Arm Helium 色彩格式轉換

NuMicro® 32 位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例程式使用 Arm Helium 來實做色彩格式轉換同時進行自我功能驗證
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
開發平台	NuMaker-M55M1(BETA) V1.1

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

本範例程式使用 Arm Helium 來實做色彩格式轉換以及功能性的自我驗證。透過 Arm Helium 提供的 M-Profile 向量擴展(M-Profile Vector Extension, MVE)指令實作 RGB565 轉 RGB888 的色彩格式轉換。範例中同時包含非向量化的轉換程式碼，可做為功能自我驗證對照使用，也可以用於向量化運算效率提升的參考值。

1.1 原理

Helium 向量化技術可以讓 CPU 同時進行 128-bit 的資料操作，包括 Load，Store，Logical Operation 以及 Vector Arithmetic。針對不同應用的情境，128-bit 的資料可能有不同的組成方式而產生若干的 lane，如圖 1-1 所示，可以由 16 個 8-bit lane，8 個 16-bit lane 或 4 個 32-bit lane 所組成。

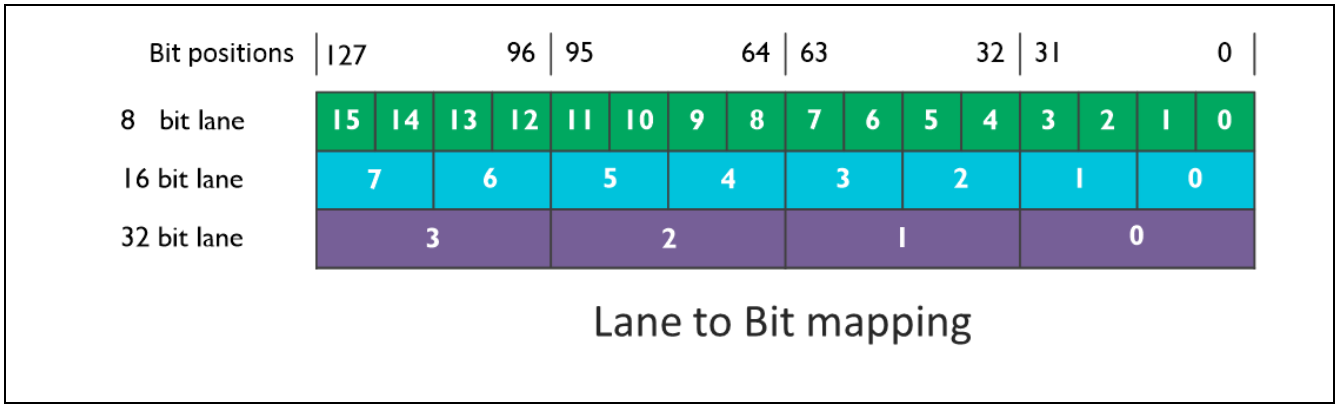

圖 1-1Helium lane 與資料寬度定義

未向量化處理的 RGB565 轉 RGB888 的流程可以參考圖 1-2。每次處理一個 RGB565 pixel 的資料，輸入資料寬度為 16-bit，轉換中所使用的資料操作皆為 Byte-based，轉換後的 RGB888 pixel 資料有三個 byte，輸出資料寬度為 24-bit。

