**DCMI Use Cases**

This report details the following typical examples of DCMI use:

• capture and display of RGB data captured in RGB565 format with QVGA (320x240) resolution, stored in the SDRAM, and displayed on the LCD‑TFT

• capture of YCbCr data captured in YCbCr format with QVGA (320x240) resolution and stored in the SDRAM

• capture of Y-only data DCMI configured to receive Y-only data to be stored in the SDRAM

**Hardware used:**

**STM32U575I-EV Board:**

The STM32U575I-EV board is a powerful development platform designed for embedded applications. It is based on the STM32U575 microcontroller, which is a high-performance device with an Arm Cortex-M33 core and a rich set of peripherals. The board is a great choice for projects that require a Digital Camera Interface (DCMI) to interface with image sensors.

The choice of such board is due to having such important features like:

1. DCMI interface: the STM32U575I-EV board features a dedicated DCMI interface that allows it to interface with a wide range of image sensors. This interface supports various image formats, including YUV, RGB, and JPEG.
2. High-performance microcontroller: The STM32U575 microcontroller is a high-performance device with an Arm Cortex-M33 core that can run at up to 480 MHz It also has a large amount of flash memory and RAM, which makes it ideal for image processing applications.
3. Rich set of peripherals: The board has a rich set of peripherals, including USB, Ethernet, CAN, and UART interfaces. These peripherals can be used to communicate with other devices or to control external components.
4. On-board sensors: The board also features a range of on-board sensors, including an accelerometer, a gyroscope, and a magnetometer. These sensors can be used to detect motion and orientation, which can be useful in image stabilization applications.
5. Expansion options: The board has a range of expansion options, including Arduino and STMod+ connectors. These connectors allow you to add additional functionality to the board, such as wireless connectivity or additional sensors.
6. Power management: The STM32U575I-EV board features advanced power management options, including low-power modes and dynamic voltage scaling. These features help to optimize power consumption and extend battery life in portable applications.

A close-up of a computer chip

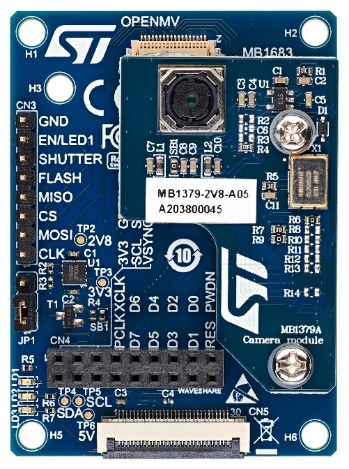
Description automatically generated

**MB1379A Camera Module:**

The MB1379A camera module is an excellent choice for video processing applications that require high performance and low power consumption. Its composite video input, sync separation capabilities, STM32 microcontroller interface, low power consumption, and small form factor make it an ideal choice for video processing applications that require high performance and low power consumption.

The MB1379A camera module is a highly integrated device that includes several features that make it ideal for video processing applications. Some of the key features of the MB1379A camera module include:

1. Composite video input: The MB1379A camera module includes a composite video input that allows it to receive video signals from a variety of sources, including cameras, VCRs, and DVD players.
2. Sync separation: The MB1379A camera module is designed to extract horizontal and vertical synchronization signals from composite video signals. These signals are used to synchronize the display of video images on a monitor or TV screen.
3. STM32 microcontroller interface: The MB1379A camera module is designed to work with the STM32 microcontroller family. It includes an interface that allows it to communicate with the microcontroller and transfer video data.
4. Low power consumption: The MB1379A camera module is designed to operate on low power, making it ideal for battery-powered applications.
5. Small form factor: The MB1379A camera module is available in a small form factor package, which makes it easy to integrate into a variety of video processing applications.



This schematic shows the camera module connector in the board:

A computer screen shot of a diagram

Description automatically generated

**MB989C LCD Module:**

The MB989C is a 3.5-inch TFT LCD display module that can be interfaced with an STM32 microcontroller. This display module features a resolution of 320x240 pixels and supports 16-bit color depth. It also includes a resistive touch panel for user input.

It uses a 16-bit parallel interface to communicate with the STM32 microcontroller. This interface allows for fast data transfer and enables the display to update quickly.



This schematic shows the LCD module connector in the board:

A computer diagram of a circuit board

Description automatically generated

**RGB Application**

Step 1: open STM32CubeMX and select “ ACCESS TO BOARD SELECTOR“

A screenshot of a computer

Description automatically generated

Step 2 : Select the correct board that you are working on. For this example we are working with the “STM32U575I-EV”

A screenshot of a computer

Description automatically generated

Step 3: Start the project and do not initialize any peripherals.  
  
