

使用赛普拉斯 EZ-USB FX3 通过 USB 配置 Xilinx FPGA

作者: Rama Sai Krishna Vakkantula

相关项目: 有

相关器件系列: **CYUSB3014**

有关最新 FX3 SDK 的信息: [请点击此处](#)

更多代码示例? 我们听到了你

要访问各种 FX3 代码示例, 请访问我们的 [USB SuperSpeed 代码示例网页](#)。

AN4868 介绍了如何使用 EZ-USB® FX3™ (新一代 USB 3.0 外设控制器) 通过一个从设备串行接口来配置 Xilinx® FPGA。该接口允许您使用 USB 2.0 或 3.0 将配置文件下载到 Xilinx FPGA 内。本应用笔记附带的固件文件是针对 Xilinx FPGA 进行设计和测试的, 但是您可以使用同样的接口为其他 FPGA 自定义这些文件。

目录

1	简介	1	3.6	将配置固件集成到设计内	10
2	Xilinx 从设备串行配置接口	2	3.7	软件的详细信息	11
3	实现	3	4	操作说明	12
3.1	硬件的详细内容	3	5	总结	19
3.2	FX3 固件	4	6	相关的项目文件	19
3.3	I/O 矩阵的配置	6	7	参考	19
3.4	从设备串行接口实现	8		文档修订记录	20
3.5	重新配置 I/O 矩阵	9		全球销售和设计支持	21

1 简介

FX3 具有一个可配置的并行通用可编程接口 (GPIO II), 可将该接口连接到外部器件上, 如图像传感器、外部处理器、ASIC 或 FPGA。因此, 用户可以将 USB 3.0 接口集成到几乎所有系统上。

另外, FX3 还具有 UART、SPI、I²C 和 I²S 等串行外设的接口。

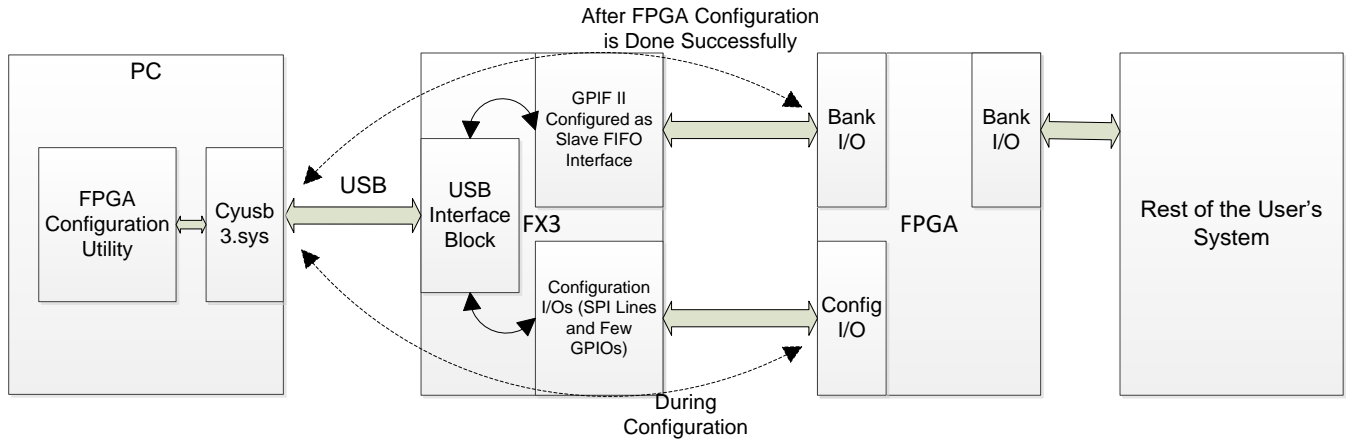
FX3 允许您将 SuperSpeed 性能添加到任何基于 FPGA 的设计。在多数应用中, FPGA 作为主设备使用, 另外 GPIO II 作为同步从设备 FIFO 接口使用。更多有关从设备 FIFO 接口的详细信息, 请查阅 [AN65974 — 使用 EZ-USB® FX3™ 从设备 FIFO 接口进行设计](#)。你可以使用与 FPGA 相连接的控制器 (在此情况下为 FX3) 来配置 FPGA。如果使用 FX3, 则不需要使用 FPGA 的专用配置芯片 (例如, 一个 PROM 或一个处理器)。此外, 这种方法可用来替代要求在电路板上 JTAG 连接器的常见 JTAG 配置接口。这样便降低了成本, 并缩小了电路板空间。FPGA 配置可以使用 FX3 从预先加载 FPGA 配置文件的外部 SPI 闪存或 EEPROM 加载。参见 [FX3 + FPGA + HelionVision ISP-Based Industrial Camera Reference Design – KBA222700](#), 其中 FX3 与用于成像应用的 FPGA 接口。

作为主设备, FX3 可以将 Xilinx FPGA 配置为两个模式: 从设备并行模式 (SelectMAP) 和从设备串行模式。请参考 [Xilinx Spartan-6 FPGA 配置用户指南](#), 以获取多种配置 FPGA 的方法。本应用笔记仅介绍了从设备串行模式, 同时也描述了 FX3 固件在完成 FPGA 配置后如何切换到从设备 FIFO 接口。图 1 中的框图显示的是 FX3 先配置 FPGA, 并且成功配置后再切换到从设备 FIFO 接口的过程。

本应用笔记使用主机应用程序使用供应商命令加载 FPGA 配置文件。供应商命令将通过 SPI 接口启动 FPGA 配置过程, 并接收来自主机的 FPGA 配置数据。

下面各节详细说明了 Xilinx 从设备串行配置接口以及使用 FX3 来进行相关设计的内容。

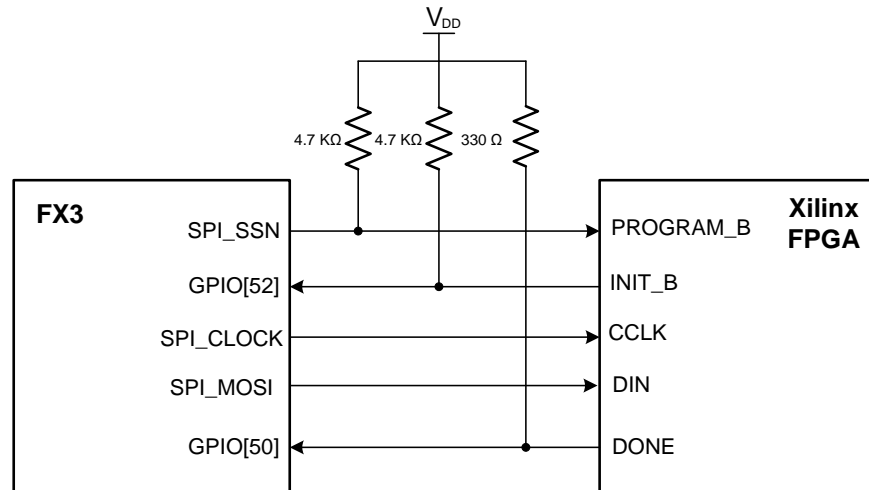
图 1. 系统级应用框图



2 Xilinx 从设备串行配置接口

本章节对 Xilinx 从设备串行接口进行了详细介绍。图 2 介绍的是与 Xilinx 从设备串行接口相关的接口引脚，表 2 则给出了从设备串行接口引脚的说明内容。

图 2. FX3 与 Xilinx Spartan-6 FPGA 之间的硬件连接



PROGRAM_B、INIT_B 以及 DONE 均为开漏信号。在这些线路上，连接至合适值的上拉电阻。图 2 中所提到的电阻值取自 [Xilinx Spartan-6 FPGA 配置用户指南](#)。请注意，如果使用了 FX3 CYUSB3KIT-001 套件，则不需要连接至这些上拉电阻，但需要将上拉电阻放置在最终设计内。

