

用于STM32 MCU的数字照相机接口（DCMI）

引言

随着市场对更高图像质量的需求不断增加，成像技术持续发展，各种技术（例如，3D、计算、运动和红外线）不断涌现。

如今的成像应用对高质量、易用性、功率效率、高度集成、快速上市和成本效益提出了全面要求。

为了满足这些要求，STM32 MCU内置数字照相机接口（DCMI），能够连接高效的并行照相机模块。

此外，STM32 MCU还提供许多性能等级（CPU、MCU子系统、DSP和FPU）。它们还提供各种功耗模式、丰富的外设和接口组合（SPI、UART、I2C、SDIO、USB、ETHERNET、I2S...）、丰富的图形产品组合（LTDC、QSPI、DMA2D...）和业界领先的开发环境，为复杂应用和连接解决方案（IOT）提供保障。

本应用笔记将向STM32用户介绍一系列基本概念，并为DCMI的功能、架构和配置提供通俗易懂的说明。此外，还提供了各种具体示例作为支持。

参考文档

本应用笔记应与STM32F2、STM32F4、STM32F7系列和STM32L4x6、STM32H7x3系列的参考手册一起阅读使用：

- *STM32F205xx, STM32F207xx, STM32F215xx和STM32F217xx基于32位MCU (RM0033) 的高级ARM®*
- *STM32F405/415, STM32F407/417, STM32F427/437和STM32F429/439基于32位MCU (RM0090) 的高级ARM®*
- *STM32F446xx基于32位MCU (RM0390) 的高级ARM®*
- *STM32F469xx和STM32F479xx基于32位MCU (RM0386) 的高级ARM®*
- *STM32F75xxx和STM32F74xxx基于32位MCU (RM0385) 的高级ARM®*
- *STM32F76xxx和STM32F77xxx基于32位MCU (RM0410) 的高级ARM®*
- *STM32L4x5和STM32L4x6基于32位MCU (RM0351) 的高级ARM®*
- *STM32H7x3基于32位MCU (RM0433) 的高级ARM®*

表1. 适用产品

类型	STM32系列
STM32F2系列	STM32F2x7
STM32F4系列	STM32F407/417、STM32F427/437、STM32F429/439、STM32F446、STM32F469/479
STM32F7系列	STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8、STM32F7x9
STM32L4系列	STM32L4x6
STM32H7系列	STM32H7x3

目录

1	概述：照相机模块和基本概念	9
1.1	成像的基本概念	9
1.2	照相机模块	10
1.2.1	照相机模块的组件	11
1.2.2	照相机模块互联（并行接口）	11
2	STM32数字照相机接口（DCMI）总览	13
2.1	数字摄像头接口 (DCMI)	13
2.2	不同STM32 MCU的DCMI可用性和特性	13
2.3	智能架构中的DCMI	14
2.3.1	STM32F2x7系列的系统架构	15
2.3.2	STM32F407/417、STM32F427/437、 STM32F429/439、STM32F446和STM32F469/479系列的系统架构	15
2.3.3	STM32F7x5、STM32F7x6、STM32F7x7、 STM32F7x8和STM32F7x9系列的系统架构	17
2.3.4	STM32L496 xx和STM32L4A6xx器件的系统架构	19
2.3.5	STM32H7x3系列的系统架构	20
2.4	具有DCMI和/或照相机模块的参考板	20
3	DCMI 描述	22
3.1	硬件接口	22
3.2	照相机模块和DCMI的互联	25
3.3	DCMI 功能说明	25
3.4	数据同步	25
3.4.1	硬件（或外部）同步	26
3.4.2	内嵌码（或内部）同步	27
3.5	捕获模式	29
3.5.1	快照模式	30
3.5.2	连续采集模式	30
3.6	数据格式和存储	31
3.6.1	单色	32
3.6.2	RGB565	32
3.6.3	YCbCr	32
3.6.4	YCbCr, 仅Y分量	33

3.6.5	JPEG	33
3.7	其他功能	34
3.7.1	裁剪功能	34
3.7.2	图像大小调整（分辨率修改）	34
3.8	DCMI 中断	35
3.9	低功耗模式	36
4	DCMI 配置	38
4.1	GPIO配置	38
4.2	时钟和定时配置	39
4.2.1	系统时钟配置（HCLK）	39
4.2.2	DCMI时钟和定时配置（DCMI_PIXCLK）	39
4.3	DCMI 配置	42
4.3.1	捕获模式 (Capture mode)	42
4.3.2	数据格式	42
4.3.3	图像分辨率和大小	42
4.4	DMA 配置	42
4.4.1	用于DCMI至存储器传输的DMA常用配置	43
4.4.2	根据图像大小和捕获模式设置DMA	44
4.4.3	DCMI通道和流的配置	45
4.4.4	DMA_SxNDTR 寄存器	45
4.4.5	FIFO和批量传输的配置	46
4.4.6	快照捕获中用于低分辨率的正常模式	46
4.4.7	连续捕获中用于低分辨率的循环模式	46
4.4.8	用于中等分辨率的双缓冲区模式（快照或连续捕获）	47
4.4.9	用于更高分辨率的DMA配置	48
4.5	相机模块配置	51
5	功耗和性能考虑	52
5.1	功耗	52
5.2	性能考虑	52
6	DCMI应用示例	54
6.1	DCMI应用场景示例	54
6.2	STM32Cube固件示例	55
6.3	基于STM32CubeMX的DCMI示例	56

6.3.1	硬件说明	57
6.3.2	常用配置示例	60
6.3.3	RGB数据的捕获和显示	74
6.3.4	YCbCr数据的捕获	74
6.3.5	仅Y分量数据的捕获	76
6.3.6	SxGA分辨率的捕获（YCbCr数据格式）	76
6.3.7	JPEG格式的捕获	79
7	支持的设备	82
8	结论	83
9	版本历史	84

表格索引

表1. 适用产品 2

表2. DCMI和相关资源可用性 13

表3. STM32F4系列中的SRAM可用性 16

表4. 各种STM32板上的DCMI和照相机模块 21

表5. 低功耗模式下的DCMI操作 37

表6. 不同STM32器件的DMA流的选择 45

表7. 一次DMA传输中传输的最大字节数 45

表8. 正常模式下的最高图像分辨率 46

表9. 双缓冲区模式下的最高图像分辨率 47

表10. 最大DCMI_PIXCLK时的最大数据流 53

表11. STM32Cube DCMI示例 55

表12. 支持的照相机模块示例 82

表13. 文档版本历史 84

表14. 中文文档版本历史 84



图片索引

图1.	原始图像与数字图像的比较.....	9
图2.	水平消隐说明	10
图3.	垂直消隐说明	10
图4.	照相机模块	10
图5.	照相机模块与MCU的连接	12
图6.	STM32F2x7系列智能架构中的DCMI从设备AHB2外设	15
图7.	STM32F407/417、STM32F427/437、STM32F429/439、 STM32F446和STM32F469/479系列智能架构中的DCMI从设备AHB2外设	16
图8.	STM32F7x5、STM32F7x6、STM32F7x7、 STM32F7x8和STM32F7x9系列智能架构中的DCMI从设备AHB2外设	18
图9.	STM32L496xx和STM32L4A6xx器件智能架构中的DCMI从设备AHB2外设智能架构	19
图10.	STM32H7x3系列智能架构中的DCMI从设备外设	20
图11.	DCMI 信号	22
图12.	DCMI 框图	23
图13.	按8位数据宽度填充的数据寄存器	24
图14.	按10位数据宽度填充的数据寄存器	24
图15.	按12位数据宽度填充的数据寄存器	24
图16.	按14位数据宽度填充的数据寄存器	24
图17.	STM32 MCU和照相机模块的互联 ⁽¹⁾	25
图18.	硬件同步模式下的帧结构	26
图19.	内嵌码字节	27
图20.	内嵌码同步模式1下的帧结构	28
图21.	内嵌码同步模式2下的帧结构	28
图22.	内嵌码取消掩码	29
图23.	快照模式下的帧接收	30
图24.	连续抓取模式下的帧接收	31
图25.	像素光栅扫描顺序	31
图26.	使用单色数据填充的DCMI数据寄存器	32
图27.	使用RGB数据填充的DCMI数据寄存器	32
图28.	使用YCbCr数据填充的DCMI数据寄存器	33
图29.	仅使用Y分量数据填充的DCMI数据寄存器	33
图30.	JPEG数据接收	34
图31.	帧分辨率修改	35
图32.	DCMI中断和寄存器	36
图33.	DCMI_ESCR寄存器字节	40
图34.	FEC结构	40
图35.	LEC结构	40
图36.	FSC结构	40
图37.	LSC结构	41
图38.	内嵌码同步模式下的帧结构	41
图39.	通过DMA传输数据	44
图40.	循环模式下的帧缓冲区和DMA_SxNDTR寄存器	47
图41.	双缓冲区模式下的帧缓冲区和DMA_SxNDTR寄存器	48
图42.	高分辨率时的DMA操作	50
图43.	STM32 DCMI应用示例	55
图44.	捕获和显示应用中的数据路径	56
图45.	32F746GDISCOVERY和STM32F4DIS-CAM的互联	57
图46.	32F746GDISCOVERY板上的照相机连接器	59