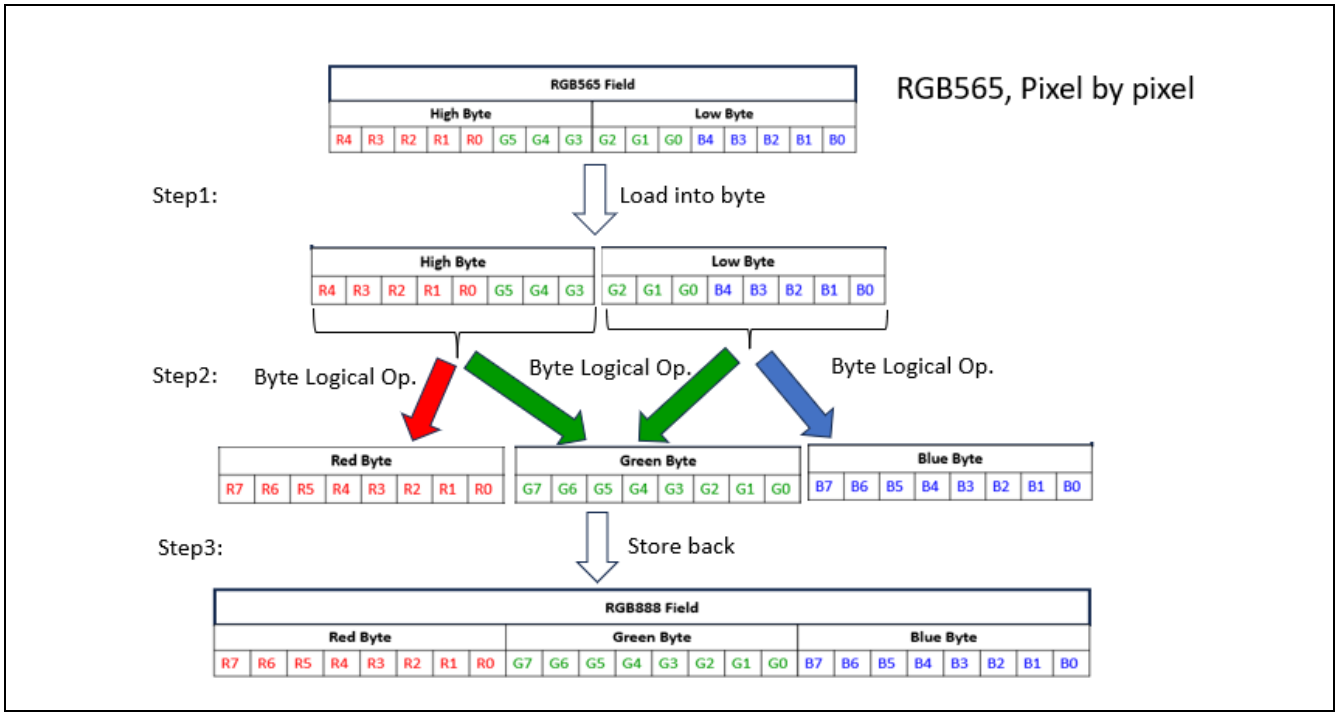

圖 1-2 RGB565 轉 RGB888 流程

如果同時載入 16 個 pixel 的資料給 Helium 的 16 個 lane，在 Step2 就可以同時處理 16 個 Byte Logical Op.，達到平行化的處理方式，流程可參照圖 1-3。

