M55M1 CMOS Image Capture and Display via HyperRAM

Example Code Introduction for 32-bit NuMicro® Family

Document Information

Application	The M55M1 series microcontroller captures CMOS images through CCAP, buffers them via HyperRAM, and outputs them to the LCD for display.
BSP Version	M55M1_Series_BSP_CMSIS_V3.01.003
Hardware	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. Overview

This document proposes a real-time image pipeline on the NuMicro® M55M1 platform, integrating a CMOS sensor (streamed via CCAP), external FrameBuffer through HyperRAM, and LCD display. The system takes QVGA (320×240) input in RGB565 format, uses HyperRAM as the external buffer memory, and performs proportional upscaling to 640×480 at the LCD display output. The program architecture uses conditional compilation flags to switch functionalities and includes a one-time FPS measurement for performance evaluation. Experimental results show that under a single-frame loop structure, the complete data path can be quickly verified, establishing a foundation for future extensions such as different resolutions, pixel formats, and AI preprocessing.

1.1 System Architecture and Data Flow

The CMOS sensor streams from the CCAP interface in RGB565/QVGA data to FrameBuffer in HyperRAM; the display side reads from the framebuffer and outputs it to LCD.

The data flow, CMOS Sensor → CCAP → HyperRAM (FrameBuffer) → Display → LCD, is shown in Figure 1-1.

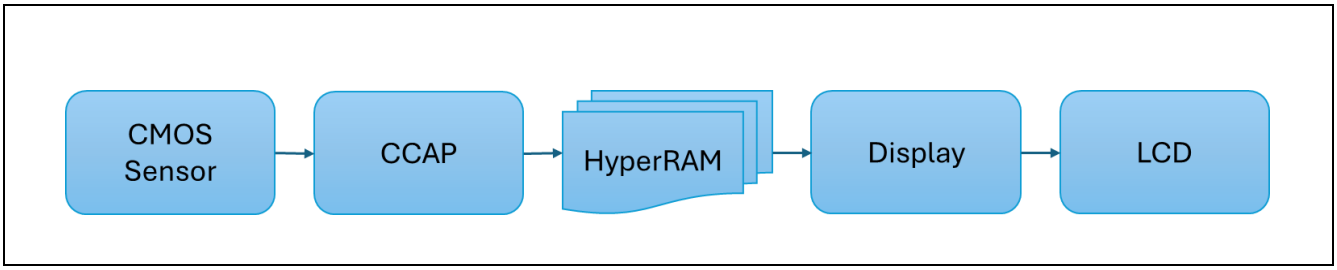

Figure 1-1 Data Flow

1.2 Module Functions

- **CCAP (Camera Capture)**: Configured for RGB565 at 320×240 resolution, triggered once per cycle to capture and directly write the output to the HyperRAM framebuffer.
- **FrameBuffer (HyperRAM)**: The main Framebuffer and JPEG buffer are placed with 32-byte alignment and custom memory sections, and global symbols required by the OpenMV minimal Framebuffer (_fb_base,_fb_end,_fballoc ,_jpeg_buf) are provided.
- **OpenMV Minimal Framebuffer**: Initializes **image_t** and **fb_alloc**, **Framebuffer** directly on the existing buffer to avoid redundant data copying.
- **Display Output (LCD)**: Uses a rectangle fill API to perform proportional upscaling of the source frame twice and output it to the LCD.
- **Performance Measurement**: Uses PMU/SysTick to time the display task of a single frame and calculate the corresponding FPS.

1.3 M55M1 Memory Configuration

The chip adopts a unified 4 Gbytes address space (0x0000_0000 – 0xFFFF_FFFF).

The primary memory and controller configuration includes:

- ITCM / DTCM: Instruction and data tightly-coupled memory
- On-chip Flash: FLASH0 and FLASH1
- Internal SRAM: SRAM0–SRAM3, LPSRAM
- External Interfaces: EBI, SPIM0

Among these, the SPIM0 HyperRAM window (0x8200_0000 – 0x83FF_FFFF, 32 MB) is the key region used in this project, serving as the external image frame buffer area. It corresponds to the scatter file and source code memory mapping.