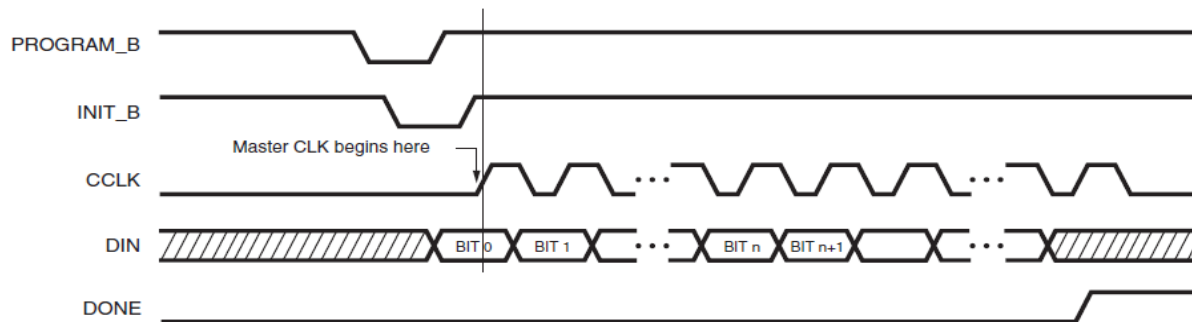
表 1 显示的是从设备串行接口的接口信号。

表 1. Xilinx 从设备串行配置引脚说明

引脚名称	引脚方向 (往 FPGA)	引脚说明
PROGRAM_B	输入	编程 FPGA。 低电平有效。 在 500 ns 或更长时间内 (在 Spartan-3 FPGA 中为 300 ns)，将该信号保持为低电平时，它将清除配置存储器中的内容，从而强制 FPGA 重启其配置过程。
INIT_B	开漏双向 I/O	FPGA 初始化指示符。 在上电复位 (POR) 后或在 FPGA 清除其配置存储器的情况下，PROGRAM_B 被置为低电平时，将该信号驱动为低电平。如果配置期间检测到一个 CRC 错误，FPGA 再次将 INIT_B 驱动为低电平。
CCLK	输入	配置时钟。
DIN	输入	数据输入。 串行数据。FPGA 在 CCLK 的上升沿上捕捉数据。
DONE	开漏双向 I/O	FPGA 配置完成。 配置期间为低电平。当 FPGA 成功完成配置时将转为高电平。

图 3 显示的是 Xilinx 从设备串行配置的时钟供给序列图。

图 3. Xilinx 从设备串行配置的时钟供给序列



3 实现

FX3 通过产生 PROGRAM_B 脉冲并监控 INIT_B 引脚来启动配置过程。当 INIT_B 为高电平时，FPGA 就绪接收数据。FX3 开始供给数据和时钟信号，直到 DONE 引脚转为高电平为止，表示成功配置，或直到 INIT_B 引脚转为低电平为止，表示配置错误。配置过程所需的时钟周期大于配置文件尺寸中指示的时钟周期。FPGA 启动时需要使用其他时钟 (请参考图 3)。

3.1 硬件的详细内容

3.1.1 硬件电路板

- Xilinx SP601 评估套件
- SuperSpeed Explorer 套件 (CYUSB3KIT-003) 或 EZ-USB FX3 DVK (CYUSB3KIT-001)
- Samtec 至 FMC 互联电路板 (对于 CYUSB3KIT-001)，或 CYUSB3ACC005 互联电路板 (对于 CYUSB3KIT-003)
- 导线互连配置信号

FX3 中的 SPI 硬件模块对来自 PC 的配置数据进行串行化处理。将 FX3 的 SPI_SSN (从设备选择)、SPI_CLOCK 和 SPI_MOSI 分别连接至 Xilinx FPGA 的 PROGRAM_B、CCLK 和 DIN。FPGA 的 INIT_B 和 DONE 引脚分别连接至 GPIO 52 和 50。图 2 显示的是 FX3 和 Xilinx FPGA 之间的连接情况。

3.2 FX3 固件

附加的固件具有以下组件：

- 通过从设备串行接口与 FX3 连接的 Xilinx FPGA 配置。
- 如果 Xilinx FPGA 配置成功，则从设备 FIFO 接口配置与 [AN65974](#) 中描述的相同。

FX3 的 SPI 硬件模块对 FX3 接收的来自 PC 应用“FPGA 配置工具”的数据进行串行化处理，如图 1 所示。FPGA 配置工具通过赛普拉斯的 VID (0x04B4) 和 PID (0x00F1) 识别 USB 器件。

欲了解应用固件的结构的信息，请查看 [FX3 编程器手册](#) 中“FX3 应用结构”章节的内容。有关 FX3 SDK API 的详细信息，请参考 [FX3 固件 API 指南](#) 中的内容。

