M55M1 CMOS 影像經 HyperRAM 輸出至 LCD

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	M55M1 系列微控制器透過 CCAP 擷取 CMOS 影像，經 HyperRAM 緩衝後輸出至 LCD 顯示。
BSP 版本	M55M1_Series_BSP_CMSIS_V3.01.003
開發平台	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

本文提出一個在 NuMicro® M55M1 平台上，整合 CMOS 感測器（CCAP 取流）、HyperRAM 外部幀緩衝(FrameBuffer)、以及 LCD 顯示的即時影像鏈結通路。系統以 QVGA (320×240) / RGB565 為輸入影像格式，採用 HyperRAM 作為外部緩衝記憶體，並在 LCD 顯示端執行等比例放大（至 640×480）。程式架構以條件編譯旗標切換功能，並內建一次性 FPS 量測以利效能評估。實驗顯示，在單幀循環架構下，可快速驗證完整資料路徑並建立後續擴充功能（不同解析度與像素格式、以及 AI 前處理）之基礎。

1.1 系統架構與資料流

資料流為 CMOS 感測器從 CCAP 介面以 RGB565/QVGA 取流至 HyperRAM 中的 FrameBuffer；顯示端從 FrameBuffer 讀取並輸出至 LCD。

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

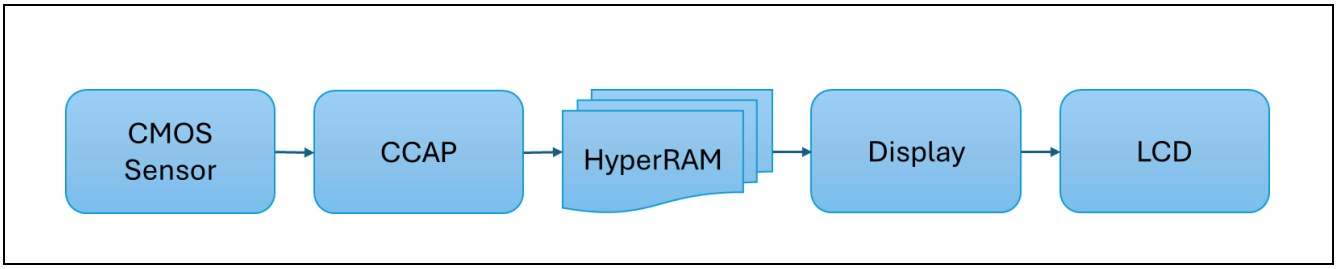

圖 1-1 資料流程

1.2 模組功能

- **CCAP（擷取）**：設定 RGB565, 320×240，每循環觸發一次擷取，輸出直寫 HyperRAM 幀緩衝。
- **FrameBuffer（HyperRAM）**：以 32-byte 對齊與自定區段放置主 FrameBuffer 與 JPEG 緩衝，並提供 OpenMV 最小化 FrameBuffer 所需之全域變數（_fb_base、_fb_end、_fballoc、_jpeg_buf）。
- **OpenMV 最小化 FrameBuffer**：在既有緩衝上建立 **image_t** 與 **fb_alloc**，**FrameBuffer** 直接位於現有緩衝區上，避免多餘搬移。
- **顯示輸出（LCD）**：以矩形區塊填充 API ，將來源幀等比例 2 倍放大輸出至 LCD。

- **效能量測**：以 PMU、SysTick 計時單幀顯示的時間，換算 FPS。

1.3 M55M1 記憶體配置

本晶片採用統一的 4 GBytes 位址空間（0x0000_0000 – 0xFFFF_FFFF）。

主要記憶體與控制器配置如下：

- ITCM / DTCM：指令與資料快取記憶體
- 內建 Flash：FLASH0、FLASH1
- 內部 SRAM：SRAM0 ~ SRAM3、LPSRAM
- 外部介面：EBI、SPIM0

其中，SPIM0 HyperRAM 區間（0x8200_0000 – 0x83FF_FFFF，容量 32 MB）為本專案主要使用區域，用作外部影像幀緩衝區，可與 scatter 檔及原始碼對應。