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 4: Got Multimedia->DCMI then select the option :”Slave 8 bits Embedded Synchro”

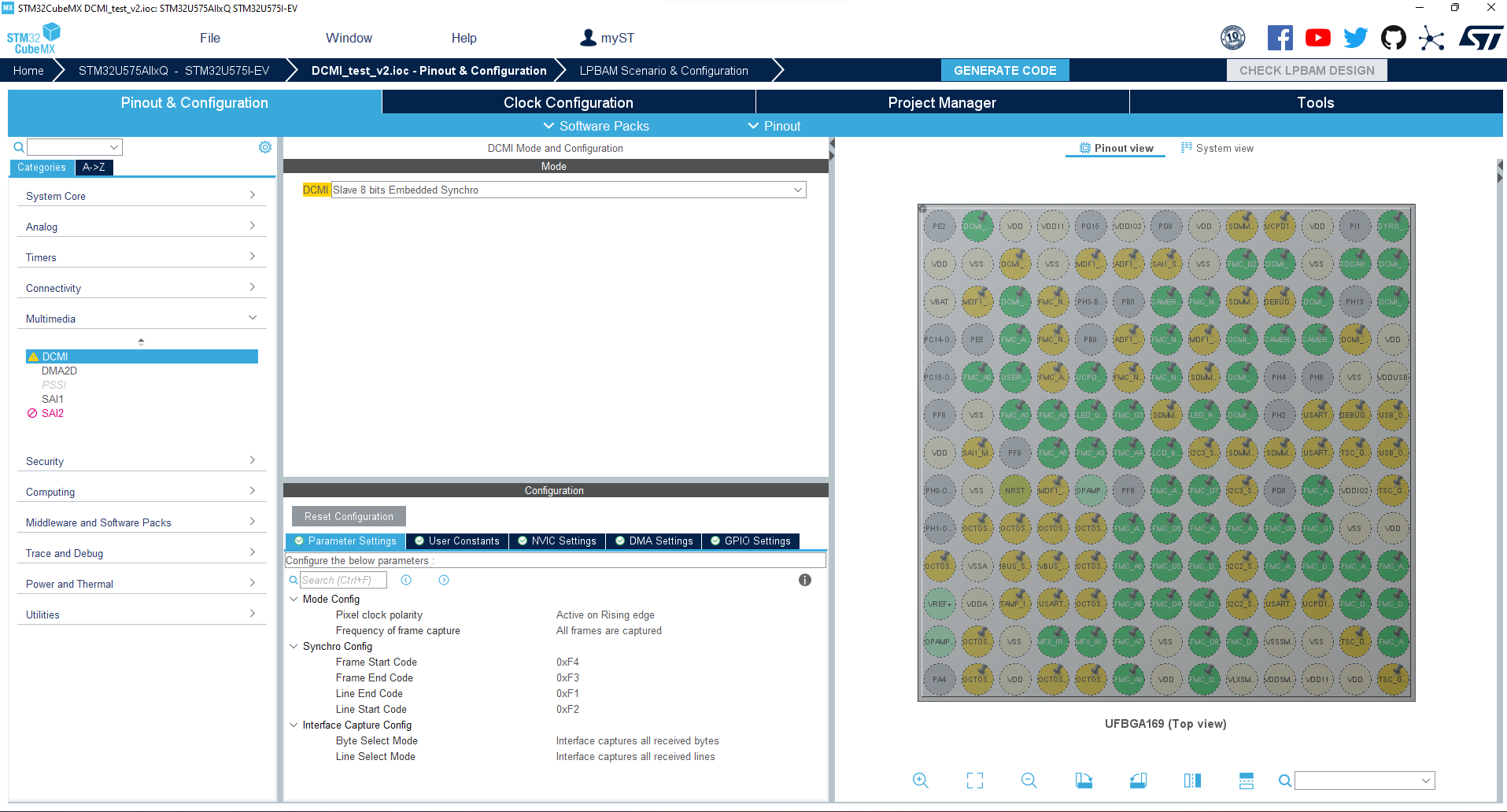
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 5: In the DCMI window select “Parameter Settings” and define the values of the necessary variables like the following configuration



Step 6: In the DCMI window select “DMA Settings” and click on the button “Go to GPDMA1”

A screenshot of a computer

Description automatically generated

Step 7: In the GPDMA1 window activate any channel from 12 to 15 to assure best performance due to image coding is a 2D operation by choosing the option : “Linked\_List Mode” because we are dealing with images that are represented by a 2D-array so 2D addressing must be used . In this example we are going to work with channel 12.