表 2 描述的是固件源代码中存在的文件，本应用笔记附带了该文件。

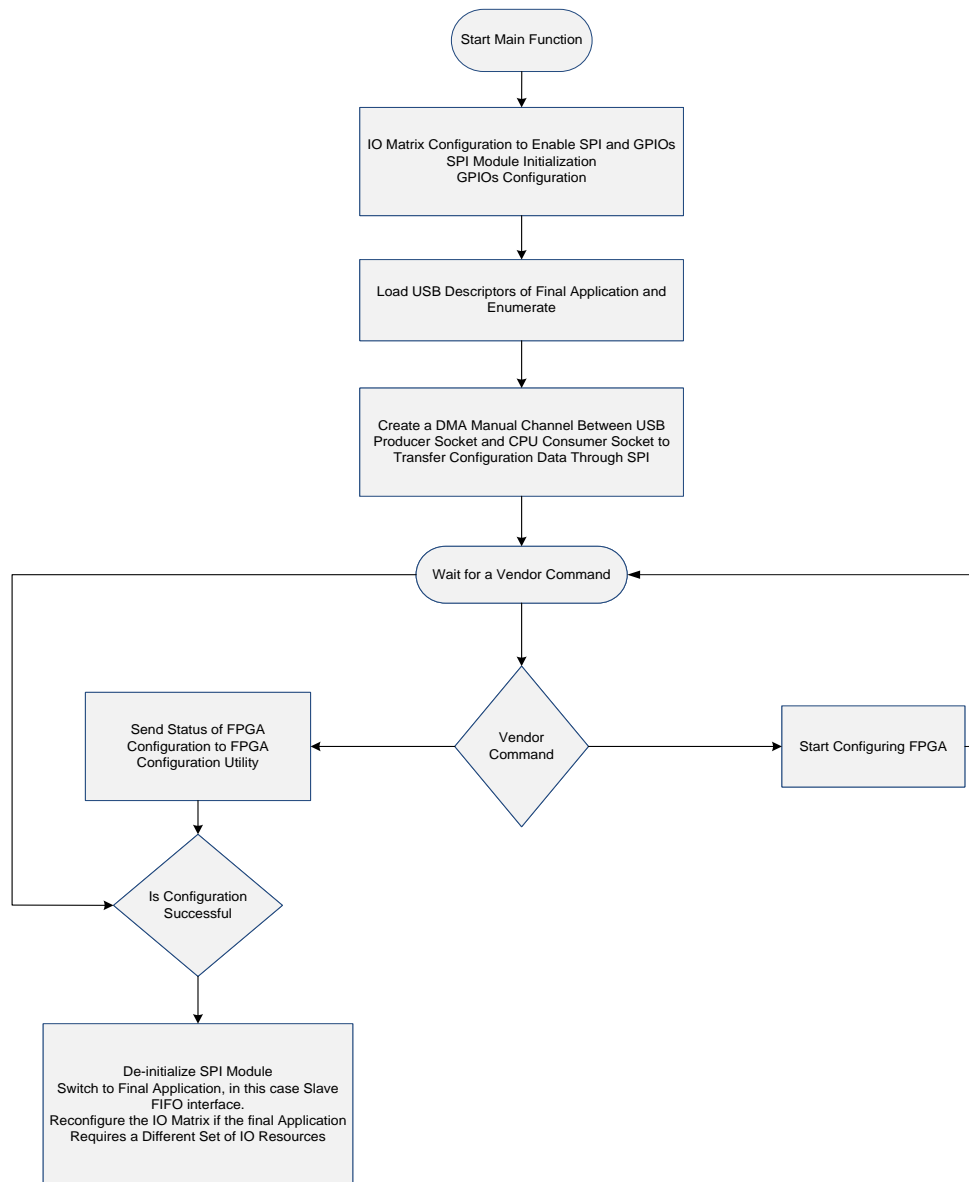
表 2. FX3 固件源文件简介

文件名称	说明
<code>cyfx_gcc_startup.S</code>	赛普拉斯 FX3 固件启动代码
<code>cyfxconfigfpga.c</code>	该文件介绍的是从设备串行模式下的配置 FPGA 示例。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。 • CyFxConfigFpgaApplnInit: 初始化 FX3 GPIO 和 SPI 模块。将 GPIO[50] 和 GPIO[52] 配置为输入信号。初始化 USB 模块，进行枚举。 • CyFxConfigFpgaApplnStart: 端点配置 (用于 USB 传输) 和 DMA 通道配置 (用于将数据从 USB 模块传输到 FX3 的 SPI 模块)。 • CyFxConfigFpgaApplnStop: 取消初始化 FX3 GPIO 和 SPI 模块，从而重新配置 I/O 矩阵。 • CyFxConfigFpga: 通过从设备串行接口将配置数据写入到 Xilinx FPGA 内。
<code>cyfxconfigfpga.h</code>	该文件包含用于配置 FPGA 应用示例的各常量和定义。
<code>cyfxslifosync.c</code>	该文件介绍的是从设备 FIFO 同步模式的示例。它包括下面各函数： <ul style="list-style-type: none"> • CyFxApplicationDefine: 用于创建一个应用线程，以便通过从设备 FIFO 接口传输数据。 • SlFifoAppThread_Entry: 应用线程函数，为 FX3 的内部模块调用初始化函数。FX3 会等待事件来配置 FPGA，在成功配置完 FPGA 后，它会切换到从设备 FIFO 接口。 • CyFxSwitchtoSlFifo: 按照从设备 FIFO 接口的要求重新配置 FX3 I/O 矩阵。 • CyFxSlFifoApplnInit: 初始化处理器接口模块，为从设备 FIFO 接口加载 GPIF 配置并启动 GPIF 状态机。 • CyFxSlFifoApplnStart: 端点配置 (用于 USB 传输) 和 DMA 通道配置 (用于在 USB 模块和 FX3 的 GPIF II 模块间进行数据传输)。 • CyFxSlFifoApplnStop: 该函数用于终止从设备 FIFO 应用。在收到从 USB 主机的 RESET 或 DISCONNECT 事件时，会调用该函数。该函数会禁用各个端点，并破坏 DMA 通道。 • CyFxSlFifoApplnUSBEventCB: 用于处理各个 USB 事件，如暂停、线缆断开连接、复位和恢复等事件。 • CyFxSlFifoApplnUSBSetupCB: 该函数会回调以便处理 USB 设置请求。 • CyFxSlFifoApplnDebugInit: 初始化用于打印调试信息的 FX3 UART 模块。调试打印将被路由到 UART 上，使用工作频率为 115200 波特率的 UART 控制台可以观察到该调试打印。
<code>cyfxslifosync.h</code>	该文件介绍了使用于从设备 FIFO 应用的各项常量和定义。
<code>cyfxslifousbdscr.c</code>	该文件介绍了从设备 FIFO 示例所需的各 USB 描述符。
<code>cyfctx.c</code>	该文件定义了 ThreadX RTOS 所需的移植。该文件以源代码形式提供，并必须与应用源代码进行编译。
<code>cyfxgpiif2config.h</code>	该文件包含 16 位和 32 位从设备 FIFO 接口的 GPIF II 描述符。

注意： 有关 FX3 中特定术语的详细信息，请参考 [EZ-USB FX3 入门](#) 应用笔记中“FX3 术语”部分的内容。

图 4 中的流程图说明的是 FX3 固件。

图 4. FX3 固件的流程图



3.3 I/O 矩阵的配置

在 `main()` 函数中, 根据应用要求, 配置 I/O 矩阵 (如以下代码所示)。将 GPIF II 接口设置为 16 位, 以使能 SPI 接口。使能 GPIO 50 和 52, 以便与 Xilinx FPGA 的 DONE 和 INIT_B 引脚连接 (有关硬件接口图, 请参考图 2)。可以在 `cyfxconfigfpga.c` 文件的 `main()` 函数中找到该代码段。

```
io_cfg.useUart    = CyTrue;
io_cfg.useI2C     = CyFalse;
io_cfg.useI2S     = CyFalse;
io_cfg.useSpi     = CyTrue;
io_cfg.isDQ32Bit  = CyFalse;
io_cfg.lppMode    = CY_U3P_IO_MATRIX_LPP_DEFAULT;
/* GPIOs 50 and 52 are enabled. */
io_cfg.gpioSimpleEn[0] = 0x00000000;
io_cfg.gpioSimpleEn[1] = 0x00140000;
io_cfg.gpioComplexEn[0] = 0;
io_cfg.gpioComplexEn[1] = 0;
status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

3.3.1 SPI 模块的初始化

使用以下代码来初始化和配置 SPI 模块。配置它, 使之在 25 MHz 时钟频率下运行。FX3 SPI 硬件模块可以支持高达 33 MHz 的时钟频率。可以在 `cyfxconfigfpga.c` 文件的 `CyFxConfigFpgaApplnInit()` 函数中找到该代码段。

```
/* Start the SPI module and configure the master. */
apiRetStatus = CyU3PSpiInit();

/* Start the SPI master block. Run the SPI clock at 25MHz and configure
the word length to 8 bits. Also configure the slave select using FW. */
CyU3PMemSet ((uint8_t *)&spiConfig, 0, sizeof(spiConfig));
spiConfig.isLsbFirst = CyFalse;
spiConfig.cpol       = CyTrue;
spiConfig.ssnPol     = CyFalse;
spiConfig.cpha       = CyTrue;
spiConfig.leadTime   = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
spiConfig.lagTime    = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
spiConfig.ssnCtrl     = CY_U3P_SPI_SSN_CTRL_FW;
spiConfig.clock       = 25000000; /* Maximum value of SPI clock is 33 MHz*/
spiConfig.wordLen     = 8;

apiRetStatus = CyU3PSpiSetConfig (&spiConfig, NULL);
```

3.3.2 GPIO 配置

使用以下代码来初始化和配置 GPIO 模块。GPIO 52 和 GPIO 50 被配置为输入, 因此 GPIO 52 可以用于监控 INIT_B 引脚, 另外 GPIO 50 可以用于监控来自 Xilinx FPGA 的 DONE 信号。可以在 `cyfxconfigfpga.c` 文件的 `CyFxConfigFpgaApplnInit()` 函数中找到该代码段。

```
/* Init the GPIO module */
gpioClock.fastClkDiv = 2;
gpioClock.slowClkDiv = 0;
gpioClock.simpleDiv  = CY_U3P_GPIO_SIMPLE_DIV_BY_2;
gpioClock.clkSrc      = CY_U3P_SYS_CLK;
gpioClock.halfDiv     = 0;
apiRetStatus = CyU3PGpioInit(&gpioClock, NULL);

/* Configure GPIO 52 as input */
gpioConfig.outValue = CyTrue;
gpioConfig.inputEn  = CyTrue;
gpioConfig.driveLowEn = CyFalse;
```

```

    gpioConfig.driveHighEn = CyFalse;
    gpioConfig.intrMode = CY_U3P_GPIO_INTR_BOTH_EDGE;
    apiRetStatus = CyU3PGpioSetSimpleConfig(FPGA_INIT_B, &gpioConfig);

    /* Configure GPIO 50 as input */
    apiRetStatus = CyU3PGpioSetSimpleConfig(FPGA_DONE, &gpioConfig);

```

3.3.3 创建 DMA 通道来设置数据传输

在发送 USB 套接字和接收 CPU 套接字之间创建 DMA 手动通道，以便将在 FX3 的 Bulk out 端点 (0x01) 上接收的配置数据手动传输到 SPI 模块。用于创建 DMA 手动通道的代码如下所示。在 *cyfxconfigfpga.c* 文件的 *CyFxConfigFpgaAppInStart()* 函数中可以找到该代码段。

```

/* Create a DMA MANUAL channel for U2CPU transfer. The DMA size is set based on the
USB speed. */
dmaCfg.size = size;
dmaCfg.count = CY_FX_SLFIFO_DMA_BUF_COUNT;
dmaCfg.prodSckId = CY_FX_PRODUCER_USB_SOCKET;
dmaCfg.consSckId = CY_U3P_CPU_SOCKET_CONS;
dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
/* Enabling the callback for produce event. */
dmaCfg.notification = 0;
dmaCfg.cb = NULL;
dmaCfg.prodHeader = 0;
dmaCfg.prodFooter = 0;
dmaCfg.consHeader = 0;
dmaCfg.prodAvailCount = 0;

apiRetStatus = CyU3PDmaChannelCreate (&glChHandleUtoCPU, CY_U3P_DMA_TYPE_MANUAL_IN,
&dmaCfg);

```

3.3.4 FPGA 配置工具和 FX3 固件之间的通信

通过两个供应商指令，可以使用在 PC “FPGA 配置工具”上运行的应用控制 FX3 固件的性能。根据所接收的供应商指令，FX3 固件对事件进行相关设置。收到供应商指令 0xB2 (VND_CMD_SLAVESER_CFGLOAD) 和配置位文件的长度后，它将设置事件 *CY_FX_CONFIGFPGAAPP_START_EVENT*，以开始进行配置 FPGA。收到供应商指令 0xB1 (VND_CMD_SLAVESER_CFGSTAT)，并且在成功配置 FPGA 后，固件还将设置事件 *CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT*，以切换到从设备 FIFO 接口。可通过以下代码段执行该操作。可以在 *cyfxslfifosync.c* 文件的 *CyFxSlFifoAppInUSBSetupCB ()* 函数中找到该代码。

