M55M1 USB UVC

NuMicro® 32位系列微控制器範例程式介紹

文件資訊

應用簡述	本範例程式展示在 M55M1 開發平台上實現 USB UVC(USB Video Device Class)功能設計。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
開發平台	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

本範例程式將展示整合 USB 視訊裝置類別 (UVC, USB Video Class) 與 NuMaker-M55M1 開發平台，並連接 HM1055 CMOS 影像感測器來捕捉影像。UVC 設備不需要安裝任何的驅動程式下，連接至 PC 即可隨插即用的攝像機功能。(本程式為 UVC 1.0 的版本為範例)

1.1 NuMaker-M55M1 開發平台

NuMicro® M55M1 系列微控制器以其卓越的性能，在數位信號處理和設備內機器學習推論方面表現出色。它搭載高達 220 MHz 運行的 Arm® Cortex®-M55 處理器，除了擁有 Cortex-M55 核心外，還整合了 Arm® Ethos®-U55 神經處理單元(NPU)作為輔助處理器，用於加速神經網路操作。開發板方面提供豐富的通信和外設介面，包括通用串列控制介面 (USCI)，可靈活設置為 UART/SPI/I²C 、USB 2.0、乙太網 MAC with RMII 和 CCAP 介面。如圖 1-1 展示。

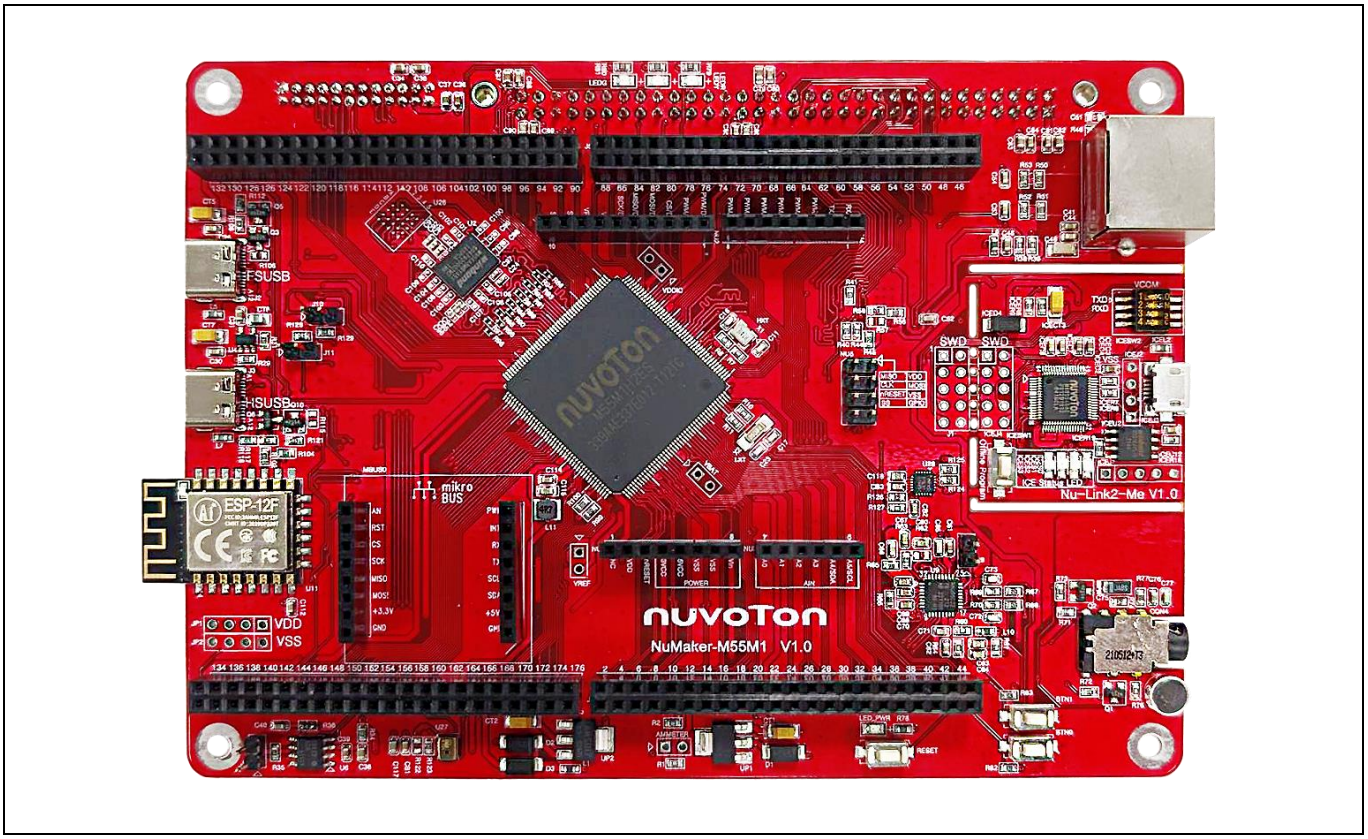

圖 1-1 NuMaker-M55M1 V1.0 開發平台

1.2 影像感測器

本範例使用 HM1055 CMOS 影像感測器，提供高品質的圖像，具有良好的靈敏度和低雜訊，非常適合多種成像應用。具有一百萬像素，影像大小為 1280x720 的 Area Scan Sensor。