The following table lists the memory areas that are more relevant to this project (capacity is expressed in decimal approximations):

Address range	Register	Description	Capacity
0x0000_0000 – 0x0000_FFFF	ITCM_BA	ITCM Instruction Cache/Tightly Coupled Memory	64 KB
0x0010_0000 – 0x001F_FFFF	FLASH0_BA	Embedded Flash0	1024 KB
0x0020_0000 – 0x002F_FFFF	FLASH1_BA	Embedded Flash1	1024 KB
0x2000_0000 – 0x2001_FFFF	DTCM_BA	DTCM Data Tightly Coupled Memory	128 KB
0x2010_0000 – 0x2017_FFFF	SRAM0_BA	Internal SRAM0	512 KB
0x2018_0000 – 0x201F_FFFF	SRAM1_BA	Internal SRAM1	512 KB
0x2020_0000 – 0x2024_FFFF	SRAM2_BA	Internal SRAM2	320 KB
0x2030_0000 – 0x2030_1FFF	SRAM3_BA	Internal SRAM3	8 KB
0x2031_0000 – 0x2031_1FFF	LPSRAM_BA	Low-Power SRAM	8 KB
0x6000_0000 – 0x602F_FFFF	EXTMEM_BA	External Parallel Memory (e.g., EBI)	3 MB
0x81F0_0000 – 0x81F7_FFFF	SRAM0_BA (Alias)	Non-Cacheable Alias of SRAM0	512 KB
0x81F8_0000 – 0x81FF_FFFF	SRAM1_BA (Alias)	Non-Cacheable Alias of SRAM1	512 KB
0x8200_0000 – 0x83FF_FFFF	SPIM0_BA	SPIM0 (HyperBUS) Memory Space	32 MB

1.4 Execution Results

After system power-on, the board-level initialization and setup of the OpenMV minimal Framebuffer are completed in sequence. The serial port outputs the starting address of the HyperRAM Framebuffer (SPIM0: 0x8200_0000) and the measured FPS value (when __PROFILE__ is enabled). When __USE_CCAP__ and __USE_DISPLAY__ are enabled, CCAP captures a single frame in RGB565/QVGA (320×240) format into HyperRAM, and the display engine scales the source frame by 2× and outputs it to the LCD within the rectangular area from (0,0) to (639,479). The screen is continuously updated in a “Capture → Display” single-loop structure to quickly validate the full data flow from CMOS → CCAP → HyperRAM → LCD. The execution environment is shown in Figure 1-2.

From a resource perspective, the single-frame Framebuffer occupies 320×240×2 = 153,600 bytes. This corresponds to a throughput of approximately 22.97 MB/s at 30 FPS, serving as a reference for subsequent panel timing or bus configuration tuning.

During actual display, if the panel resolution exceeds 640×480, the 2× upscaled image is placed within the specified rectangle (with the remaining area filled based on panel driver settings). If the panel resolution is equal to or smaller than 640×480, the display driver will fill or crop based on the rectangle boundaries. When __PROFILE__ is enabled, the calculated FPS is printed, allowing performance comparison across different clock rates, scaling factors, or bus configurations. The terminal screen is shown in Figure 1-3. The evaluation board result is shown in Figure 1-4.