下表列出與本專案較相關的記憶體區（容量以十進位約數表示）：

位址範圍	暫存器	說明	容量
0x0000_0000 – 0x0000_FFFF	ITCM_BA	ITCM 指令快取/緊耦合記憶體	64 KB
0x0010_0000 – 0x001F_FFFF	FLASH0_BA	內建 Flash0	1024 KB
0x0020_0000 – 0x002F_FFFF	FLASH1_BA	內建 Flash1	1024 KB
0x2000_0000 – 0x2001_FFFF	DTCM_BA	DTCM 資料緊耦合記憶體	128 KB
0x2010_0000 – 0x2017_FFFF	SRAM0_BA	內部 SRAM0	512 KB
0x2018_0000 – 0x201F_FFFF	SRAM1_BA	內部 SRAM1	512 KB
0x2020_0000 – 0x2024_FFFF	SRAM2_BA	內部 SRAM2	320 KB
0x2030_0000 – 0x2030_1FFF	SRAM3_BA	內部 SRAM3	8 KB
0x2031_0000 – 0x2031_1FFF	LPSRAM_BA	低功耗 SRAM	8 KB
0x6000_0000 – 0x602F_FFFF	EXTMEM_BA	外部平行記憶體（如 EBI）	3 MB
0x81F0_0000 – 0x81F7_FFFF	SRAM0_BA (Alias)	SRAM0 非快取別名	512 KB

0x81F8_0000 – 0x81FF_FFFF	SRAM1_BA (Alias)	SRAM1 非快取別名	512 KB
0x8200_0000 – 0x83FF_FFFF	SPIM0_BA	SPIM0（HyperBUS）記憶體空間	32 MB

1.4 執行結果

系統上電後依序完成開發板初始化與 OpenMV 最小化 FrameBuffer 建立，序列埠會印出 HyperRAM 幀緩衝起始位址（SPIM0：0x8200_0000） 與 FPS 量測值（啟用 __PROFILE__ 時）。啟用 __USE_CCAP__ 與 __USE_DISPLAY__ 後，CCAP 以 RGB565/QVGA（320×240）擷取單幀到 HyperRAM，顯示端將來源幀緩衝區等比例放大 2×，於 LCD 以 (0,0) → (639,479) 的矩形區域輸出。畫面更新以「擷取 → 顯示」的單迴圈形式持續進行，用以快速驗證 CMOS→CCAP→HyperRAM→LCD 的整體資料流。執行環境如圖 1-2 所示。

從資源觀點，單幀幀緩衝佔用 320×240×2 = 153,600 Bytes。預期為 22.97 MB/s（30 FPS），可作為後續面板時序或匯流排設定的參考基準。

實際顯示時，若面板解析度大於 640×480，2×放大影像將置於指定矩形內（其餘區域依面板驅動設定保留底色）；若面板解析度等於或小於 640×480，顯示驅動會依矩形邊界進行填滿或裁切。啟用 __PROFILE__ 時，FPS 計算後打印至序列埠，可據此比較不同時脈、放大倍率或匯流排設定對效能的影響。終端機畫面如圖 1-3 所示。開發板結果如圖 1-4 所示。