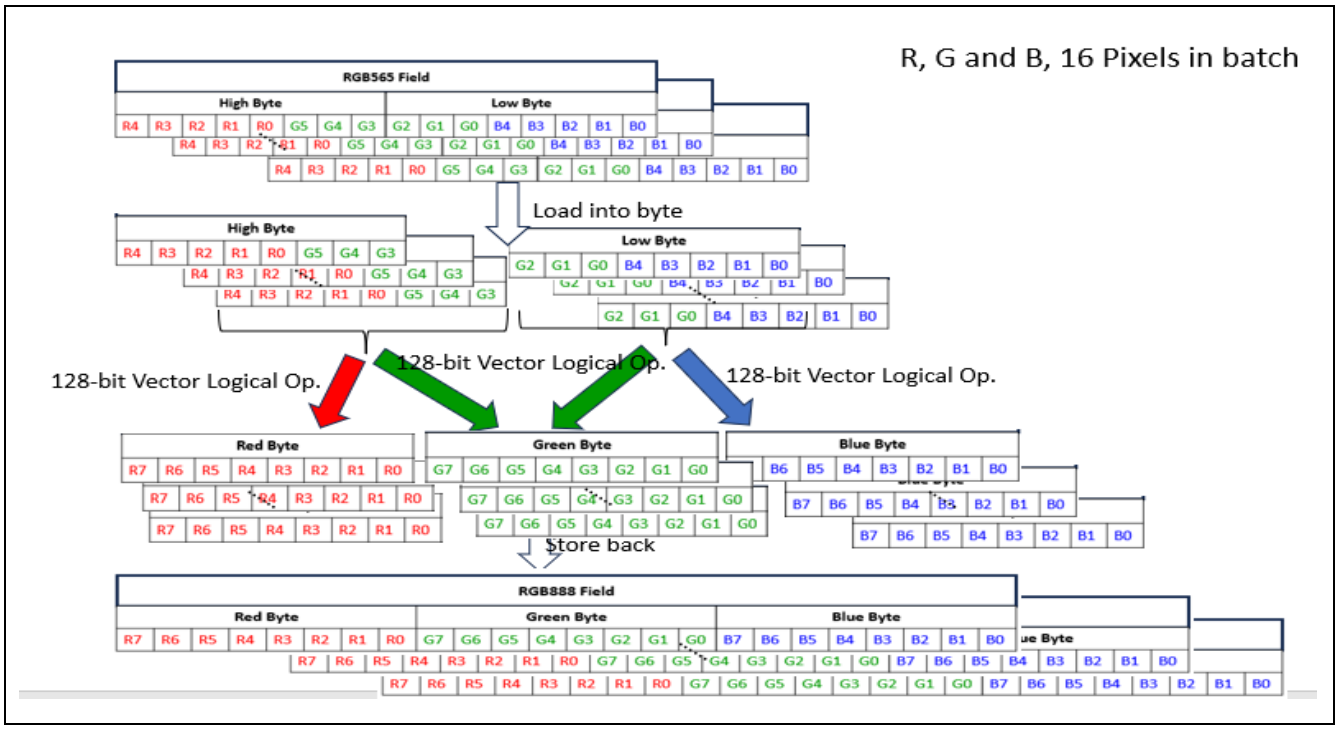

圖 1-3 RGB565 轉 RGB888 向量化處理示意圖

像素資料在記憶體上是以連續的位址存放，向量暫存器進行 RGB566 高低 Byte 的讀取，可以透過 Helium Scatter Load 方式來進行，讀取到兩個向量暫存器中。Scatter Load 須給定位址偏移量，以下圖 1-4 為例，要讀取 RGB565 的高 Byte 須指定 offset 為 *0x0e0c0a0806040200*，要讀取 RGB565 的低 Byte 須指定 offset 為 *0x0f0d0b0907050301*，每一 offset 用 1 個 byte 來表示。

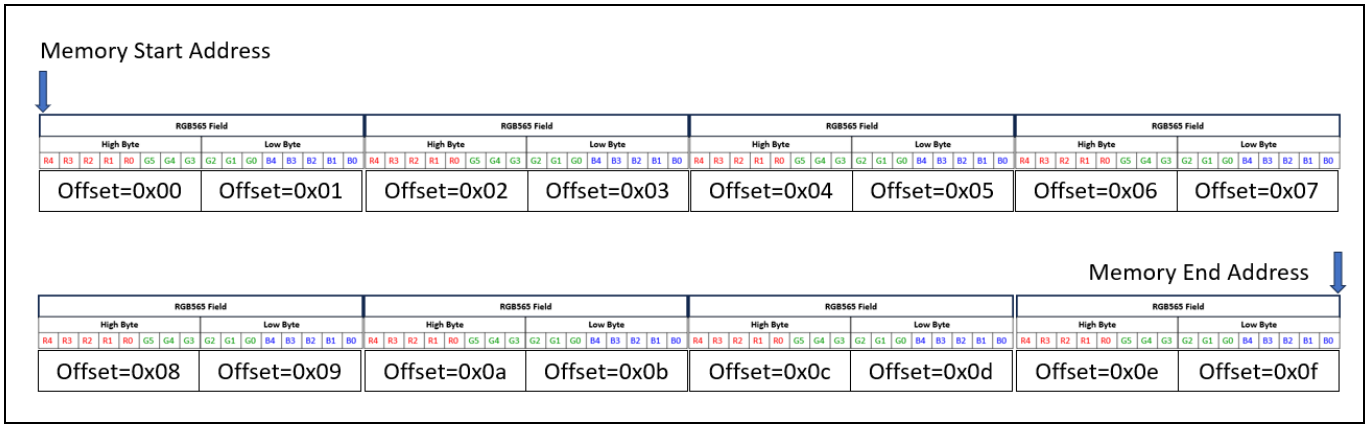

圖 1-4 RGB565 Scatter Load Offset 示意圖

完成格式轉換後，因為轉換後的 R,G 和 B 的資料分別存放在三個向量暫存器，可透過 Helium Scatter Store 方式將資料擺放至連續的記憶體位址上，以圖 1-5 為例，要將 RGB888 放回連續的記憶體位址上，需要指定三組 offset，R 須指定 offset 為 0x0300，G 須指定 offset 為 0x0401，B 須指定 offset 為 0x0502。