HM1055 CMOS 影像感測器如圖 1-2 所示，使用者僅需把感測器安裝到 M55M1 開發平台的 CCAP Connector 相連接，即完成硬體接線。

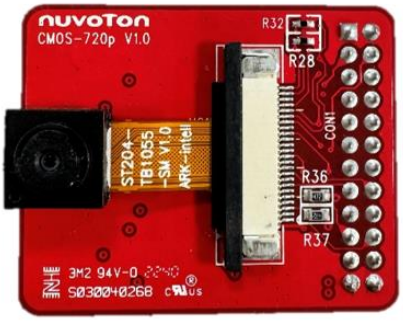

圖 1-2 HM1055 CMOS Image Sensor

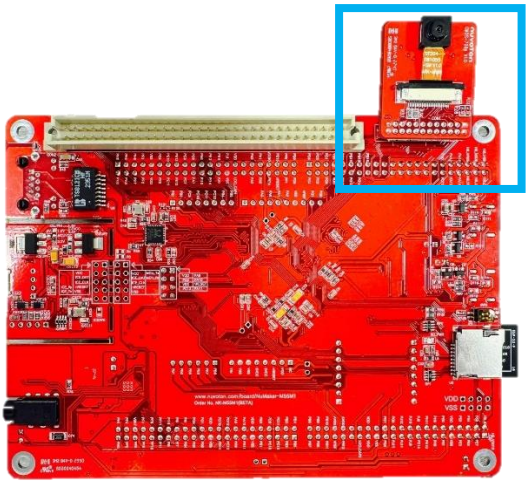

圖 1-3 NuMaker-M55M1 V1.0 連接 HM1055 CMOS Image Sensor

1.3 UVC 及 USB Descriptor 的工作原理解析

1.3.1 背景介紹

影像處理和傳輸系統從 CMOS 感測器開始捕捉影像，經過處理模組處理，並最終通過 USB 傳輸到 PC 顯示。這個過程涉及多個關鍵元件和功能，包括 CMOS 感測器(如 HM1055)、影像處理模組 (CCAP) 以及 USB 傳輸模組。USB 視頻類 (UVC) 是一個標準類規範，標準化了 USB 上的視頻流功能，使得網路攝影機、數位攝像機、視頻轉換器和數位電視調諧器等設備能夠與主機無縫連接。UVC 支持多種視頻格式的串流傳輸，包括 YUV、MJPEG、MPEG-2 TS、H.264、DV 等，並提供了描述視頻設備功能的結構，定義了 USB 根據請求控制設備不同參數和視頻流特徵的方法。UVC 還提供了支持多種視頻分辨率、格式和幀速率的靈活性，這對設備和主機之間的帶寬協商至關重要。大多數操作系統平台都支持 UVC 驅動程式，這大大減少了開發人員創建 USB 視頻設備所需的時間。

1.3.2 UVC 模組示意圖

圖 1-4 UVC 架構圖

影像系統由以下組成：相機感測器模組負責使用光學儀器捕捉現實世界的影像；UVC 裝置處理這些數據並符合 UVC 規範；PC則識別韌體資訊，載入相應驅動程式，與韌體溝通，並實現相機數據的傳輸與顯示。

- 相機的 Sensor 模組，使用光學儀器採集實景。
- UVC 設備：對採集到的資料進行處理,並實現 UVC 相關的規範。
- PC：識別韌體訊息，載入驅動，與韌體通訊，實現攝影機資料的傳輸及顯示。

1.3.3 影像處理和傳輸過程

整個影像處理和傳輸過程從 CMOS 感測器(Camera Sensor Module)的初始化開始。在 Sensor_HM1055.c 檔案中，CMOS 感測器的初始化由 InitSensor() 函式完成。這個函式負責設置感測器的輸出格式，確保感測器能夠正確地捕捉影像數據。隨後，感測器開始捕捉影像數據，並通過 CaptureFrame() 函式將影像數據進行數位化，準備進入下一步的處理階段。

捕捉到的影像數據隨後會進入 M55M1 CCAP 模組進行處理。在這個階段，PacketFormatDownScale() 函式負責將影像數據調整到所需的格式和大小。這裡使用了 CCAP_Start() 函式來啟動影像捕捉介面，並通過 CCAP_SetPacketBuf() 和 CCAP_SetPacketScaling() 函式將數據儲存到記憶體中，為後續的 USB 傳輸做好準備。