图47.	STM32F4DIS-CAM上的照相机连接器	60
图48.	STM32CubeMX - DCMI同步模式选择	61
图49.	STM32CubeMX - 选择“配置选项卡”	61
图50.	STM32CubeMX - “配置”选项卡上的DCMI按钮	61
图51.	STM32CubeMX - 选择“GPIO设置”	61
图52.	STM32CubeMX - 选择DCMI引脚	62
图53.	STM32CubeMX - 选择“GPIO无上拉且无下拉”	62
图54.	STM32CubeMX - 选择“参数设置”选项卡	62
图55.	STM32CubeMX - DCMI控制信号和捕获模式配置	63
图56.	STM32CubeMX - 配置DCMI中断	63
图57.	STM32CubeMX - 选择“DMA设置”选项卡	64
图58.	STM32CubeMX - 选择“添加”按钮	64
图59.	STM32CubeMX - DMA流配置	64
图60.	STM32CubeMX: DMA配置	64
图61.	STM32CubeMX - 配置PH13引脚	65
图62.	STM32CubeMX - “配置”选项卡上的“GPIO”按钮	65
图63.	STM32CubeMX - 配置DCMI电源引脚	66
图64.	STM32CubeMX - HSI配置	66
图65.	STM32CubeMX - 时钟配置	67

1 概述：照相机模块和基本概念

本节提供照相机模块及其主要元件的概述，并介绍了并行照相机模块的外部接口。

1.1 成像的基本概念

本节提供像场的简介以及基本概念和原理（例如像素、分辨率、色深和消隐）的概述。

- **像素**：图像的每个点均体现了彩色图像的颜色或黑白照片的灰度。通过数值逼近重建最终图像。该数字图像是由物理点组成的二维数组。每个点称为一个像素（源自图像元素）。换句话说，像素是图像的最小可控元素。每个像素均可寻址。

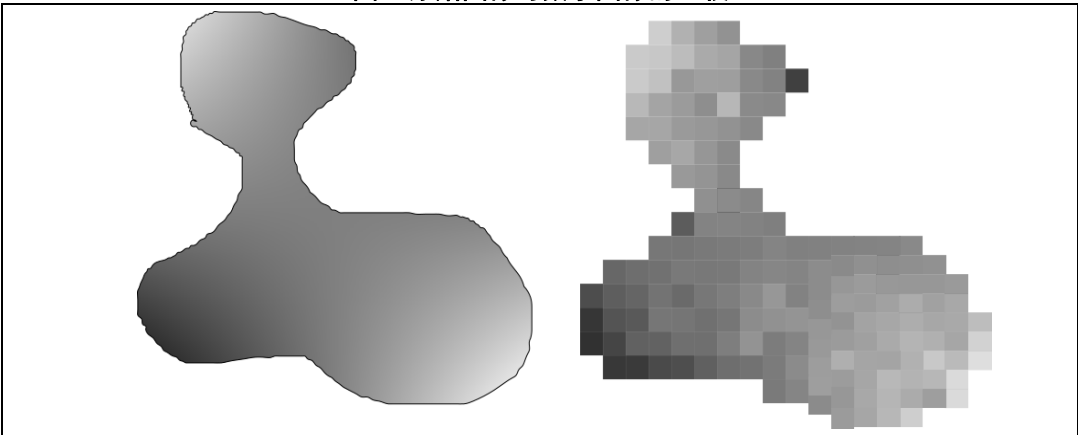
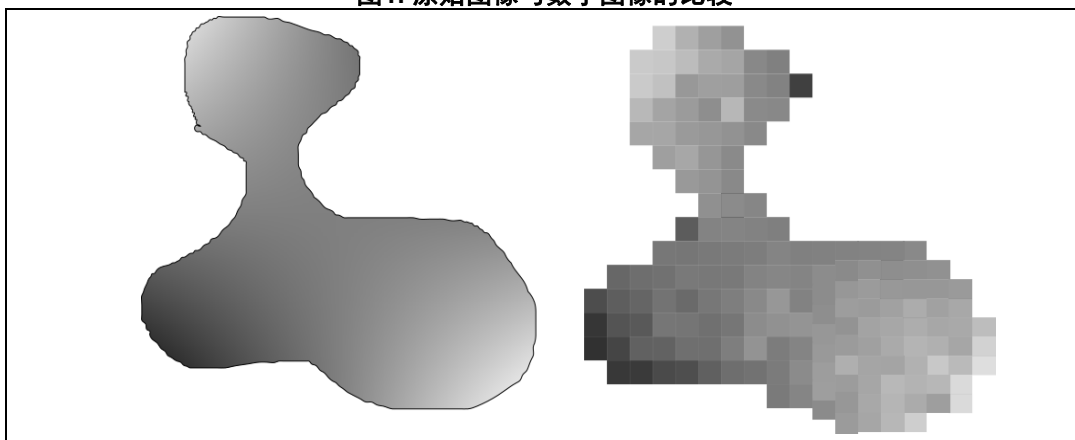
 1显示了原始图像与数值逼近之间的差异。

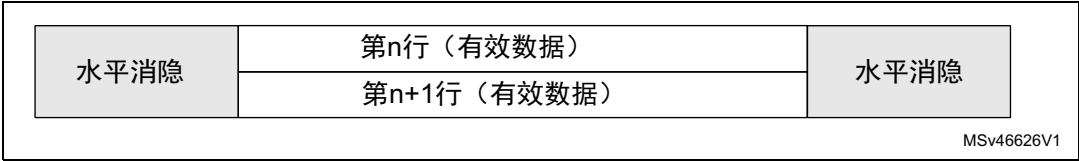
图1. 原始图像与数字图像的比较



- **分辨率**：图像中像素的数量。像素越多，图像尺寸越大。当图像尺寸相同时，像素的数量越多，图像包含的细节越丰富。
- **色深（位深）**：用于指示像素颜色的位数。它也被称为每像素位数（bpp）。
示例：
 - 对于二值图像，每个像素包含一位。每个像素为黑色或白色（0或1）。
 - 对于灰度图，图像通常为2 bpp（每个像素可以有4级灰阶中的1级）至8 bpp（每个像素可以有256级灰阶中的1级）。
 - 对于彩色图像，每个像素的位数为8至24不等（每个像素最多可以有16777216种可能的颜色）。
- **帧率（视频）**：每秒传输的帧（或图像）数，表示为帧每秒（FPS）。

