Arm Helium 色彩格式转换

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例程序使用 Arm Helium 来实做色彩格式转换同时进行自我功能验证
BSP 版本	M55M1_Series_BSP_CMSIS_V3.00.001
开发平台	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

本范例程序使用 Arm Helium 来实做色彩格式转换以及功能性的自我验证。透过 Arm Helium 提供的 M-Profile 向量扩展(M-Profile Vector Extension, MVE)指令实作 RGB565 转 RGB888 的色彩格式转换。范例中同时包含非向量化的转换程序代码，可做为功能自我验证对照使用，也可以用于向量化运算效率提升的参考值。

1.1 原理

Helium 向量化技术可以让 CPU 同时进行 128-bit 的资料操作，包括 Load，Store，Logical Operation 以及 Vector Arithmetic。针对不同应用的情境，128-bit 的数据可能有不同的组成方式而产生若干的 lane，如图 1-1 所示，可以由 16 个 8-bit lane，8 个 16-bit lane 或 4 个 32-bit lane 所组成。

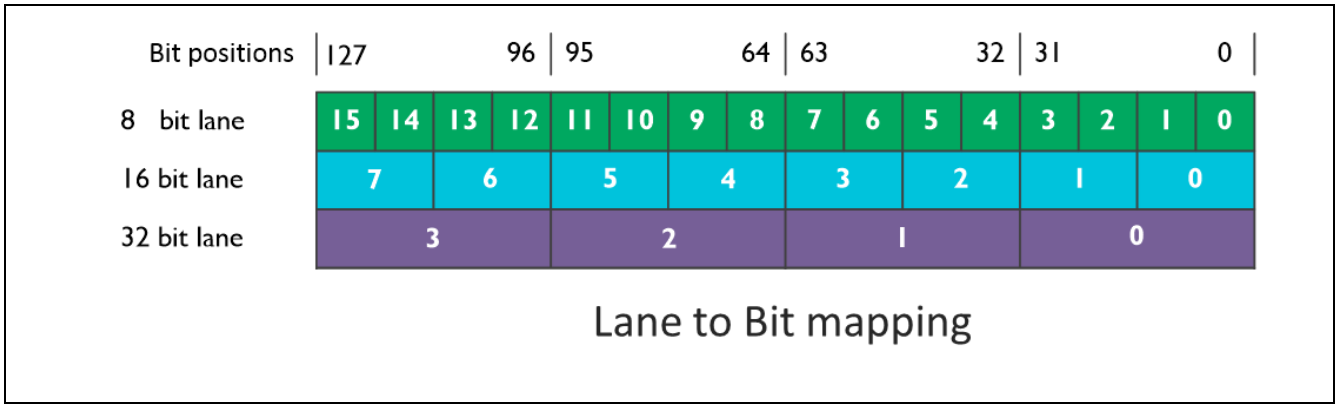

图 1-1Helium lane 与数据宽度定义

未向量化处理的 RGB565 转 RGB888 的流程可以参考图 1-2。每次处理一个 RGB565 pixel 的数据，输入数据宽度为 16-bit，转换中所使用的数据操作皆为 Byte-based，转换后的 RGB888 pixel 数据有三个 byte，输出数据宽度为 24-bit。