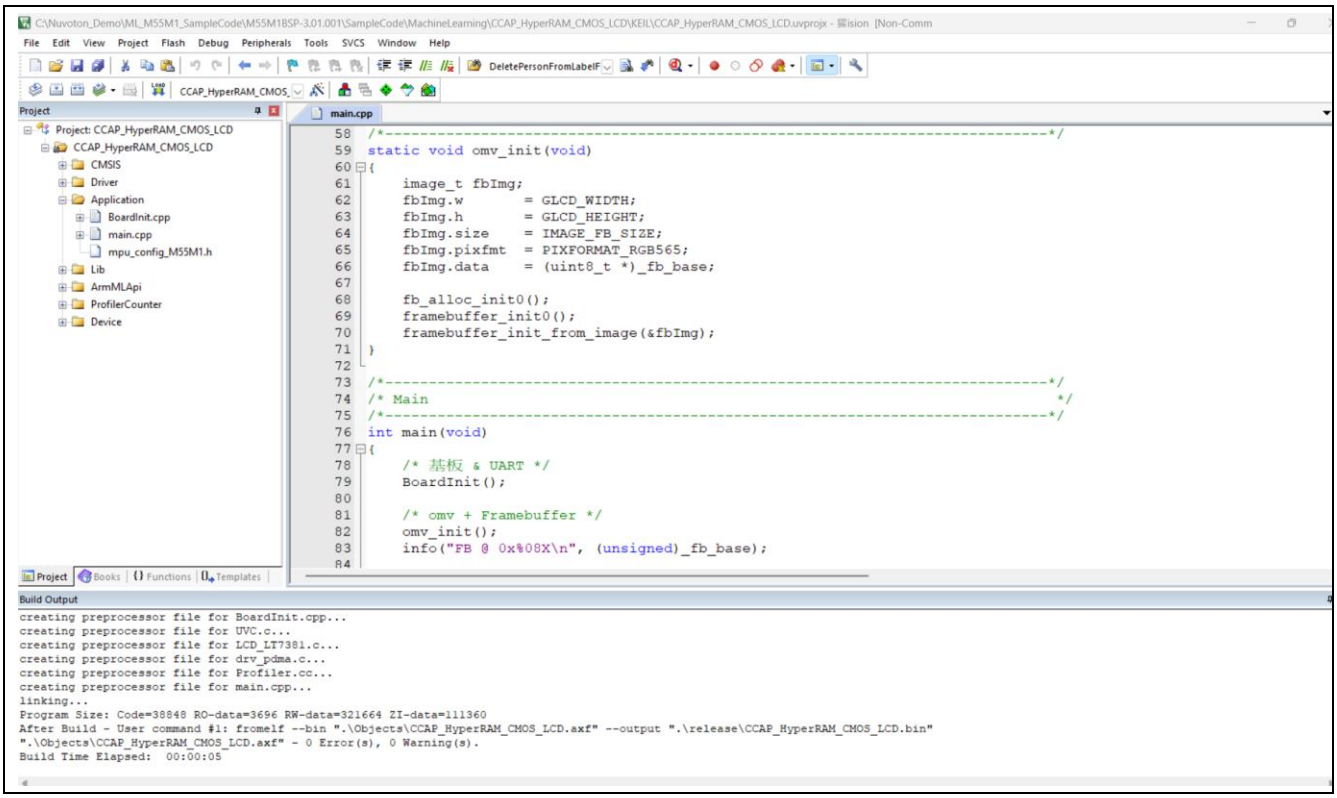

圖 1-2 選擇編譯選項與執行編譯

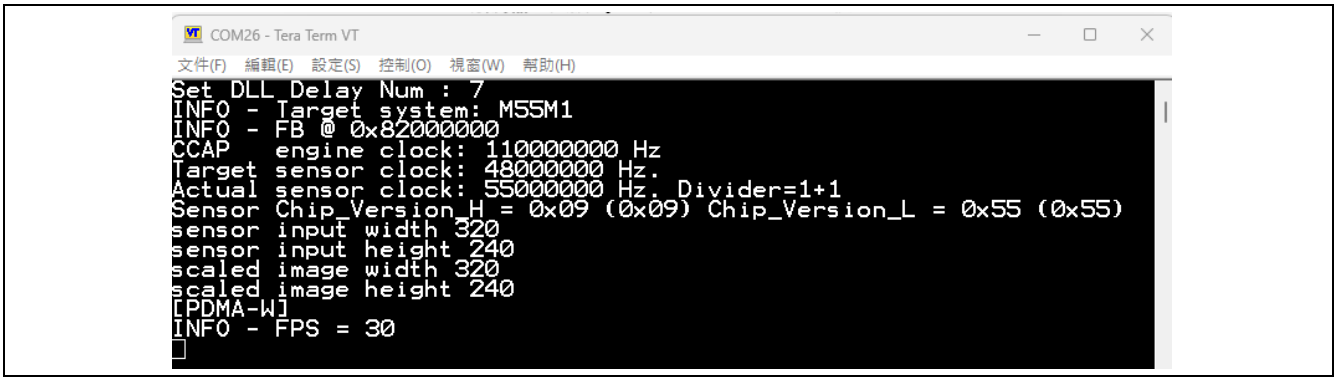

圖 1-3 執行畫面

圖 1-4 屏幕呈現圖像

1.5 使用 HyperRAM 注意內容

將 SPIM0_START / SPIM0_SIZE 調整為板子上 HyperRAM 的 SPIM0 位址與容量，並保留 UNINIT 以避免上電時清空大型緩衝；SRAM 非快取區用於 CCAP、LCD 等由 DMA 直接讀寫的緩衝，可避免快取不一致。為了容量保險，建議在 scatter 文件檔尾加入連結期檢查，在編譯過程中捕獲過度分配記憶體。

