M55M1 CMOS 影像经 HyperRAM 输出至 LCD

NuMicro® 32 位系列微控制器范例程序代码介绍

文件信息

应用简述	M55M1 系列微控制器通过 CCAP 撷取 CMOS 影像，经 HyperRAM 缓冲后输出至 LCD 显示。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.003
开发平台	NuMaker-X-M55M1D 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

本文提出一个在 NuMicro® M55M1 平台上，整合 CMOS 传感器（CCAP 取流）、HyperRAM 外部帧缓冲(FrameBuffer)、以及 LCD 显示的实时影像连接通路。系统以 QVGA (320×240) / RGB565 为输入影像格式，采用 HyperRAM 作为外部缓冲存储器，并在 LCD 显示端执行等比例放大（至 640×480）。程序架构以条件编译旗标切换功能，并内建一次性 FPS 量测以利效能评估。实验显示，在单帧循环架构下，可快速验证完整数据路径并建立后续扩充功能（不同分辨率与像素格式、以及 AI 前处理）之基础。

1.1 系统架构与数据流

数据流为 CMOS 传感器从 CCAP 界面以 RGB565/QVGA 取流至 HyperRAM 中的 FrameBuffer；显示端从 FrameBuffer 读取并输出至 LCD。

CMOS Sensor → CCAP → HyperRAM (FrameBuffer) → Display → LCD 如图 1-1 所示。

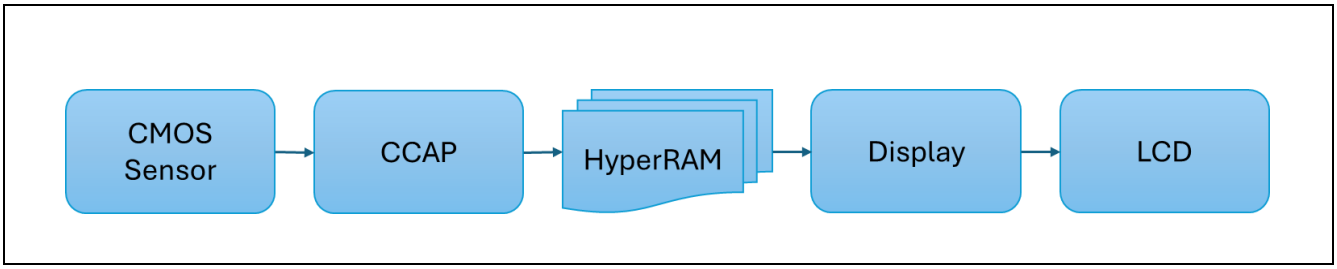

图 1-1 数据流

1.2 模块功能

- **CCAP（撷取）**：设定 RGB565, 320×240，每循环触发一次撷取，输出直写 HyperRAM 帧缓冲。
- **FrameBuffer（HyperRAM）**：以 32-byte 对齐与自定区段放置主 FrameBuffer 与 JPEG 缓冲，并提供 OpenMV 最小化 FrameBuffer 所需之全局变量（_fb_base、_fb_end、_fballoc、_jpeg_buf）。
- **OpenMV 最小化 FrameBuffer**：在既有缓冲上建立 **image_t** 与 **fb_alloc**，**FrameBuffer** 直接位于现有缓冲区上，避免多余搬移。
- **显示输出（LCD）**：以矩形区块填充 API ，将来源帧等比例 2 倍放大输出至 LCD。

- **效能量测**：以 PMU、SysTick 计时单帧显示的时间，换算 FPS。

1.3 M55M1 内存配置

本芯片采用统一的 4 GBytes 地址空间（0x0000_0000 – 0xFFFF_FFFF）。

主要内存与控制器配置如下：

- ITCM / DTCM：指令与数据高速缓存
- 内建 Flash：FLASH0、FLASH1
- 内部 SRAM：SRAM0 ~ SRAM3、LPSRAM
- 外部接口：EBI、SPIM0

其中，SPIM0 HyperRAM 区间（0x8200_0000 – 0x83FF_FFFF，容量 32 MB）为本项目主要使用区域，用作外部影像帧缓冲区，可与 scatter 档及原始码对应。