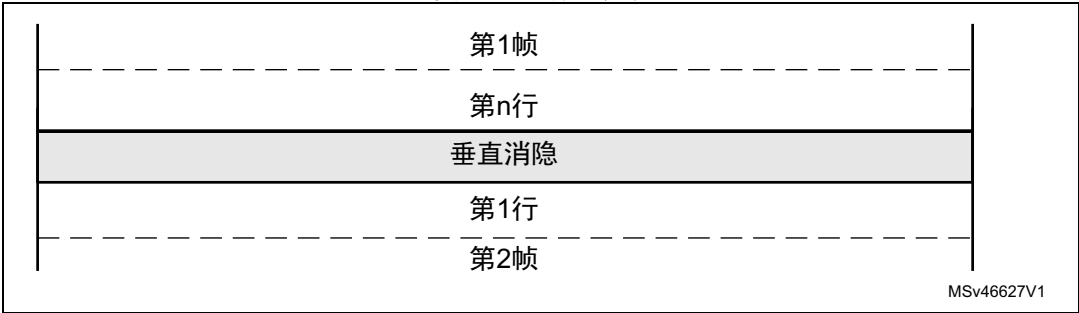
- **水平消隐：**一行末尾与下一行开头之间被忽略的行。

图2. 水平消隐说明



- **垂直消隐：**帧最后一行末尾与下一帧第一行开头之间被忽略的行。

图3. 垂直消隐说明



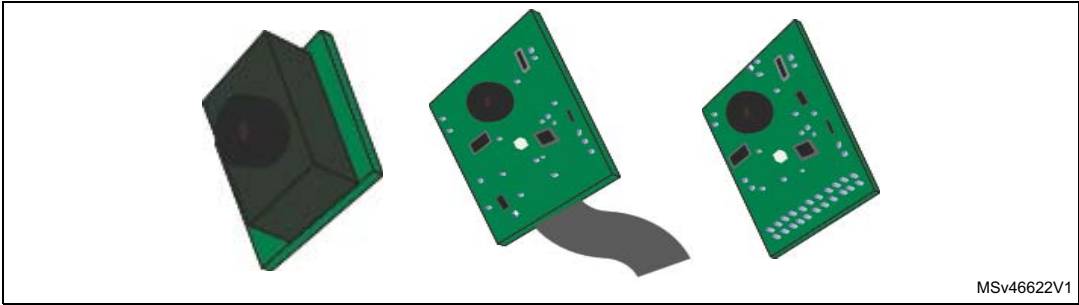
- **逐行扫描：**一种处理移动图像的方式。它可以按顺序逐一扫描行，无需像隔行扫描一样将奇数行与偶数行分开。构建图像：
 - 在逐行扫描时，先绘制第一行，然后是第二行，再到第三行。
 - 在隔行扫描时，每一帧分为两个场，即奇数行和偶数行。两个场交替显示。

1.2 照相机模块

照相机模块由四部分组成：图像传感器、镜头、印刷电路板（PCB）和接口。

图 4所示为一些常用照相机模块的示例。

图4. 照相机模块



1.2.1 照相机模块的组件

下文描述了照相机模块的四个组件：

图像传感器

它是一种模拟设备，能够将接收到的光转换为电子信号。这些信号传输构成数字图像的信息。

数字照相机中可以使用两种类型的传感器：

- CCD（电荷耦合器件）传感器
- CMOS（互补金属氧化物半导体）传感器。

二者都将光转换为电子信号，但是有各自的转换方法。由于性能持续改进且成本不断下降，CMOS成像装置已在数字摄影领域占据主导地位。

镜头

镜头是一种光学镜片，能够严格复制图像传感器捕获的实际图像。挑选合适的镜头是用户创造力的一部分，并会显著影响图像质量。

印刷电路板 (PCB)

PCB是一种由电子元件组成的板，用于确保良好的极化并保护图像传感器。

PCB还为照相机模块的所有其他部分提供支持。

照相机模块的互联

照相机接口是一种桥接器，能够将图像传感器连接到嵌入式系统并发送或接收信号。照相机与嵌入式系统之间传输的信号主要是：

- 控制信号
- 图像数据信号
- 电源信号
- 照相机配置信号。

根据数据信号的传输方式，可将照相机接口分为两种类型：**并行**和**串行接口**。

1.2.2 照相机模块互联（并行接口）

如上文所述，照相机模块需要四种主要类型的信号来正确发送图像数据：控制信号、图像数据信号、电源信号和照相机配置信号。


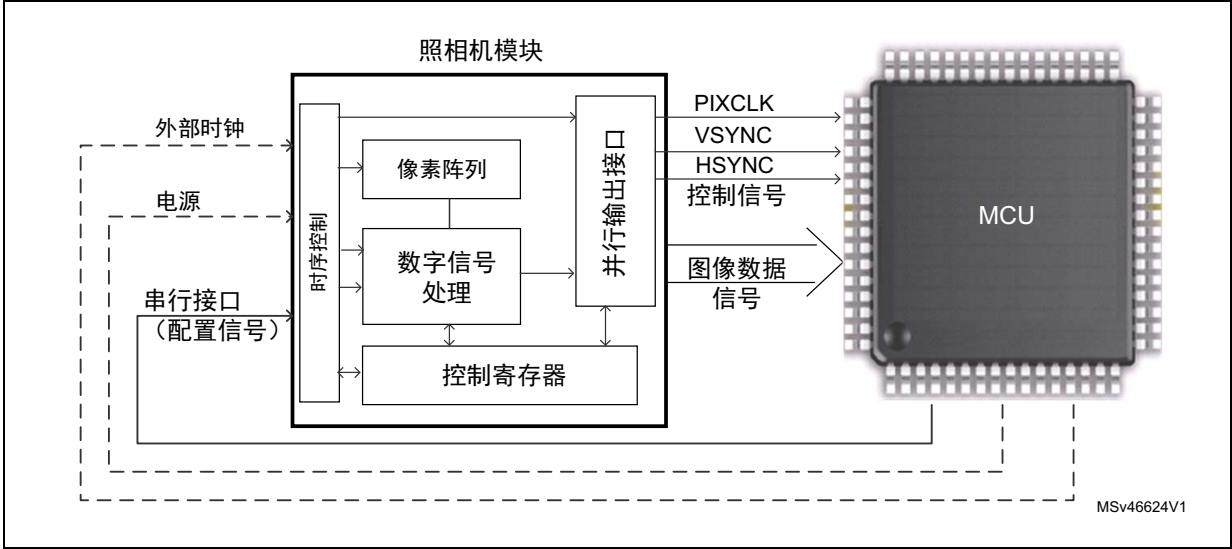
 图 5所示为CMOS传感器及其与MCU互联的典型框图。

图5. 照相机模块与MCU的连接



控制信号

这些信号用于时钟生成和数据传输同步。必须根据照相机规格提供照相机时钟。

照相机还提供数据同步信号：

- **HSYNC**，用于行同步
- **VSYNC**，用于帧同步。

图像数据信号

每一个这样的信号发送一位图像数据。图像数据信号宽度代表每个像素时钟要传输的位数。该数值取决于照相机模块的并行接口和嵌入式系统接口。

电源信号

与任何嵌入式电子系统一样，照相机模块需要电源。照相机模块的数据手册中指定了模块的工作电压。

配置信号

这些信号用于：

- 配置合适的图像特性，例如分辨率、格式和帧率
- 配置对比度和亮度
- 选择接口类型（照相机模块可支持多个接口：一个并行接口和一个串行接口。然后，用户应根据应用选择最方便的接口。）

大多数照相机模块通过I²C通信总线实现参数化。

2 STM32数字照相机接口（DCMI）总览

本节提供了不同STM32器件的数字照相机接口（DCMI）可用性的综合概述，并给出了关于STM32 MCU架构中DCMI集成的简单易懂的说明。

2.1 数字摄像头接口 (DCMI)

数字照相机接口（DCMI）是一种同步并行数据总线。它可以轻松集成并轻松适应应用的特殊要求。DCMI连接8、10、12和14位CMOS照相机模块，并支持多种数据格式。

2.2 不同STM32 MCU的DCMI可用性和特性

[表 2](#)对内置DCMI的STM32器件进行了汇总；它还显示了方便DCMI操作或可以在同一应用中与DCMI一起使用的其他硬件资源的可用性。

DCMI应用需要帧缓冲区来存储采集的图像。因此，必须根据图像大小和传输速度使用合适的目标存储区。

在某些应用中，必须连接外部存储器，以便提供较大空间用于数据存储。因此，可以使用Quad-SPI。有关更多详细信息，请参考应用笔记*STM32微控制器上的Quad-SPI接口*（AN4760）。

DMA2D（Chrom-ART Accelerator™控制器）可用于色彩空间转换（例如RGB565至ARGB8888），或从一个存储区到另一个存储区的数据转移。