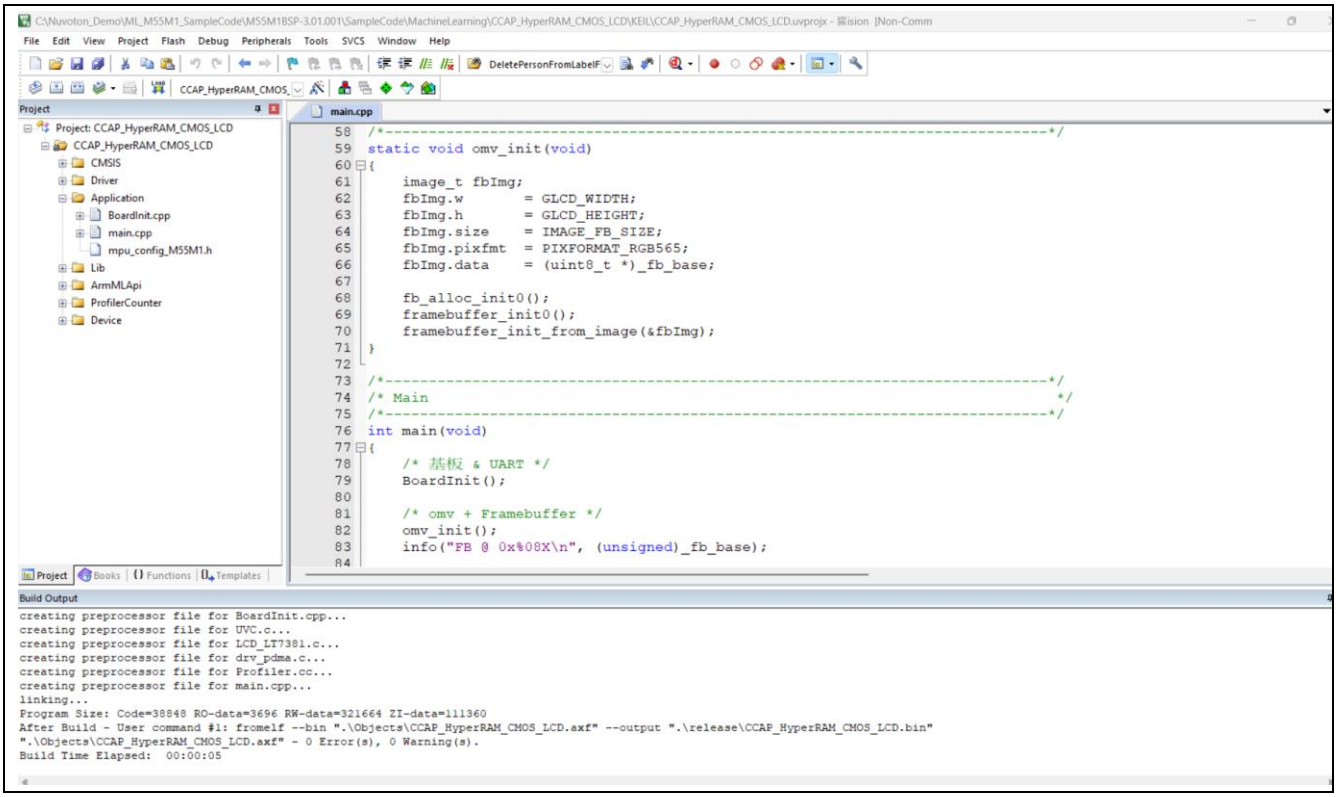

Figure 1-2 Select Compilation Options and Execute Compilation

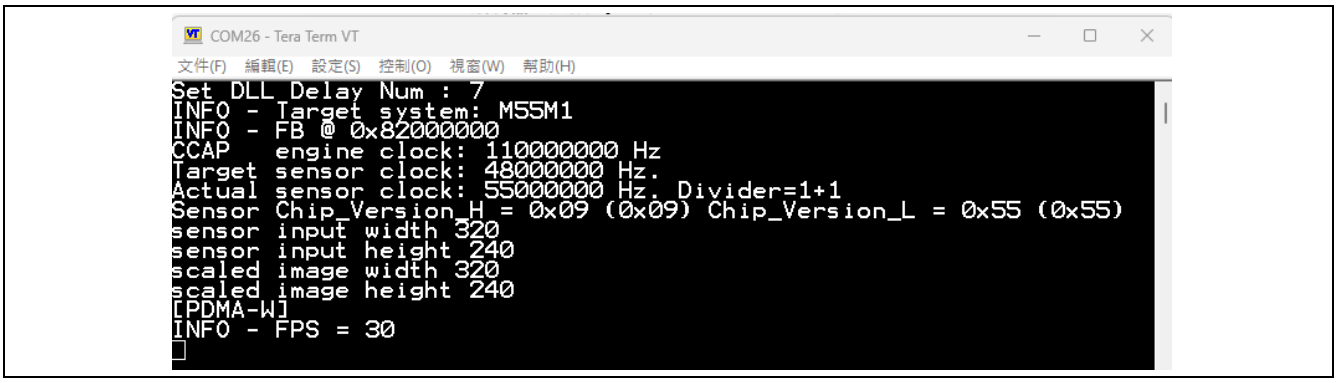

Figure 1-3 Execution Screen

Figure 1-4 Screen Display Image

1.5 HyperRAM Usage Notes

Set SPIM0_START / SPIM0_SIZE to your board's actual HyperRAM base/size, and keep UNINIT to avoid clearing large buffers at boot. The non-cacheable region is for buffers directly accessed by DMA, CCAP, LCD, preventing cache incoherency. For capacity safety, add a link-time check at the end of the scatter file to catch over-allocation during the build.

- Scatter File Configuration

The following maps .vram.hyper into HyperRAM (SPIM0: 0x8200_0000, 32 MB) with UNINIT, and provides a non-cacheable region for peripheral-shared buffers and an optional internal SRAM2 bucket for scratch data.

```
/* HyperRAM: place FrameBuffer here (UNINIT prevents zeroing on reset) */
```

```
SPIM_HYPERRAM 0x82000000 UNINIT 0x02000000 { .ANY(.vram.hyper) }

/* Non-cacheable region: for DMA/CCAP/LCD shared buffers, avoid cache issues */
SRAM_NONCACHEABLE 0x81F00000 0x00100000
{ .ANY(NonCacheable.Init, .bss.NonCacheable.ZeroInit) }

/* Internal SRAM2 (optional): small scratch, also UNINIT */
SRAM2 0x20200000 UNINIT 0x00050000 { .ANY(.bss.vram.data) }
```

- Image Sensor Frame-Size Alignment

Frame size (RGB565) = width × height × 2. If the sensor is not QVGA (320×240), adjust width/height/scale accordingly, and ensure .vram.hyper (HyperRAM) fits

OMV_FB_SIZE = GLCD_WIDTH * GLCD_HEIGHT * 2 + 1024; multiply by 2 if using double buffering.

- Link-Time Capacity Check

Add the following at the end of the scatter file to ensure the total usage of the Non-cacheable + HyperRAM regions stay within capacity. If you use macros, rewrite the RHS as (SPIM_ALIASE_SIZE + SPIM0_SIZE) for maintainability.

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

