M55M1 USB UVC

NuMicro® 32位系列微控制器范例程序介绍

文件信息

应用简述	本范例程式展示在 M55M1 开发平台上实现 USB UVC(USB Video Device Class)功能设计。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
开发平台	NuMaker-M55M1 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例程式将展示整合 UVC 设备 (UVC, USB Video Class) 与 NuMaker-M55M1 开发平台，并连接 HM1055 CMOS 影像传感器来捕捉影像。UVC 设备不需要安装任何的驱动程序下，连接至 PC 即可即插即用的摄像机功能。(本程序为 UVC 1.0 的版本为范例)

1.1 NuMaker-M55M1 开发平台

NuMicro® M55M1 系列微控制器以其卓越的性能，在数字元信号处理和设备内机器学习推论方面表现出色。它搭载高达 220 MHz 运行的 Arm® Cortex®-M55 处理器，除了拥有 Cortex-M55 核心外，还整合了 Arm® Ethos®-U55 神经处理单元(NPU)作为协处理器，用于加速神经网络操作。开发板方面提供丰富的通信和外设接口，包括通用串行控制接口(USCI)，可灵活设置为 UART/SPI/I²C 、USB 2.0、以太网 MAC with RMII 和 CCAP 界面。如图 1-1 展示。

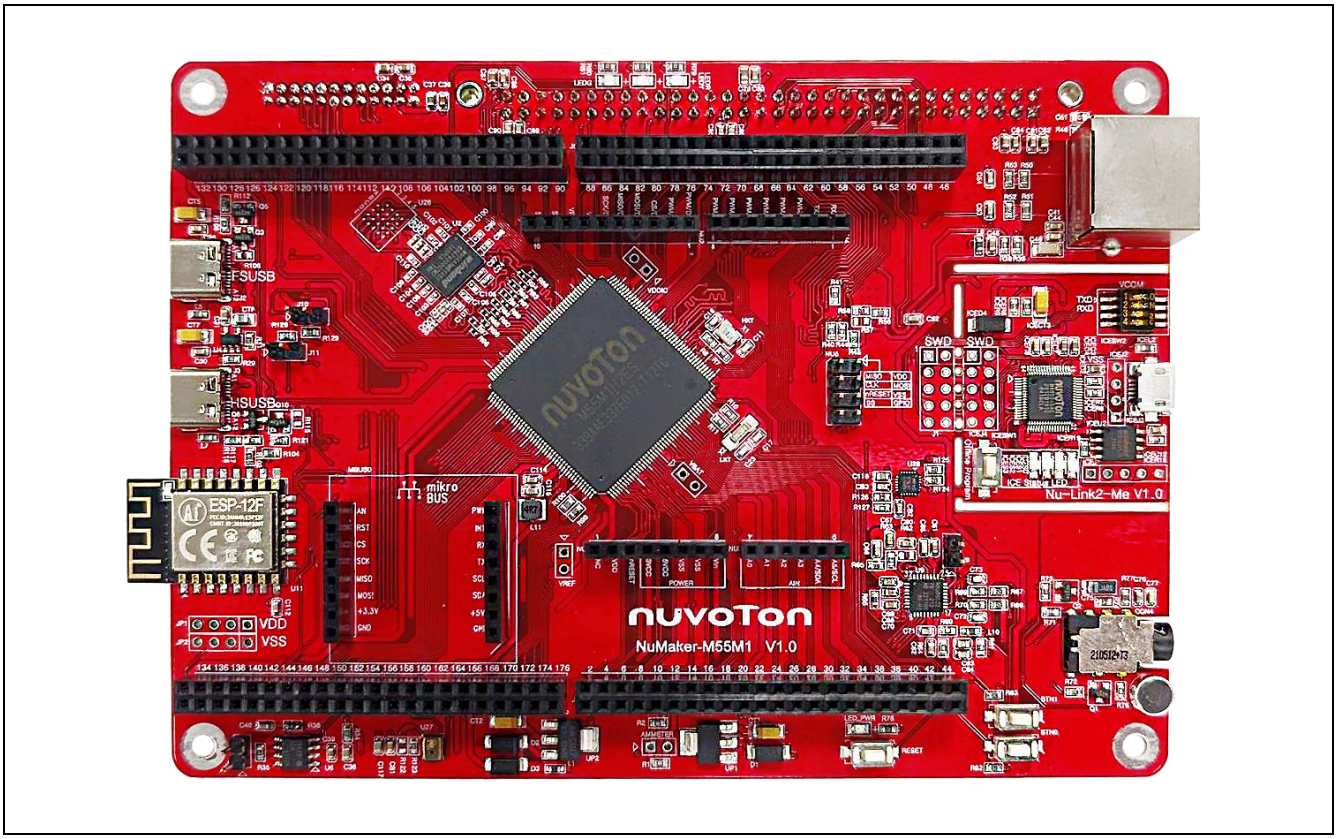

图 1-1 NuMaker-M55M1 V1.0 开发平台

1.2 影像传感器

本范例使用 HM1055 CMOS 影像传感器，提供高质量的图像，具有良好的灵敏度和低噪声，非常适合多种成像应用。具有一百万像素，影像大小为 1280x720 的 Area Scan Sensor。