下表列出与本项目较相关的内存区（容量以十进制约数表示）：

地址范围	缓存器	说明	容量
0x0000_0000 – 0x0000_FFFF	ITCM_BA	ITCM 指令快取/紧耦合内存	64 KB
0x0010_0000 – 0x001F_FFFF	FLASH0_BA	内建 Flash0	1024 KB
0x0020_0000 – 0x002F_FFFF	FLASH1_BA	内建 Flash1	1024 KB
0x2000_0000 – 0x2001_FFFF	DTCM_BA	DTCM 数据紧耦合内存	128 KB
0x2010_0000 – 0x2017_FFFF	SRAM0_BA	内部 SRAM0	512 KB
0x2018_0000 – 0x201F_FFFF	SRAM1_BA	内部 SRAM1	512 KB
0x2020_0000 – 0x2024_FFFF	SRAM2_BA	内部 SRAM2	320 KB
0x2030_0000 – 0x2030_1FFF	SRAM3_BA	内部 SRAM3	8 KB
0x2031_0000 – 0x2031_1FFF	LPSRAM_BA	低功耗 SRAM	8 KB
0x6000_0000 – 0x602F_FFFF	EXTMEM_BA	外部并行储存器（如 EBI）	3 MB
0x81F0_0000 – 0x81F7_FFFF	SRAM0_BA (Alias)	SRAM0 非快取别名	512 KB

0x81F8_0000 – 0x81FF_FFFF	SRAM1_BA (Alias)	SRAM1 非快取别名	512 KB
0x8200_0000 – 0x83FF_FFFF	SPIM0_BA	SPIM0（HyperBUS）内存空间	32 MB

1.4 执行结果

系统上电后依序完成开发板初始化与 OpenMV 最小化 FrameBuffer 建立，串行端口会印出 HyperRAM 帧缓冲起始地址（SPIM0：0x8200_0000）与 FPS 量测值（启用 __PROFILE__ 时）。启用 __USE_CCAP__ 与 __USE_DISPLAY__ 后，CCAP 以 RGB565/QVGA（320×240）撷取单帧到 HyperRAM，显示端将来源帧缓冲区等比例放大 2×，于 LCD 以 (0,0) → (639,479) 的矩形区域输出。画面更新以「撷取 → 显示」的单循环形式持续进行，用以快速验证 CMOS→CCAP→HyperRAM→LCD 的整体数据流。执行环境如图 1-2 所示。

从资源观点，单帧帧缓冲占用 320×240×2 = 153,600 Bytes。预期为 22.97 MB/s（30 FPS），可作为后续面板时序或总线设定的参考基准。

实际显示时，若面板分辨率大于 640×480，2×放大影像将置于指定矩形内（其余区域依面板驱动设定保留底色）；若面板分辨率等于或小于 640×480，显示驱动会依矩形边界进行填满或裁切。启用 __PROFILE__ 时，FPS 计算后打印至串行端口，可据此比较不同频率、放大倍率或总线设定对效能的影响。终端机画面如图 1-3 所示。开发板结果如图 1-4 所示。