```

if (bRequest == VND_CMD_SLAVESER_CFGLOAD)
{
    if ((bReqType & 0x80) == 0)
    {
        CyU3PUsbGetEP0Data (wLength, glEp0Buffer, NULL);
        filelen = uint32_t) (glEp0Buffer[3]<<24) | (glEp0Buffer[2]<<16) |
            (glEp0Buffer[1]<<8) | glEp0Buffer[0];
        glConfigDone = CyTrue;
    /* Set CONFIGFPGAAPP_START_EVENT to start configuring FPGA */
        CyU3PEventSet (&glFxConfigFpgaAppEvent,
            CY_FX_CONFIGFPGAAPP_START_EVENT, CYU3P_EVENT_OR);
        isHandled = CyTrue;
    }
}

if (bRequest == VND_CMD_SLAVESER_CFGSTAT)
{
    if ((bReqType & 0x80) == 0x80)
    {
        glEp0Buffer [0]= glConfigDone;
    }
}

```



```

    CyU3PUsbSendEP0Data (wLength, glEp0Buffer);
/* Switch to slaveFIFO interface when FPGA is configured successfully*/
if (glConfigDone)
    CyU3PEventSet (&glFxConfigFpgaAppEvent,
                  CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT,
                  CYU3P_EVENT_OR);
    isHandled = CyTrue;
}
}

```

3.3.5 根据事件进行的操作

FX3 固件连续寻找上述提及的事件，并执行与这些事件相对应的操作。*cyfxslfifosync.c* 中的 *SlFifoAppThread_Entry()* 包含以下代码：

```

/* Wait for events to configure FPGA */
txApiRetStatus = CyU3PEventGet (&glFxConfigFpgaAppEvent,
                                (CY_FX_CONFIGFPGAAPP_START_EVENT |
                                 CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT),
                                CYU3P_EVENT_OR_CLEAR, &eventFlag,
                                CYU3P_WAIT_FOREVER);
if (txApiRetStatus == CY_U3P_SUCCESS)
{
    if (eventFlag & CY_FX_CONFIGFPGAAPP_START_EVENT)
    {
        /* Start configuring FPGA */
        CyFxConfigFpga(filelen);
    }
    else if ((eventFlag & CY_FX_CONFIGFPGAAPP_SW_TO_SLFIFO_EVENT))
    {
        /* Switch to SlaveFIFO interface */
        CyFxConfigFpgaApplnStop();
        CyFxSwitchtoslFifo();
        CyFxSlFifoApplnInit();
        CyFxSlFifoApplnStart();
    }
}
}

```

3.4 从设备串行接口实现

CyFxConfigFpga 是执行 Xilinx 从设备串行接口的函数。为了开始配置操作，FX3 将 PROGRAM_B 驱动为低电平。然后，FX3 等到 INIT_B 转为低电平，并且在 INIT_B 再次转为高电平时，它将开始传输数据。将所有配置数据发送到 FPGA 后，FX3 将根据 DONE 信号来决定配置是否成功。如果配置成功，DONE 信号将被设为高电平。要想深入了解时序图，请参见图 1。在 *cyfxconfigfpga.c* 中可以找到该函数。

```

/* This is the function that writes configuration data to the Xilinx FPGA */
CyU3PReturnStatus_t CyFxConfigFpga(uint32_t uiLen)
{
    uint32_t uiIdx;
    CyU3PReturnStatus_t apiRetStatus;
    CyU3PDmaBuffer_t inBuf_p;
    CyBool_t xFpga_Done, xFpga_Init_B;

    /* Pull PROG_B line to reset FPGA */
    apiRetStatus = CyU3PSpiSetSsnLine (CyFalse);
    CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
    CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
    if (xFpga_Init_B)
    {
        glConfigDone = CyFalse;
    }
}

```



```

        return apiRetStatus;
    }

    CyU3PThreadSleep(10);
/* Release PROG_B line */
    apiRetStatus |= CyU3PSpiSetSsnLine (CyTrue);
    CyU3PThreadSleep(10);    // Allow FPGA to startup

/* Check if FPGA is now ready by testing the FPGA_Init_B signal */
    apiRetStatus |= CyU3PGpioSimpleGetValue (FPGA_INIT_B, &xFpga_Init_B);
    if( (xFpga_Init_B != CyTrue) || (apiRetStatus != CY_U3P_SUCCESS) ){

        return apiRetStatus;
    }
/* Start shifting out configuration data */
    for(uiIdx = 0; (uiIdx < uiLen) && glIsApplnActive; uiIdx += uiPacketSize )
    {
        if(CyU3PDmaChannelGetBuffer (&glChHandleUtoCPU, &inBuf_p, 2000) !=
        CY_U3P_SUCCESS){
            glConfigDone = CyFalse;
            apiRetStatus = CY_U3P_ERROR_TIMEOUT;
            break;
        }
        apiRetStatus = CyU3PSpiTransmitWords(inBuf_p.buffer , uiPacketSize);
        if (apiRetStatus != CY_U3P_SUCCESS)
        {
            glConfigDone = CyFalse;
            break;
        }
        if(CyU3PDmaChannelDiscardBuffer (&glChHandleUtoCPU) != CY_U3P_SUCCESS)
        {
            glConfigDone = CyFalse;
            apiRetStatus = CY_U3P_ERROR_TIMEOUT;
            break;
        }
    }
    CyU3PThreadSleep(1);

    apiRetStatus |= CyU3PGpioSimpleGetValue (FPGA_DONE, &xFpga_Done);
    if( (xFpga_Done != CyTrue) )
    {
        glConfigDone = CyFalse;
        apiRetStatus = CY_U3P_ERROR_FAILURE;
    }
    return apiRetStatus;
}

```

3.5 重新配置 I/O 矩阵

将所有配置数据发送到 FX3 后, FPGA 配置工具会自动发送供应商指令 0xB1 (VND_CMD_SLAVESER_CFGSTAT)。如果 FPGA 的配置成功, 则 FX3 固件仅会切换到从设备 FIFO 接口。使用以下代码段重新配置 I/O 矩阵。如果最终应用中使用了同一个 I/O 资源, 则不需要执行该操作。但是, 在这种情况下, 需要重新配置 I/O 矩阵, 因为从设备 FIFO 固件 (取自 AN65974) 使用了 GPIF II 中的 32 位接口。请确保在重新配置 I/O 矩阵前, 已经取消初始化所有受影响的外设模块。在该应用中, 重新配置 I/O 矩阵前, 已经取消了初始化 GPIO 和 SPI 模块。I/O 矩阵的配置需要与 32 位从设备 FIFO 接口相同, 如下显示。在 `cyfxslifosync.c` 文件的 `CyFxSwitchtoSlFifo ()` 函数中可以找到该代码段。

```

    io_cfg.useUart    = CyTrue;
    io_cfg.useI2C     = CyFalse;
    io_cfg.useI2S     = CyFalse;
    io_cfg.useSpi     = CyFalse;
    #if (CY_FX_SLFIFO_GPIF_16_32BIT_CONF_SELECT == 0)

```

```
io_cfg.isDQ32Bit = CyFalse;
io_cfg.lppMode = CY_U3P_IO_MATRIX_LPP_UART_ONLY;
#else
io_cfg.isDQ32Bit = CyTrue;
io_cfg.lppMode = CY_U3P_IO_MATRIX_LPP_DEFAULT;
#endif
/* No GPIOs are enabled. */
io_cfg.gpioSimpleEn[0] = 0x00000000;
io_cfg.gpioSimpleEn[1] = 0;
io_cfg.gpioComplexEn[0] = 0;
io_cfg.gpioComplexEn[1] = 0;
status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

3.5.1 端点配置与恢复序号

使用同一个发送终端 (EP1 OUT BULK) 来配置 FPGA, 并且在成功配置 FPGA 后, 通过从设备 FIFO 接口将数据从 USB 发送到 FX3 上连接的 FPGA 内。从设备 FIFO 接口被使能后, EP1 被重新配置, 以使能突发传输, 从而支持高带宽数据传输。因此, 将调用两次 CyU3PSetEpConfig API 来配置同一个终端。该 API 会清除与终端相关的序号。如果 USB 3.0 主机和 FX3 器件检测到它们的序号不匹配, 则数据传输会失败。因此, 需要恢复序号, 从而在重新配置 EP1 后, USB 3.0 主机仍能够成功执行数据传输。该内容仅对数据传输有效。