HM1055 CMOS 影像传感器如图 1-2 所示，使用者仅需把传感器安装到 M55M1 开发平台的 CCAP Connector 相连接，即完成硬件接线。

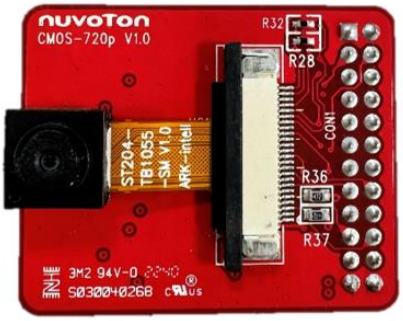

图 1-2 HM1055 CMOS Image Sensor

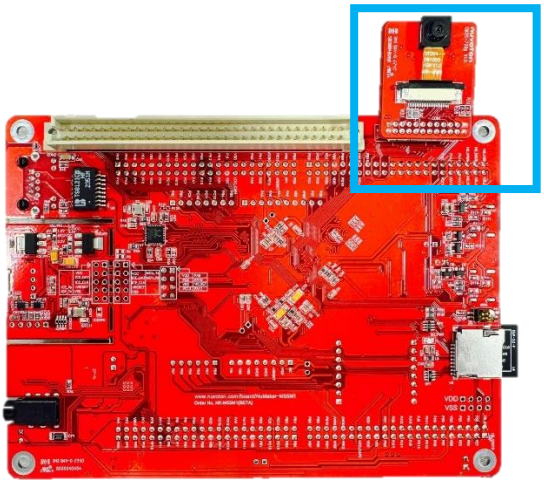

图 1-3 NuMaker-M55M1 V1.0 连接 HM1055 CMOS Image Sensor

1.3 UVC 及 USB Descriptor 的工作原理解析

1.3.1 背景介绍

图像处理和传输系统从 CMOS 传感器开始捕捉影像，经过处理模块处理，并最终通过 USB 传输到 PC 显示。这个过程涉及多个关键组件和功能，包括 CMOS 传感器(如 HM1055)、图像处理模块(CCAP)以及 USB 传输模块。USB 视频类(UVC)是一个标准类规范，标准化了 USB 上的视频流功能，使得网络摄像头、数字摄像机、视频转换器、和数字电视调谐器等设备能够与主机无缝连接。UVC 支持多种视频格式的流式传输，包括 YUV、MJPEG、MPEG-2 TS、H.264、DV 等，并提供了描述视频设备功能的结构，定义了 USB 根据请求控制设备不同参数和视频流特征的方法。UVC 还提供了支持多种视频分辨率、格式和帧速率的灵活性，这对设备和主机之间的带宽协商至关重要。大多数操作系统平台都支持 UVC 驱动程序，这大大减少了开发人员创建 USB 视频设备所需的时间。

1.3.2 UVC 模块示意图

图 1-4 UVC 架构图

影像系统由以下组成：相机传感器模块负责使用光学仪器捕捉现实世界的图像；UVC 设备处理这些数据并符合 UVC 规范；PC 则识别固件信息，加载相应驱动程序，与固件通信，并实现相机数据的传输与显示。

- 相机的 Sensor 模块，使用光学仪器采集实景。
- UVC 设备：对采集到的数据进行处理,并实现 UVC 相关的规范。
- PC：识别韧体讯息，加载驱动，与韧体通讯，实现摄影机数据的传输及显示。

1.3.3 图像处理和传输过程

整个图像处理和传输过程从 CMOS 传感器(Camera Sensor Module)的初始化开始。在 Sensor_HM1055.c 档案中，CMOS 传感器的初始化由 InitSensor() 函数完成。这个函数负责设置传感器的输出格式，确保传感器能够正确地捕捉影像数据。随后，传感器开始捕捉影像数据，并通过 CaptureFrame() 函数将影像数据进行数字化，准备进入下一步的处理阶段。

捕捉到的影像数据随后会进入 M55M1 CCAP 模块进行处理。在这个阶段，PacketFormatDownScale() 函数负责将影像数据调整到所需的格式和大小。这里使用了 CCAP_Start() 函数来启动影像捕捉接口，并通过 CCAP_SetPacketBuf() 和 CCAP_SetPacketScaling() 函数将数据存储到内存中，为后续的 USB 传输做好准备。

一旦影像数据准备好，接下来的步骤是将其打包并透过 USB 传输。在 uvc.c 中，这个过程由 UVC_SendImage() 函数负责。首先，该函数重置 DMA 传输，然后根据是否为静态影像或视频预览来设置相应的数据头。随后，数据通过 UVC_IsoIn() 函数以等时传输的方式发送到主机。这里使用了 UVC_ActiveDMA() 函数来启动 USB DMA 传输，确保数据可以稳定地传输到主机。

当数据到达主机端后，操作系统和驱动程序负责将数据包重组并展示为连续的视频流。这一过程主要在主机端进行，通过操作系统的驱动和软件来完成。

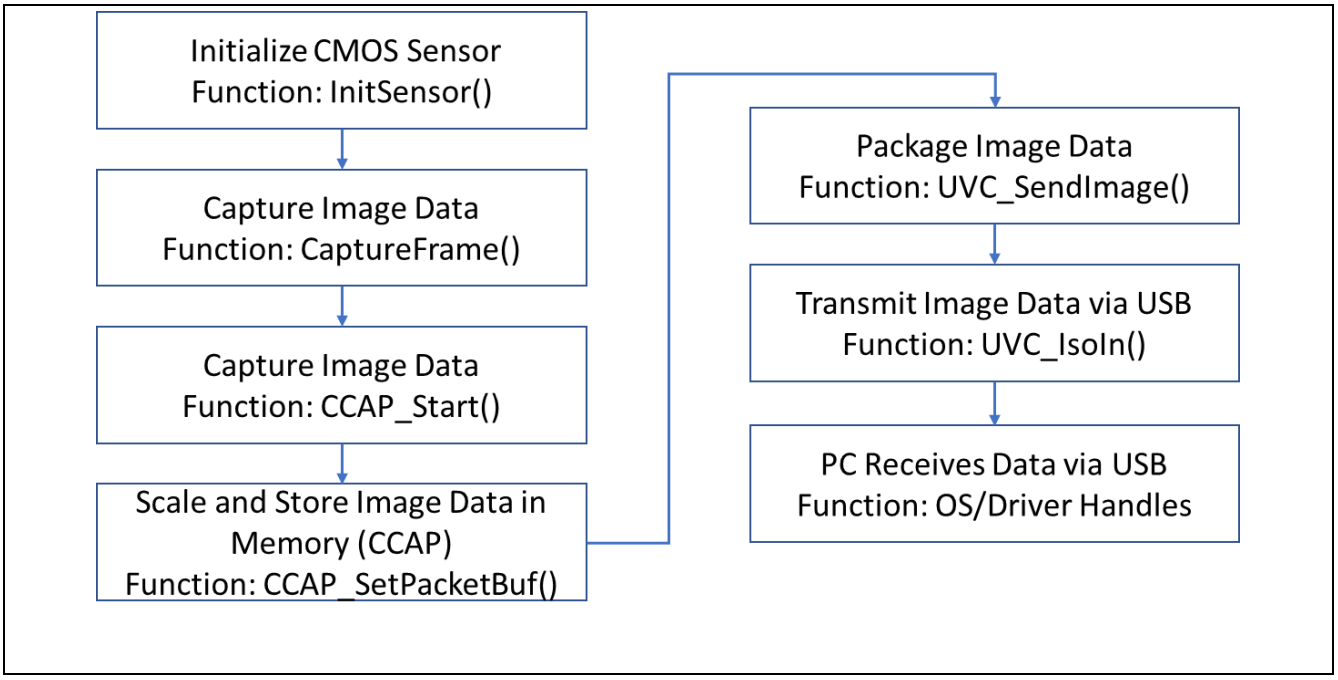

图 1-5 程序流程图

1.3.4 UVC(USB Video Class)的概念

UVC 是一个标准的 USB 装置类型，用于处理视频数据流，例如从网络摄像头将视频传输到计算器。UVC 装置使用标准协议与主机交互，许多操作系统原生支持它，因此 UVC 装置不需要特定驱动程序。