一旦影像數據準備好，接下來的步驟是將其打包並透過 USB 傳輸。在 uvc.c 中，這個過程由 UVC_SendImage() 函式負責。首先，該函式重置 DMA 傳輸，然後根據是否為靜態影像或視頻預覽來設置相應的資料表頭。隨後，數據通過 UVC_IsoIn() 函式以等時傳輸的方式發送到主機。這裡使用了 UVC_ActiveDMA() 函式來啟動 USB DMA 傳輸，確保數據可以穩定地傳輸到主機。

當數據到達主機端後，操作系統和驅動程式負責將數據包重組並展示為連續的視頻流。這一過程主要在主機端進行，通過操作系統的驅動和軟體來完成。

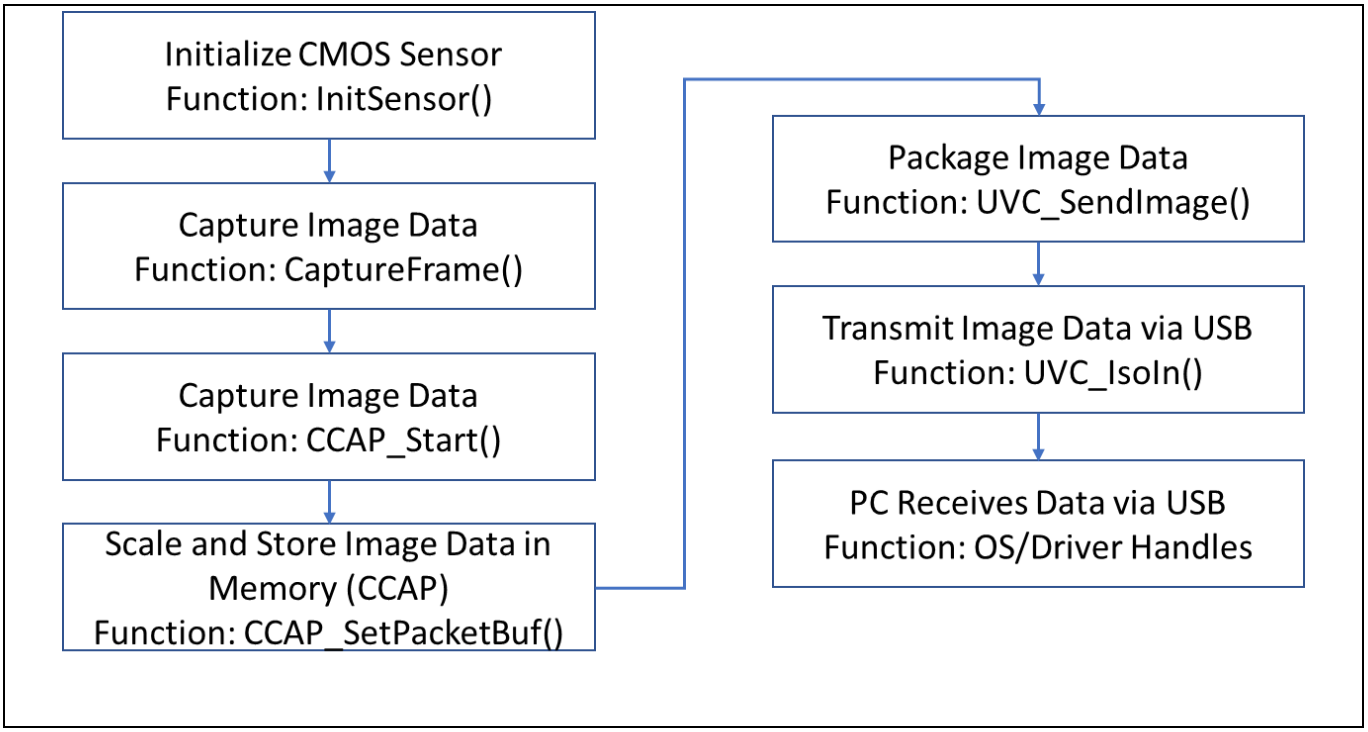

圖 1-5 程式流程圖

1.3.4 UVC (USB Video Class) 的概念

UVC 是一個標準的 USB 裝置類型，用於處理視頻數據流，例如從網路攝影機將視頻傳輸到計算機。UVC 裝置使用標準協議與主機交互，許多操作系統原生支持它，因此 UVC 裝置不需要特定驅動程式。

1.3.5 USB 描述符 (Descriptor) 概述

USB 描述符是一組結構化的數據，用來描述 USB 裝置的屬性、功能和需求。這些描述符幫助主機識別裝置類型、可用功能和傳輸需求。主要的描述符包括：

- 設備描述符(Device Descriptor)：描述 USB 裝置的基本資訊，如裝置類型、製造商和產品 ID。
- 配置描述符(Configuration Descriptor)：描述裝置的配置，如電源需求、介面數量等。
- 介面描述符(Interface Descriptor)：描述裝置的各種介面，如視頻控制介面或視頻流介面。
- 端點描述符(Endpoint Descriptor)：定義數據傳輸的具體端點和方式。