A screenshot of a computer

Description automatically generated

Step 8: Change the Execution Mode to “Circular” and the allocated Port For Transfer to “Port 1” as Port1 responsible of data transfert from GPDMA to AHB2 (refer to data sheet) :

A screenshot of a computer

Description automatically generated

Step 9: In the NVIC Settings enable the global interrupt

A screen shot of a computer

Description automatically generated

Step 10: Activate the ICACHE by choosing the mode “1-way” to boost the performance of the DCMI.

A screenshot of a computer

Description automatically generated

Step 11: In SYS window change the timebase source to SysTick to boost the performance ( APB2 peripheral could attain 160 MHz )

A screenshot of a computer

Description automatically generated

Step 12: In the window Utilities->LINKEDLIST add a new linked list and configure it as the figure below

A screenshot of a computer

Description automatically generated

Step 13: Edit the existing Node to the following configuration ( we use Port zero as it is recommended by the application note : [How to use the GPDMA for STM32U5 Series microcontrollers - Application note](https://www.st.com/resource/en/application_note/an5593-how-to-use-the-gpdma-for-stm32u575585-microcontrollers-stmicroelectronics.pdf) page 13 ) :

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 14: Add an other node with the same exact configuration

A screenshot of a computer

Description automatically generated

Step 15: Enable the “I2C2” by uncommenting the following line in “Core/inc/stm32u5xx\_hal\_conf.h”:

**#define** **HAL\_I2C\_MODULE\_ENABLED**

Step 16: Enable the “SRAM” as follows(note that you can just to allocate the maximum address size and not 24bits exactly) :A screenshot of a computer

Description automatically generatedStep 17: Edit the NVIC table to this configuration :   
A screenshot of a computer

Description automatically generated

Step 18: Edit the Clock Configuration as follows:  
  
A computer screen shot of a computer program

Description automatically generated

Step 19: Clear all the unused pins and generate the project we used for this use case the STM32CubeIDE as a preferred toolchain :

A screenshot of a computer

Description automatically generated

Step 20: Add to your project the necessary packages to your project directory:

Step 20.1: Clone this Github repository <https://github.com/STMicroelectronics/STM32CubeU5.git>

Step 20.2: Create a folder called “Utilities” and copy from the repository the folder “Fonts” and “lcd” from the “STM32Cube\_FW\_U5/Utilities” folder of the repository

A screenshot of a computer

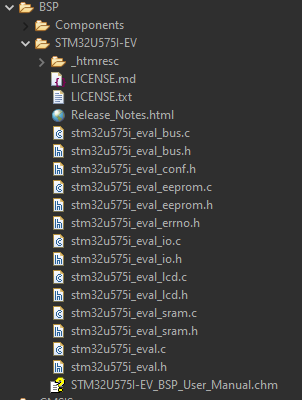
Description automatically generated

Step 20.3: In folder “Drivers” of your project create a new folder called “BSP” then transfer from the repository folder “STM32Cube\_FW\_U5\Drivers\BSP” the following driver folders:

A screenshot of a computer

Description automatically generated

Step 20.4: In folder “Drivers/BSP/STM32U575I-EV” of your project make sure to only have the necessary files to assure the correct building of the project . The files are as follow :



Step 21: Add the new folders of the project to the include paths of the project simply by right clicking the project on the STM32CubeIDE and then going to properties->C/C++ General->Paths and Symbols then adding the following paths

A screenshot of a computer

Description automatically generated

Step 22: In the file “Drivers/BSP/ STM32U575I-EV/ stm32u575i\_eval\_conf.h” edit the following variable “USE\_BSP\_IO\_CLASS” from 0 to 1( depending on the number of layers you are going to use in the LCD for this example 1 layer is sufficient ) :

**#define** USE\_BSP\_IO\_CLASS 1U

Step 23: In file “Core/Inc/main.h” add the following lines in the Private includes section:   
  
/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

**#include** "linked\_list.h"

**#include** "stm32u575i\_eval.h"

**#include** "stm32u575i\_eval\_io.h"

**#include** "stm32u575i\_eval\_lcd.h"

**#include** "stm32u575i\_eval\_bus.h"

**#include** "stm32\_lcd.h"

**#include** "../Components/ov5640/ov5640.h"

/\* USER CODE END Includes \*/

Step 24: In file “Core/Inc/main.h” add the following lines in the Exported constants section:

/\* Exported constants --------------------------------------------------------\*/