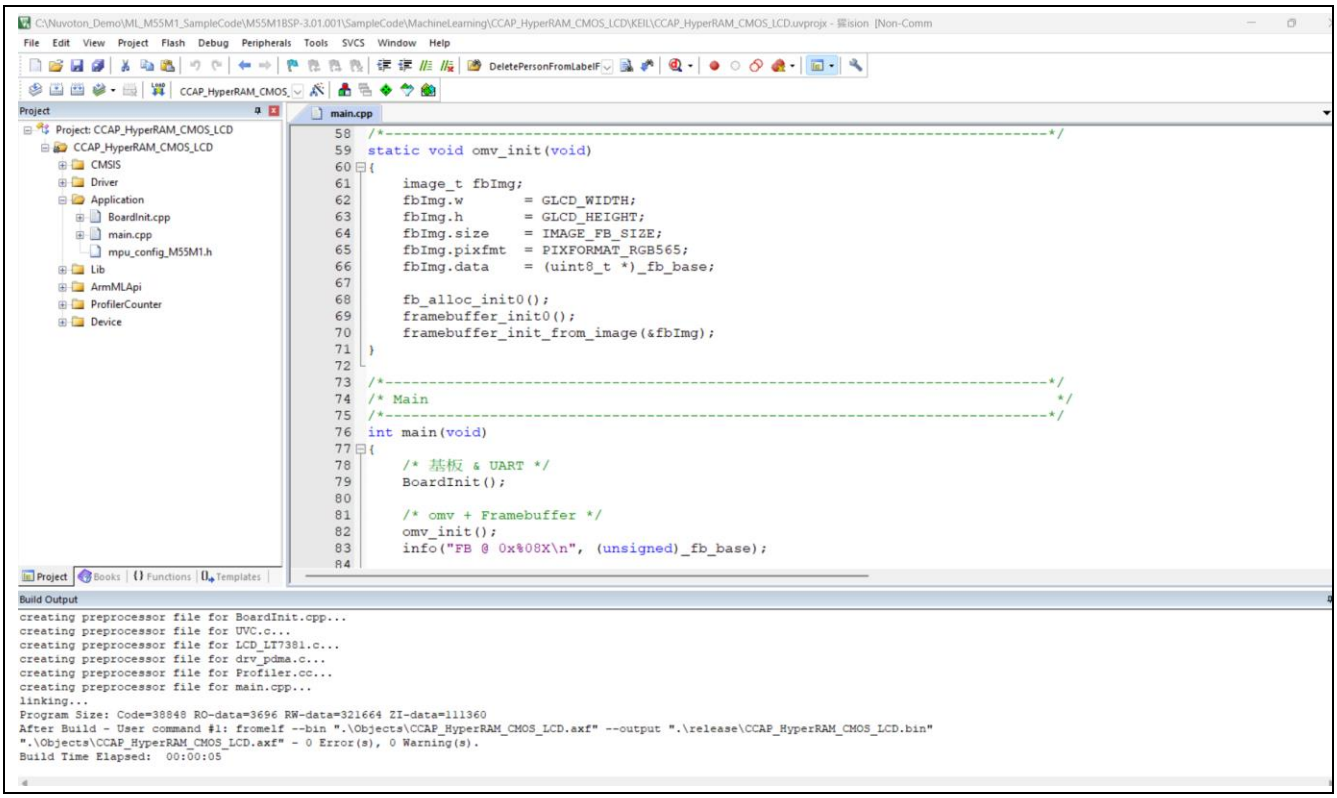

图 1-2 选择编译选项与执行编译

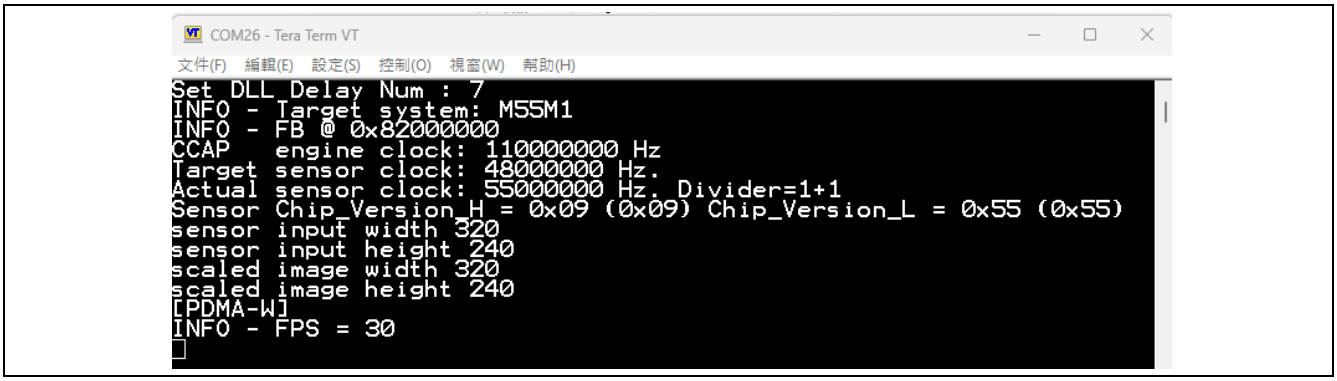

图 1-3 执行画面

图 1-4 屏幕呈现图像

1.5 使用 HyperRAM 注意内容

将 SPIM0_START / SPIM0_SIZE 调整为板子上 HyperRAM 的 SPIM0 地址与容量，并保留 UNINIT 以避免上电时清空大型缓冲；SRAM 非快取区用于 CCAP、LCD 等由 DMA 直接读写的缓冲，可避免快取不一致。为了容量保险，建议在 scatter 文档最后加入连结期检查，在编译过程中捕获过度分配内存。