CyU3PUsbGetEpSeqNum API 为端点获取当前序号, CyU3PUsbSetEpSeqNum 则为终端设置有效序号。

3.6 将配置固件集成到设计内

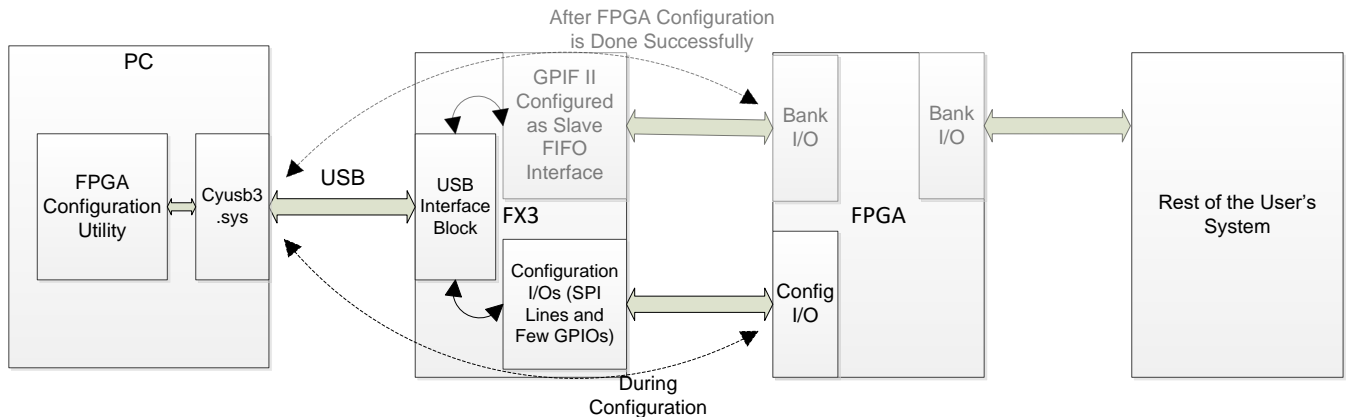
本节向您介绍了如何将配置固件集成到设计内。当您阅读下面各步骤时, 请参考附带本应用笔记的项目:

1. 将 *cyfxconfigfpga.c* 和 *cyfxconfigfpga.h* 文件导入到您的项目中。
2. 在您的设计中备注 *main()* 函数, 因为 *main()* 是在 *cyfxconfigfpga.c* 中实现的。
3. 在您的应用初始化函数的位置上, 调用线程入口函数的 *CyFxConfigFpgaApplnInit()*。在这个示例中, *SlFifoAppThread_Entry()* 函数 (由 *CyFxSlFifoApplnInit()* 调用) 调用了 *CyFxConfigFpgaApplnInit()*。
4. USB 事件回调函数 (由您的应用启动函数调用) 调用了 *CyFxConfigFpgaApplnStart()*。在这个示例中, *CyFxSlFifoApplnUSBEventCB* 函数 (由 *CyFxSlFifoApplnStart()* 调用) 调用了 *CyFxConfigFpgaApplnStart()*。
5. 因为 *CyFxConfigFpgaApplnInit()* 已经处理了 USB 枚举部分, 所以注释掉 *CyFxSlFifoApplnInit()* 中处理它的代码段。
6. 在该示例中执行供应商指令和事件时, 会添加对它们的支持。
7. 如果您的应用程序需要不同的资源, 则需要重新配置 I/O 矩阵。在这个示例中, 在 *cyfxslfifosync.c* 位置的 *CyFxSwitchToSlFifo()* 函数中可以找到 I/O 矩阵重新配置代码。
8. 改变您应用程序线程入口函数使之与 *SlFifoAppThread_Entry()* 相似。

3.7 软件的详细信息

该章节介绍了执行本应用笔记附带的项目文件时所需的主机应用和 USB 3.0 驱动器。图 5 显示的是系统级框图，它包括 PC 配置连接到 FX3 的 FPGA 时所需的主机应用和驱动器。

图 5. 显示 PC 端上软件的详细信息的系统级应用框图



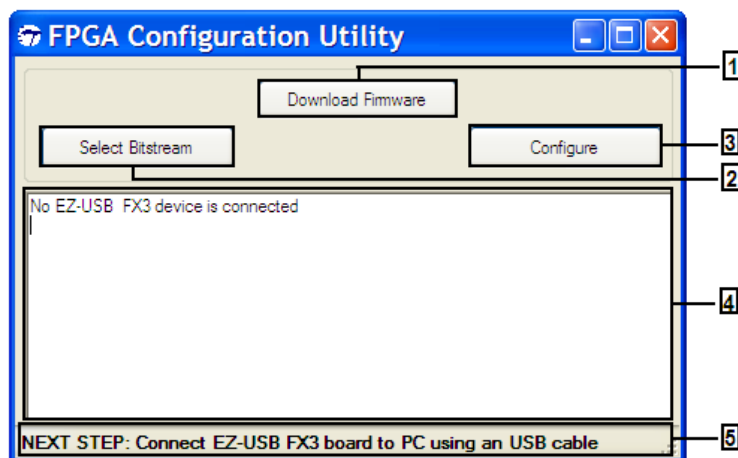
3.7.1 主机应用

FPGA 配置工具是针对该应用程序而特别开发的，并作为附件使用。

USB 驱动器: *cyusb3.inf* 和 *cyusb3.sys* 是 EZ-USB FX3 SDK 的一部分。

设计中包含了主机应用示例 (即 FPGA 配置工具，针对配置 FPGA 而创建的)。该应用程序使用了赛普拉斯应用开发库 *CyUSB.dll* (包含在赛普拉斯 *SuperSpeed USB Suite* 中) 在 Visual C# 2008 Express Edition 中开发。该器件必须与 *CyUSB3.sys* (由赛普拉斯开发的通用驱动程序) 绑定。随本应用笔记附带的主机应用程序作为参考示例，用于开发 FPGA 配置工具。它提供了一个用于将固件图像下载到 FX3 的 RAM 内的选项，并为 Xilinx FPGA 配置选择 bitstream (.bin) 文件提供了灵活性。此外，本应用还列出了各步骤的状态，并且指出了下一个步骤以成功运行演示。图 6 显示的是 FPGA 配置工具的备注元素。

图 6. FPGA 配置工具的备注元素



- 1: 将固件图像下载到 FX3 的 RAM 内
- 2: 向 Xilinx FPGA (.bin:文件) 选择配置文件
- 3: 通过 FX3 下载选中的配置文件

- 4: 在配置 Xilinx FPGA 过程中, 显示每个步骤的状态
- 5: 显示下一个步骤

4 操作说明

本节向您介绍了通过使用由本应用笔记提供的软件和固件项目来如何配置连接至 FX3 Superspeed Explorer 套件的 Xilinx FPGA。使 Xilinx Spartan-6 SP601 评估套件和 FX3 SuperSpeed Explorer 套件 (或 CYUSB3KIT-001) 之间建立硬件连接, 如表 3 所示。这些连接与硬件互连图 (图 2) 所示的连接情况相同。此外, 通过 Samtec-to-FMC 连接器将 FX3 SuperSpeed Explorer 套件 (或 CYUSB3KIT-001) 连接至 Xilinx Spartan-6 SP601 评估套件。请注意, 用于本应用笔记的硬件设置与 AN65974 中的硬件设置相同, 但是您需要使用五根导线连接所需信号, 以配置 FPGA。

表 3. Xilinx SP601 评估套件和 FX3 SuperSpeed Explorer 套件 (或 CYUSB3KIT-001) 之间的硬件连接

信号名称	SP601 评估套件上的引脚位置	FX3 SuperSpeed Explorer 套件上的引脚位置	CYUSB3KIT-001 上的引脚位置
PROGRAM_B	J12 的引脚 1	J7 的引脚 23	J102 的引脚 2
INIT_B	电阻 R90 的一端 (如图 7 所示)	J7 的引脚 31	J20 的引脚 6
CCLK	J12 的引脚 7	J7 的引脚 27	J101 的引脚 2
DIN	J12 的引脚 6	J7 的引脚 19	J104 的引脚 2
DONE	R113 或 LED DS9 的一端 (如图 7 所示)	J7 的引脚 37	J20 的引脚 4