1.3.5 USB 描述符(Descriptors)概述

USB 描述符是一组结构化的数据，用来描述 USB 装置的属性、功能和需求。这些描述符帮助主机识别装置类型、可用功能和传输需求。主要的描述符包括：

- 设备描述符(Device Descriptor)：描述 USB 装置的基本信息，如装置类型、制造商和产品 ID。
- 配置描述符(Configuration Descriptor)：描述装置的配置，如电源需求、接口数量等。
- 界面描述符(Interface Descriptor)：描述装置的各种接口，如视频控制接口或视频流接口。
- 端点描述符(Endpoint Descriptor)：定义数据传输的具体端点和方式。

1.3.6 描述符在 UVC 装置中的作用

在 UVC 装置中，描述符详细说明了装置如何与主机交互并传输视频数据。以下是每个描述符在视频传输中的具体作用，以及程序代码中的实现方式：

1.3.6.1 设备描述符(Device Descriptor)

设备描述符提供了 USB 装置的基本信息，如装置类型、厂商 ID、产品 ID 和支持的 USB 版本等。这是主机获取装置信息的第一步，通过这个描述符，主机能识别装置的基本属性。

```
const uint8_t gu8DeviceDescriptor[] = {
    0x12,                   // bLength: 描述符的长度（18 bytes）
    USB_DT_DEVICE,          // bDescriptorType: 描述符类型（Device）
    0x10, 0x01,             // bcdUSB: USB 版本（USB 1.1）
    0x00,                   // bDeviceClass: 装置类型（由接口定义）
    0x00,                   // bDeviceSubClass: 子类型
    0x00,                   // bDeviceProtocol: 协议
    EP0_MAX_PKT_SIZE,       // bMaxPacketSize0: 端点 0 的最大封包大小
    LOBYTE(USBD_VID),       // idVendor: 装置的厂商 ID
    HIBYTE(USBD_VID),       // idProduct: 装置的产品 ID
    0x00, 0x01,             // bcdDevice: 装置版本号
    0x01,                   // iManufacturer: 厂商名称字符串索引
    0x02,                   // iProduct: 产品名称字符串索引
    0x03,                   // iSerialNumber: 序列号字符串索引
    0x01                    // bNumConfigurations: 支持的配置数量
```

```
};
```

1.3.6.2 配置描述符(Configuration Descriptor)

配置描述符描述了装置的配置选项，包括使用的电源、支持的接口数量等。主机通过这个描述符来配置装置，使其能正常工作。

```
const uint8_t gu8ConfigDescriptor[] = {
    0x09,                   // bLength: 描述符的长度（9 bytes）
    USB_DT_CONFIG,          // bDescriptorType: 描述符类型（Configuration）
    USB_CONFIG_DESC_SIZE,   // wTotalLength: 包含所有子描述符的总长度
    0x02,                   // bNumInterfaces: 支持的接口数量
    0x01,                   // bConfigurationValue: 配置值（用于选择此配置）
    0x00,                   // iConfiguration: 配置描述字符串索引
    0x80,                   // bmAttributes: 配置特性（如是否总线供电）
    0x32                    // MaxPower: 最大电流（100mA）
};
```

bNumInterfaces 表示此配置支持两个接口，一个是视频控制接口(Video Control Interface)，另一个是视频流界面(Video Streaming Interface)。

1.3.6.3 界面描述符(Interface Descriptor)

接口描述符详细描述了配置中的每个接口，包括它们所提供的功能(如视频控制或视频流)、使用的端点数量等。

```
const uint8_t gu8InterfaceDescriptor[] = {
    0x09,                   // bLength: 描述符的长度（9 bytes）
    USB_DT_INTERFACE,       // bDescriptorType: 描述符类型（Interface）
    0x00,                   // bInterfaceNumber: 界面编号
    0x00,                   // bAlternateSetting: 替代设置
    0x01,                   // bNumEndpoints: 此接口使用的端点数量（1个端点）
    USB_CLASS_VIDEO,        // bInterfaceClass: 界面类型（视频类）
    USB_SUBCLASS_VIDEOCONTROL, // bInterfaceSubClass: 子类型（视频控制）
    USB_PROTOCOL_UNDEFINED, // bInterfaceProtocol: 协议
    0x00                    // iInterface: 接口描述字符串索引
};
```

这里的 bInterfaceClass 设置为 USB_CLASS_VIDEO，表示这是一个视频类界面。

bNumEndpoints 设置为 1，表明此接口仅使用一个端点来传输数据。

1.3.6.4 端点描述符(Endpoint Descriptor)

端点描述符定义了数据在主机和装置之间传输的具体方式。对于视频流，端点通常配置为等时传输(Isochronous Transfer)以确保数据的稳定性和实时性。

```
const uint8_t gu8EndpointDescriptor[] = {
    0x07,                   // bLength: 描述符的长度（7 bytes）
    USB_DT_ENDPOINT,        // bDescriptorType: 描述符类型（Endpoint）
```

```
    ISO_IN_EP_NUM,          // bEndpointAddress: 端点地址（IN端点）
    USB_ENDPOINT_TYPE_ISOCHRONOUS, // bmAttributes: 端点属性（等时传输）
    LOBYTE(MAX_ISO_PACKET_SIZE),  // wMaxPacketSize: 最大封包大小
    HIBYTE(MAX_ISO_PACKET_SIZE),
    0x01                    // bInterval: 轮询间隔（1ms）
};
```

bEndpointAddress 表示这是一个 IN 端点，用于将数据从装置传输到主机。

bmAttributes 设置为 USB_ENDPOINT_TYPE_ISOCHRONOUS，表示这是等时端点，适合传输视频流。

1.4 CMOS 影像数据传输的完整过程

1.4.1 CMOS 影像捕捉

CMOS 传感器负责捕捉影像并将其数字元化，这是整个视频传输的第一步。

在 Sensor_HM1055.c 中，初始化并启动影像捕捉的部分：

```
/* Initialize sensor and set output format as YUV422 */
if (psSensorInfo->pfnInitSensor(0) == FALSE) {
    printf("Init sensor failed !\n");
    return -1;
}
```