/\* USER CODE BEGIN EC \*/

**#define** FRAME\_WIDTH 320

**#define** FRAME\_HEIGHT 240

**#define** FMC\_ADDRESS FMC\_BANK1\_4

**#define** REGION\_SIZE 0x0CFFFFFF

**#define** XSDN\_PIN GPIO\_PIN\_3

**#define** XSDN\_PORT GPIOI

**#define** RSTI\_PIN GPIO\_PIN\_2

**#define** RSTI\_PORT GPIOI

**#define** CAM\_PLUG\_PIN GPIO\_PIN\_10

**#define** CAM\_PLUG\_PORT GPIOG

**#define** CAMERA\_OV5640\_ADDRESS 0x78U

**#define** CAMERA\_R320x240 1U /\* QVGA Resolution \*/

/\* Camera Pixel Format \*/

**#define** CAMERA\_PF\_RGB565 0U /\* Pixel Format RGB565 \*/

/\* USER CODE END EC \*/

Step 25: In file “Core/Inc/main.h” edit the Private defines section to the following to assure a more understandable project structure:

/\* Private defines -----------------------------------------------------------\*/

**#define** USER\_BUTTON\_Pin GPIO\_PIN\_13

**#define** USER\_BUTTON\_GPIO\_Port GPIOC

**#define** USER\_BUTTON\_EXTI\_IRQn EXTI13\_IRQn

/\* USER CODE BEGIN Private defines \*/

Step 26: In file “Core/Src/main.c” add the following lines in the “USER CODE BEGIN PV” section:

/\* USER CODE BEGIN PV \*/

\_\_IO uint32\_t frame\_suspended = 0;

\_\_IO uint32\_t frame\_captured = 0;

uint32\_t FRAME\_BUFFER\_SIZE = (FRAME\_WIDTH\*FRAME\_HEIGHT\*2)/4;

uint32\_t CAMERA\_FRAME\_BUFFER[(FRAME\_WIDTH\*FRAME\_HEIGHT\*2)/4];

OV5640\_SyncCodes\_t pSyncroCodes;

DCMI\_SyncUnmaskTypeDef SyncUnmask;

**extern** DMA\_QListTypeDef DCMIQueue;

\_\_IO FlagStatus UserButtonPressed = *RESET*;

/\* USER CODE END PV \*/

Step 27: In file “Core/Src/main.c” add the following lines in the USER CODE BEGIN PFP section:

/\* USER CODE BEGIN PFP \*/

**static** **void** **MPU\_Config**(**void**);

**static** **void** **Example\_Description**(**void**);

**static** uint32\_t **OV5640\_Config**(**void**);

/\* USER CODE END PFP \*/

Step 28: In file “Core/Src/main.c” add the necessary code to the main function

Step 28.1: In file “Core/Src/main.c” add the following code in the “USER CODE BEGIN SysInit” section :

/\* USER CODE BEGIN SysInit \*/

/\* Configure the MPU attributes for SRAM \*/

MPU\_Config();

/\*##-1- LEDs and User Button initialization #################################################\*/

BSP\_LED\_Init(*LED5*);

BSP\_LED\_Init(*LED6*);

/\*##-2- LCD configuration #################################################\*/

/\* LCD initialization and display enable\*/

BSP\_LCD\_Init(0,LCD\_ORIENTATION\_LANDSCAPE);

BSP\_LCD\_DisplayOn(0);

Example\_Description();

/\* USER CODE END SysInit \*/

Step 28.2: In file “Core/Src/main.c” add the following code to the “USER CODE 2” section:

/\* USER CODE BEGIN 2 \*/

MX\_DCMIQueue\_Config();

HAL\_DMAEx\_List\_LinkQ(&handle\_GPDMA1\_Channel12, &DCMIQueue);

\_\_HAL\_LINKDMA(&hdcmi, DMA\_Handle, handle\_GPDMA1\_Channel12);

/\*##-4- Camera Initialization ############################\*/

/\* Initialize the Camera in QVGA mode \*/

**if**(OV5640\_Config() != 0)

{

Error\_Handler();

}

/\* Wait for User button press \*/

**while** (UserButtonPressed != *SET*);

/\* Reset for next operation \*/

UserButtonPressed = *RESET*;

/\*

LCD size is 320 x 240 and format is RGB565 i.e. 16 bpp or 2 bytes/pixel.

The LCD frame size is therefore 320 \* 240 half-words of (320\*240\*2) 8-bit long bytes .