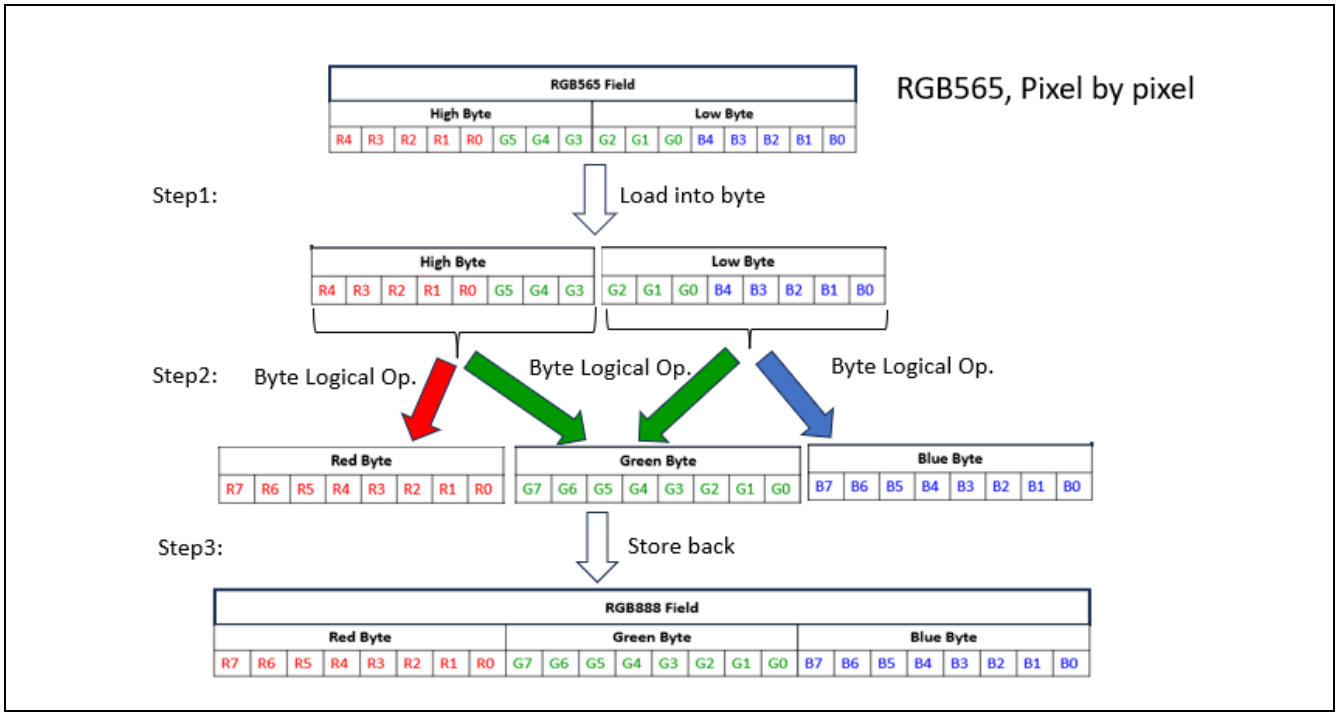

图 1-2 RGB565 转 RGB888 流程

如果同时加载 16 个 pixel 的数据给 Helium 的 16 个 lane，在 Step2 就可以同时处理 16 个 Byte Logical Op.，达到平行化的处理方式，流程可参照图 1-3。