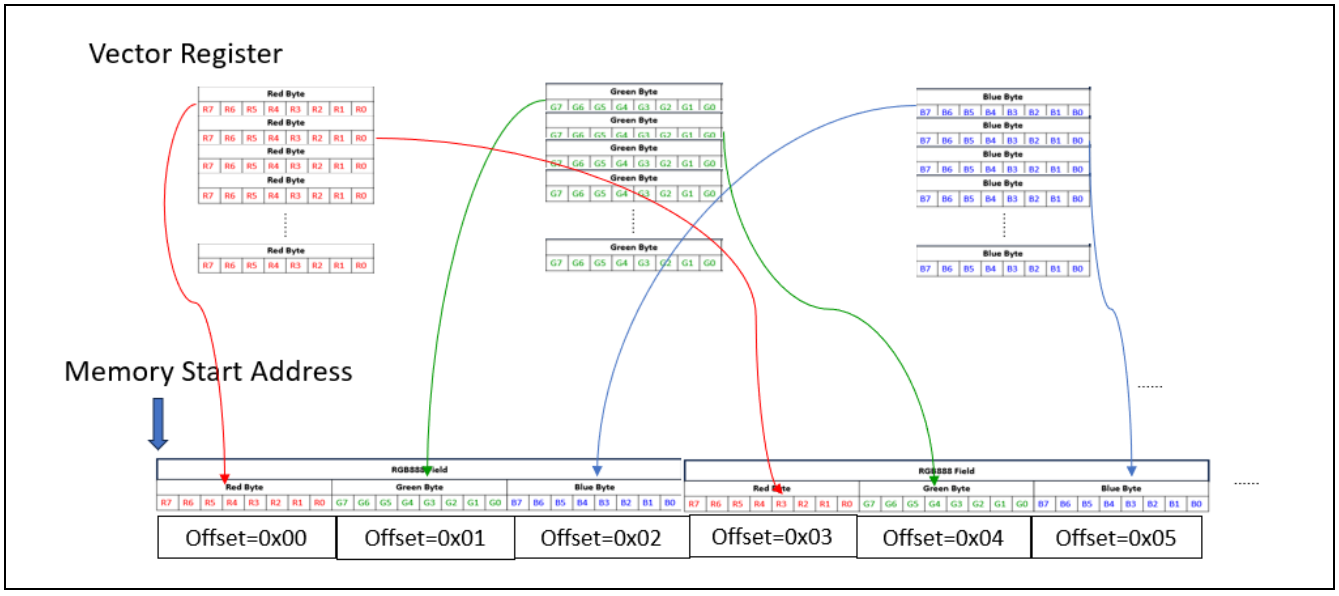

圖 1-5 RGB565 Scatter Store Offset示意圖

1.2 執行結果

本範例程式執行後會印出以下資訊，包含向量化與非向量化色彩格式轉換的執行 cycle 數目，同時也會逐筆比對色彩轉換後的結果是否正確。

```
+----------------------------------------+
|   Helium RGB565 to 888 Sample Code KEIL |
+----------------------------------------+

 Execute RGB565to888

 Tick Counters are:558!

 Execute VRGB565to888_TP

 Tick Counters are:382!

 Compare RGB565 Results
 Match: image_dst[0] = 0xf0
 Match: image_dst[1] = 0x58
 Match: image_dst[2] = 0xf8
 Match: image_dst[3] = 0x30
 Match: image_dst[4] = 0x94
 Match: image_dst[5] = 0x90
 Match: image_dst[6] = 0xa0
 Match: image_dst[7] = 0x94
 Match: image_dst[8] = 0x10
 Match: image_dst[9] = 0x38
 Match: image_dst[10] = 0x50
 Match: image_dst[11] = 0x90
 Match: image_dst[12] = 0xc0
 Match: image_dst[13] = 0x98
 Match: image_dst[14] = 0x10
 Match: image_dst[15] = 0xa0
 Match: image_dst[16] = 0x94
 Match: image_dst[17] = 0x10
 Match: image_dst[18] = 0xa0
 Match: image_dst[19] = 0x8c
 Match: image_dst[20] = 0x10
 Match: image_dst[21] = 0x30
 Match: image_dst[22] = 0x94
 Match: image_dst[23] = 0x10
 Match: image_dst[24] = 0x30
 Match: image_dst[25] = 0xb4
 Match: image_dst[26] = 0x10
 Match: image_dst[27] = 0x70
 Match: image_dst[28] = 0x80
 Match: image_dst[29] = 0xe0
 Match: image_dst[30] = 0x30
```

```
Match: image_dst[31] = 0x9c
Match: image_dst[32] = 0x90
Match: image_dst[33] = 0x30
Match: image_dst[34] = 0x84
Match: image_dst[35] = 0x90
Match: image_dst[36] = 0x30
Match: image_dst[37] = 0x88
Match: image_dst[38] = 0x10
Match: image_dst[39] = 0x30
Match: image_dst[40] = 0x88
Match: image_dst[41] = 0x90
Match: image_dst[42] = 0x30
Match: image_dst[43] = 0x8c
Match: image_dst[44] = 0x10
Match: image_dst[45] = 0x70
Match: image_dst[46] = 0xec
Match: image_dst[47] = 0x90

RGB565 Speedup 1.46 X

Test Done
```

