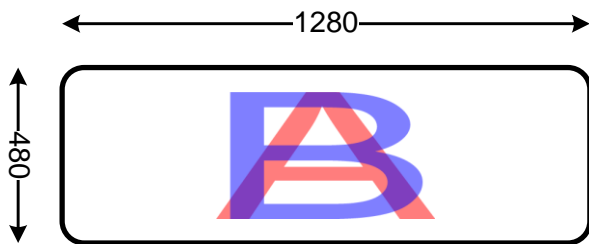
通过两个彩色传感器发送分辨率为 640 x 480 的 YUY2 数据为例。FX3 在每帧中接收到两个完整的彩色图像（图 53）。

图 53. FX3 接收到两个彩色图像



与示例 1 唯一不同点是 YUY2 格式中的每像素使用了两个字节，而不是一个字节。为了适应双倍像素，报告的图像尺寸也为双倍（图 54）。

图 54. 报告的图像尺寸



由于现在每个像素占用了两个字节，因此每帧的数据长度为：

$$\text{每帧的字节数} = 2 \times 2 \times 640 \times 480 = 1,228,800 \text{ 字节}$$

请注意，只要反映的是双倍像素，那么便可以报告任意宽度和高度的帧。只双倍一个维将简化下行计算。双倍宽度（相对于高度）可以覆盖所有垂直线，从而简化应用的图像处理。

这些描述符的修改允许以交错方式将图像从图像传感器传递给主机应用：任何两个连续字节均来自不同的图像传感器。

上述修改已经通过 UVC 驱动程序的一致性检查，因此允许驱动程序将视频数据传递给主机应用。当直接观看它们时，以这种方式进行的数据流并不清晰。您可以使用标准的 UVC 主机应用执行完整性检查。但是，由于图像以非标准的方式进行流处理，因此应用不能正确显示它们。需要编译一个自定义应用，以便区分这些图像，并查看和计算交错视频的有益信息。

9.2 固件修改检查表

总之，当修改已给示例固件时，为了确保已经实现所有所需的修改项，请按照下面步骤进行操作：

- 图像传感器控制：示例将参考 I²C 代码作为控制接口。需要修改设置或要求其它接口类型。
- 其它代码，用于控制作为控制接口的图像传感器选择的 GPIO。当 FX3 对本应用笔记中所使用的传

感器进行写操作时，该代码将两个图像传感器作为多播信息。但是，当 FX3 读取图像传感器中 I²C 寄存器时，需要单独访问。

- 其它代码，用于监控控制图像传感器的待机模式或低功耗模式的 GPIO。
- 修改 PROBE/COMMIT 控制结构以反映所修改的帧速率和帧分辨率。
- 修改 UVC 特定的高速和超速 USB 配置描述符的帧和格式，用于报告所修改的帧分辨率、帧速率以及长宽比。
- 修改 GPIF II 描述符，以便增大总线宽度，并根据总线宽度更新相应的计数器限制。通过 GPIF II Designer 实现上述修改，并生成头文件。需要将新生成的文件复制到您的项目中，以确保这些修改生效。

10. 总结

本应用笔记描述了如何使用赛普拉斯的 EZ-USB FX3 实现符合 USB 视频类别的图像传感器。尤其重要的是，它显示了：

- 主机应用和驱动程序如何与 UVC 器件交互作用
- UVC 器件如何管理 UVC 特定的请求
- 如何使用 GPIF II Designer 编程 FX3 接口，以便接收到典型的图像传感器的数据
- 如何显示视频流并修改主机应用中的照相机属性

- 如何向 UVC 器件增加用于调试的 USB 接口
- 如何查找各种平台上的主机应用，包括开源主机应用项目
- 如何使用 UVC 连接多个图像传感器、外部同步它们，并同步流操作
- 如何排除故障并调试 FX3 固件（若需要）。

11. 关于作者

姓名：Karnik Shah

职务：应用工程师

12. 文档修订记录

文档标题：如何使用 EZ-USB® FX3™ 在 USB 视频类别（UVC）框架内实现图像传感器接口— AN75779

文档编号：001-92220

| 修订版 | ECN | 原始变更 | 提交日期 | 修订说明 |
|-----|---------|----------|------------|--------------------------------------|
| ** | 4354623 | GKL | 04/21/2014 | 本文档版本号为 Rev**，译自英文版 001-75779 Rev*D。 |
| *A | 5708781 | AESATP12 | 04/28/2017 | Updated logo and copyright. |

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、原厂代表和经销商组成的全球性网络。如欲查找离您最近的办事处，请访问 [赛普拉斯所在地](#)。

产品

| | |
|-------------------|--|
| ARM® Cortex® 微控制器 | cypress.com/arm |
| 汽车级产品 | cypress.com/automotive |
| 时钟与缓冲器 | cypress.com/clocks |
| 接口 | cypress.com/interface |
| 物联网 | cypress.com/iot |
| 存储器 | cypress.com/memory |
| 微控制器 | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| 电源管理 IC | cypress.com/pmic |
| 触摸感应 | cypress.com/touch |
| USB 控制器 | cypress.com/usb |
| 无线连接 | cypress.com/wireless |

PSoC® 解决方案

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

赛普拉斯开发者社区

[论坛](#) | [WICED IoT 论坛](#) | [项目](#) | [视频](#) | [博客](#) | [培训](#) | [组件](#)

技术支持

cypress.com/support

EZ-USB® 和 FX3™ 是赛普拉斯半导体公司的注册商标。此处引用的所有其他商标或注册商标归其各自所有者所有。


CYPRESS
 EMBEDDED IN TOMORROW™

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

赛普拉斯半导体公司，2012-2017 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC (“赛普拉斯”) 的财产。本文件，包括其包含或引用的任何软件或固件 (“软件”)，根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可 (无再许可权) (1) 在赛普拉斯特软件著作权项下的下列许可权 (一) 对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和 (二) 仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供 (无论直接提供或通过经销商和分销商间接提供)，和 (2) 在被软件 (由赛普拉斯公司提供，且未经修改) 侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统 (包括急救设备和手术植入物)、污染控制或有有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途 (“非预期用途”)。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产