图 7. Xilinx SP601 评估套件上的硬件连接



1. 运行存在于 `FPGA_Config_Utility\bin\Debug` 文件夹中的 `Template.exe` 性能, 并查看屏幕上显示的工具。下面的状态信息显示: **No EZ-USB FX3 device is connected** (未连接任何 EZ-USB FX3 器件)。
2. 使用 USB 线缆将 EZ-USB FX3 Explorer 套件或 CYUSB3KIT-001 连接至 PC, 如图 8 所示。然后, 您会观察到文本框所显示的状态信息 **EZ-USB FX3 Bootloader device connected** (已连接的 EZ-USB FX3 Bootloader 器件), 如图 9 所示。

3. 点击 **Download Firmware** 按键，将固件图像下载到 FX3 的 RAM 内，并浏览 *ConfigFpgaSlaveFifoSync.img* 文件所在的位置，如图 10 所示。然后，点击 **Open** 按键。

图 8. 未连接任何 EZ-USB FX3 器件的 FPGA 配置工具

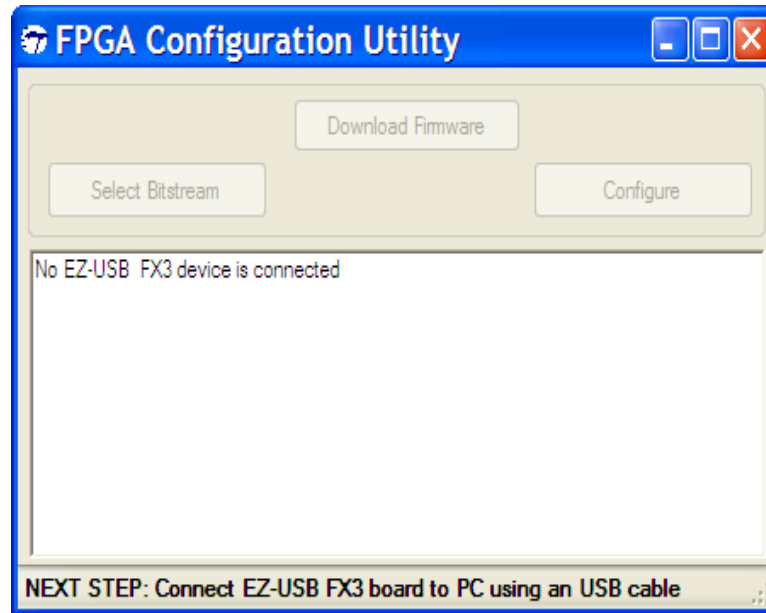


图 9. 将 EZ-USB FX3 DVK 连接到 PC 后的 FPGA 配置工具

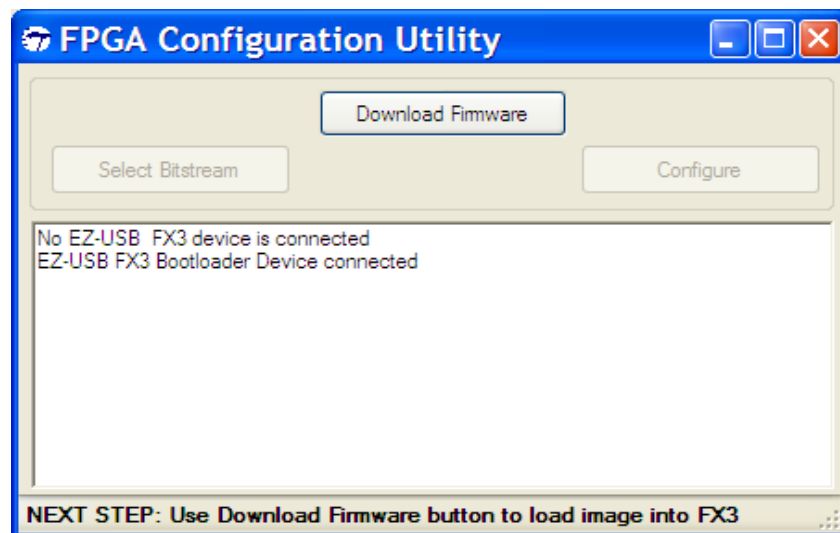
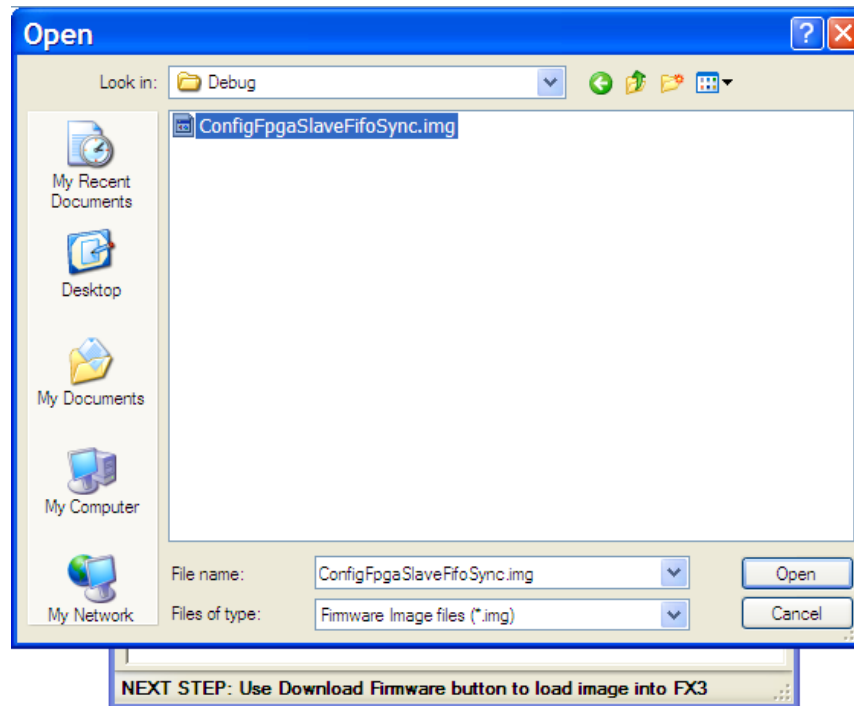


图 10. 选择 FX3 的固件图像



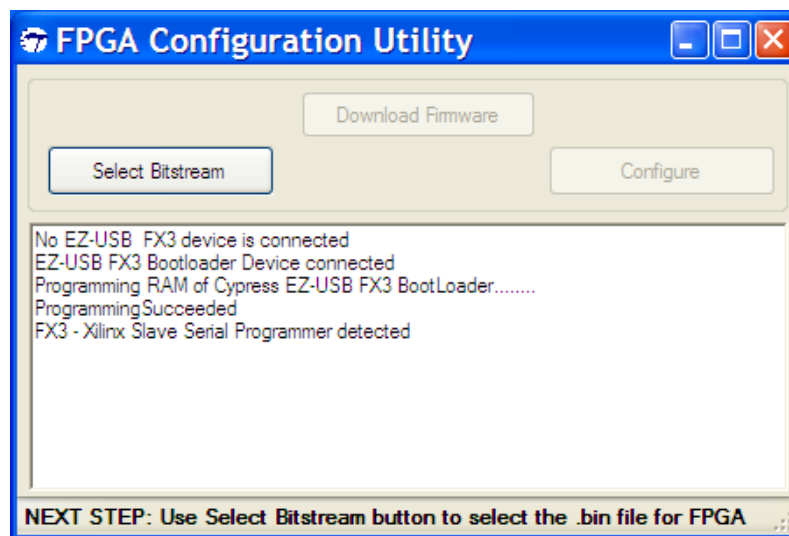
显示了下面的状态信息，如图 11 所示：

Programming RAM of Cypress EZ-USB FX3 BootLoader..... (编程赛普拉斯 EZ-USB FX3 BootLoader 的 RAM.....)