Since the unit of the DMA associated to DCMI IP is with word , the last parameter of

HAL\_DCMI\_Start\_DMA is set to:

FRAME\_BUFFER\_SIZE = ((FRAME\_WIDTH\*FRAME\_HEIGHT)\*2)/4, that is (320 \* 240 \* 2)/4

\*/

HAL\_DCMI\_Start\_DMA(&hdcmi, DCMI\_MODE\_CONTINUOUS, (uint32\_t)CAMERA\_FRAME\_BUFFER, FRAME\_BUFFER\_SIZE);

/\* USER CODE END 2 \*/

Step 28.3: In file “Core/Src/main.c” add the following code to the Infinite loop section :

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* Display the continuous grap \*/

UTIL\_LCD\_SetFuncDriver(&LCD\_Driver);

BSP\_LCD\_FillRGBRect(0,0,0,(uint8\_t \*)CAMERA\_FRAME\_BUFFER,FRAME\_WIDTH,FRAME\_HEIGHT);

UTIL\_LCD\_SetBackColor(UTIL\_LCD\_COLOR\_GRAY);

UTIL\_LCD\_SetTextColor(UTIL\_LCD\_COLOR\_RED);

UTIL\_LCD\_DisplayStringAt(0, 5, (uint8\_t \*)"RGB Format", *CENTER\_MODE*);

/\* for any Press Check whether the Continuous capture should be suspended or resumed \*/

**while** (UserButtonPressed != *RESET*)

{

**if**(frame\_suspended == 1)

{

**if**(HAL\_DCMI\_Resume(&hdcmi) != *HAL\_OK*)

{

Error\_Handler();

}

frame\_suspended=0;

}

**else**

{

**if**(HAL\_DCMI\_Suspend(&hdcmi) != *HAL\_OK*)

{

Error\_Handler();

}

frame\_suspended++;

}

UserButtonPressed = *RESET*;

}

/\* Check each time for new frame\*/

**if**(frame\_captured != 0)

{

frame\_captured = 0;

BSP\_LED\_Toggle(*LED5*);

}

**else**

{

BSP\_LED\_Off(*LED5*);

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

Step 29: In file “Core/Src/main.c” add the following functions in the USER CODE BEGIN 4 section :

/\*\*

\* @brief Configure the MPU attributes.

\* @note The Base Address is External SRAM

\* @param None

\* @retval None

\*/

**static** **void** **MPU\_Config**(**void**)

{

MPU\_Attributes\_InitTypeDef attr;

MPU\_Region\_InitTypeDef region;

/\* Disable MPU before perloading and config update \*/

HAL\_MPU\_Disable();

/\* Define cacheable memory via MPU \*/

attr.Number = MPU\_ATTRIBUTES\_NUMBER0;

attr.Attributes = 0 ;

HAL\_MPU\_ConfigMemoryAttributes(&attr);

/\* BaseAddress-LimitAddress configuration \*/

region.Enable = MPU\_REGION\_ENABLE;

region.Number = MPU\_REGION\_NUMBER0;

region.AttributesIndex = MPU\_ATTRIBUTES\_NUMBER0;

region.BaseAddress = FMC\_ADDRESS;

region.LimitAddress = FMC\_ADDRESS + REGION\_SIZE - 1;

region.AccessPermission = MPU\_REGION\_ALL\_RW;

region.DisableExec = MPU\_INSTRUCTION\_ACCESS\_ENABLE;

region.IsShareable = MPU\_ACCESS\_NOT\_SHAREABLE;

HAL\_MPU\_ConfigRegion(&region);

/\* Enable the MPU \*/

HAL\_MPU\_Enable(MPU\_PRIVILEGED\_DEFAULT);

}

/\*\*

\* @brief Camera Frame Event callback.

\*/

**void** **HAL\_DCMI\_FrameEventCallback**(DCMI\_HandleTypeDef \*hdcmi)

{

frame\_captured++;

}

/\*\*

\* @brief Configure the Camera Module in continuous

\* in Embedded Synchronization mode :

\* Use embedded codes (SynchroCodes)

\* for Synchronization the same

\* set to the DCMI IP

\* @param None

\* @retval 0 OK

\* !=0 KO

\*/

**static** uint32\_t **OV5640\_Config**(**void**)

{

OV5640\_IO\_t IOCtx;

uint32\_t id;

uint32\_t status = 0;

**static** OV5640\_Object\_t OV5640Obj;

GPIO\_InitTypeDef gpio\_init\_structure;

/\* Configure the Camera driver \*/

IOCtx.Address = CAMERA\_OV5640\_ADDRESS;

IOCtx.Init = BSP\_I2C2\_Init;

IOCtx.DeInit = BSP\_I2C2\_DeInit;

IOCtx.ReadReg = BSP\_I2C2\_ReadReg16;

IOCtx.WriteReg = BSP\_I2C2\_WriteReg16;

IOCtx.GetTick = BSP\_GetTick;

/\* Reset sensor \*/

gpio\_init\_structure.Pin = RSTI\_PIN | XSDN\_PIN;

gpio\_init\_structure.Pull = GPIO\_NOPULL;

gpio\_init\_structure.Mode = GPIO\_MODE\_OUTPUT\_PP;

gpio\_init\_structure.Alternate = GPIO\_AF10\_DCMI;