- Scatter 文件设定内容

下列设定将 .vram.hyper 放入 HyperRAM（SPIM0：0x8200_0000）并标示 UNINIT；同时提供一块非快取区给外设共享缓冲，及（选用）在内部 SRAM2 放暂存数据。

```
/* HyperRAM：放影像 FrameBuffer（UNINIT，避免上电清空） */
SPIM_HYPERRAM 0x82000000 UNINIT 0x02000000 { .ANY(.vram.hyper) }

/* 非快取区：给 DMA/CCAP/LCD 共享缓冲，避免快取不同步 */
SRAM_NONCACHEABLE 0x81F00000 0x00100000
{ .ANY(NonCacheable.Init, .bss.NonCacheable.ZeroInit) }

/* 内部 SRAM2（选用）：小型暂存，也标 UNINIT */
SRAM2 0x20200000 UNINIT 0x00050000 { .ANY(.bss.vram.data) }
```

- 影像传感器大小确认

RGB565 单帧大小 = 宽 × 高 × 2。若传感器分辨率非 QVGA（320×240），请同步调整宽、高与放大倍率，并确认 .vram.hyper（HyperRAM）可容纳 OMV_FB_SIZE = GLCD_WIDTH * GLCD_HEIGHT * 2 + 1024；若采用双缓冲，容量需再乘以 2。

- 链接期容量检查

在 scatter 文档最后加入检查，确保「非快取区 + HyperRAM」的总占用不超过可用容量；若已用宏定义容量，可改写为 (SPIM_ALIASE_SIZE + SPIM0_SIZE) 以便维护。

```
ScatterAssert((ImageLength(SRAM_NONCACHEABLE) + ImageLength(SPIM_HYPERRAM))
              < (0x00100000 + 0x02000000))
```

- 程序使用方法

将缓冲变数以 .**vram.hyper** 区段宣告，链接器即依 Scatter 对映到 HyperRAM；并提供影像框架所需的全局变量（名称请勿更动、亦勿设为 static），即可直接以这些变量作为来源 / 目的缓冲。

```
__attribute__((section(".vram.hyper"), aligned(32)))
static char fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE];

__attribute__((section(".vram.hyper"), aligned(32)))
static char jpeg_array[OMV_JPEG_BUF_SIZE];

char* _fb_base  = fb_array;
char* _fb_end   = fb_array + OMV_FB_SIZE - 1;
char* _fballoc  = fb_array + OMV_FB_SIZE;
char* _jpeg_buf = jpeg_array;
```

2. 程序代碼介紹

在 程序代码端 (main.cpp)，使用 __attribute__((section(".vram.hyper"), aligned(32))) 宣告两个主要的全局缓冲区：

- fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE]，作为 FrameBuffer 与 fb_alloc 的储存空间。
- jpeg_array[OMV_JPEG_BUF_SIZE]，作为 JPEG 压缩暂存缓冲区。

并提供 OpenMV 所需的全局符号 _fb_base = fb_array 与 _jpeg_buf = jpeg_array，让系统能正确定位 FrameBuffer 与 JPEG Buffer。

在 Linker/Scatter 档 (M55M1.scatter) 中，则将 .vram.hyper 区段对应至 HyperRAM，确保上述缓冲区能配置在外部 HyperRAM 内存，避免占用内部 SRAM，满足 OpenMV 图像处理对大容量缓冲的需求。

- Linker/Scatter（*M55M1.scatter*）重点节录

```
#define SPIM0_START 0x82000000
#define SPIM0_SIZE  0x02000000

SPIM_HYPERRAM SPIM0_START UNINIT SPIM0_SIZE
{
    .ANY  (.vram.hyper)
    .ANY1 (.bss.NoInit.activation_buf_sram)
}
```