JPEG编解码器能够进行数据压缩（JPEG编码）或解压缩（JPEG解码）。

表2. DCMI和相关资源可用性

STM32系列	最大闪存大小 (字节)	片上SRAM (字节)	QUAD SPI	最高FMC SRAM和SDRAM 频率 (MHz) ⁽¹⁾	最大DCMI 像素时钟输入 (MHz) ⁽²⁾	JPEG 编解码器	DMA2D	LCD_ TFT控制器 ⁽³⁾	MIPI- DSI主机 ⁽⁴⁾	最高 AHB频率 (MHz)
STM32F2x7	1 M	128	无	60	48	无	无	无	无	120
STM32F407/417	1 M	192	无	60	54	无	无	无	无	168
STM32F427/437	2 M	256	无	90	54	无	有	无	无	180
STM32F429/439	2 M	256	无	90	54	无	有	有	无	180
STM32F446	512 K	128	有	90	54	无	无	无	无	180
STM32F469/479	2 M	384	有	90	54	无	有	有	有	180
STM32F7x5	2 M	512	有	100	54	无	有	无	无	216
STM32F7x6	1 M	320	有	100	54	无	有	有	无	216
STM32F7x7	2 M	512	有	100	54	有	有	有	无	216
STM32F7x8 STM32F7x9	2 M	512	有	100	54	有	有	有	有	216

表2. DCMI和相关资源可用性（续）

STM32系列	最大闪存大小 (字节)	片上SRAM (字节)	QUAD SPI	最高FMC SRAM和SDRAM 频率(MHz) ⁽¹⁾	最大DCMI 像素时钟 输入(MHz) ⁽²⁾	JPEG 编解码器	DMA2D	LCD_ TFT控制 器 ⁽³⁾	MIPI- DSI主 机 ⁽⁴⁾	最高 AHB频 率 (MHz)
STM32L4x6	1 M	320	有	40	32	无	有	无	无	80
STM32H7x3	2 M	1000	有	133	80	有	有	有	无	400

- 1. 用于STM32F2x7和STM32F407/417系列的FSMC。
- 2. 对于像素时钟频率（DCMI_PIXCLK），请参考相应器件的数据手册。
- 3. 关于STM32 LTDC外设的更多信息，请参考应用笔记AN4861。
- 4. 关于STM32的MIPI-DSI主机的更详细信息，请参见应用笔记AN4860。

2.3 智能架构中的DCMI

DCMI通过AHB2外设总线连接到AHB总线矩阵。DMA将访问它以便传输接收到的图像数据。所接收数据的目标位置取决于应用。

STM32 MCU的智能架构允许：

- DMA（作为AHB主设备）在CPU处理之前采集的图像（图像编号n）时自主访问AHB2外设并将接收的数据（图像编号n+1）传输到存储器。
- 使用DMA2D（作为AHB主设备）传输或修改接收的数据并将CPU资源保留用于其他任务
- 通过多层总线矩阵实现存储器吞吐量改善和性能改进。



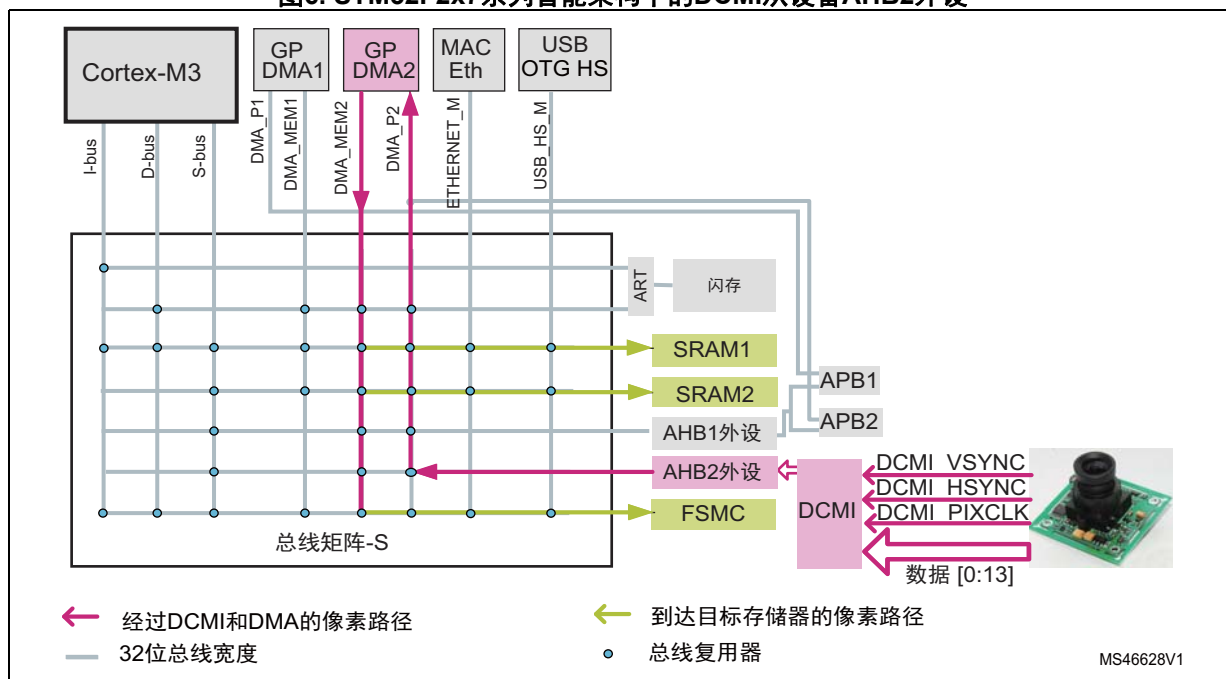
2.3.1 STM32F2x7系列的系统架构

STM32F2x7系列器件基于32位多层总线矩阵，允许八台主设备和七台从设备之间的互联。

DCMI是从设备AHB2外设。DMA2通过FSMC执行从DCMI至内部SRAM或外部存储器的数据传输。

图6所示为STM32F2x7xx器件中的DCMI互联和数据路径。

图6. STM32F2x7系列智能架构中的DCMI从设备AHB2外设



2.3.2 STM32F407/417、STM32F427/437、STM32F429/439、STM32F446和STM32F469/479系列的系统架构

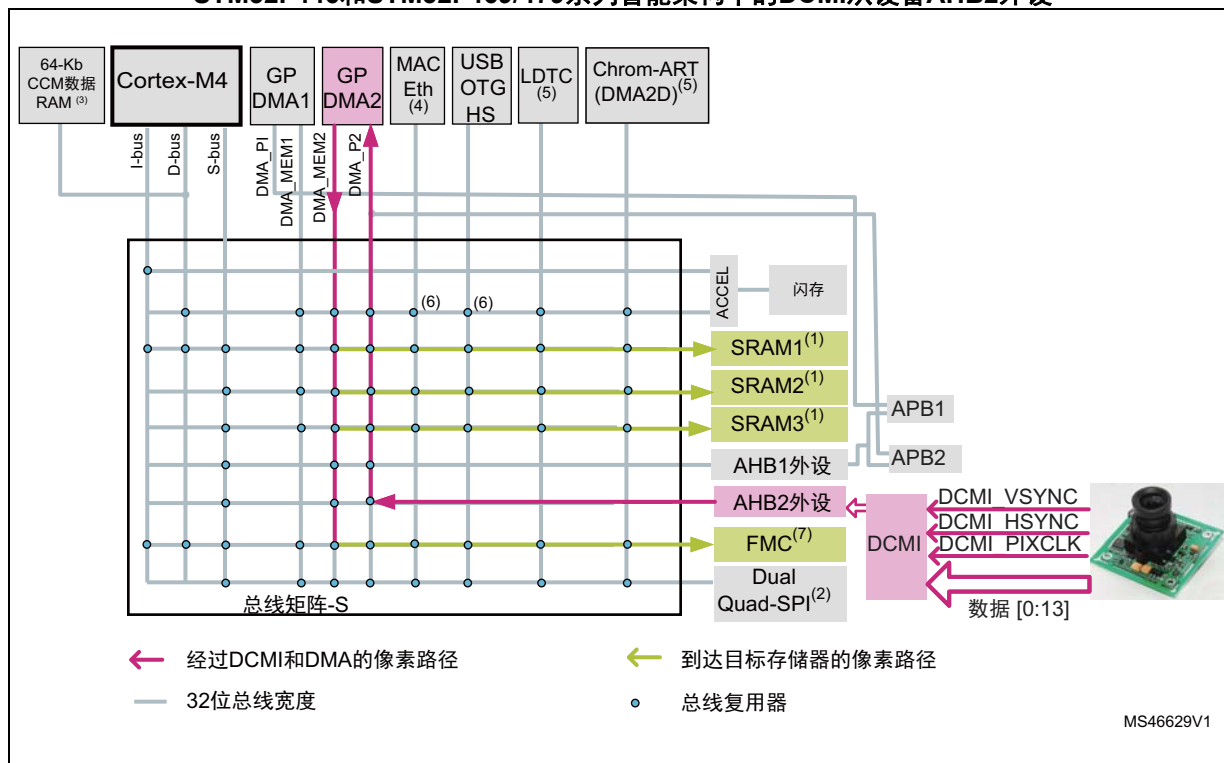
STM32F407/417、STM32F427/437、STM32F429/439、STM32F446和STM32F469/479系列器件基于32位多层总线矩阵，能够实现以下设备之间的互联：