HAL\_GPIO\_Init(GPIOI, &gpio\_init\_structure);

/\* Camera sensor RESET sequence \*/

/\* Assert the camera STANDBY pin (active high) \*/

HAL\_GPIO\_WritePin(XSDN\_PORT, XSDN\_PIN, *GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(RSTI\_PORT, RSTI\_PIN, *GPIO\_PIN\_RESET*);

HAL\_Delay(100); /\* RST and XSDN signals asserted during 100ms \*/

/\* De-assert the camera STANDBY pin (active high) \*/

HAL\_GPIO\_WritePin(XSDN\_PORT, XSDN\_PIN, *GPIO\_PIN\_RESET*);

HAL\_Delay(3); /\* RST de-asserted and XSDN asserted during 3ms \*/

/\* De-assert the camera RSTI pin (active low) \*/

HAL\_GPIO\_WritePin(RSTI\_PORT, RSTI\_PIN, *GPIO\_PIN\_SET*);

HAL\_Delay(20); /\* RST de-asserted during 20ms \*/

/\* Verify that the Camera module is plugged in \*/

gpio\_init\_structure.Pin = CAM\_PLUG\_PIN;

gpio\_init\_structure.Pull = GPIO\_PULLUP;

gpio\_init\_structure.Mode = GPIO\_MODE\_INPUT;

HAL\_GPIO\_Init(CAM\_PLUG\_PORT, &gpio\_init\_structure);

**if**(((uint32\_t)HAL\_GPIO\_ReadPin(CAM\_PLUG\_PORT, CAM\_PLUG\_PIN) & CAM\_PLUG\_PIN) == CAM\_PLUG\_PIN)

{

status = 1;

}

/\*\*/

**if**(OV5640\_RegisterBusIO (&OV5640Obj, &IOCtx) != OV5640\_OK)

{

status = OV5640\_ERROR;

}

/\*Read ID\*/

**if**(OV5640\_ReadID(&OV5640Obj, &id) != OV5640\_OK)

{

status = OV5640\_ERROR;

}

**if**(id == OV5640\_ID)

{

/\*Initialize\*/

**if**(OV5640\_Init(&OV5640Obj, CAMERA\_R320x240, CAMERA\_PF\_RGB565) != OV5640\_OK)

{

status = OV5640\_ERROR;

}

/\* Enable and set SynchroCodes CCIR to the camera Module \*/

pSyncroCodes.FrameStartCode = 0xF4;

pSyncroCodes.FrameEndCode = 0xF3;

pSyncroCodes.LineStartCode = 0xF2;

pSyncroCodes.LineEndCode = 0xF1;

**if**(OV5640\_EmbeddedSynchroConfig(&OV5640Obj,&pSyncroCodes) !=OV5640\_OK)

{

status = OV5640\_ERROR;

}

HAL\_Delay(100);

/\* Disable Flip and mirror effect \*/

**if**(OV5640\_MirrorFlipConfig(&OV5640Obj, OV5640\_MIRROR\_FLIP\_NONE)!= OV5640\_OK)

{

status = OV5640\_ERROR;

}

HAL\_Delay(100);

}

**else**

{

status = OV5640\_ERROR;

}

**return** status;

}

/\*\*

\* @brief Display main example messages

\* @param None

\* @retval None

\*/

**static** **void** **Example\_Description**(**void**)

{

/\* Set GUI functions \*/

UTIL\_LCD\_SetFuncDriver(&LCD\_Driver);

/\* Clear the LCD \*/

UTIL\_LCD\_Clear(UTIL\_LCD\_COLOR\_WHITE);

/\* Set font \*/

UTIL\_LCD\_SetFont(&Font16);

/\* Clear the LCD \*/

UTIL\_LCD\_Clear(UTIL\_LCD\_COLOR\_WHITE);

/\* Set the LCD Text Color \*/

UTIL\_LCD\_SetTextColor(UTIL\_LCD\_COLOR\_WHITE);

UTIL\_LCD\_SetBackColor(UTIL\_LCD\_COLOR\_DARKBLUE);

/\* Display LCD messages \*/

UTIL\_LCD\_FillRect(0, 5, 320, 80, UTIL\_LCD\_COLOR\_DARKBLUE);

UTIL\_LCD\_DisplayStringAt(0, 5, (uint8\_t \*)" Continuous Capture in ", *CENTER\_MODE*);

UTIL\_LCD\_DisplayStringAt(0, 20, (uint8\_t \*)"Embedded Synchronization Mode", *CENTER\_MODE*);

UTIL\_LCD\_DisplayStringAt(0, 35, (uint8\_t \*)" and Suspend-Resume ", *CENTER\_MODE*);