2. 代碼介紹

計算 RGB565 轉 RGB888 的程式，代碼位於 *main.c*。

```
#define BYTENUMBER_RGB565  2
……
/**
  * @brief      Convert RGB565 to RGB888 Color Format
  * @param[in]  src is the source data(RGB565) address
  * @param[in]  dst is the destination data(RGB88) address
  * @param[in]  len is the source data length in packets
  * @return     void
  * @details    This function converts RGB565 to RGB888 color format in non vector manner.
  */

void RGB565to888(uint8_t *src, uint8_t *dst, uint8_t len)
{
    uint8_t buf888_high, buf888_low;
    uint8_t dst888_r, dst888_g, dst888_b;
    int count = len;

    while (count >= 0)
    {
        //RGB565_IN_UINT8

        //Sequential Load
        buf888_high = *src;
        buf888_low = *(src + 1);

        //Separate RGB
        dst888_r  = (buf888_high & 0xF8) ;
        dst888_g  = ((buf888_high & 0x07) << 5) | ((buf888_low & 0xE0) >> 3) ;
        dst888_b  = (buf888_low << 3) ;

        //Sequential Store
        *(dst++) = dst888_r;
        *(dst++) = dst888_g;
        *(dst++) = dst888_b;

        src += BYTENUMBER_RGB565;
        count --;
    }
}
```