- 对于STM32F429/439系列，为十台主设备和八台从设备
- 对于STM32F469/479系列，为十台主设备和九台从设备
- 对于STM32F446系列，为七台主设备和七台从设备
- 对于STM32F407/417系列，为八台主设备和七台从设备
- 对于STM32F427/437系列，为八台主设备和八台从设备。

DCMI是从设备AHB2外设。DMA2通过FMC执行从DCMI至内部SRAM或外部存储器的数据传输（对于STM32F407/417系列，为FSMC）。

图7所示为STM32F407/417、STM32F427/437、STM32F429/439、STM32F446和STM32F469/479系列微控制器中的DCMI互联和数据路径。

图7. STM32F407/417、STM32F427/437、STM32F429/439、STM32F446和STM32F469/479系列智能架构中的DCMI从设备AHB2外设



1. 有关SRAM1、SRAM2和SRAM3的更多信息, 请参见表 3。

表3. STM32F4系列中的SRAM可用性

STM32系列	SRAM1 (KB)	SRAM2 (KB)	SRAM3 (KB)
STM32F407/417	112	16	x
STM32F427/437 - STM32F429/439	112	16	64
STM32F446	112	16	x
STM32F469/479	160	32	128

2. 仅STM32F469/479和STM32F446系列提供双路Quad-SPI接口。
3. STM32F446xx器件不提供64-Kb CCM数据RAM。
4. STM32F446xx器件不提供以太网MAC接口。
5. 内置LTDC和DMA2D的系列只有STM32F429/439和STM32F469/479。
6. 对于STM32F407/417系列：
 - 以太网主设备与闪存DCode总线之间不互联
 - USB主设备与闪存DCode总线之间不互联。对于STM32F446系列，USB主设备与闪存的DCode总线之间不互联。
7. 用于STM32F407/417系列的FSMC。

2.3.3 STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列的系统架构

STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列器件基于32位多层总线矩阵，能够实现以下设备之间的互联：

- 对于STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列，为十二台主设备和八台从设备
- 对于STM32F7x5系列，为十一台主设备和八台从设备。

DCMI是从设备AHB2外设。DMA2通过FMC执行从DCMI至内部SRAM或外部存储器的数据传输。

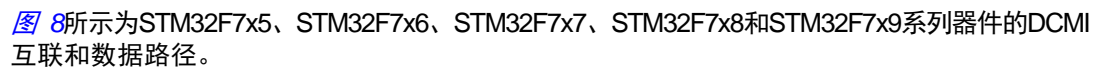
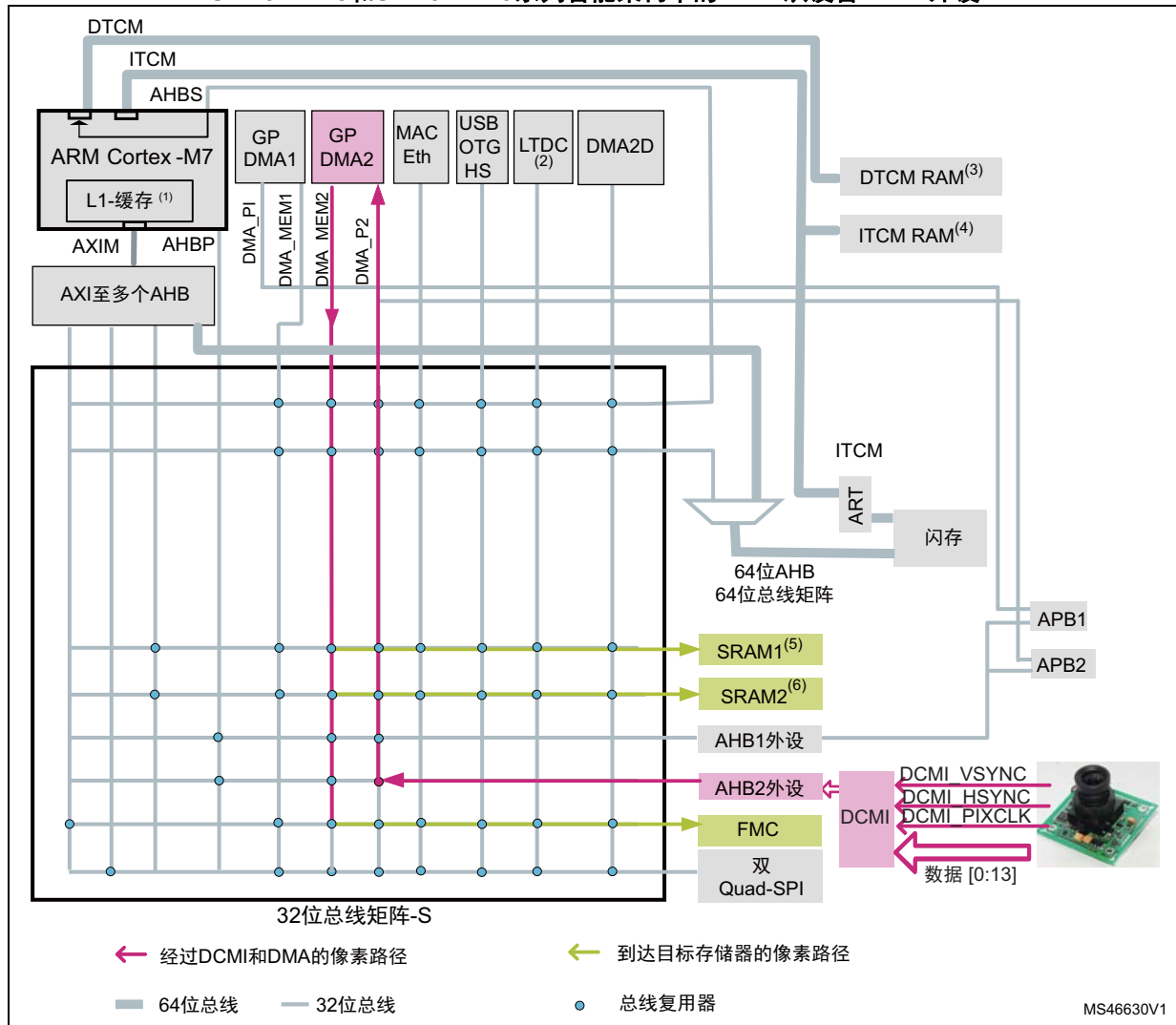
 图 8所示为STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列器件的DCMI互联和数据路径。

图8. STM32F7x5、STM32F7x6、STM32F7x7、
STM32F7x8和STM32F7x9系列智能架构中的DCMI从设备AHB2外设



- I/D缓存空间大小：
 - STM32F7x5和STM32F7x6系列为4 Kb
 - STM32F7x7、STM32F7x8和STM32F7x9系列为16 Kb。
- 仅STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列提供LTDC（LCD-TFT控制器）。
- DTCM RAM大小：
 - STM32F7x5和STM32F7x6系列为64 Kb
 - STM32F7x7、STM32F7x8和STM32F7x9系列为128 Kb。
- STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列的ITCM RAM大小为16 KB。
- SRAM1大小：
 - STM32F7x5和STM32F7x6系列为240 Kb
 - STM32F7x7、STM32F7x8和STM32F7x9系列为368 Kb。
- STM32F7x5、STM32F7x6、STM32F7x7、STM32F7x8和STM32F7x9系列的SRAM2大小为16 KB。