UTIL\_LCD\_DisplayStringAt(0, 50, (uint8\_t \*)" example: RGB Format ", *CENTER\_MODE*);

UTIL\_LCD\_SetFont(&Font12);

UTIL\_LCD\_FillRect(0, 150, 320, 50, UTIL\_LCD\_COLOR\_DARKBLUE);

UTIL\_LCD\_SetTextColor(UTIL\_LCD\_COLOR\_WHITE);

UTIL\_LCD\_SetBackColor(UTIL\_LCD\_COLOR\_DARKBLUE);

UTIL\_LCD\_DisplayStringAt(0, 150, (uint8\_t \*)"Press USER push-button", *CENTER\_MODE*);

UTIL\_LCD\_DisplayStringAt(0, 165, (uint8\_t \*)"to start:", *CENTER\_MODE*);

UTIL\_LCD\_DisplayStringAt(0, 180, (uint8\_t \*)"then to Suspend/Resume ", *CENTER\_MODE*);

UTIL\_LCD\_SetBackColor(UTIL\_LCD\_COLOR\_GRAY);

UTIL\_LCD\_SetTextColor(UTIL\_LCD\_COLOR\_ORANGE);

UTIL\_LCD\_DisplayStringAt(0, 210, (uint8\_t \*)"CREATED BY : JAAFER HOSNI ", *CENTER\_MODE*);

}

/\*\*

\* @brief EXTI line rising detection callback.

\* @param GPIO\_Pin: Specifies the port pin connected to corresponding EXTI line.

\* @retval None

\*/

**void** **HAL\_GPIO\_EXTI\_Rising\_Callback**(uint16\_t GPIO\_Pin)

{

/\* Prevent unused argument(s) compilation warning \*/

UNUSED(GPIO\_Pin);

UserButtonPressed = *SET*;

}

Step 30: In file “Core/Src/main.c” add the following code to the **Error\_Handler** function:

**void** **Error\_Handler**(**void**)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

**while** (1)

{

BSP\_LED\_Toggle(*LED6*);

HAL\_Delay(100);

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

Step 31: In file “Core->Src->main.c” add the following code to the assert\_failed function:

**void** assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* Infinite loop \*/

**while** (1)

{

}

/\* USER CODE END 6 \*/

}

Step 32: Build the project and enjoy the demonstration.

**YCrCb Application**

To do an YCrCb application you need to do the same steps from 1 to 28.2 of the RGB appplication

Then add to the end of “User Code 2” section of the main function in file “Core/Src/main.c” the following code:

**uint32\_t** pixelCount = FRAME\_BUFFER\_SIZE / 2; // Assuming RGB565 pixel format

**uint16\_t** \*rgbData = (**uint16\_t** \*)CAMERA\_FRAME\_BUFFER;

**uint16\_t** \*ycrcbData = (**uint16\_t** \*)CAMERA\_FRAME\_BUFFER; // Reuse the same buffer for YCrCb data

/\* Perform RGB to YCrCb conversion \*/

**for** (**uint32\_t** i = 0; i < pixelCount; ++i)

{

**uint16\_t** rgbPixel = rgbData[i];

**uint8\_t** r = (rgbPixel >> 11) & 0x1F;

**uint8\_t** g = (rgbPixel >> 5) & 0x3F;

**uint8\_t** b = rgbPixel & 0x1F;

**uint8\_t** y = (**uint8\_t**)(0.299 \* r + 0.587 \* g + 0.114 \* b);

**uint8\_t** cr = (**uint8\_t**)(128 + 0.564 \* (b - y));

**uint8\_t** cb = (**uint8\_t**)(128 + 0.713 \* (r - y));

ycrcbData[i] = (y << 8) | (cr << 3) | (cb >> 3);

}

In file “Core/Src/main.c” add the following code to the “Infinite loop” section of the main function :

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* Display the continuous grap \*/

**UTIL\_LCD\_SetFuncDriver**(&LCD\_Driver);

**BSP\_LCD\_FillRGBRect**(0,0,0,(**uint8\_t** \*)ycrcbData,FRAME\_WIDTH,FRAME\_HEIGHT);

**UTIL\_LCD\_SetBackColor**(UTIL\_LCD\_COLOR\_GRAY);

**UTIL\_LCD\_SetTextColor**(UTIL\_LCD\_COLOR\_RED);

**UTIL\_LCD\_DisplayStringAt**(0, 5, (**uint8\_t** \*)"YCrCb Format", CENTER\_MODE);

/\* for any Press Check whether the Continuous capture should be suspended or resumed \*/