1.4.2 图像处理和数据打包

捕捉到的影像数据需要进行处理并打包成 USB 能够传输的数据包。这包括数据缩放和格式转换，并最终将数据存储在准备传输的内存中。

在 PacketFormatDownScale() 函数中：

```
/* Set System Memory Packet Base Address Register */
CCAP_SetPacketBuf((uint32_t)u8FrameBuffer0);

/* Set Packet Scaling Vertical/Horizontal Factor Register */
CCAP_SetPacketScaling(SYSTEM_HEIGHT, psSensorInfo->m_u16Height, SYSTEM_WIDTH,
psSensorInfo->m_u16Width);

/* Set Packet Frame Output Pixel Stride Width */
CCAP_SetPacketStride(SYSTEM_WIDTH);

/* Start Image Capture Interface */
CCAP_Start();
```

这段程序代码将捕捉的影像数据进行缩放并准备好内存地址，供后续的 USB 传输使用。

1.4.3 数据传输到主机

影像数据被打包后，通过 USB 传输到主机端。这里使用了等时传输模式，确保视频流的实时性。

在 UVC_SendImage() 函数中：

```
HSUSBD_ResetDMA();

if (bStillImage) {
    g_u8UVC_PD = UVC_PH_Snapshot;  /* Payload Header for Snapshot */
    uvcStatus.StillImage = 0;
} else {
    g_u8UVC_PD = UVC_PH_Preview;  /* Payload Header for Preview */
}

UVC_IsoIn(u32Addr, u32transferSize);
```

UVC_SendImage() 函数负责将数据传输到主机，其中 UVC_IsoIn() 函数通过 USB 等时传输将数据发送出去。输入的 u32Addr 是影像数据的地址，而 u32transferSize 则是传输数据的大小。

1.4.4 主机重组数据和显示视频

主机接收到这些数据包后，会根据 UVC 协议重组数据并显示视频。在程序代码层面上，这个过程主要由主机端的操作系统和驱动程序完成，具体的程序代码在装置端是不可见的。

在 UVC_IsoIn() 函数中：

```
HSUSBD_SET_DMA_READ(ISO_IN_EP_NUM);
UVC_ActiveDMA(u32Addr, u32Len);
```

UVC_ActiveDMA() 函数启动 USB DMA 传输，这是数据传输到主机的重要一步。

1.5 影像数据打包过程的详细说明

根据 USB UVC 的封包内容，分为许多的字段，需要填入对应的数据，然后打包发送给 PC，封包各字段说明如下：

- UVC Payload Header：这是封包的头部，包含了帧的控制信息。
- bHeaderLength：描述了 UVC Payload Header 的长度。
- bmHeaderInfo：包含帧标志位，表示这是帧的开始或结束，也可以标记帧的类型(例如静态图片)。
- dwPresentationTime：一个 4 字节的时间戳，用来同步影像的播放时间。

- dwSourceClock：4 字节的时钟时间戳，用来同步影像的捕捉时间。
- Image Data Payload：这是封包的主要内容，包含实际的影像数据。
- Video Data：这一部分存储了捕捉到的影像数据，其大小取决于影像的分辨率和帧格式。
- USB Packet Footer：这部分是可选的，用来存放一些附加信息，例如错误检查码(如 CRC)。
- Error Check (CRC)：如果系统支持，可以在封包结尾加入错误检查码，以确保数据的完整性。

1.6 封包结构在程序代码中的对应

在程序代码中，这些部分由 UVC_SendImage() 函数负责打包和传输。例如，UVC_PH_Preview 和 UVC_PH_Snapshot 对应了 bmHeaderInfo，而影像数据则是从影像缓冲区中读取并存放在封包中。

```
if (bStillImage) {
    g_u8UVC_PD = UVC_PH_Snapshot;  /* Payload Header for Snapshot */
    uvcStatus.StillImage = 0;
} else {
    g_u8UVC_PD = UVC_PH_Preview;  /* Payload Header for Preview */
}
```

这段程序代码设定了 UVC Payload Header 中的 bmHeaderInfo，并根据是否是静态影像或视频来决定封包的类型。

1.7 UVC 的 Frame Rate

这段 UVC 的程序，传送 1 张 image 到 PC 的时间是 75ms，FPS 大约是 13 FPS

```
while (1)
{

    u32TimeVal = g_u32Ticks;

    if (gAppConnected == 1)
    {
        if (s_u32XferBufAddr) //Trigger ISO-IN DMA
        {
            UVC_SendImage(s_u32XferBufAddr, s_u32XferSize, uvcStatus.StillImage);
            s_u32XferBufAddr = 0;  // Set ISO-IN IDLE
        }
    }

    u32TimeVal_Vec = g_u32Ticks;
```

```
printf("Time = %d ms\n", (u32TimeVal_Vec - u32TimeVal) / 1   );
```