上述函式並未使用向量化處理，其處理方式是將資料逐筆進行色彩格式轉換。

使用向量化進行色彩轉換的函示如下:

```
….
#define OFFSET_R_LOW     0x15120F0C09060300
#define OFFSET_R_HIGH    0x2D2A2724211E1B18
#define OFFSET_G_LOW     0x1613100D0A070401
#define OFFSET_G_HIGH    0x2E2B2825221F1C19
#define OFFSET_B_LOW     0x1714110E0B080502
#define OFFSET_B_HIGH    0x2F2C292623201D1A
……
```

```
#define OFFSET_565HIGHBYTE_LOW     0x0E0C0A0806040200
#define OFFSET_565HIGHBYTE_HIGH    0x1E1C1A1816141210
#define OFFSET_565LOWBYTE_LOW    0x0F0D0B0907050301
#define OFFSET_565LOWBYTE_HIGH   0x1F1D1B1917151311

#define BYTENUMBER_RGB888  3
#define PIXELS_LOOP       16
#define PIXELS_CONVERT_EACHLOOP       16
…
/**
  * @brief      Convert RGB565 to RGB888 Color Format
  * @param[in]  src is the source data(RGB565) address
  * @param[in]  dst is the destination dat(RGB88) address
  * @param[in]  size_h is the height of source image data
  * @param[in]  size_w is the width of source image data
  * @return     void
  * @details    This function converts RGB565 to RGB888 color format in vector manner.
  */

void VRGB565to888_TP(uint8_t *src, uint8_t *dst, uint16_t size_h, uint16_t size_w)
{
    int8_t count;
    uint8x16_t vsrc_8_16_lo;
    uint8x16_t vsrc_8_16_hi;
    uint8x16_t vdst_8_16_r;
    uint8x16_t vdst_8_16_g;
    uint8x16_t vdst_8_16_b;
    uint8x16_t offset_8_16;

    count = size_h * size_w;

    while (count > 0)
    {
        //RGB565_IN_UINT8

        //Include tail predication to handle improper tailing.
        mve_pred16_t p = vctp8q(count);

        //Scatter Load//
        offset_8_16 = vcreateq_u8(OFFSET_565LOWBYTE_LOW, OFFSET_565LOWBYTE_HIGH);

        vsrc_8_16_lo = vldrbq_gather_offset_z_u8(src, offset_8_16, p);

        offset_8_16 = vcreateq_u8(OFFSET_565HIGHBYTE_LOW, OFFSET_565HIGHBYTE_HIGH);

        vsrc_8_16_hi = vldrbq_gather_offset_z_u8(src, offset_8_16, p);

        //Separate RGB//
        vdst_8_16_r  =  vrshlq(vshrq(vsrc_8_16_hi, 3), 3);//vrshrq results in rounded.
        vdst_8_16_g =  vorrq_u8(vrshlq(vsrc_8_16_hi, 5),
                                vrshlq(vshrq(vsrc_8_16_lo, 5), 2));

        vdst_8_16_b =  vrshlq(vsrc_8_16_lo, 3);
        //Scatter Store//
        offset_8_16 = vcreateq_u8(OFFSET_R_LOW, OFFSET_R_HIGH);

        vstrbq_scatter_offset_p(dst, offset_8_16, vdst_8_16_r, p);

        offset_8_16 = vcreateq_u8(OFFSET_G_LOW, OFFSET_G_HIGH);
```

```
        vstrbq_scatter_offset_p(dst, offset_8_16, vdst_8_16_g, p);

        offset_8_16 = vcreateq_u8(OFFSET_B_LOW, OFFSET_B_HIGH);

        vstrbq_scatter_offset_p(dst, offset_8_16, vdst_8_16_b, p);

        //Update data source and destination pointer

        //Source: shift one RGB566 pixel size(2 byte)
        src += BYTENUMBER_RGB565 * PIXELS_CONVERT_EACHLOOP;

        //Destination: shift one RGB888 pixel size(3 byte)
        dst += BYTENUMBER_RGB888 * PIXELS_CONVERT_EACHLOOP;

        count -= PIXELS_CONVERT_EACHLOOP;

    }
}
```

將上述範例中所使用到的 Helium 相關函式整理摘要依邏輯運算和資料載入整理如下:

Intrinsic 函式名稱	參數	返回值	函式說明
vshrq	(uint8x16_t a, const int imm)	uint8x16_t	將參數中第一向量暫存器 a 的每個元素右移一常數值 imm，結果寫入返回值向量暫存器。
vrshlq	(uint8x16_t a, int32_t b)	uint8x16_t	將參數中第一個向量暫存器 a 的每個元素左移位第二個變數 b 的最低位元組的值，結果寫入返回值向量暫存器。
vorrq_u8	(uint8x16_t a, uint8x16_t b)	uint8x16_t	計算參數中兩個向量暫存器 a 和 b 的 bitwise logic OR 結果，結果寫入返回值向量暫存器。

表 2-1Helium 邏輯運算部分相關函式表

此部分和非向量的差別就在於同時進行了 16 個 Lane 的邏輯運算，以藍色為例，非向量運算:

```
dst888_b = (buf888_low << 3) ;
```

向量運算:

```
vdst_8_16_b = vrshlq(vsrc_8_16_lo, 3);
```

Intrinsic 函式名稱	參數	返回值	函式說明
vcreateq_u8	(uint64_t a, uint64_t b)	uint8x16_t	傳回一個返回值向量，其暫存器表示形式與參數指定的兩個 64 位元數值 a 和 b 的串聯相同。
vldrbq_gather_offset_z_u8	(uint8_t const * base, uint8x16_t offset, mve _pred16_t p)	uint8x16_t	將連續元素從記憶體載入到目標向量暫存器。在此模式下，目標位址是根據基址暫存器 base 和偏移量 offset 計算得出的。資料長度不足的部分會參考 p 由 Helium 填零補足。
vstrbq_scatter_offset_p	(uint8_t * base, uint8x 16_t offset, uint8x16_t value, mve_pred16_t p)	void	將連續元素從向量暫存器 value 儲存到記憶體基址位置。在此模式下，目標位址是根據基址暫存器 base 和偏移量 offset 計算得出資料長度不足的部分會參考 p 由 Helium 填零補足。