1.3.6 描述符在 UVC 裝置中的作用

在 UVC 裝置中，描述符詳細說明了裝置如何與主機交互並傳輸視頻數據。以下是每個描述符在視頻傳輸中的具體作用，以及程式碼中的實現方式：

1.3.6.1 設備描述符 (Device Descriptor)

設備描述符提供了 USB 裝置的基本資訊，如裝置類型、廠商 ID、產品 ID 和支持的 USB 版本等。這是主機獲取裝置資訊的第一步，通過這個描述符，主機能識別裝置的基本屬性。

```
const uint8_t gu8DeviceDescriptor[] = {
    0x12,                   // bLength: 描述符的長度 ( 18 bytes )
    USB_DT_DEVICE,          // bDescriptorType: 描述符類型 ( Device )
    0x10, 0x01,             // bcdUSB: USB 版本 ( USB 1.1 )
    0x00,                   // bDeviceClass: 裝置類型 ( 由介面定義 )
    0x00,                   // bDeviceSubClass: 子類型
    0x00,                   // bDeviceProtocol: 協議
    EP0_MAX_PKT_SIZE,       // bMaxPacketSize0: 端點 0 的最大封包大小
    LOBYTE(USBD_VID),       // idVendor: 裝置的廠商 ID
    HIBYTE(USBD_VID),       // idProduct: 裝置的產品 ID
    0x00, 0x01,             // bcdDevice: 裝置版本號
    0x01,                   // iManufacturer: 廠商名稱字符串索引
    0x02,                   // iProduct: 產品名稱字符串索引
    0x03,                   // iSerialNumber: 序列號字符串索引
    0x01                    // bNumConfigurations: 支持的配置數量
```

```
};
```

1.3.6.2 配置描述符 (Configuration Descriptor)

配置描述符描述了裝置的配置選項，包括使用的電源、支持的介面數量等。主機通過這個描述符來配置裝置，使其能正常工作。

```
const uint8_t gu8ConfigDescriptor[] = {
    0x09,                   // bLength: 描述符的長度 ( 9 bytes )
    USB_DT_CONFIG,          // bDescriptorType: 描述符類型 ( Configuration )
    USB_CONFIG_DESC_SIZE,   // wTotalLength: 包含所有子描述符的總長度
    0x02,                   // bNumInterfaces: 支持的介面數量
    0x01,                   // bConfigurationValue: 配置值 ( 用於選擇此配置 )
    0x00,                   // iConfiguration: 配置描述字符串索引
    0x80,                   // bmAttributes: 配置特性 ( 如是否總線供電 )
    0x32                    // MaxPower: 最大電流 ( 100mA )
};
```

bNumInterfaces 表示此配置支持兩個介面，一個是視頻控制介面(Video Control Interface)，另一個是視頻流介面(Video Streaming Interface)。

1.3.6.3 介面描述符 (Interface Descriptor)

介面描述符詳細描述了配置中的每個介面，包括它們所提供的功能(如視頻控制或視頻流)、使用的端點數量等。

```
const uint8_t gu8InterfaceDescriptor[] = {
    0x09,                   // bLength: 描述符的長度 ( 9 bytes )
    USB_DT_INTERFACE,       // bDescriptorType: 描述符類型 ( Interface )
    0x00,                   // bInterfaceNumber: 介面編號
    0x00,                   // bAlternateSetting: 替代設置
    0x01,                   // bNumEndpoints: 此介面使用的端點數量 ( 1個端點 )
    USB_CLASS_VIDEO,        // bInterfaceClass: 介面類型 ( 視頻類 )
    USB_SUBCLASS_VIDEOCONTROL, // bInterfaceSubClass: 子類型 ( 視頻控制 )
    USB_PROTOCOL_UNDEFINED, // bInterfaceProtocol: 協議
    0x00                    // iInterface: 介面描述字符串索引
};
```

這裡的 bInterfaceClass 設置為 USB_CLASS_VIDEO，表示這是一個視頻類介面。

bNumEndpoints 設置為 1，表明此介面僅使用一個端點來傳輸數據。

1.3.6.4 端點描述符 (Endpoint Descriptor)