執行結果：

```
Time = 75 ms
```

2. 程序介绍

主程序如下，初始化系统后设置 HM1055 并启动影像抓取，而后设置 USB Device 设备：

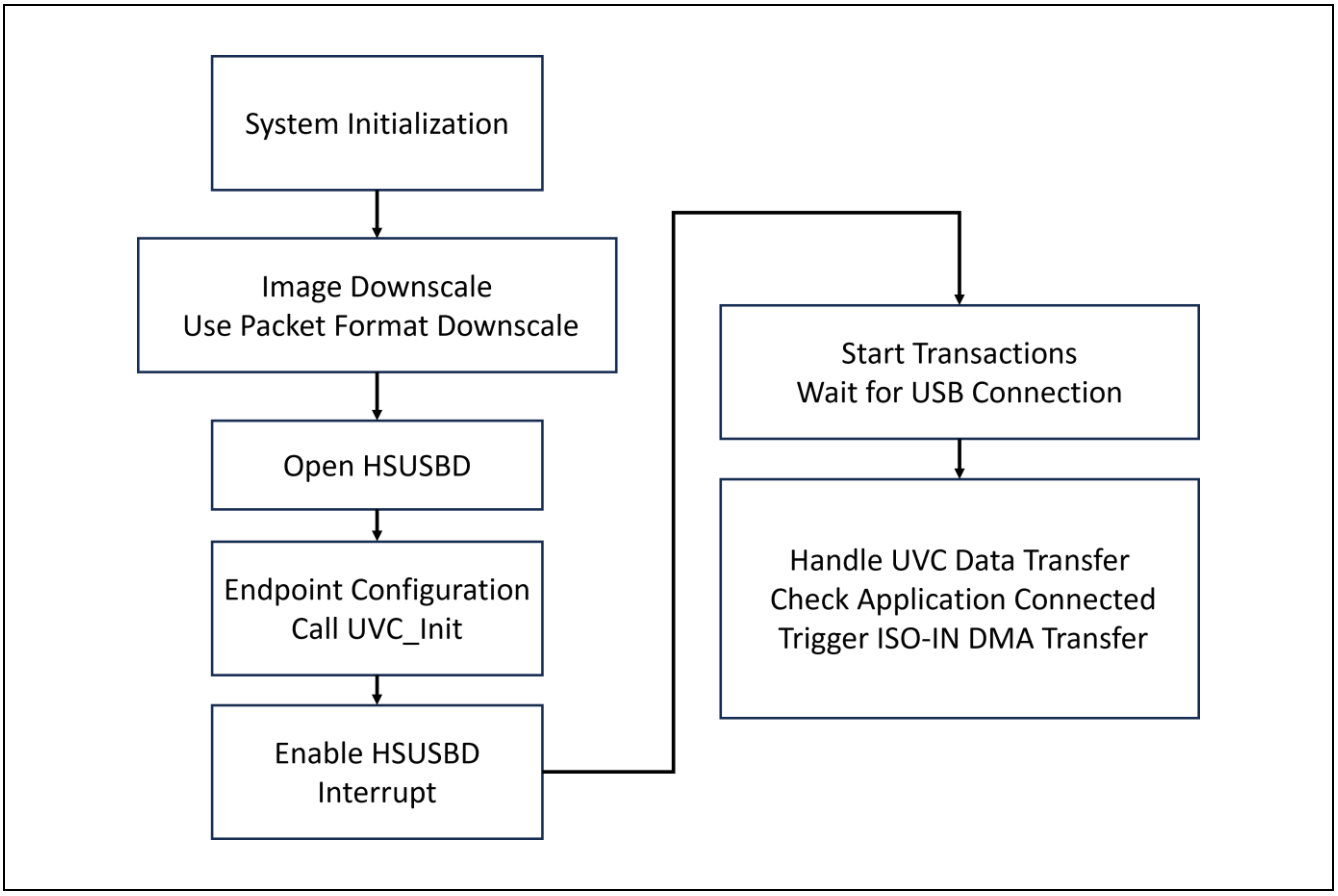

图 2-1 程序流程图

```
int32_t main(void)
{
    /* Init System, peripheral clock and multi-function I/O */
    SYS_Init();

    /* Init UART to 115200-8n1 for print message */
    InitDebugUart();

    printf("NuMicro HSUSBD UVC YUV422 Streaming\n");

    /* Using Packet format to Image down scale */
    if (PacketFormatDownScale(&g_sSensorHM1055) != 0)
        printf("Capture frame failed !\n");

    HSUSBD_Open(&gsHSInfo, UVC_ClassRequest, UVC_SetInterface);

    /* Endpoint configuration */
    UVC_Init();
```

```
    /* Enable HSUSBD interrupt */
    NVIC_EnableIRQ(HSUSBD_IRQn);

    /* Start transaction */
    while (1)
    {
        if (!HSUSBD_IS_ATTACHED())
        {
            HSUSBD_Start();
            break;
        }
    }

    while (1)
    {
        if (gAppConnected == 1)
        {
            if (s_u32XferBufAddr) //Trigger ISO-IN DMA
            {
                UVC_SendImage(s_u32XferBufAddr, s_u32XferSize, uvcStatus.StillImage);
                s_u32XferBufAddr = 0; // Set ISO-IN IDLE
            }
        }
    }
}
```

HM1055 的设置并初始化，配置影像捕捉 (CCAP) 与输出格式为 YUV422：

```
int32_t PacketFormatDownScale(S_SENSOR_INFO *psSensorInfo)
{
    uint32_t u32Frame;

    /* Init CCAP clock and Sensor clock */
    CCAP_SetFreq(12000000, 12000000);

    /* Initialize sensor and set output format as YUV422 */
    if (psSensorInfo->pfnInitSensor(0) == FALSE)
    {
        printf("Init sensor failed !\n");
        return -1;
    }

    /* Enable External CCAP Interrupt */
    NVIC_EnableIRQ(CCAP_IRQn);

    /* Enable CCAP Frame End Interrupt */
    CCAP_EnableInt(CCAP_INT_VIEN_ENABLE);

    /* Set Vsync polarity, Hsync polarity, pixel clock polarity, Sensor Format and Order
*/
    CCAP_Open(psSensorInfo->m_u32Polarity, psSensorInfo->m_u32InputFormat,
CCAP_PAR_OUTFMT_YUV422);

    /* Set Cropping Window Vertical/Horizontal Starting Address and Cropping Window Size
*/
    CCAP_SetCroppingWindow(0, 0, psSensorInfo->m_u16Height, psSensorInfo->m_u16Width);
```

```
    /* Set System Memory Packet Base Address Register */
    CCAP_SetPacketBuf((uint32_t)u8FrameBuffer0);

    /* Set Packet Scaling Vertical/Horizontal Factor Register */
    CCAP_SetPacketScaling(SYSTEM_HEIGHT, psSensorInfo->m_u16Height, SYSTEM_WIDTH,
psSensorInfo->m_u16Width);

    /* Set Packet Frame Output Pixel Stride Width */
    CCAP_SetPacketStride(SYSTEM_WIDTH);

    /* Start Image Capture Interface */
    CCAP_Start();

    return 0;
}
```

函式 ChangeFrame 用于改变视讯输入设置的函数，调整框架大小：

```
/*---------------------------------------------------------------------------------------------*/
/* Change VideoIN Setting for Frame size */
/*---------------------------------------------------------------------------------------------*/

void ChangeFrame(char bChangeSize, uint32_t u32Address, uint16_t u16Width, uint16_t
u16Height)
{
    if (bChangeSize)    /* Change Frame Size */
    {
        CCAP_SetPacketScaling(u16Height, SENSOR_IN_HEIGHT, u16Width, SENSOR_IN_WIDTH);
    }
    /* Set Buffer Address */
    CCAP_SetPacketBuf((uint32_t)u32Address);
    CCAP_SetPacketStride(u16Width);
}
```