表 2-2Helium 資料載入部分相關函式表

此部分可對照圖 1-4 圖 1-5 關於 Scatter Load/Store 的說明，程式中非向量一次載入 1 個 pixel:

```
//Sequential Load
buf888_high = *src;
buf888_low = *(src + 1);
```

向量一次載入 16 個 pixel，並將 high/low byte 放在兩個不同的向量暫存器，因為相鄰 pixel 的 low byte 地址非連續，因此需提供 pixel 的基底地址與偏移量:

```
……
#define OFFSET_565LOWBYTE_LOW    0x0F0D0B0907050301
#define OFFSET_565LOWBYTE_HIGH   0x1F1D1B1917151311

//Scatter Load//
offset_8_16 = vcreateq_u8(OFFSET_565LOWBYTE_LOW, OFFSET_565LOWBYTE_HIGH);

vsrc_8_16_lo = vldrbq_gather_offset_z_u8(src, offset_8_16, p);
```

OFFSET_565LOWBYTE_LOW 和 OFFSET_565LOWBYTE_HIGH 各為 64-bit 寬，依表 2-2Helium 資料載入部分相關函式表表 2-2 說明，執行上述兩行程式碼後，向量暫存器 offset_8_16 的內容值為:

Offset_8_16(uint8x16t)

0x1F	0x1D	0x1B	0x19	0x17	0x15	0x13	0x11	0xF	0xD	0xB	0x9	0x7	0x5	0x3	0x1

vsrc_8_16_lo(uint8x16t)

src [0x1F]	src [0x1D]	src [0x1B]	src [0x19]	src [0x17]	src [0x15]	src [0x13]	src [0x11]	src [0xF]	src [0xD]	src [0xB]	src [0x9]	src [0x7]	src [0x5]	src [0x3]	src [0x1]

資料儲存部分:

非向量儲存時一次存放 1 個 RGB888 pixel(3 bytes):

```
//Sequential Store
*(dst++) = dst888_r;
*(dst++) = dst888_g;
*(dst++) = dst888_b;
```

向量儲存時，向量暫存器的內容(16 lanes)會被同時儲存至記憶體，需提供儲存位址的基底位址和偏移量，讓 RGB888 在記憶體採連續方式排列。

以 R 為例:

```
….
#define OFFSET_R_LOW     0x15120F0C09060300
#define OFFSET_R_HIGH    0x2D2A2724211E1B18
……
//Scatter Store//
offset_8_16 = vcreateq_u8(OFFSET_R_LOW, OFFSET_R_HIGH);
```

Offset_8_16(uint8x16t)

0x2D	0x2A	0x27	0x24	0x21	0x1E	0x1B	0x18	0x15	0x12	0xF	0xC	0x9	0x6	0x3	0x0

```
vstrbq_scatter_offset_p(dst, offset_8_16, vdst_8_16_r, p);
```

上述程式碼執行後，記憶體 dst 的資料如下:

位址	0x2F	0x2E	0x2D	……	……	……	……	……	……	……
內容			*vdst_8_16_r[15]*	……	……	……	……	……	……	……
位址	……	……	0x9	……	0x5	0x4	0x3	0x2	0x1	0x0
內容	……	……	*vdst_8_16_r[3]*	……			*vdst_8_16_r[1]*			*vdst_8_16_r[0]*

G 和 B 的資料儲存方式和 R 相似，可自行參考 R 部分。

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.00.001
- IDE 版本
  - Keil uVersion 5.28

3.2 硬體需求

- 電路元件
  - NuMaker-M55M1(Beta V1.1)
- 線路示意圖
  - 如下圖 3-1 將 UART0 TX (PB.13) pin 腳連接到 Nu_Link Me，再透過 VCOM 連接 PC，BaudRate 設定為 115200 bps，以顯示範例代碼的執行結果。