- Program Usage

Declare buffers in the **.vram.hyper** section so the linker maps them into HyperRAM, and expose the required global variables (do not rename them or mark as static) to use these symbols directly as source/destination buffers.

```
__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. Code Introduction

In the code (main.cpp), two main global buffers are declared using __attribute__((section(".vram.hyper"), aligned(32))):

- fb_array[OMV_FB_SIZE + OMV_FB_ALLOC_SIZE], serves as the storage space for the FrameBuffer and fb_alloc
- jpeg_array[OMV_JPEG_BUF_SIZE], serves as the temporary buffer for JPEG compression.

Global symbols required by OpenMV, _fb_base = fb_array and _jpeg_buf = jpeg_array, are provided to allow the system to correctly locate the FrameBuffer and JPEG Buffer.

In the Linker/Scatter file (*M55M1.scatter*), the **.vram.hyper** section is mapped to HyperRAM to ensure that the above buffers are allocated in external memory. This prevents them from occupying internal SRAM and meets OpenMV's requirement for large-capacity buffers in image processing.

- Key excerpt from the Linker/Scatter side (*M55M1.scatter*)

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

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

The **.vram.hyper** section (which contains **fb_array** and **jpeg_array**) is fully mapped to HyperRAM starting at **0x8200_0000**, and marked with **UNINIT** to avoid zero-initialization or data copying of large buffers during system startup.

- Non-cacheable region of SRAM (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)
```

```
}
```

Used for data that requires non-cacheable attributes (e.g., buffers shared with peripheral DMA). An empty section **SRAM_NONCACHEABLE_END** is defined to export boundary symbols for calculating the maximum capacity.