端點描述符定義了數據在主機和裝置之間傳輸的具體方式。對於視頻流，端點通常配置為等時傳輸 (Isochronous Transfer)以確保數據的穩定性和實時性。

```
const uint8_t gu8EndpointDescriptor[] = {
    0x07,                   // bLength: 描述符的長度 ( 7 bytes )
```

```
    USB_DT_ENDPOINT,         // bDescriptorType: 描述符類型（Endpoint）
    ISO_IN_EP_NUM,           // bEndpointAddress: 端點地址（IN端點）
    USB_ENDPOINT_TYPE_ISOCHRONOUS, // bmAttributes: 端點屬性（等時傳輸）
    LOBYTE(MAX_ISO_PACKET_SIZE),  // wMaxPacketSize: 最大封包大小
    HIBYTE(MAX_ISO_PACKET_SIZE),
    0x01                     // bInterval: 輪詢間隔（1ms）
};
```

bEndpointAddress 表示這是一個 IN 端點，用於將數據從裝置傳輸到主機。

bmAttributes 設置為 USB_ENDPOINT_TYPE_ISOCHRONOUS，表示這是等時端點，適合傳輸視頻流。

1.4 CMOS 影像數據傳輸的完整過程

1.4.1 CMOS 影像捕捉

CMOS 感測器負責捕捉影像並將其數位化，這是整個視頻傳輸的第一步。

在 Sensor_HM1055.c 中，初始化並啟動影像捕捉的部分：

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

1.4.2 影像處理和數據打包

捕捉到的影像數據需要進行處理並打包成 USB 能夠傳輸的數據包。這包括數據縮放和格式轉換，並最終將數據儲存在準備傳輸的記憶體/緩衝區中。

在 PacketFormatDownScale() 函式中：

```
/* 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();
```

這段程式碼將捕捉的影像數據進行縮放並準備好記憶體/緩衝區地址，供後續的 USB 傳輸使用。

1.4.3 數據傳輸到主機

影像數據被打包後，通過 USB 傳輸到主機端。這裡使用了等時傳輸模式，確保視頻流的實時性。

在 UVC_SendImage() 函式中：

```
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() 函式負責將數據傳輸到主機，其中 UVC_IsoIn() 函式通過 USB 等時傳輸將數據發送出去。輸入的 u32Addr 是影像數據的地址，而 u32transferSize 則是傳輸數據的大小。

1.4.4 主機重組數據和顯示視頻

主機接收到這些數據包後，會根據 UVC 協議重組數據並顯示視頻。在程式碼層面上，這個過程主要由主機端的操作系統和驅動程式完成，具體的程式碼在裝置端是不可見的。

在 UVC_IsoIn() 函式中：

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

UVC_ActiveDMA() 函式啟動 USB DMA 傳輸，這是數據傳輸到主機的重要一步。

1.5 影像數據打包過程的詳細說明

根據 USB UVC 的封包內容，分為許多的欄位，需要填入對應的數據，然後打包發送給 PC，封包各欄位說明如下：

- UVC Payload Header：這是封包的表頭，包含了幀的控制資訊。
- bHeaderLength：描述了 UVC Payload Header 的長度。
- bmHeaderInfo：包含幀標誌位，表示這是幀的開始或結束，也可以標記幀的類型(例如靜態圖片)。

- dwPresentationTime：一個 4 位元組的時間戳，用來同步影像的播放時間。
- dwSourceClock：4 位元組的時鐘時間戳，用來同步影像的捕捉時間。
- Image Data Payload：這是封包的主要內容，包含實際的影像數據。
- Video Data：這一部分儲存了捕捉到的影像數據，其大小取決於影像的分辨率和幀格式。
- USB Packet Footer：這部分是可選的，用來存放一些附加資訊，例如錯誤檢查碼(如 CRC)。
- Error Check (CRC)：如果系統支持，可以在封包結尾加入錯誤檢查碼，以確保數據的完整性。

1.6 封包結構在程式碼中的對應

在程式碼中，這些部分由 UVC_SendImage() 函式負責打包和傳輸。例如，UVC_PH_Preview 和 UVC_PH_Snapshot 對應了 bmHeaderInfo，而影像數據則是從影像緩衝區中讀取並存放在封包中。