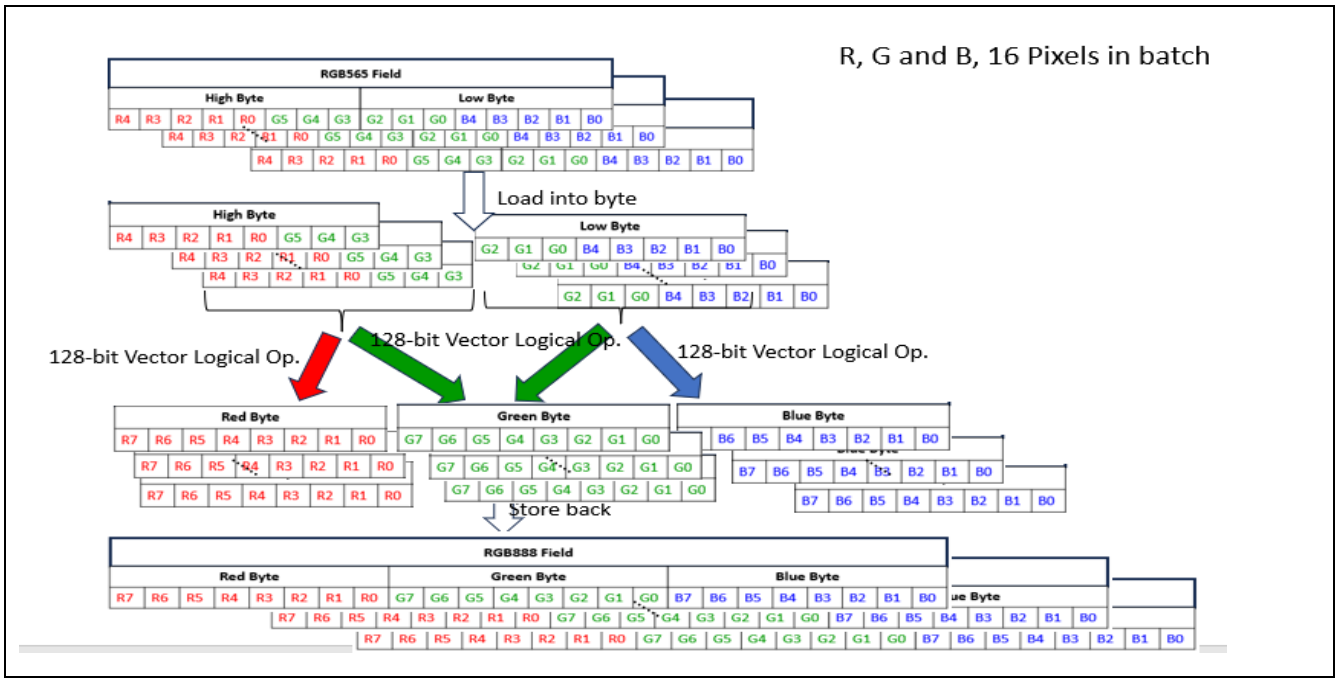

图 1-3 RGB565 转 RGB888 向量化处理示意图

像素数据在内存上是以连续的地址存放，向量寄存器进行 RGB566 高低 Byte 的读取，可以透过 Helium Scatter Load 方式来进行，读取到两个向量寄存器中。Scatter Load 须给定地址偏移量，以下图 1-4 为例，要读取 RGB565 的高 Byte 须指定 offset 为 *0x0e0c0a0806040200*，要读取 RGB565 的低 Byte 须指定 offset 为 *0x0f0d0b0907050301*，每一 offset 用 1 个 byte 来表示。