USB UVC 的初始化设置：

```
void UVC_Init(void)
{
    /* Configure USB controller */
    /* Enable USB BUS, CEP and EPA global interrupt */
    HSUSBD_ENABLE_USB_INT(HSUSBD_GINTEN_USBIEN_Msk | HSUSBD_GINTEN_CEPIEN_Msk |
HSUSBD_GINTEN_EPAIEN_Msk | HSUSBD_GINTEN_EPCIEN_Msk);
    /* Enable BUS interrupt */
    HSUSBD_ENABLE_BUS_INT(HSUSBD_BUSINTEN_DMADONEIEN_Msk | HSUSBD_BUSINTEN_RESUMEIEN_Msk |
HSUSBD_BUSINTEN_RSTIEN_Msk | HSUSBD_BUSINTEN_VBUSDETIEN_Msk);
    /* Reset Address to 0 */
    HSUSBD_SET_ADDR(0);


    /* Control endpoint */
    HSUSBD_SetEpBufAddr(CEP, CEP_BUF_BASE, CEP_BUF_LEN);
```

```
    HSUSBD_ENABLE_CEP_INT(HSUSBD_CEPINTEN_SETUPPKIEN_Msk |
HSUSBD_CEPINTEN_STSDONEIEN_Msk);

    UVC_InitForHighSpeed();

    uvcInfo.u8ErrCode = 0;
    uvcInfo.bChangeData = 0;
    uvcStatus.StillImage = 0;

    uvcInfo.IsoMaxPacketSize[0] = gMaxPacketSize;
    uvcInfo.IsoMaxPacketSize[1] = gMaxPacketSize;
    uvcInfo.IsoMaxPacketSize[2] = gMaxPacketSize;

    //Initialize Video Capture Filter Control Value
    uvcInfo.CapFilter.Brightness = 1;
    uvcInfo.CapFilter.POWER_LINE_FREQUENCY = 2;
    uvcInfo.CapFilter.Brightness = 1;
    uvcInfo.CapFilter.Contrast = 2;
    uvcInfo.CapFilter.Hue = 3;
    uvcInfo.CapFilter.Saturation = 4;
    uvcInfo.CapFilter.Sharpness = 5;
    uvcInfo.CapFilter.Gamma = 3;
    uvcInfo.CapFilter.Backlight = 7;

    /* Initialize Video Probe and Commit Control data */
    memset((uint8_t *)&uvcInfo.VSCmdCtlData, 0x0, sizeof(VIDEOSTREAMCMDDATA));
    uvcInfo.VSCmdCtlData.bmHint = 0x0100;
    uvcInfo.VSCmdCtlData.bFormatIndex = 1;
    uvcInfo.VSCmdCtlData.bFrameIndex = 2;
    uvcStatus.FormatIndex = 1;
    uvcStatus.FrameIndex = 2;
    uvcInfo.VSCmdCtlData.dwFrameInterval = 0x051615;

    /* Initialize Still Image Command data */
    memset((uint8_t *)&uvcInfo.VSStillCmdCtlData, 0x0, sizeof(VIDEOSTREAMSTILLCMDDATA));

    uvcInfo.VSStillCmdCtlData.bFormatIndex = 1;
    uvcInfo.VSStillCmdCtlData.bFrameIndex = 1;
    uvcStatus.snapshotFormatIndex = 1;
    uvcStatus.snapshotFrameIndex = 1;
}
```

影像捕捉后，透过 USB 传输至 PC 端：

```
int UVC_SendImage(uint32_t u32Addr, uint32_t u32transferSize, uint32_t bStillImage)
{
    if (u32transferSize != 0)
    {

        uvcStatus.bReady = 0;
        HSUSBD_ResetDMA();

        if (bStillImage)
        {
            g_u8UVC_PD = UVC_PH_Snapshot;       /* Payload Header for Snapshot */
            uvcStatus.StillImage = 0;
```

```
        }
        else
            g_u8UVC_PD = UVC_PH_Preview;        /* Payload Header for Preview */

        UVC_IsoIn(u32Addr, u32transferSize);

        return 1;
    }
    return 0;
}
```

其中 CCAP_IRQHandler 是一个中断服务程序，用于处理影像捕获模块(CCAP)的中断：

在下列程序代码中，我们可以看到使用 2 块 frame buffer 在此切换，使用了两个 Frame Buffer(即 u8FrameBuffer0 和 u8FrameBuffer1)来实现双缓冲机制，主要目的是为了避免影像撕裂(tearing)问题并提升图像处理的稳定性。当一块缓冲区正在用来显示或传输影像时，另一块缓冲区则负责接收和写入新的一帧数据，这样可以确保不会同时读写同一块缓冲区，防止画面在读取过程中被修改导致撕裂现象。透过交替使用两个缓冲区，系统能够实现平滑的影像帧切换，避免出现部分显示新数据、部分显示旧数据的问题，同时也提升了数据传输的效率，确保图像处理的连续性与流畅度。