Programming Succeeded (编程成功)

FX3 – Xilinx Slave Serial Programmer detected (检测到 FX3 – Xilinx 从设备串行编程器)

图 11. 图像文件被下载到 FX3 的 RAM 内后的 FPGA 配置工具

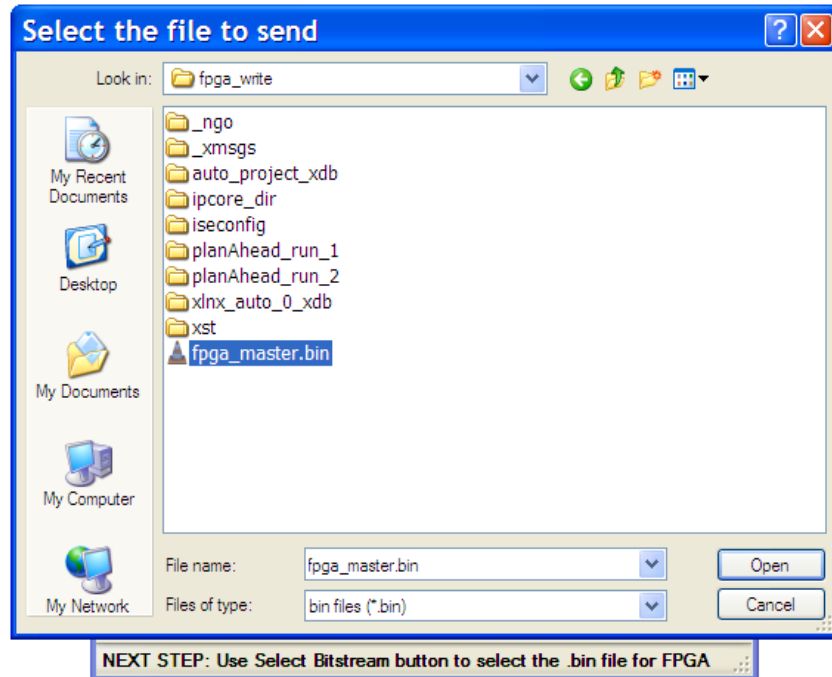


4. 当信息被显示在工具上时，点击 **Select Bitstream**，实现为 FPGA 选择 .bin 文件。

如果 *.bin* 文件不可用，并且您只有一个 *.bit* 文件，那么使用 **PromGen** 指令行将 *.bit* 文件转换为 *.bin* 文件。此外，您还可以使用 **iMPACT PROM File Formatter**，为 Xilinx PROM 创建一个 *.bin* 文件。请访问 www.xilinx.com/support.html，从而支持生成 *.bin* 文件。

5. 浏览到 *fpga_master.bin* 文件所在的位置，如图 12 所示。点击 **Open** 按钮。

图 12. 为 Xilinx FPGA 选择配置位文件 (.bin)



6. 点击 **Configure**，以对 Xilinx FPGA 进行配置，如图 13 所示。如果成功配置 FPGA，则 FX3 的固件会切换到从设备 FIFO 接口。显示了下面的状态信息，如图 14 所示。

Writing data to FPGA (正在将数据写入到 FPGA 内)

Configuration data has been sent to FPGA (配置的数据已发送到 FPGA)

Configurations Successful (配置成功)

FX3 Slave FIFO interface is activated (FX3 从设备 FIFO 接口被激活)

图 13. 选择“.bin”文件后的 FPGA 配置工具

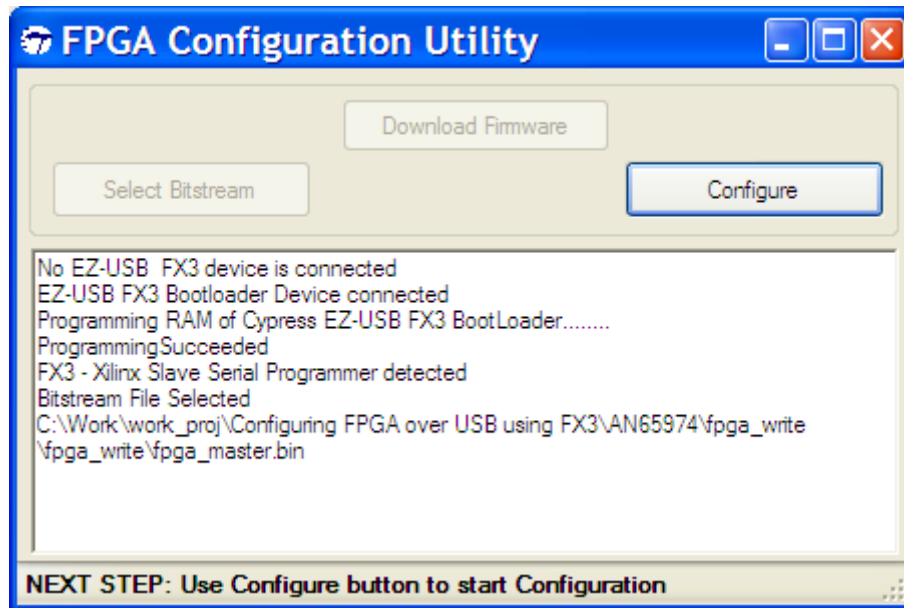
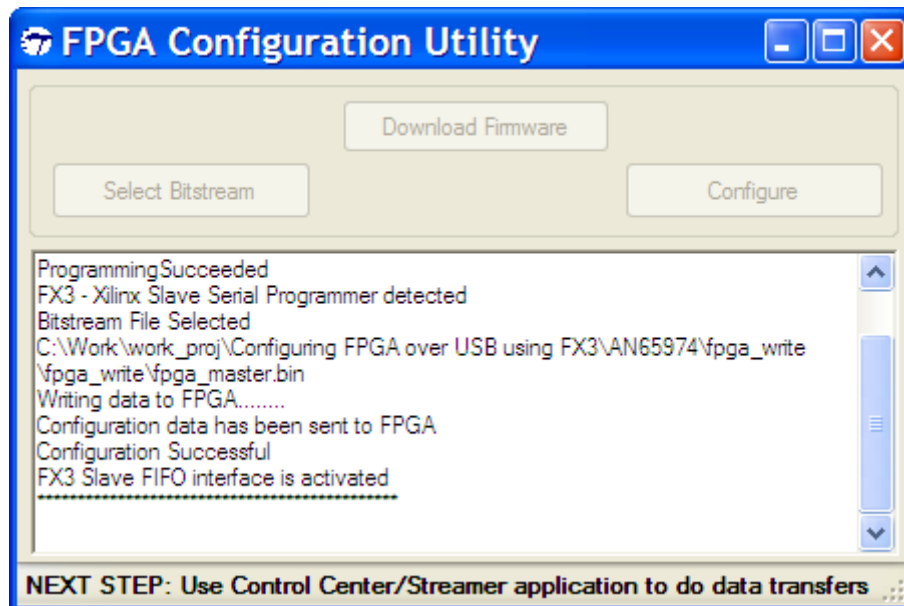
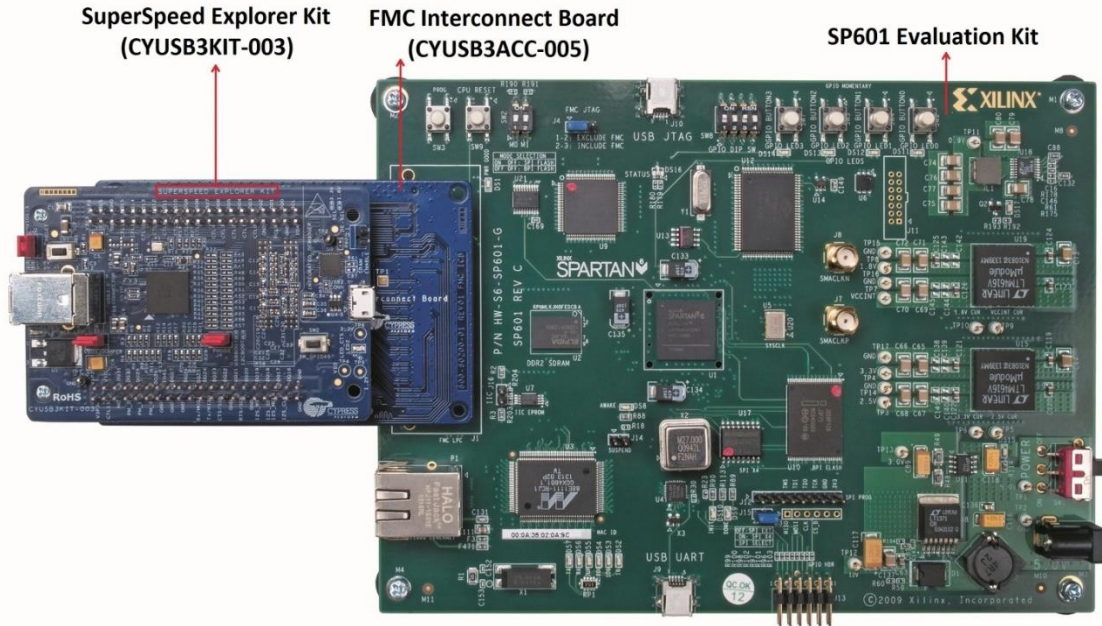