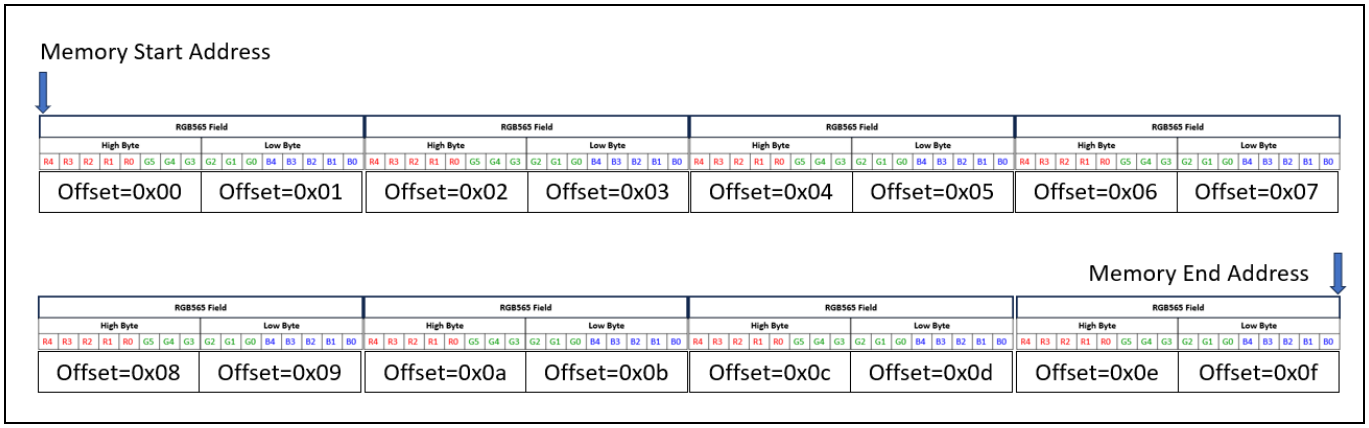

图 1-4 RGB565 Scatter Load Offset 示意图

完成格式转换后，因为转换后的 R,G 和 B 的数据分别存放在三个向量寄存器，可透过 Helium Scatter Store 方式将数据摆放至连续的内存地址上，以图 1-5 为例，要将 RGB888 放回连续的内存地址上，需要指定三组 offset，R 须指定 offset 为 0x0300，G 须指定 offset 为 0x0401，B 须指定 offset 为 0x0502。

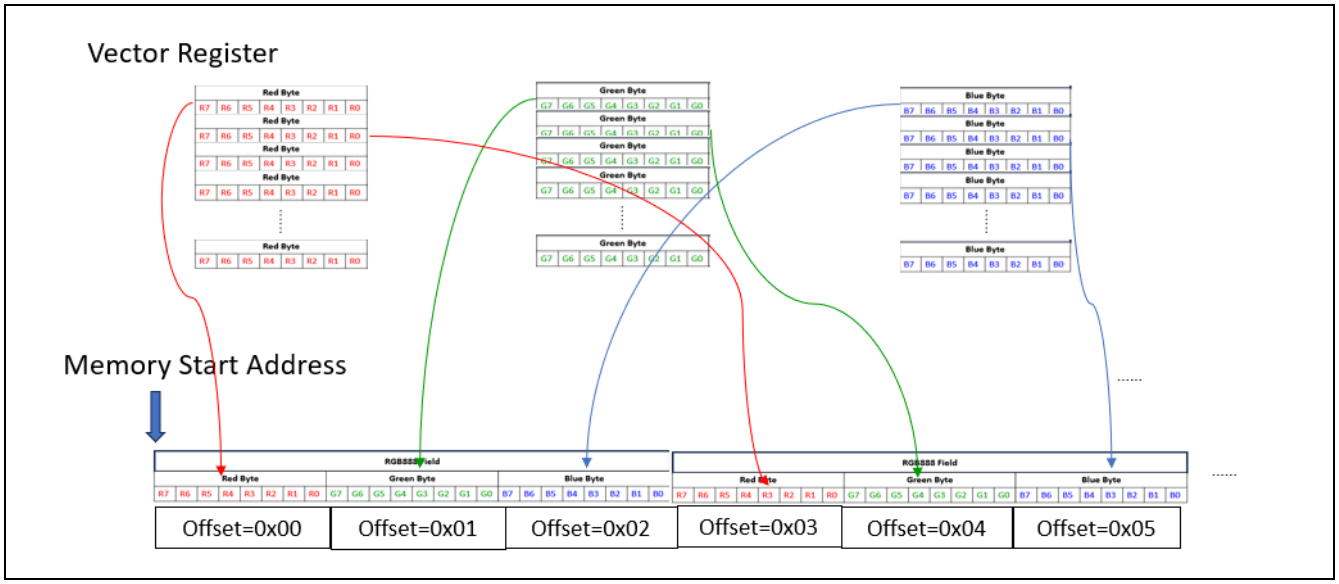

图 1-5 RGB565 Scatter Store Offset 示意图

1.2 执行结果

本范例程序执行后会印出以下信息，包含向量化与非向量化色彩格式转换的执行 cycle 数目，同时也会逐笔比对色彩转换后的结果是否正确。

```
+----------------------------------------+
|   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. 代码介绍

计算 RGB565 转 RGB888 的程序，代码位于 *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 --;
    }
}
```