当 u32UVCXferIdx 为奇数时，将资料写入 u8FrameBuffer1，并预设下一帧将使用 u8FrameBuffer0。

当 u32UVCXferIdx 为偶数时，将资料写入 u8FrameBuffer0，并预设下一帧将使用 u8FrameBuffer1。

```
/*---------------------------------------------------------------------------*/
/* To run CCAPInterruptHandler, when CAP frame end interrupt                 */
/*---------------------------------------------------------------------------*/
void CCAPInterruptHandler(void)
{
    /* UVC transfer counter */
    static uint32_t u32UVCXferIdx = 0;

    /* Current Frame Index */
    static uint32_t u32CurFrameIndex = 2;

    /* Current Format Index */
    static uint32_t u32CurFormatIndex = 1;

    /* Current Width & Height */
    static uint16_t u16CurWidth = 320, u16CurHeight = 240;

    uint32_t u32NextFrameBufferAddr;

    if (s_u32XferBufAddr == 0) // UVC ISO-IN IDLE
    {
```

```
        if (u32UVCXferIdx & 0x1u)
        {
            s_u32XferBufAddr = (uint32_t)u8FrameBuffer1;
            u32NextFrameBufferAddr = (uint32_t)u8FrameBuffer0;
        }
        else
        {
            s_u32XferBufAddr = (uint32_t)u8FrameBuffer0;
            u32NextFrameBufferAddr = (uint32_t)u8FrameBuffer1;
        }

        /* UVC_Foramt_YUY2 */
        switch (uvcStatus.FormatIndex)
        {
        case UVC_Format_YUY2:       /* UVC_Foramt_YUY2 */
        {
            s_u32XferSize = uvcStatus.MaxVideoFrameSize;
            if (u32CurFrameIndex != uvcStatus.FrameIndex || u32CurFormatIndex !=
uvcStatus.FormatIndex)
            {
                /* Frame Size Changed */
                switch (uvcStatus.FrameIndex)
                {
                case UVC_VGA:
                    u16CurWidth = 640;
                    u16CurHeight = 480;
                    break;
                case UVC_QVGA:
                    u16CurWidth = 320;
                    u16CurHeight = 240;
                    break;
                case UVC_QQVGA:
                    u16CurWidth = 160;
                    u16CurHeight = 120;
                    break;
                }
                /* Set VideoIn Setting for Frame Size */
                ChangeFrame(TRUE, u32NextFrameBufferAddr, u16CurWidth, u16CurHeight);
                u32CurFrameIndex = uvcStatus.FrameIndex;
                u32CurFormatIndex = uvcStatus.FormatIndex;
            }
            else