将 **.vram.hyper**（**fb_array / jpeg_array** 所在区段）整个配置到 0x8200_0000 起始的 **HyperRAM**，并以 **UNINIT** 避免开机时清零 / 拷贝庞大缓冲。

- SRAM 非快取区（SRAM01 aliased）

```
#define SRAM01_ALIASE_START 0x81F00000
#define SRAM01_ALIASE_SIZE  0x00100000

SRAM_NONCACHEABLE SRAM01_ALIASE_START SRAM01_ALIASE_SIZE
{
    .ANY (NonCacheable.Init)
    .ANY (.bss.NonCacheable.ZeroInit)
}

; Empty section to get maximum size of NonCacheable region with
Image$$SRAM_NONCACHEABLE_END$$Base
```

```
SRAM_NONCACHEABLE_END AlignExpr(+0, 1024)
{
    .ANY (SRAM_NONCACHEABLE_END)
}
```

供需要非快取属性的数据使用（例如与外设 DMA 共享缓冲）。定义一个空部分 **SRAM_NONCACHEABLE_END** 来导出边界符号以便计算容量上限。

- SRAM2（内部 SRAM 区）

```
#define SRAM2_START 0x20200000
#define SRAM2_SIZE  0x00050000

SRAM2 SRAM2_START UNINIT SRAM2_SIZE
{
    .ANY (.bss.vram.data)
}
```

此段设定将 **.bss.vram.data** 区段配置于内部 SRAM2（地址 0x2020_0000，容量 320 KB），并标注为未初始化（UNINIT）区域，用于储存执行时期的暂存数据或影像缓冲。

- 超额保护

```
ScatterAssert((ImageLength(SRAM_NONCACHEABLE) + ImageLength(SPIM_HYPERRAM))
              < (SRAM01_ALIASE_SIZE + SPIM0_SIZE))
```

连结期检查非快取区与 HyperRAM 区的总占用，不超出内存容量。

开发时若要使用外部内存（如 HyperRAM）存放大型影像或数据缓冲区，可用 __attribute__((section)) 放入自定义区段，并在 scatter file 中对应到外部内存；若不想每次上电被清空，可加上 UNINIT 属性。当外部内存同时被 DMA、显示控制器或 CMOS 传感器使用时，建议在 MPU 中将该区设为「非快取」（Non-Cacheable），或在每次数据传输后进行快取清理，以避免画面异常或数据错乱。

2.1 主循环流程

在 **main.cpp** 系统启动后，首先进行初始化，可选择执行显示初始化与传感器/CCAP 初始化。在进入主循环 (LOOP) 后，流程如下：

1. 触发 CCAP 撷取，将影像数据存入 **_fb_base**（配置于 HyperRAM 的 FrameBuffer）。

2. 构造输出矩形，范围从 (0,0) 至 (GLCD_WIDTH×2 - 1, GLCD_HEIGHT×2 - 1)。

3. 呼叫 **Display_FillRect()**，将来源 FrameBuffer 数据绘制至 LCD，并进行 2 倍放大输出。

4. 若定义了 **__PROFILE__**，则透过 SysTick 或 PMU 纪录图像处理时间，并计算 FPS。

流程结束后重复循环，持续进行影像撷取与显示。

```
初始化 → 显示初始化 → 传感器/CCAP 初始化
LOOP:
  1) 触发 CCAP 撷取至 _fb_base（HyperRAM）
  2) 构造输出矩形:  (0,0) ~ (GLCD_WIDTH*2-1, GLCD_HEIGHT*2-1)
  3) Display_FillRect(来源=framebuffer, 目标=LCD, 放大=2×)
  4) 若 __PROFILE__ 启用: 以 SysTick/PMU 记录时间并计算 FPS
END
```

2.2 Scatter file 与「放到指定地址 / 内存区」

- 命名区段与 Scatter 对应配置

在本范例中，特定数据区段会以命名方式标注，并于 scatter 文档中对应至指定的内存区域。

例如：

```
__attribute__((section(".vram.hyper"), aligned(32)))
static char fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE];

__attribute__((section(".vram.hyper"), aligned(32)))
static char jpeg_array[OMV_JPEG_BUF_SIZE];
```