图 14. 成功执行 FPGA 配置后的 FPGA 配置工具



成功配置后，在 Xilinx FPGA 电路板上可以观察到 DS9 LED 发光。如果在配置过程中出现错误，它不会发光。图 15 显示 DS9 LED 发光。

图 15. 成功配置 FPGA 后的硬件设置

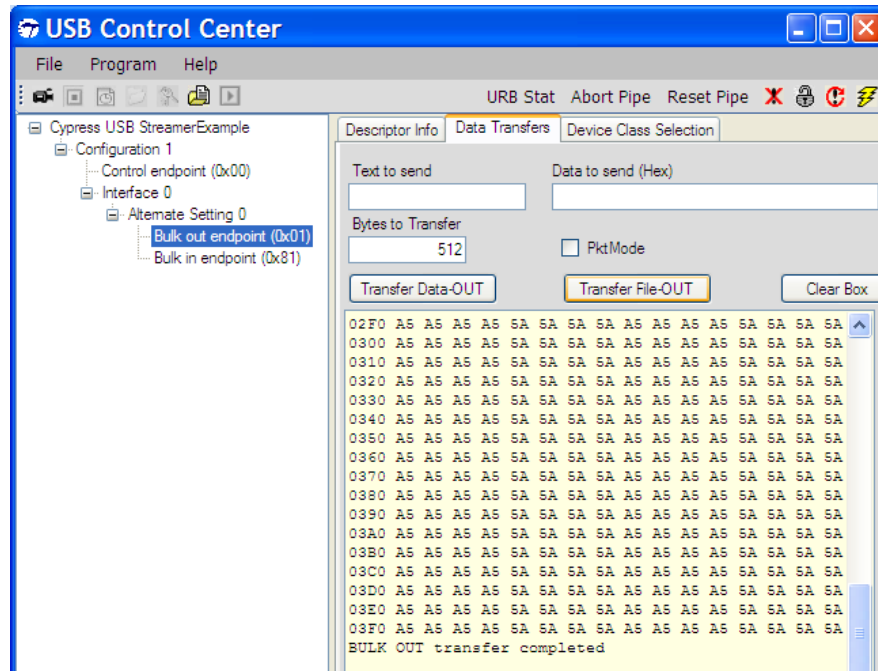


7. 使用控制中心应用来验证 FX3 和 Xilinx FPGA 间的回送操作。确保 SP601 评估套件上的 SW8 开关保持以下模式

SW8[1]	SW8[2]	SW8[3]	SW8[4]
OFF	OFF	OFF	ON

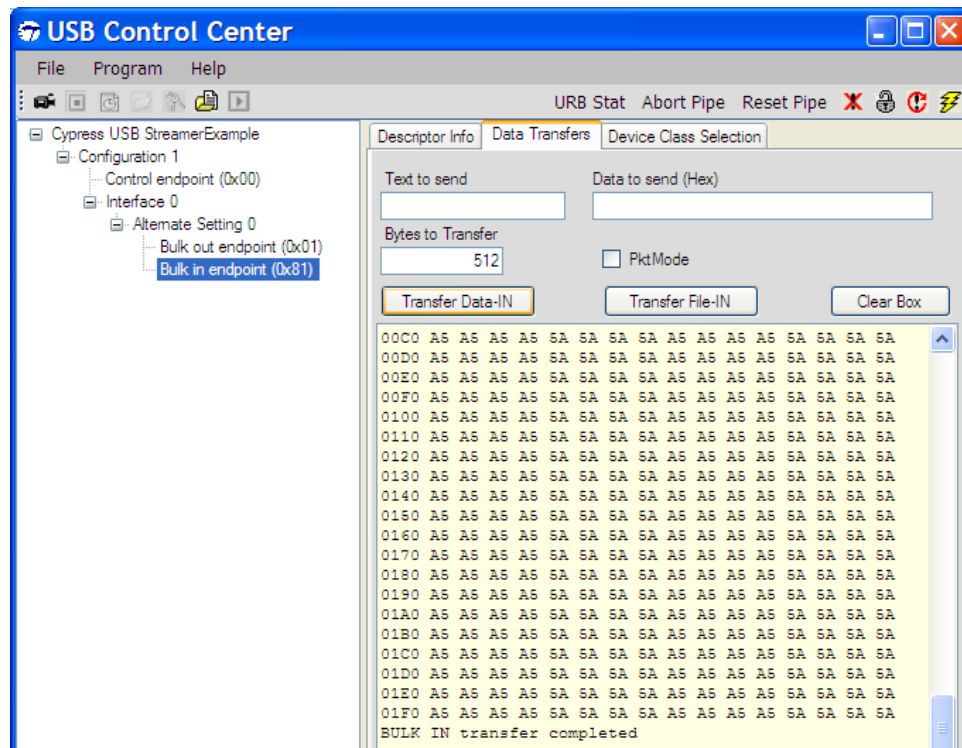
进入 **Bulk out endpoint (0x01)**，并点击 **Transfer File-OUT** 按键，以传输名称为 **TEST.txt** 的文件，它位于同一个文件夹中。然后，您可以看见成功地将一系列 **A5 A5 A5 A5 5A 5A 5A 5A** 传输到 **Bulk out endpoint**，如图 16 所示。

图 16. 通过 Bulk Out Endpoint (批量输出端点) 传输“TEST.txt”文件后的 USB 控制中心



- 选择 Bulk in endpoint，并点击 Transfer Data IN 按钮。然后，您可以观察到已接收的数据同已传送到 Bulk outendpoint 中的数据相类似，如图 17 所示。数据路径具体如下：**Control Center > Bulk out endpoint of FX3 > FPGA reads the data from Bulk out endpoint > FPGA writes the same data to Bulk in endpoint of FX3 > Control Center**。

图 17. 执行数据输入 (Data-IN) 以得到来自 Bulk in endpoint (批量输入端点) 的数据后的 USB 控制中心



5 总结

本应用笔记展示了采用赛普拉斯 FX3 通过 USB 有效配置 Xilinx FPGA 的解决方案。您可以将该解决方案集成到某个系统中，其中为实现 USB 3.0 功能将 FPGA 作为 FX3 接口使用，而不需要专用的编程电路来配置 FPGA。

6 相关的项目文件

表 4 显示了附带本应用笔记的文件。

表 4. 附件中的文件说明

文件夹名称	说明
FPGA 配置工具	PC 端应用的源代码
FX3 固件	FX3 固件中的源代码
fpga_write	Xilinx FPGA 的源代码作为主设备使用。它类似于 AN65974 中提供的 FPGA 代码。
bin	它包括以下文件： <i>TEST.txt</i> — 用于测试 FX3 和 FPGA 间的回送操作的数据文件。 <i>ConfigFpgaSlaveFifoSync.img</i> — FX3 固件的图像文件。 <i>Template.exe</i> — FPGA 配置工具的可执行文件。

7 参考

- [CYUSB3014 数据手册](#)
- [FX3 入门](#)
- [FX3 从设备 FIFO 接口](#)
- [Spartan-6 新一代配置用户指南 — Xilinx UG380](#)
- [Xilinx Spartan-6 FPGA SP601 评估套件](#)
- [使用微处理器通过从设备串行或 SelectMAP 模式来配置 Xilinx FPGA](#)
- [FX3 + FPGA + HelionVision ISP-Based Industrial Camera Reference Design – KBA222700](#)

关于作者

姓名: Rama Sai Krishna
职务: 应用工程师

文档修订记录

文档标题: AN84868 — 使用赛普拉斯 EZ-USB FX3 通过 USB 配置 Xilinx FPGA

文档编号: 001-91956

版本	ECN	变更者	提交日期	变更说明
**	4337816	HENG	04/09/2014	本文档版本号为 Rev**, 译自英文版 001-84868 Rev**。
*A	4885103	HENG	08/17/2015	本文档版本号为 Rev*A, 译自英文版 001-84868 Rev*B。
*B	5165278	HENG	02/23/2016	本文档版本号为 Rev*B, 译自英文版 001-84868 Rev*C。
*C	5716009	AESATMP9	04/27/2017	更新徽标和版权。
*D	6485043	JSLI	02/14/2019	本文档版本号为 Rev*D, 译自英文版 001-84868 Rev*E。

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 MCU](#)

赛普拉斯开发者社区

[社区](#) | [项目](#) | [视频](#) | [博客](#) | [培训](#) | [组件](#)

技术支持

cypress.com/support

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 Semiconductor
198 Champion Court
San Jose, CA 95134-1709

赛普拉斯半导体公司，2013-2019 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。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的安全问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）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。