2.3.4 STM32L496xx和STM32L4A6xx器件的系统架构

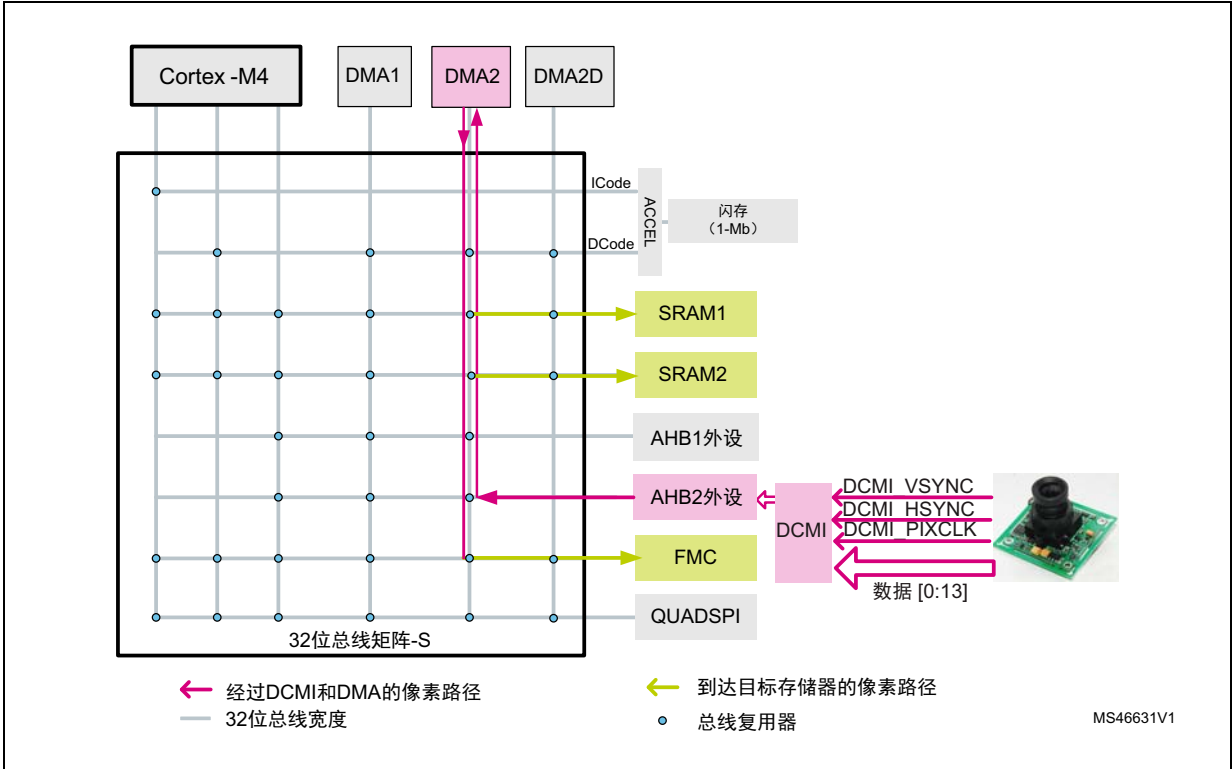
STM32L496xx和STM32L4A6xx器件基于32位多层总线矩阵，允许六台主设备和八台从设备之间的互联。

DCMI是从设备AHB2外设。DMA2通过FMC执行从DCMI至内部SRAM或外部存储器的数据传输。

在STM32L496xx和STM32L4A6xx MCU中，DMA只有一个端口（有别于外设端口独立于存储器端口的STM32F2、STM32F4、STM32F7和STM32H7系列），但是它支持圆形缓冲区管理以及外设至存储器、存储器至外设和外设至外设的数据传输。

图 9所示为STM32L496xx和STM32L4A6xx器件中的DCMI互联和数据路径。

图9. STM32L496xx和STM32L4A6xx器件智能架构中的DCMI从设备AHB2外设智能架构



2.3.5 STM32H7x3系列的系统架构

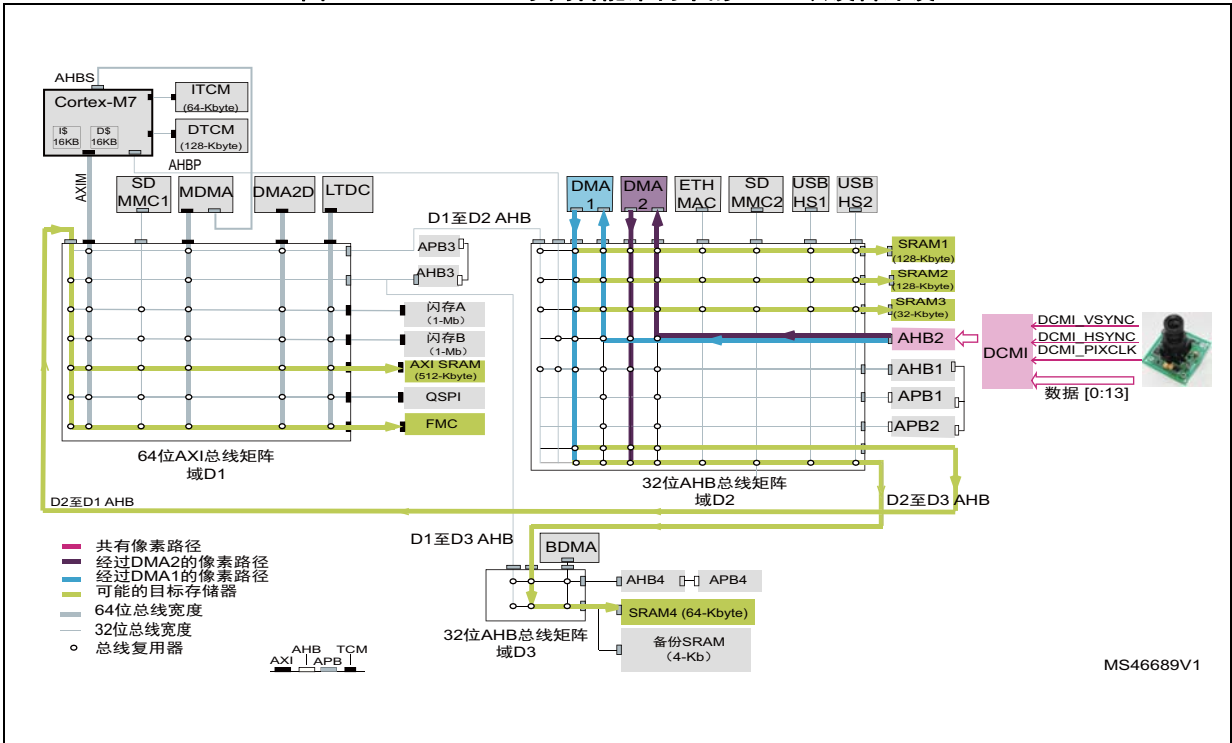
STM32H7x3xx器件基于一个AXI总线矩阵、两个AHB总线矩阵和总线桥，允许18台主设备和20台从设备之间的互联。

DCMI是从设备AHB2外设。DMA1或DMA2可通过FMC执行从DCMI至内部SRAM或外部存储器的数据传输。

DMA1和DMA2位于域D2中。它们能够访问域D1和域D3中的从设备。因此，DMA1或DMA2能够将DCMI（位于域2中）接收的数据传输至域1或域3中的存储区。

图 10所示为STM32H7x3xx器件中的DCMI互联和数据路径。

图10. STM32H7x3系列智能架构中的DCMI从设备外设



2.4 具有DCMI和/或照相机模块的参考板

ST提供许多STM32参考板，例如NUCLEO、Discovery和EVAL板。其中的大多数内置DCMI，有些则具有板载照相机模块。

可根据应用和硬件资源选择板。

表 4总结了各种STM32参考板的DCMI、照相机模块和存储器可用性。

表4. 各种STM32板上的DCMI和照相机模块⁽¹⁾

STM32系列	板	照相机模块	CMOS传感器	内部SRAM (KB)	外部SDRAM 总线宽度 (位)	外部SRAM 总线宽度 (位)
STM32F2x7	STM3220G-EVAL	有 ⁽²⁾	OV2640或 OV9655	132	NA	
	STM3221G-EVAL	有 ⁽²⁾				
STM32F407/417	STM32F4DISCOVERY	有 ⁽³⁾ 或 ⁽⁴⁾	OV9655	196		
	STM3240G-EVAL	有 ⁽²⁾				
	STM3241G-EVAL					
STM32F429/439	32F429IDISCOVERY	NA ⁽³⁾	NA	256	16	NA
	STM32429I-EVAL	有 ⁽²⁾	OV2640或 OV9655		32	16
	STM32439I-EVAL					
STM32F446	STM32446E-EVAL	有 ⁽²⁾	S5k5CAGA	128	16	NA
STM32F469/479	32F469IDISCOVERY	NA ⁽³⁾	NA	324	32	NA
	STM32469I-EVAL	有 ⁽²⁾	S5k5CAGA			16
	STM32479I-EVAL					
STM32F7x6	32F746GDISCOVERY	有 ⁽⁴⁾	OV9655	320	16	NA
	STM32746G-EVAL	有 ⁽²⁾	S5k5CAGA		32	16
	STM32756G-EVAL					
STM32F7x9	32F769IDISCOVERY	NA ⁽³⁾	NA	512	32	NA
	STM32F769I-EVAL	有 ⁽²⁾	S5k5CAGA			16
	STM32F779I-EVAL					
STM32L4x6	32L496GDISCOVERY	有 ⁽⁴⁾	OV9655	320	NA	NA
STM32H7x3	STM32H743I-EVAL STM32H753I-EVAL	NA ⁽³⁾	NA	864	32	16