- SRAM2 (internal SRAM area)

```
#define SRAM2_START 0x20200000
#define SRAM2_SIZE  0x00050000  /* 320 KB */

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

This configuration places the **.bss.vram.data** section in the internal SRAM2 (address 0x2020_0000, size 320 KB) and marks it as an uninitialized (UNINIT) region used for temporary data or image buffering during runtime.

- Excess protection

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

During the link period, check that the total usage of the non-cache area and HyperRAM area does not exceed the memory capacity.

When using external memory (e.g., HyperRAM) to store large image or data buffers, you can place the buffer in a custom section with __attribute__((section)) and map it through the scatter file; to keep its content after power-up, add the UNINIT attribute. When the external memory is shared by DMA, display controllers, or camera sensors, it's recommended to mark the region as Non-Cacheable in the MPU or perform cache cleaning after each data transfer to avoid glitches display or data corruption.

2.1 Main Loop Process

After system startup in **main.cpp**, initialization is performed first, with optional execution of display initialization and sensor/CCAP initialization.
Once entering the main loop (LOOP), the process is as follows:

1. Trigger CCAP capture to store image data into **_fb_base** (the FrameBuffer allocated in HyperRAM).
2. Construct the output rectangle, ranging from (0,0) to (GLCD_WIDTH×2 - 1, GLCD_HEIGHT×2 - 1).
3. Call **Display_FillRect()** to draw the source FrameBuffer data onto the LCD with 2× scaling.

4. If __**PROFILE**__ is defined, use SysTick or PMU to measure the image processing time and calculate the FPS.

After the process is completed, the loop repeats continuously to perform image capture and display.

```
Initialization → Display Initialization → Sensor/CCAP Initialization
LOOP:
  1) Trigger CCAP capture to _fb_base (HyperRAM)
  2) Construct output rectangle: (0,0) ~ (GLCD_WIDTH×2 - 1, GLCD_HEIGHT×2 - 1)
  3) Display_FillRect (source = framebuffer, target = LCD, scale = 2×)
  4) If __PROFILE__ is enabled: use SysTick/PMU to record timing and calculate FPS
END
```

2.2 Scatter File and Assigning Data to Specific Memory Regions

- Named Sections and Scatter Mapping

In this example, specific data sections are explicitly named and mapped to designated memory regions through the scatter file.

For example:

```
__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];
```

The above code places the variables fb_array and jpeg_array into a custom section named **.vram.hyper**.

In the scatter file, this section is assigned to the SPIM0 (HyperRAM) region starting at address 0x8200_0000 (SPIM_HYPERRAM) and marked as UNINIT, preventing the system from clearing this large buffer during startup and thereby reducing boot time.

- Complete Code

```
/**************************************************************************//**
 * @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. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M55M1_Series_BSP_CMSIS_V3.01.003
- IDE version
  - Keil uVision 5.40

3.2 Hardware Requirements

- Circuit components
  - NuMaker-X-M55M1D V1.0
  - HM1055 CMOS Sensor
  - NuMaker-TFT-LCD5 V1.2
- Circuit diagram
  - Assemble the NuMaker-X-M55M1D V1.0, NuMaker-TFT-LCD adapter board, and CMOS module.
  - Use a USB Type-C to Type-A cable to connect the NuMaker-X-M55M1D evaluation board to the PC for power and communication.