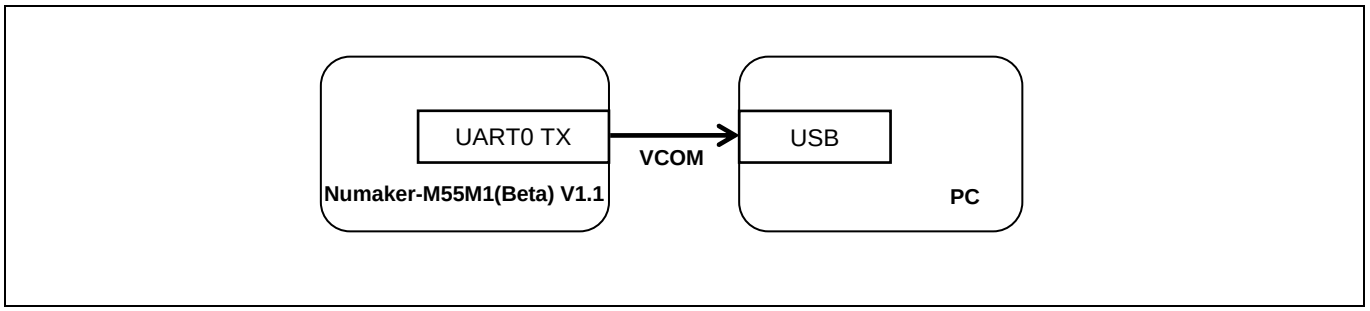

圖 3-1 線路示意圖

4. 目錄資訊

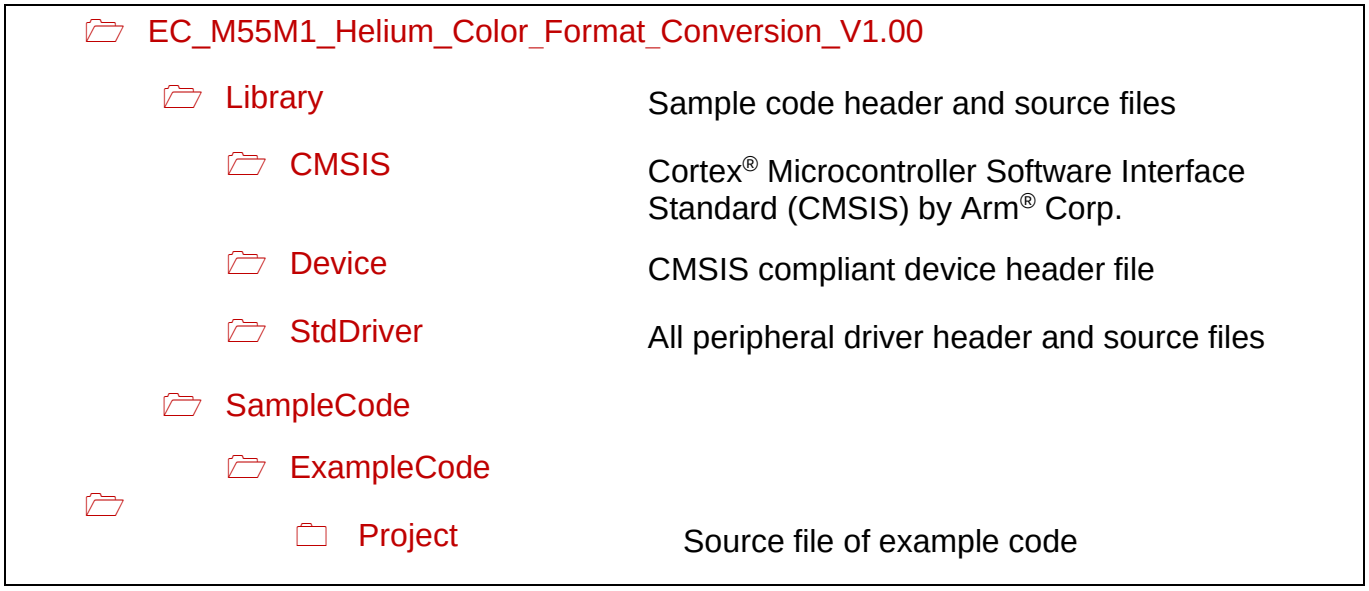

圖 4-1 目錄資訊

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 M55M1_Helium_Color_Format_Conversion.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

6. 修訂紀錄

Date	Revision	Description
2024.06.21	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.