上述程序将 fb_array 与 jpeg_array 变量放入自定义的 .vram.hyper 区段。

在 scatter 文件中，该区段会被配置至 SPIM0（HyperRAM）地址 0x8200_0000 起始的 SPIM_HYPERRAM 区域，并设定为 UNINIT，以避免系统开机时清空此大容量缓冲区，从而缩短启动时间。

- 完整程序代码

```
/**************************************************************************//**
 * @file     main.cpp
 * @brief    CCAP → HyperRAM Framebuffer → LCD Demo
 * This demo captures a QVGA RGB565 frame from a CMOS sensor via CCAP,
 * stores it in HyperRAM as a framebuffer, then displays it on an LCD
 * with 2× upscaling. Optional one-shot FPS profiling is provided.
 *****************************************************************************/
#include "BoardInit.hpp"
#include "log_macros.h"
```

```
#include "imlib.h"          // OpenMV basic image structures
#include "framebuffer.h"
#include "Display.h"
#include "ImageSensor.h"
#include "Profiler.hpp"
#include "NuMicro.h"

/*---------------------------------------------------------------------------*/
/* Feature toggles                                                           */
/*---------------------------------------------------------------------------*/
#define __PROFILE__              /* Enable one-shot FPS profiling            */
#define __USE_CCAP__             /* Enable CMOS Sensor + CCAP capture        */
#define __USE_DISPLAY__          /* Enable LCD output (e.g., LT7381)         */

/*---------------------------------------------------------------------------*/
/* Image / Framebuffer configuration                                         */
/*---------------------------------------------------------------------------*/
#define GLCD_WIDTH                   320
#define GLCD_HEIGHT                  240
#define IMAGE_FB_SIZE               (GLCD_WIDTH * GLCD_HEIGHT * 2)  /* RGB565 */
#define IMAGE_DISP_UPSCALE_FACTOR     2   /* Display upscale factor (2x)      */

/* OpenMV defaults (do not re-define elsewhere) */
#define OMV_FB_SIZE         (IMAGE_FB_SIZE + 1024)

/*---------------------------------------------------------------------------*/
/* HyperRAM framebuffer region (scatter maps .vram.hyper to 0x8200_0000)     */
/*---------------------------------------------------------------------------*/
__attribute__((section(".vram.hyper"), aligned(32)))
static char fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE];

__attribute__((section(".vram.hyper"), aligned(32)))
static char jpeg_array[OMV_JPEG_BUF_SIZE];

/* Global symbols required by OpenMV framebuffer/fb_alloc */
char* _fb_base  = fb_array;
char* _fb_end   = fb_array + OMV_FB_SIZE - 1;
char* _fballoc  = fb_array + OMV_FB_SIZE;
char* _jpeg_buf = jpeg_array;

/*---------------------------------------------------------------------------*/
/* Minimal OpenMV framebuffer initialization                                 */
/*---------------------------------------------------------------------------*/
static void omv_init(void)
{
    image_t fbImg;
    fbImg.w      = GLCD_WIDTH;
    fbImg.h      = GLCD_HEIGHT;
    fbImg.size   = IMAGE_FB_SIZE;
    fbImg.pixfmt = PIXFORMAT_RGB565;
    fbImg.data   = reinterpret_cast<uint8_t*>(_fb_base);

    fb_alloc_init0();
    framebuffer_init0();
    framebuffer_init_from_image(&fbImg);
}

/*---------------------------------------------------------------------------*/
/* Main                                                                      */
/*---------------------------------------------------------------------------*/
```

```
int main(void)
{
    /* Board & UART init */
    BoardInit();

    /* OpenMV framebuffer setup */
    omv_init();
    info("FB @ 0x%08X\n", (unsigned)(uintptr_t)_fb_base);

#if defined(__USE_DISPLAY__)
    Display_Init();
    Display_ClearLCD(C_WHITE);
#endif

#if defined(__USE_CCAP__)
    /* Sensor & CCAP init (QVGA RGB565) */
    ImageSensor_Init();
    ImageSensor_Config(eIMAGE_FMT_RGB565, GLCD_WIDTH, GLCD_HEIGHT, true);
#endif

#ifdef __PROFILE__
    bool printed_fps = false;
    pmu_reset_counters();
#endif

    while (1)
    {
#if defined(__USE_CCAP__)
        /* 1) Capture one frame to HyperRAM */
        ImageSensor_TriggerCapture((uint32_t)_fb_base);
        ImageSensor_WaitCaptureDone();
#endif

#if defined(__USE_DISPLAY__)
        /* 2) Upscale and display on LCD */
        S_DISP_RECT rect;
        rect.u32TopLeftX     = 0;
        rect.u32TopLeftY     = 0;
        rect.u32BottonRightX = (GLCD_WIDTH  * IMAGE_DISP_UPSCALE_FACTOR) - 1;
        rect.u32BottonRightY = (GLCD_HEIGHT * IMAGE_DISP_UPSCALE_FACTOR) - 1;

#ifdef __PROFILE__
        uint64_t t0 = pmu_get_systick_Count();
#endif

        Display_FillRect(reinterpret_cast<uint16_t*>(_fb_base),
                         &rect,
                         IMAGE_DISP_UPSCALE_FACTOR);

#ifdef __PROFILE__
        uint64_t t1 = pmu_get_systick_Count();
        if (!printed_fps) {
            info("FPS = %llu\n", FREQ_220MHZ / (t1 - t0));
            printed_fps = true;
        }
#endif
#endif
```

```
    }
}
```