- Scatter 文件設定內容

下列設定將 .vram.hyper 放入 HyperRAM（SPIM0：0x8200_0000）並標示 UNINIT；同時提供一塊非快取區給外設共享緩衝，及（選用）在內部 SRAM2 放暫存資料。

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

/* 非快取區：給 DMA/CCAP/LCD 共享緩衝，避免快取不同步 */
SRAM_NONCACHEABLE 0x81F00000 0x00100000
{ .ANY(NonCacheable.Init, .bss.NonCacheable.ZeroInit) }

/* 內部 SRAM2（選用）：小型暫存，也標 UNINIT */
SRAM2 0x20200000 UNINIT 0x00050000 { .ANY(.bss.vram.data) }
```

- 影像感測器大小確認

RGB565 單幀大小 = 寬 × 高 × 2。若感測器解析度非 QVGA（320×240），請同步調整寬、高與放大倍率，並確認 .vram.hyper（HyperRAM）可容納 OMV_FB_SIZE = GLCD_WIDTH * GLCD_HEIGHT * 2 + 1024；若採雙緩衝，容量需再乘以 2。

- 連結期容量檢查

在 scatter 檔尾加入檢查，確保「非快取區 + HyperRAM」的總佔用不超過可用容量；若已用巨集定義容量，可改寫為 (SPIM_ALIASE_SIZE + SPIM0_SIZE) 以利維護。

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

- 程式使用方法

將緩衝變數以 .**vram.hyper** 區段宣告，連結器即依 Scatter 對映到 HyperRAM；並提供影像框架所需的全域變數（名稱請勿更動、亦勿設為 static），即可直接以這些變數作為來源 / 目的緩衝。

```
__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. 程式碼介紹

在 **程**式碼端 (main.cpp)，使用 __attribute__((section(".vram.hyper"), aligned(32))) 宣告兩個主要的全域緩衝區：

- fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE]，作為 FrameBuffer 與 fb_alloc 的儲存空間。
- jpeg_array[OMV_JPEG_BUF_SIZE]，作為 JPEG 壓縮暫存緩衝區。

並提供 OpenMV 所需的全域符號 _fb_base = fb_array 與 _jpeg_buf = jpeg_array，讓系統能正確定位 FrameBuffer 與 JPEG Buffer。

在 Linker/Scatter 檔 (M55M1.scatter) 中，則將 .vram.hyper 區段對應至 HyperRAM，確保上述緩衝區能配置在外部 HyperRAM 記憶體，避免佔用內部 SRAM，滿足 OpenMV 影像處理對大容量緩衝的需求。

- Linker/Scatter（*M55M1.scatter*）重點節錄

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

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

將 **.vram.hyper**（**fb_array / jpeg_array** 所在區段）整個配置到 0x8200_0000 起始的 HyperRAM，並以 **UNINIT** 避免開機時清零 / 拷貝龐大緩衝。