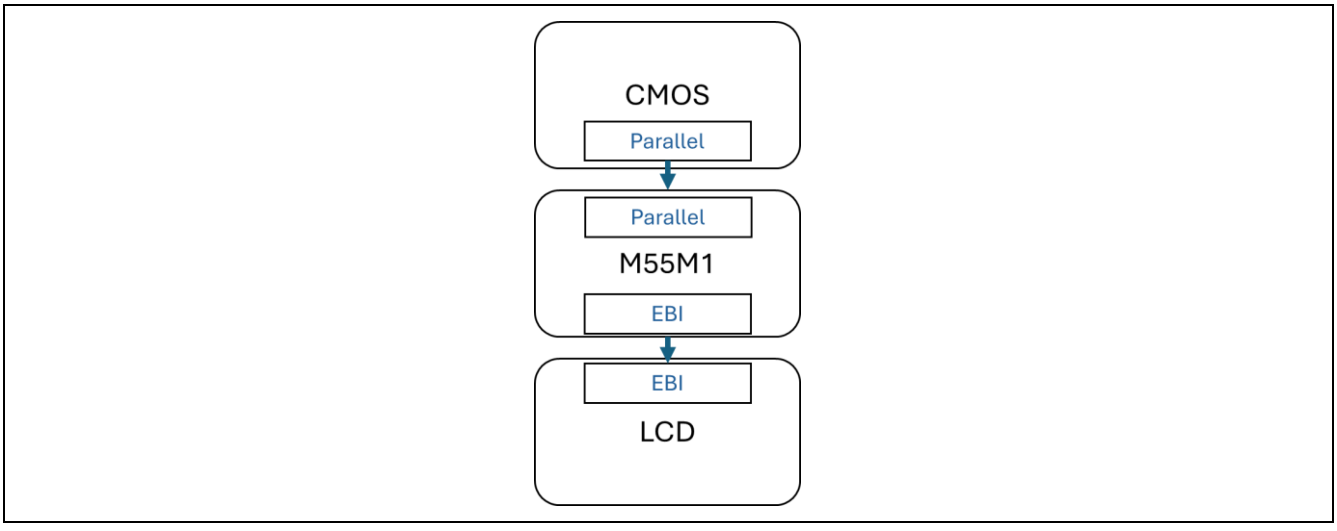

Figure 3-1 Circuit Diagram

4. Directory Structure

The directory structure is shown below.

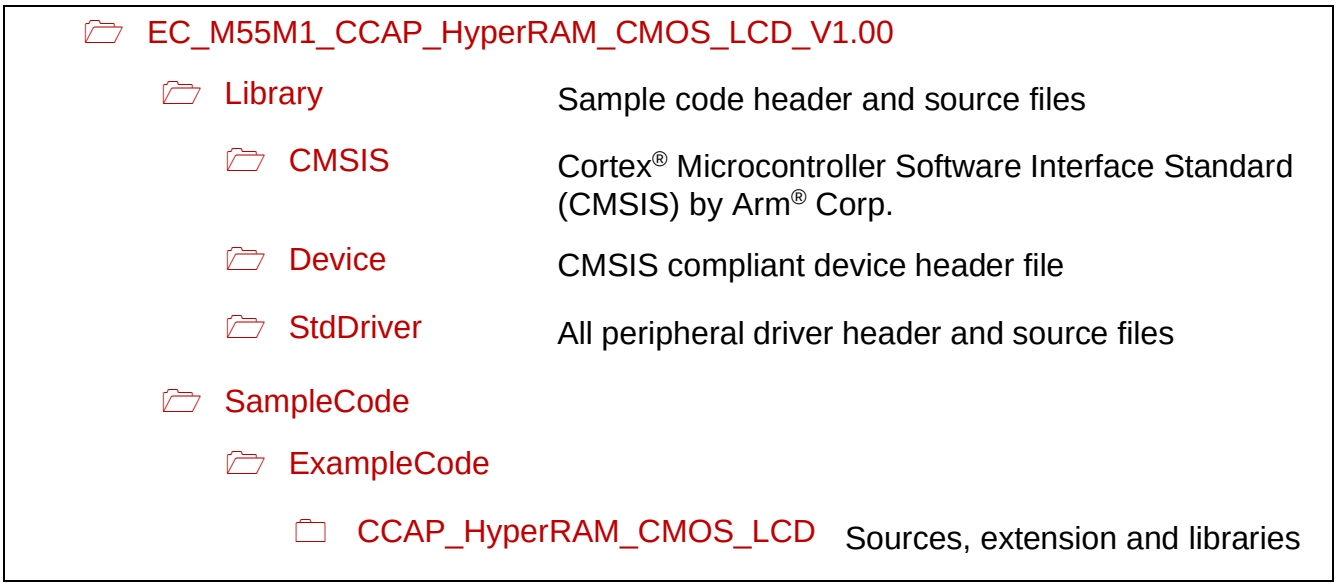

Figure 4-1 Directory Structure

5. Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *CCAP_HyperRAM_CMOS_LCD.uvprojx*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run

6. Revision History

Date	Revision	Description
2025.08.21	1.00	Initial version

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.