```
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 */
}
```

這段程式碼設定了 UVC Payload Header 中的 bmHeaderInfo，並根據是否是靜態影像或視頻來決定封包的類型。

1.7 UVC 的 Frame Rate

這段 UVC 的程式，傳送 1 張 image 到 PC 的時間是 75ms，FPS 大約是 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. 程式介紹

主程式如下，初始化系統後設置 HM1055 並啟動影像抓取，而後設置 USB Device 設備：

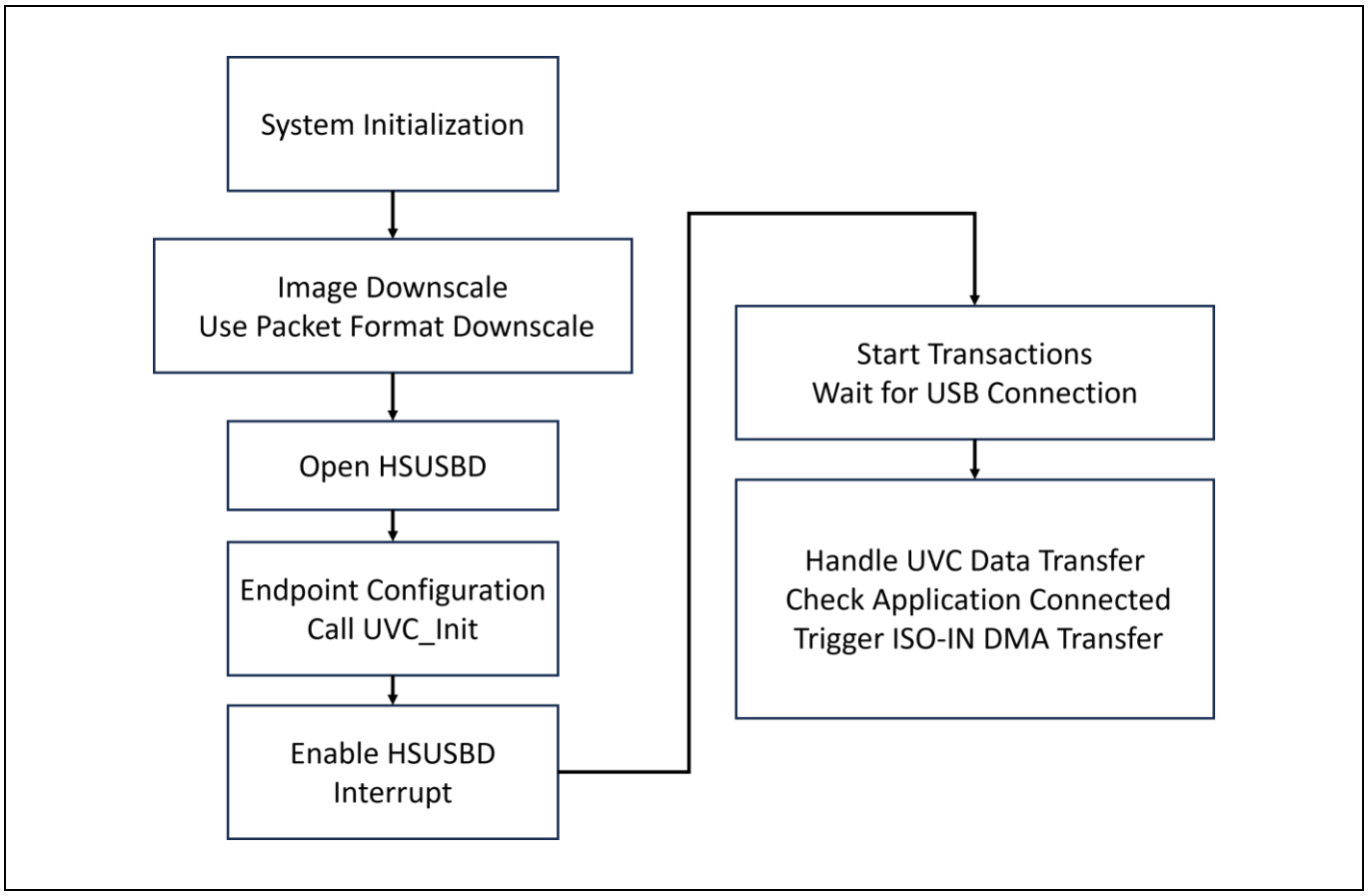

圖 2-1 程式流程圖

```
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 的設置並初始化，配置影像捕捉 (CCAP) 與輸出格式為 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()用於改變視訊輸入設置的函式，調整框架大小：

```
/*---------------------------------------------------------------------------------------------*/
/* 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 的初始化設置:

```
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;
}
```

影像捕捉後，透過 USB 傳輸至 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 是一個中斷服務程序，用於處理影像捕獲模組 (CCAP) 的中斷:

在下列程式碼中，我們可以看到使用 2 塊 frame buffer 在此切換，使用了兩個 Frame Buffer (即 u8FrameBuffer0 和 u8FrameBuffer1) 來實現雙緩衝機制，主要目的是為了避免影像撕裂 (tearing) 問題並提升影像處理的穩定性。當一塊緩衝區正在用來顯示或傳輸影像時，另一塊緩衝區則負責接收和寫入新的一幀資料，這樣可以確保不會同時讀寫同一塊緩衝區，防止畫面在讀取過程中被修改導致撕裂現象。透過交替使用兩個緩衝區，系統能夠實現平滑的影像幀切換，避免出現部分顯示新資料、部分顯示舊資料的問題，同時也提升了資料傳輸的效率，確保影像處理的連續性與流暢度。

當 u32UVCXferIdx 為奇數時，將資料寫入 u8FrameBuffer1，並預設下一幀將使用 u8FrameBuffer0。

當 u32UVCXferIdx 為偶數時，將資料寫入 u8FrameBuffer0，並預設下一幀將使用 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. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.00.001

3.2 硬體需求

- 電路元件
  - NuMaker-M55M1 V1.0
  - HM1055 CMOS Sensor

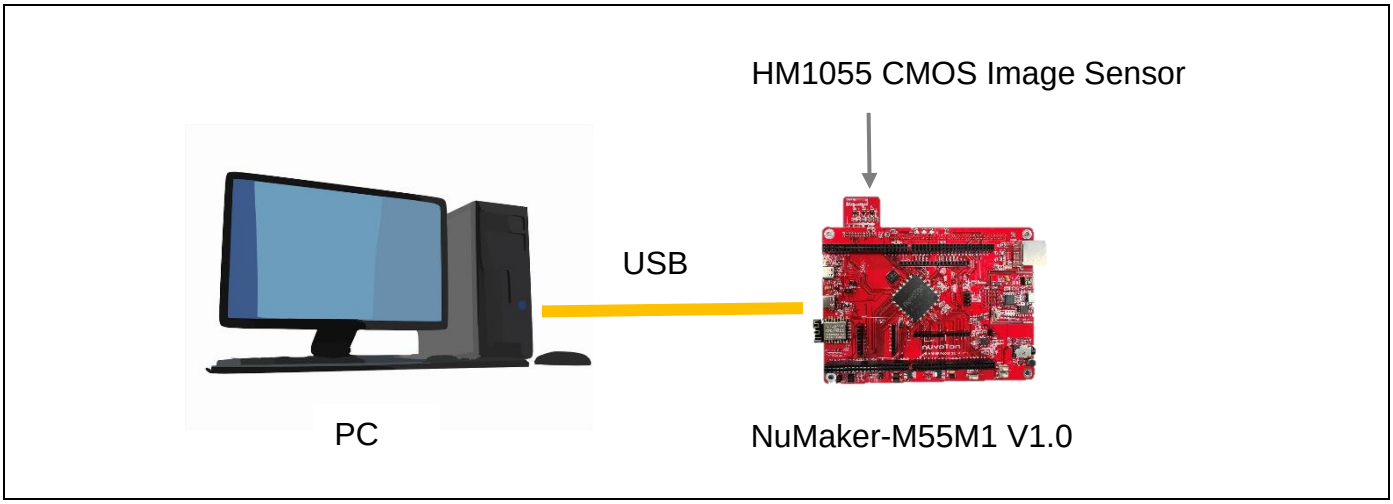

圖 3-1 USB UVC 連接執行示意圖

- CMOS 電路接線示意圖
  - M55M1 使用 I^2C 來傳輸控制命令至 CMOS 感測器，使用 CCAP_SCLK 作為 SCLK 供應頻率給 CMOS 感測器，CMOS 感測器會根據設定回傳 Pixel Clock (PIXCLK)、HSYNC、VSYNC 和 D0~D7 資料數據。

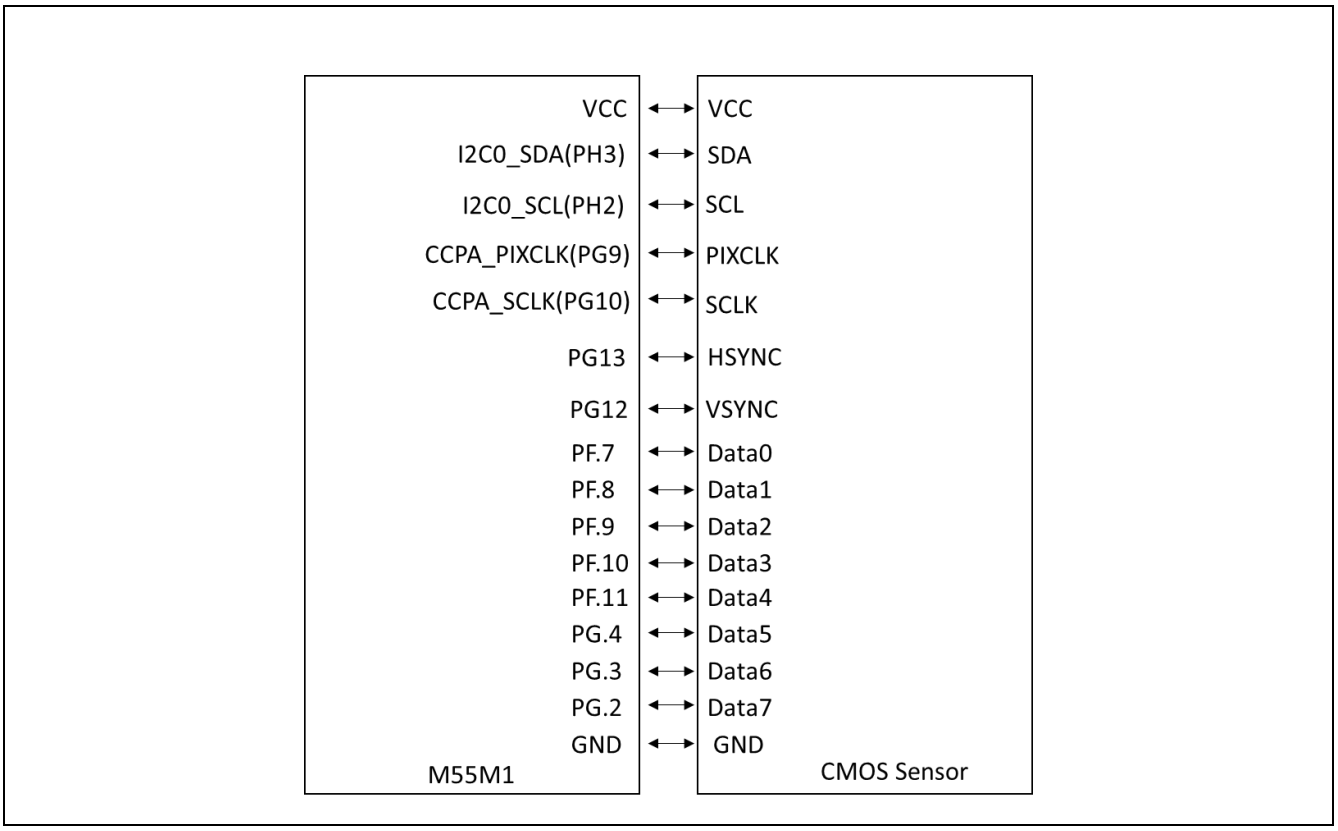

圖 3-2 CMOS 線路示意圖

4. 目錄資訊

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

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *HSUSBD_UVC_VideoSensor.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行代碼
   - 執行結果

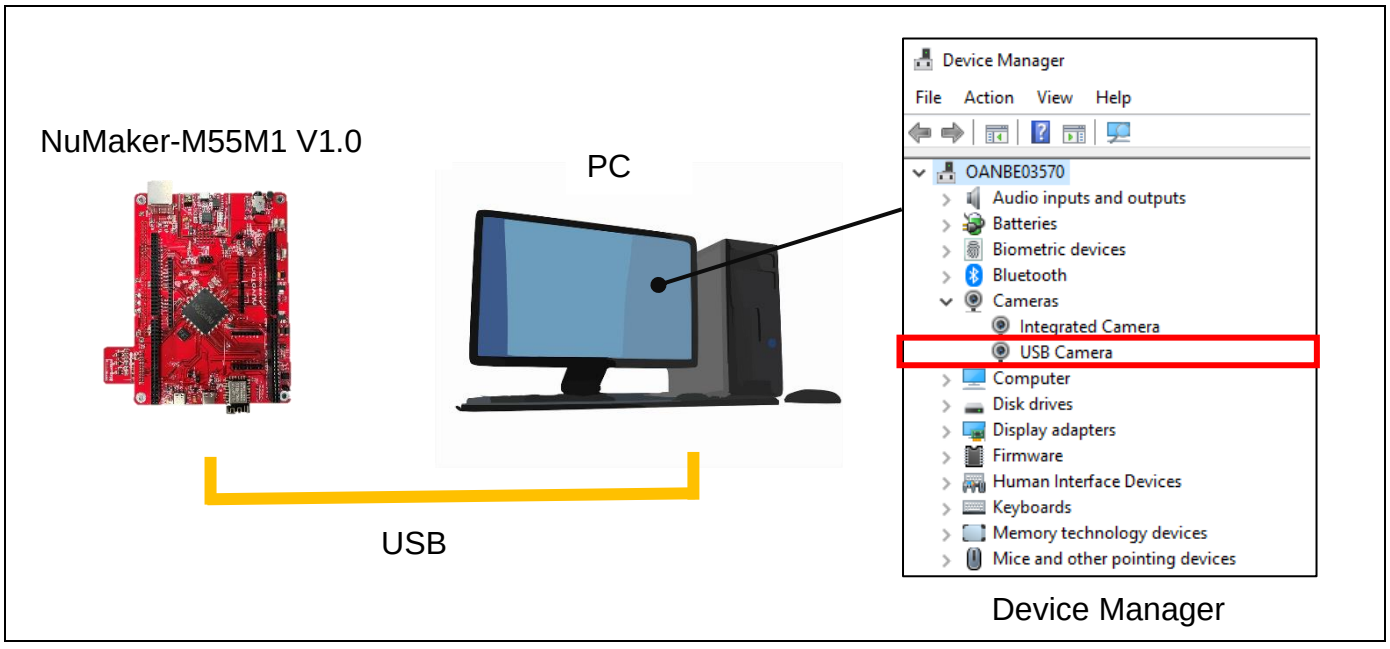

圖 5-1 USB UVC 執行結果

6. 修訂紀錄

Date	Revision	Description
2024.06.14	1.00	初始發布。

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.