- SRAM 非快取區（SRAM01 aliased）

```
#define SRAM01_ALIASE_START 0x81F00000
#define SRAM01_ALIASE_SIZE  0x00100000  /* 1 MB */

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**非快取屬性**的資料使用（例如與外設 DMA 共享緩衝）。定義一個空部分 **SRAM_NONCACHEABLE_END** 來匯出邊界符號以便計算容量上限。

- SRAM2（內部 SRAM 區）

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

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

此段設定將 **.bss.vram.data** 區段配置於內部 SRAM2（位址 0x2020_0000，容量 320 KB），並標註為未初始化（UNINIT）區域，用於儲存執行時期的暫存資料或影像緩衝。

- 超額保護

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

連結期檢查非快取區與 HyperRAM 區的總佔用，不超出記憶體容量。

開發時若要使用外部記憶體（如 HyperRAM）存放大型影像或資料緩衝區，可用 __attribute__((section)) 放入自訂區段，並在 scatter file 中對應到外部記憶體；若不想每次上電被清空，可加上 UNINIT 屬性。當外部記憶體同時被 DMA、顯示控制器或 CMOS 感測器使用時，建議在 MPU 中將該區設為「非快取」（Non-Cacheable），或在每次資料傳輸後進行快取清理，以避免畫面異常或資料錯亂。

2.1 主迴圈流程

在 **main.cpp** 系統啟動後，首先進行初始化，可選擇執行顯示初始化與感測器/CCAP 初始化。在進入主循環 (LOOP) 後，流程如下：

1. 觸發 CCAP 擷取，將影像資料存入 **_fb_base**（配置於 HyperRAM 的 FrameBuffer）。
2. 構造輸出矩形，範圍從 (0,0) 至 (GLCD_WIDTH×2 - 1, GLCD_HEIGHT×2 - 1)。
3. 呼叫 **Display_FillRect()**，將來源 FrameBuffer 資料繪製至 LCD，並進行 2 倍放大輸出。
4. 若定義了 **__PROFILE__**，則透過 SysTick 或 PMU 紀錄影像處理時間，並計算 FPS。

流程結束後重複循環，持續進行影像擷取與顯示。

```
初始化 → 顯示初始化 → 感測器/CCAP 初始化
LOOP:
  1) 觸發 CCAP 擷取至 _fb_base（HyperRAM）
  2) 構造輸出矩形： (0,0) ~ (GLCD_WIDTH*2-1, GLCD_HEIGHT*2-1)
  3) Display_FillRect(來源=framebuffer, 目標=LCD, 放大=2×)
  4) 若 __PROFILE__ 啟用：以 SysTick/PMU 記錄時間並計算 FPS
END
```

2.2 Scatter file 與「放到指定位址 / 記憶體區」

- 命名區段與 Scatter 對應配置

在本範例中，特定資料區段會以命名方式標註，並於 scatter 檔中對應至指定的記憶體區域。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];
```

上述程式將 fb_array 與 jpeg_array 變數放入自訂的 **.vram.hyper** 區段。

在 scatter 檔中，該區段會被配置至 SPIM0（HyperRAM）位址 0x8200_0000 起始的 SPIM_HYPERRAM 區域，並設定為 UNINIT，以避免系統開機時清空此大容量緩衝區，從而縮短啟動時間。