**while** (UserButtonPressed != *RESET*)

{

**if**(frame\_suspended == 1)

{

**if**(**HAL\_DCMI\_Resume**(&hdcmi) != *HAL\_OK*)

{

**Error\_Handler**();

}

frame\_suspended=0;

}

**else**

{

**if**(**HAL\_DCMI\_Suspend**(&hdcmi) != *HAL\_OK*)

{

**Error\_Handler**();

}

frame\_suspended++;

}

UserButtonPressed = *RESET*;

}

/\* Check each time for new frame\*/

**if**(frame\_captured != 0)

{

frame\_captured = 0;

**BSP\_LED\_Toggle**(*LED5*);

}

**else**

{

**BSP\_LED\_Off**(*LED5*);

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

Next do all the remaining steps from 29 to 32 of the RGB application

**Y-Only Application**

To do an Y-Only application you need to do the same steps from 1 to 28.2 of the RGB application

Then add to the end of “User Code 2” section of the main function in file “Core/Src/main.c” the following code:

**uint16\_t** \*rgbData = (**uint16\_t** \*)CAMERA\_FRAME\_BUFFER;

**uint8\_t** \*yData = (**uint8\_t** \*)CAMERA\_FRAME\_BUFFER; // Reuse the same buffer for YCrCb data

**for** (**uint16\_t** y = 0; y < FRAME\_HEIGHT; y++)

{

**for** (**uint16\_t** x = 0; x < FRAME\_WIDTH; x++)

{

// Calculate the index for the current pixel in the RGB image data array

**uint32\_t** pixelIndex = (y \* FRAME\_WIDTH + x) \* 3;

// Get the RGB components of the pixel

**uint8\_t** r = rgbData[pixelIndex];

**uint8\_t** g = rgbData[pixelIndex + 1];

**uint8\_t** b = rgbData[pixelIndex + 2];

// Convert RGB to YCbCr

**uint8\_t** yCbCr[3];

yCbCr[0] = (**uint8\_t**)(0.299 \* r + 0.587 \* g + 0.114 \* b);

yCbCr[1] = (**uint8\_t**)(-0.1687 \* r - 0.3313 \* g + 0.5 \* b + 128);

yCbCr[2] = (**uint8\_t**)(0.5 \* r - 0.4187 \* g - 0.0813 \* b + 128);

// Extract the Y component

**uint8\_t** yValue = yCbCr[0];

// Store the Y component in the grayscale image data array

**uint32\_t** grayIndex = y \* FRAME\_WIDTH + x;

yData[grayIndex] = yValue;

}

}

In file “Core/Src/main.c” add the following code to the “Infinite loop” section of the main function :

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

/\* Display the continuous grap \*/

**UTIL\_LCD\_SetFuncDriver**(&LCD\_Driver);

**for** (**uint16\_t** y = 0; y < FRAME\_HEIGHT; y++)

{

**for** (**uint16\_t** x = 0; x < FRAME\_WIDTH; x++)

{

// Calculate the index for the current pixel in the grayscale image data array

**uint32\_t** pixelIndex = y \* FRAME\_WIDTH + x;

// Get the grayscale value of the pixel

**uint8\_t** grayValue = yData[pixelIndex];

// Convert the grayscale value to an RGB565 color value

**uint16\_t** colorValue = ((grayValue >> 3) << 11) | ((grayValue >> 2) << 5) | (grayValue >> 3);

// Set the pixel on the LCD

**BSP\_LCD\_WritePixel**(0, x, y, colorValue);

}

}

**UTIL\_LCD\_SetBackColor**(UTIL\_LCD\_COLOR\_GRAY);

**UTIL\_LCD\_SetTextColor**(UTIL\_LCD\_COLOR\_RED);

**UTIL\_LCD\_DisplayStringAt**(0, 5, (**uint8\_t** \*)"Y-Only Format", CENTER\_MODE);

/\* for any Press Check whether the Continuous capture should be suspended or resumed \*/

**while** (UserButtonPressed != *RESET*)

{

**if**(frame\_suspended == 1)

{

**if**(**HAL\_DCMI\_Resume**(&hdcmi) != *HAL\_OK*)

{

**Error\_Handler**();

}

frame\_suspended=0;

}

**else**

{

**if**(**HAL\_DCMI\_Suspend**(&hdcmi) != *HAL\_OK*)

{

**Error\_Handler**();

}

frame\_suspended++;

}

UserButtonPressed = *RESET*;

}

/\* Check each time for new frame\*/

**if**(frame\_captured != 0)

{

frame\_captured = 0;

**BSP\_LED\_Toggle**(*LED5*);

}

**else**

{

**BSP\_LED\_Off**(*LED5*);

}

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

  /\* USER CODE END 3 \*/

Next do all the remaining steps from 29 to 32 of the RGB application