- Scatter File

```
/* HyperRAM base/size */
#define SPIM0_START 0x82000000
#define SPIM0_SIZE  0x02000000

/* SRAM01 aliased (non-cacheable) */
#define SRAM01_ALIASE_START 0x81F00000
#define SRAM01_ALIASE_SIZE  0x00100000

/* SRAM2 base/size */
#define SRAM2_START 0x20200000
#define SRAM2_SIZE  0x00050000

/* Place .vram.hyper in HyperRAM, keep uninitialized on reset */
SPIM_HYPERRAM SPIM0_START UNINIT SPIM0_SIZE
{
    .ANY  (.vram.hyper)
    .ANY1 (.bss.NoInit.activation_buf_sram)
}

/* Non-cacheable bucket */
SRAM_NONCACHEABLE SRAM01_ALIASE_START SRAM01_ALIASE_SIZE
{
    .ANY (NonCacheable.Init)
    .ANY (.bss.NonCacheable.ZeroInit)
}

/* Optional internal SRAM bucket for .bss.vram.data */
SRAM2 SRAM2_START UNINIT SRAM2_SIZE
{
    .ANY (.bss.vram.data)
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - M55M1_Series_BSP_CMSIS_V3.01.003。
- IDE 版本
  - Keil uVision 5.40。

3.2 硬件需求

- 电路组件
  - NuMaker-X-M55M1D V1.0
  - HM1055 CMOS Sensor
  - NuMaker-TFT-LCD5 V1.2
- 线路示意图
  - 将 NuMaker-X-M55M1D V1.0、NuMaker-TFT-LCD 连接板和 CMOS 进行组装。
  - 使用 USB Type-C to Type-A 连接线，将 NuMaker-X-M55M1D V1.0 开发板连接至计算机上电。

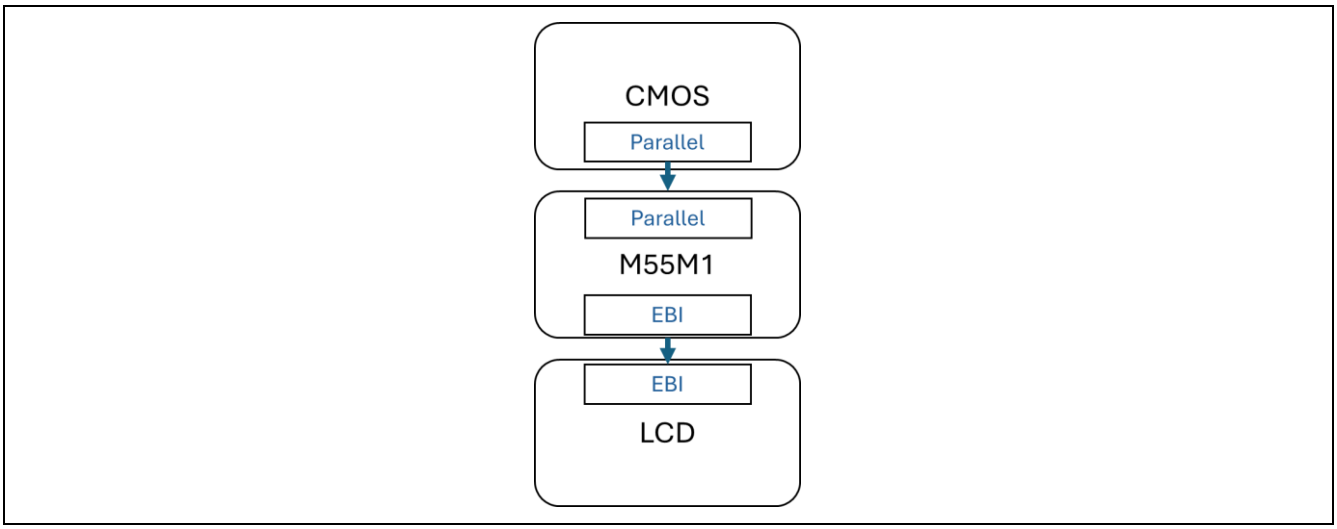

图 3-1 线路示意图

4. 目录信息

目录结构如下图所示。

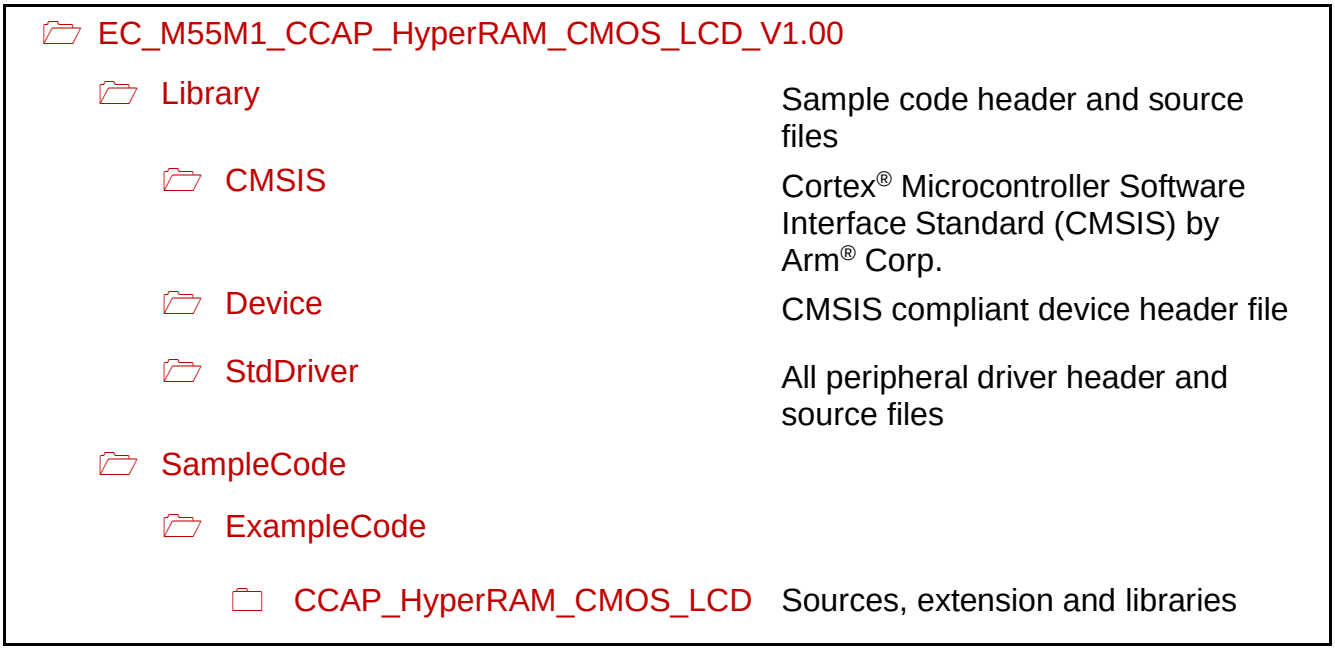

图 4-1 目录信息

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 *CCAP_HyperRAM_CMOS_LCD.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
2025.08.21	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.