                ChangeFrame(FALSE, u32NextFrameBufferAddr, u16CurWidth, u16CurHeight);

            break;
        }
        default:
            break;
        }

        u32UVCXferIdx++;
    }
}

/*---------------------------------------------------------------------------------------*/
/*  CCAP_IRQHandler                                                                      */
```

```
/*---------------------------------------------------------------------------------------------*/
NVT_ITCM void CCAP_IRQHandler(void)
{
    uint32_t u32IntStatus = CCAP_GET_INT_STS();

    if (CCAP_IsIntEnabled(CCAP_INT_VIEN_ENABLE) && (u32IntStatus & CCAP_INTSTS_VINTF_Msk)
== CCAP_INTSTS_VINTF_Msk)
    {
        CCAPInterruptHandler();
        CCAP_CLR_INT_FLAG(CCAP_INTSTS_VINTF_Msk); /* Clear Frame end interrupt */
    }

    if (CCAP_IsIntEnabled(CCAP_INT_ADDRMIEN_ENABLE) && (u32IntStatus &
CCAP_INTSTS_ADDRMINTF_Msk) == CCAP_INTSTS_ADDRMINTF_Msk)
    {
        CCAP_CLR_INT_FLAG(CCAP_INTSTS_ADDRMINTF_Msk);  /* Clear Address match interrupt */
    }

    if (CCAP_IsIntEnabled(CCAP_INT_MEIEN_ENABLE) && (u32IntStatus &
CCAP_INTSTS_MEINTF_Msk) == CCAP_INTSTS_MEINTF_Msk)
    {
        CCAP_CLR_INT_FLAG(CCAP_INTSTS_MEINTF_Msk);    /* Clear Memory error interrupt */
    }

    CCAP_EnableUpdate();

    // CPU read interrupt flag register to wait write(clear) instruction completement.
    u32IntStatus = CCAP_GET_INT_STS();
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.00.001

3.2 硬件需求

- 电路组件
  - NuMaker-M55M1 V1.0
  - HM1055 CMOS Sensor

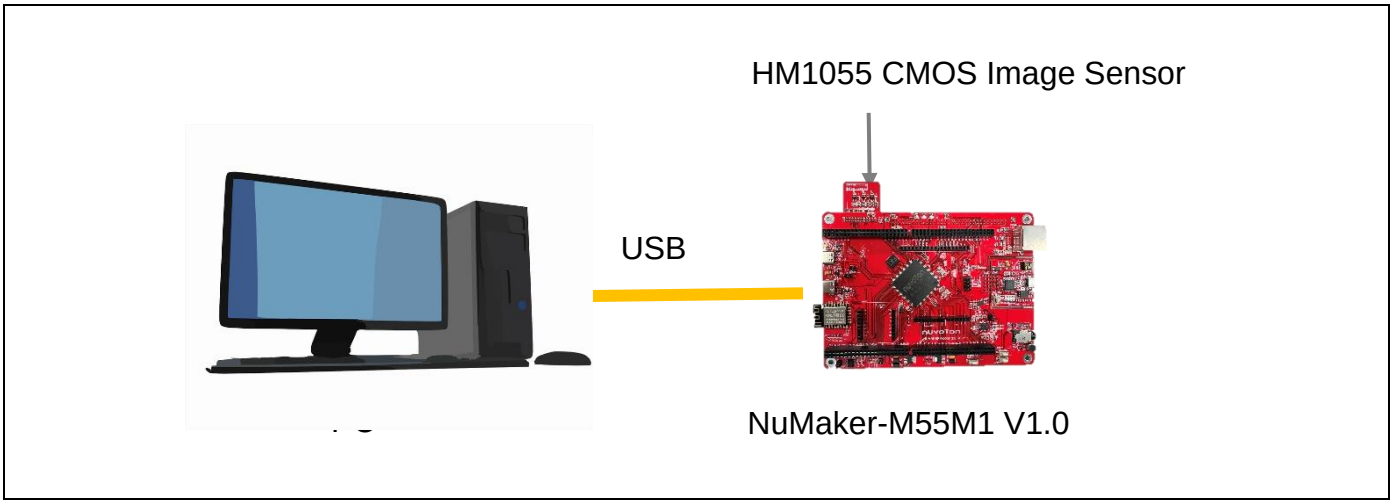

图 3-1 USB UVC 连接执行示意图

- CMOS 电路接线示意图
  - M55M1 使用 I²C 来传输控制命令至 CMOS 传感器，使用 CCAP_SCLK 作为 SCLK 供应频率给 CMOS 传感器，CMOS 传感器会根据设定回传 Pixel Clock (PIXCLK)、HSYNC、VSYNC 和 D0~D7 数据数据。

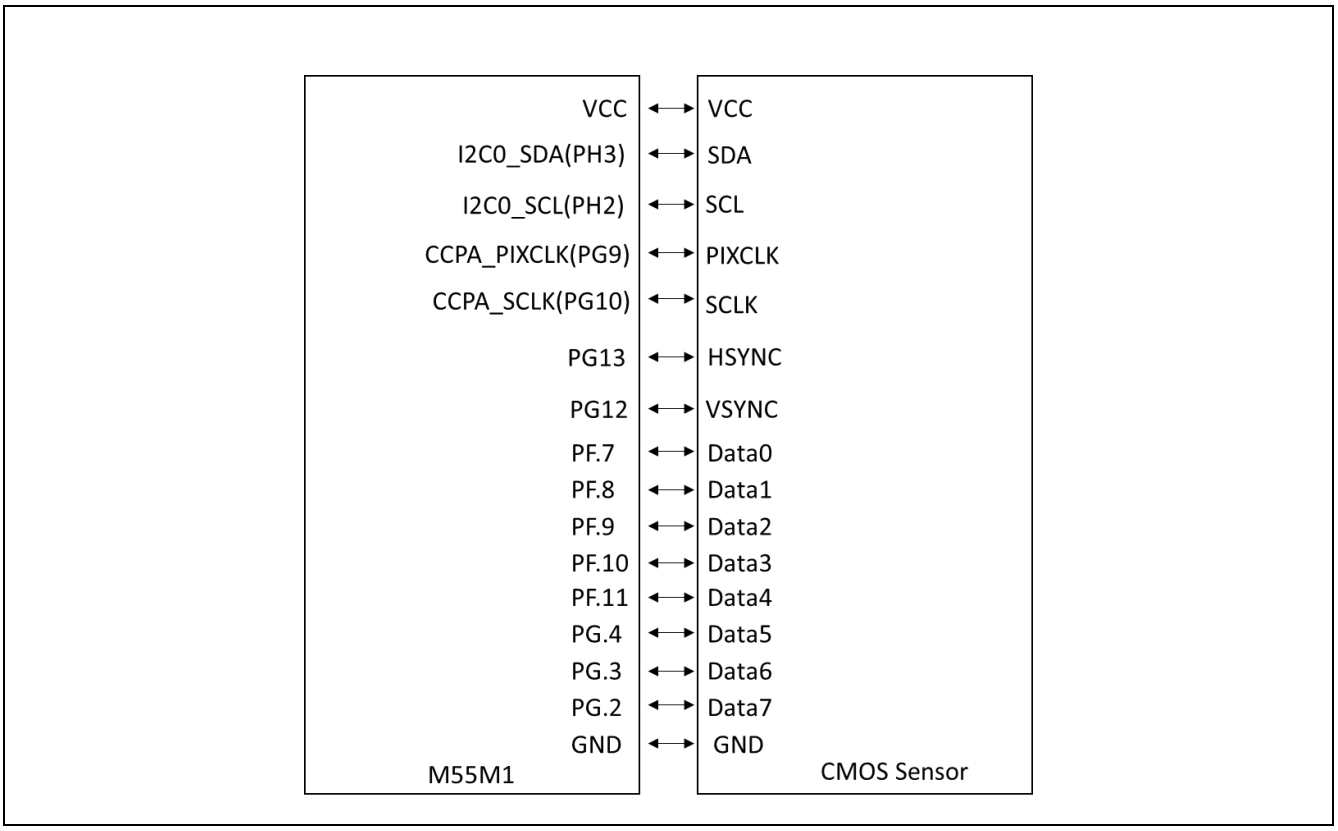

图 3-2 CMOS 线路示意图

4. 目录信息

目录结构如下图所示。

EC_M55M1_HSUSBD_UVC_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
HSUSBD_UVC	Source file of example code

图 4-1 目录信息

5. 范例程式执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 *HSUSBD_UVC_VideoSensor.uvprojx*。
2. 进入编译模式接口
   - 编译
   - 下载代码至内存
   - 进入 / 离开除错模式
3. 进入除错模式接口
   - 执行代码
   - 执行结果

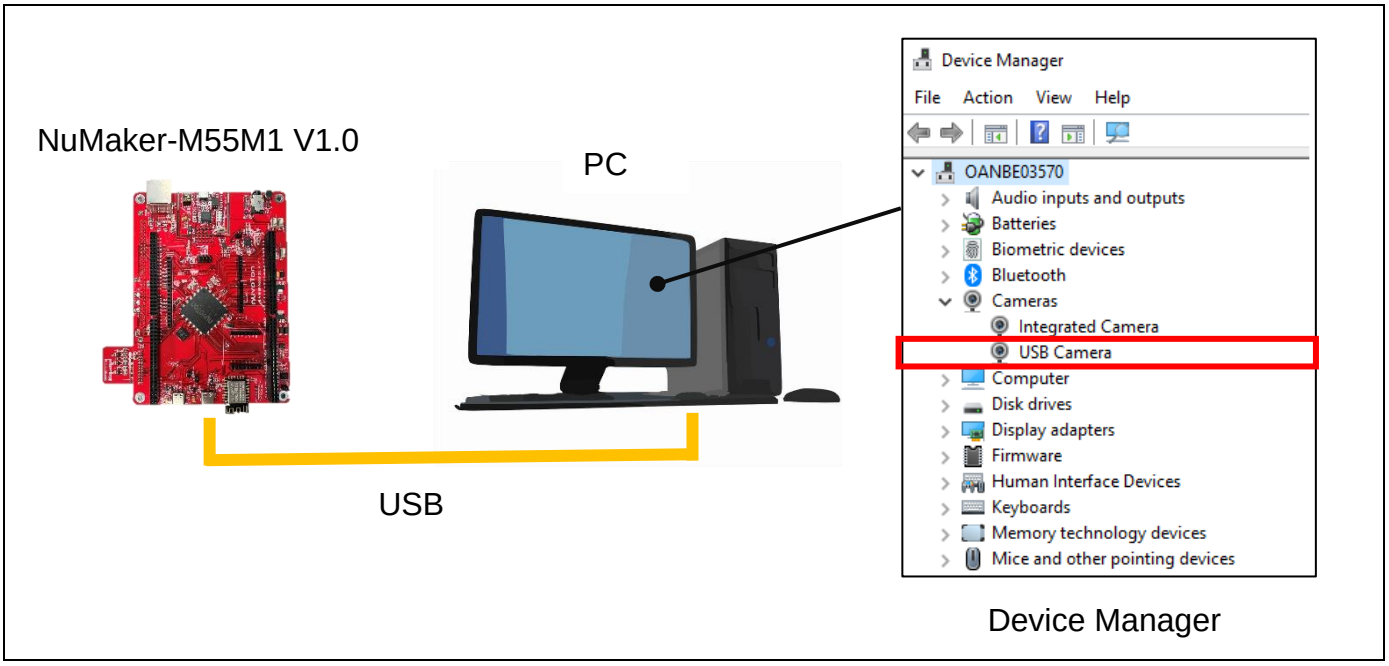

图 5-1 USB UVC 执行结果

6. 修订纪录

Date	Revision	Description
2024.06.14	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.