- 完整程式碼

```
/**************************************************************************//**
 * @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  /* 1 MB */

/* 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. 軟體與硬體需求

3.1 軟體需求

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

3.2 硬體需求

- 電路元件
  - NuMaker-X-M55M1D V1.0
  - HM1055 CMOS Sensor
  - NuMaker-TFT-LCD5 V1.2
- 線路示意圖
  - 將 NuMaker-X-M55M1D V1.0、NuMaker-TFT-LCD 連接板和 CMOS 進行組裝。
  - 使用 USB Type-C to Type-A 連接線，將 NuMaker-X-M55M1D V1.0 開發板連接至電腦上電。

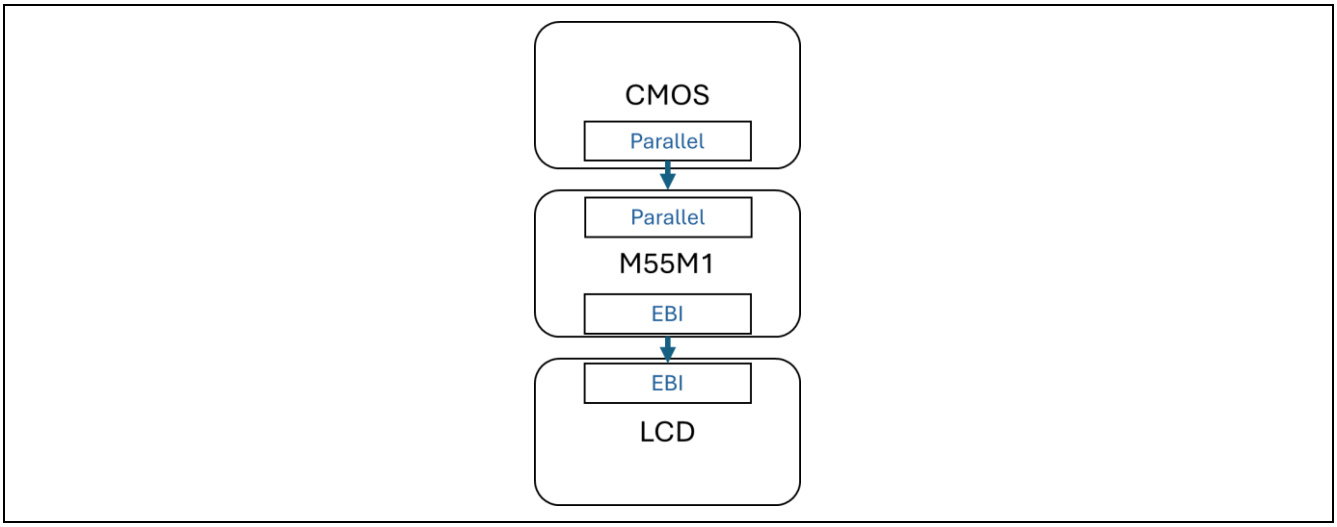

圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如下圖所示。

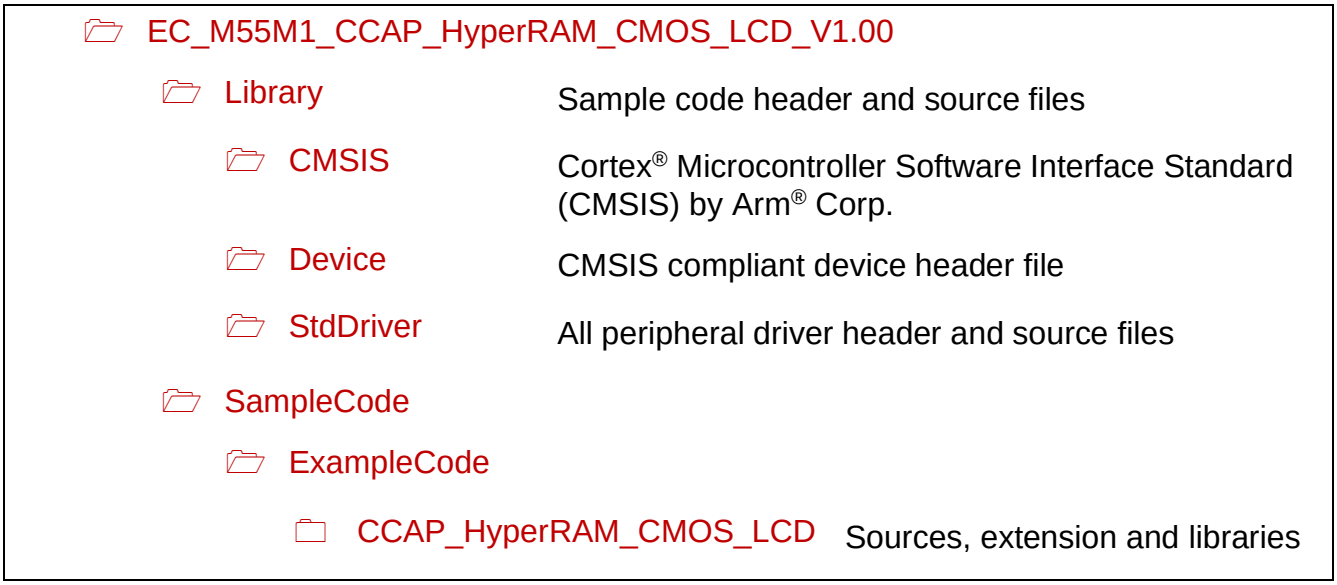

圖 4-1 目錄資訊

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 *CCAP_HyperRAM_CMOS_LCD.uvprojx*。

1. 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
2. 進入除錯模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2025.08.21	1.00	初始發佈。

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.