上述函数并未使用向量化处理，其处理方式是将资料逐笔进行色彩格式转换。

使用向量化进行色彩转换的函数如下:

```
….
#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;

    }
}
```

将上述范例中所使用到的 Helium 相关函数整理摘要依逻辑运算和数据加载整理如下:

Intrinsic 函数名称	参数	返回值	函数说明
vshrq	(uint8x16_t a, const int imm)	uint8x16_t	将参数中第一向量寄存器 a 的每个元素右移一常数值 imm，结果写入返回值向量寄存器。
vrshlq	(uint8x16_t a, int32_t b)	uint8x16_t	将参数中第一个向量寄存器 a 的每个元素左移位第二个变量 b 的最低字节的值，结果写入返回值向量寄存器。
vorrq_u8	(uint8x16_t a, uint8x16_t b)	uint8x16_t	计算参数中两个向量寄存器 a 和 b 的 bitwise logic OR 结果，结果写入返回值向量寄存器。

表 2-1Helium 逻辑运算部分相关函数表

此部分和非向量的差别就在于同时进行了 16 个 Lane 的逻辑运算，以蓝色为例，非向量运算:

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

向量运算:

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

Intrinsic 函数名称	参数	返回值	函数说明
vcreateq_u8	(uint64_t a, uint64_t b)	uint8x16_t	传回一个返回值向量，其寄存器表示形式与参数指定的两个 64 位数值 a 和 b 的串联相同。
vldrbq_gather_offset_z_u8	(uint8_t const * base, uint8x16_t offset, mve_pred16_t p)	uint8x16_t	将连续元素从内存加载到目标向量寄存器。在此模式下，目标地址是根据基地址寄存器 base 和偏移量 offset 计算得出的。数据长度不足的部分会参考 p 由 Helium 填零补足。
vstrbq_scatter_offset_p	(uint8_t * base, uint8x16_t offset, uint8x16_t value, mve_pred16_t p)	void	将连续元素从向量寄存器 value 储存到内存基址位置。在此模式下，目标地址是根据基址寄存器 base 和偏移量 offset 计算得出数据长度不足的部分会参考 p 由 Helium 填零补足。

表 2-2Helium 数据加载部分相关函数表

此部分可对照图 1-4 图 1-5 关于 Scatter Load/Store 的说明，程序中非向量一次加载 1 个 pixel :

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

向量一次载入 16 个 pixel，并将 high/low byte 放在两个不同的向量寄存器，因为相邻 pixel 的 low byte 地址非连续，因此需提供 pixel 的基地址与偏移量:

```
……
#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 各为 64-bit 宽，依表 2-2 表 2-2 说明，执行上述两行程序代码后，向量寄存器 offset_8_16 的内容值为:

Offset_8_16(uint8x16t)

0x1F	0 x 1 D	0 x 1 B	0x19	0x17	0x15	0x13	0x11	0xF	0xD	0xB	0x9	0x7	0x5	0x3	0x1

vsrc_8_16_lo(uint8x16t)

src [0 x 1 F]	src [0 x 1 D]	src [0 x 1 B]	src [0 x 1 9]	src [0 x 1 7]	src [0 x 1 5]	src [0 x 1 3]	src [0 x 1 1]	src [0xF]	src [0xD]	src [0xB]	src [0x9]	src [0x7]	src [0x5]	src [0x3]	src [0x1]

数据储存部分:

非向量储存时一次存放 1 个 RGB888 pixel(3 bytes):

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

向量储存时，向量寄存器的内容(16 lanes)会被同时储存至内存，需提供储存地址的基地址和偏移量，让 RGB888 在内存采连续方式排列。

以 R 为例:

```
……
#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);
```

上述程序代码执行后，内存 dst 的资料如下:

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

G 和 B 的数据储存方式和 R 相似，可自行参考 R 部分。

3. 软件与硬件需求

3.1 软件需求

- BSP 版本

  M55M1_Series_BSP_CMSIS_V3.00.001

- IDE 版本

  Keil uVersion 5.28

3.2 硬件需求

- 电路组件

  NuMaker-M55M1(Beta V1.1)

- 线路示意图

  如下

图 3-1 将 UART0 TX (PB.13) pin 脚连接到 Nu_Link Me，再透过 VCOM 连接 PC，BaudRate 设定为 115200 bps，以显示范例代码的执行结果。

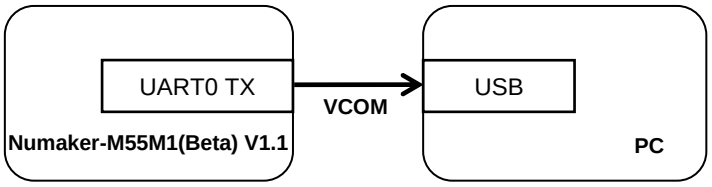

图 3-1 线路示意图

4. 目录信息

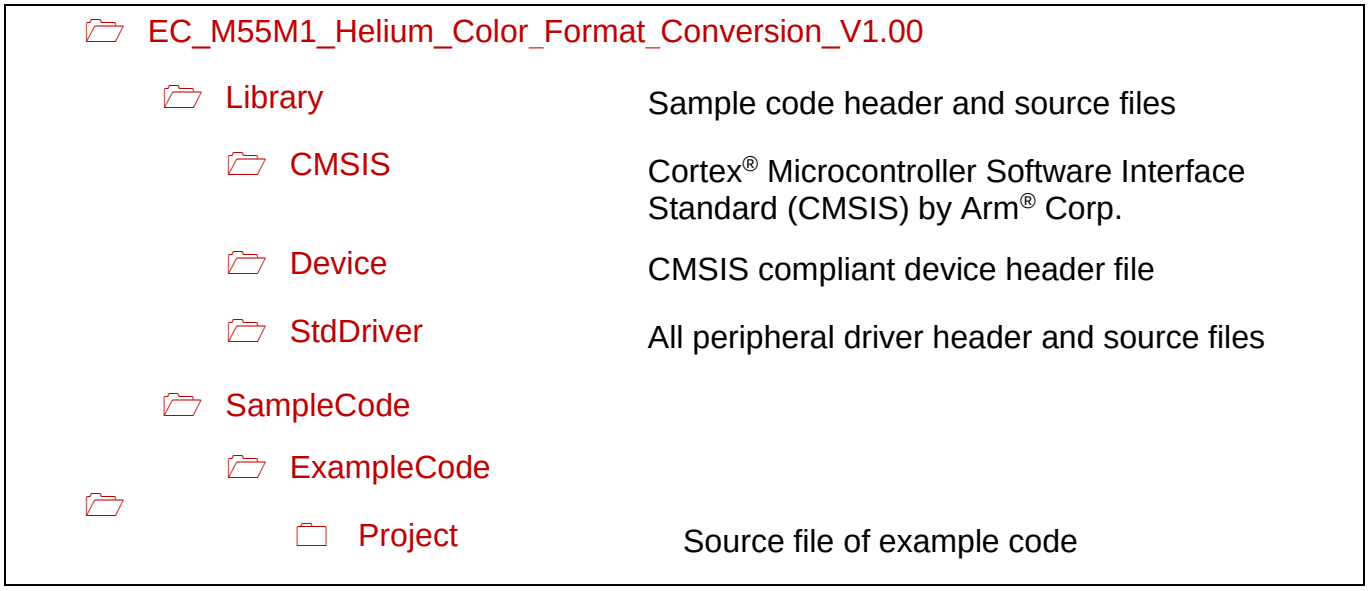

图 4-1 目录信息

5. 范例程序执行

1. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击 M55M1_Helium_Color_Format_Conversion.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

6. 修订纪录

Date	Revision	Description
2024.06.21	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.