1. NA：不适用。用户应使用与DCMI接口兼容的合适照相机模块。
2. 对于不同的EVAL板，可使用特定的连接器连接DCMI和照相机模块。
 - 对于STM3220G-EVAL、STM3221G-EVAL、STM32F40G-EVAL和STM32F41G-EVAL，将可以连接两种照相机：模块CN01302H1045-C（CMOS传感器OV9655，130万像素）和模块CN020VAH2554-C（CMOS传感器OV2640，200万像素）。
 - 对于STM32429I-EVAL和STM32439I-EVAL，连接照相机模块子板MB1066。
 - 对于STM32446E-EVAL、STM32469I-EVAL、STM32F479I-EVAL、STM32746G-EVAL、STM32756G-EVAL、STM32F769I-EVAL和STM32F779I-EVAL，连接照相机模块子板MB1183。
3. 照相机模块可通过GPIO引脚连接到DCMI。
4. 照相机模块可通过FFC（柔性扁平电缆）连接到DCMI：
 - 对于STM32F4DISCOVERY，应使用STM32F4DIS-EXT扩展板连接STM32F4DIS-CAM照相机模块。
 - 对于32F746IDISCOVERY和32L496GDISCOVERY，可直接连接STM32F4DIS-CAM板。
 关于STM32F4DIS-EXT和STM32F4DIS-CAM的更多信息，请访问意法半导体网站

3 DCMi 描述

本节将详细描述DCMI及其处理图像数据和同步信号的方式。

注：DCMI仅支持从设备输入模式。

3.1 硬件接口

DCMI 包括：

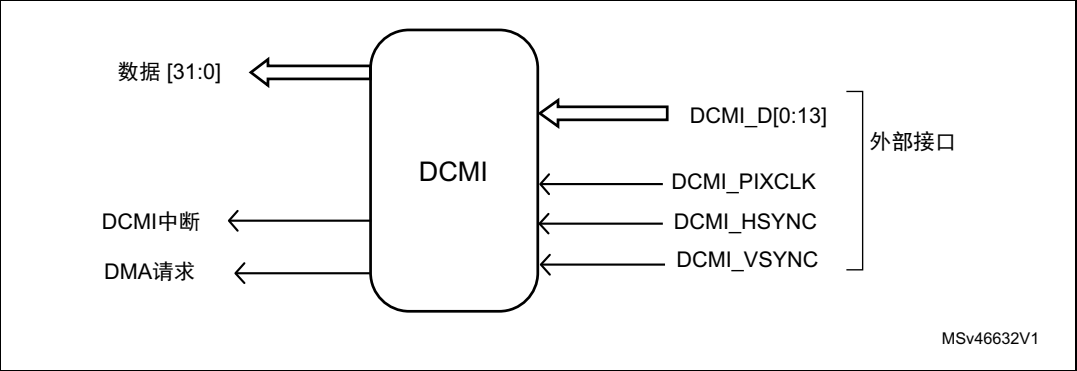
- 最多14条数据线（D13-D0）
- 像素时钟线DCMI_PIXCLK
- DCMI_HSYNC线（水平同步）
- DCMI_VSYNC线（垂直同步）。

DCMI包含最多17路输入。DCMI输入的数量（11、13、15或17路信号）随用户使能的数据线数量（8、10、12或14）而变化。

如果使用小于14位的数据宽度，则不得通过GPIO复用功能将未使用引脚分配给DCMI。可将未使用输入引脚分配给其他外设。

对于内嵌码同步，DCMI只需要9路输入（8路数据线和DCMI_PIXCLK）就能正常工作。8个未使用引脚可用于GPIO或其他功能。

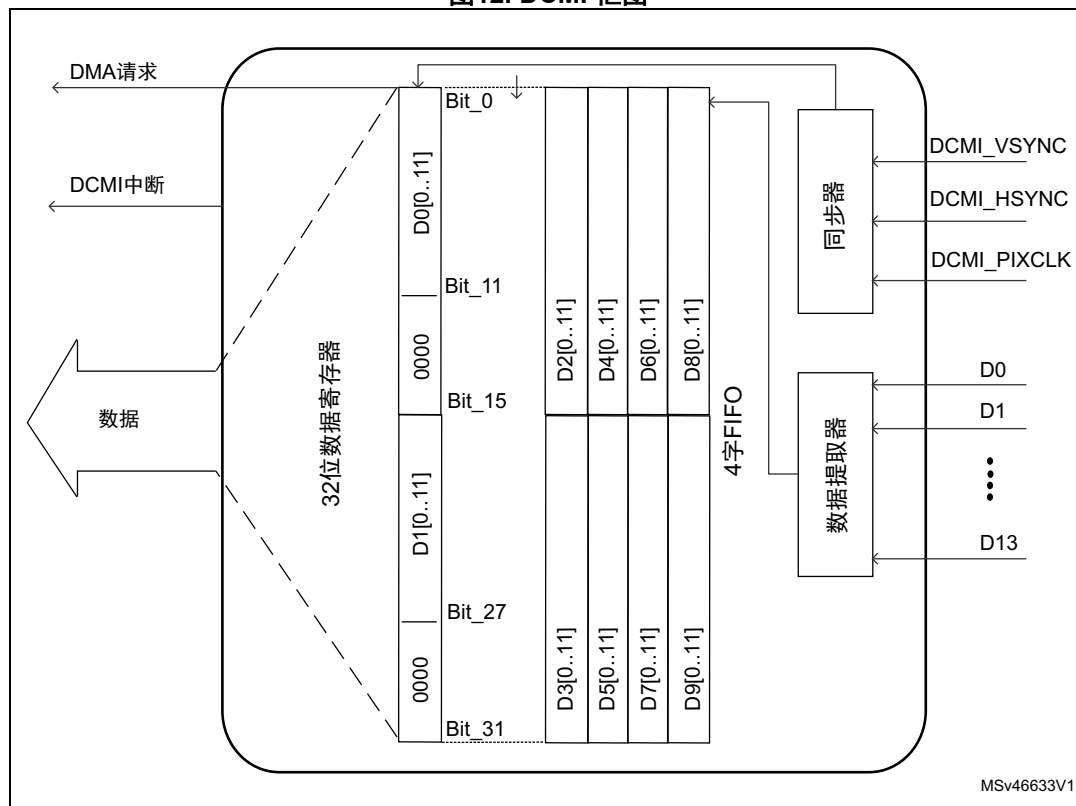
图11. DCMi 信号



如果选择x位数据宽度（使能x路数据线，x为8、10、12或14），每个DCMI_PIXCLK周期将传输x位图像（或视频）数据，并压缩到32位寄存器中。

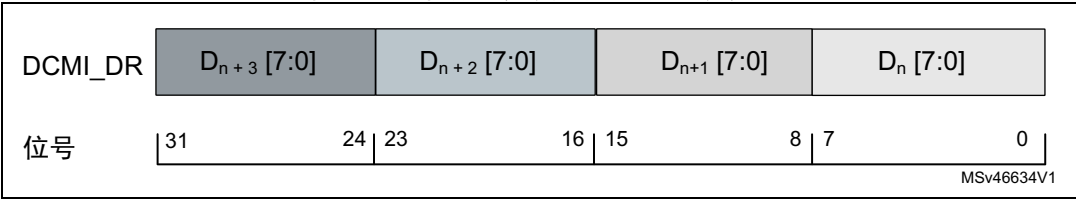
如图 12 所示, DDCMI 由四个主要元件组成:

图12. DCMI 框图



- **DCMI同步器**：确保对通过DCMI的数据流有序排列的控制。它控制数据提取器、FIFO和32位寄存器。
- **数据提取器**：确保对通过DCMI接收的数据的提取。
- **FIFO**：实现该4字FIFO的目的是调整传输到AHB的数据率。如果AHB不维持数据传输率，就没有溢出保护来防止数据被覆盖。如果同步信号溢出或出错，FIFO将复位，DCMI接口将等待新的帧开始。
- **32位寄存器**：数据寄存器，数据位在这里被打包以便通过通用DMA通道进行传输。捕获数据在32位寄存器中的定位放置取决于数据宽度：
 - 对于**8位数据宽度**，DCMI捕获8个LSB（忽略其他6路输入D[13:8]）。捕获的第一个数据字节放置在32位字的LSB位置，捕获的第四个数据字节放置在MSB位置。
因此，在此情况下，每四个像素时钟周期会生成一个32位数据字。

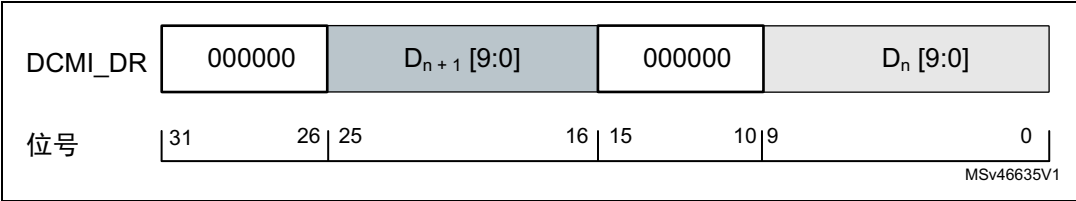
图13. 按8位数据宽度填充的数据寄存器



有关详细信息，请参见第 3.6 节：数据格式和存储。

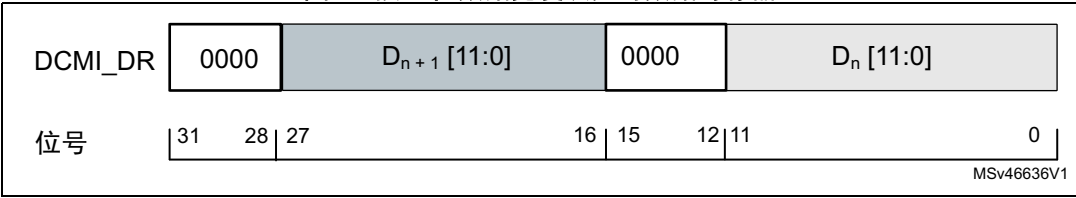
- 对于**10位数据宽度**，DCMI捕获10个LSB（忽略其他4路输入D[13:10]）。捕获的前10位作为16位字的10个LSB。DCMI_DR寄存器16位字中的剩余MSB（位10至15）将被清零。
因此，在此情况下，每两个像素时钟周期会生成一个 32 位数据字。

图14. 按10位数据宽度填充的数据寄存器



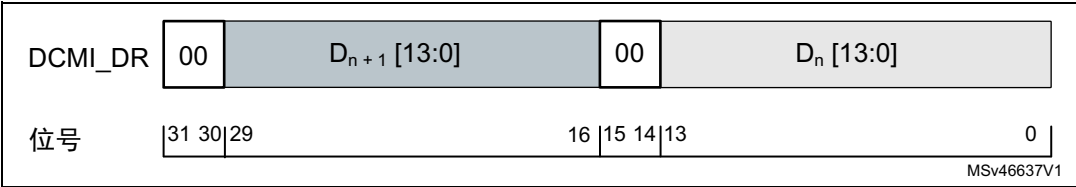
- 对于**12位数据宽度**，DCMI捕获12个LSB（忽略其他2路输入D[13:12]）。捕获的前12位作为16位字的12个LSB。DCMI_DR寄存器16位字中的剩余MSB（位12至15）将被清零。
因此，在此情况下，每两个像素时钟周期会生成一个 32 位数据字。

图15. 按12位数据宽度填充的数据寄存器



- 对于**14位数据宽度**，DCMI捕获所有接收到的位。捕获的前14位作为16位字的14个LSB。DCMI_DR寄存器16位字中的剩余MSB（位14和15）将被清零。
因此，在此情况下，每两个像素时钟周期会生成一个 32 位数据字。

图16. 按14位数据宽度填充的数据寄存器

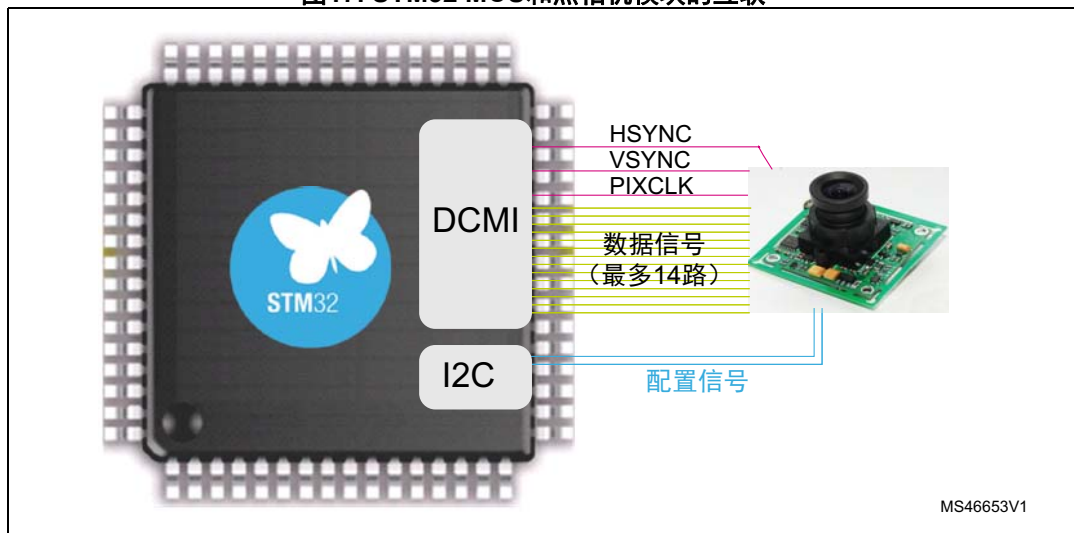


3.2 照相机模块和DCMI的互联

如第 1.2.2 节：照相机模块互联（并行接口）所述，照相机模块通过三类信号连接到DCMI：

- DCMI时钟和数据信号
- I2C配置信号

图17. STM32 MCU和照相机模块的互联⁽¹⁾



1. 对于内嵌码同步，将忽略DCMI_HSYNC和DCMI_VSYNC信号，只使用8路数据信号

3.3 DCMI 功能说明

以下步骤总结了内部DCMI元件操作，并给出了通过系统总线矩阵的数据流的示例：

- 在接收到不同信号后，同步器控制通过DCMI不同元件（数据提取器、FIFO和32位数据寄存器）的数据流。
- 通过提取器提取的数据在4字FIFO中打包，然后在32位寄存器中排序。
- 在寄存器中将32位数据块打包后，将生成DMA请求。
- DMA将数据传输至相应的目标存储区。
- 存储器中保存的数据的处理方式可能存在差异，具体取决于应用。

注：假定在照相机模块中执行所有图像预处理。

3.4 数据同步

照相机接口具有可配置并行数据接口（8至14路数据线），以及像素时钟线DCMI_PIXCLK（上升/下降沿配置）、水平同步线DCMI_HSYNC和垂直同步线DCMI_VSYNC，可设定极性。

DCMI_PIXCLK和AHB时钟必须满足2.5的最小AHB/DCMI_PIXCLK比值。

某些照相机模块支持两种类型的同步，而其他则支持硬件或内嵌码同步。

3.4.1 硬件（或外部）同步

在该模式下，使用DCMI_VSYNC和DCMI_HSYNC两个信号进行同步：

- 行同步一直被称为DCMI_HSYNC（也称为LINE VALID）。
- 帧同步一直被称为DCMI_VSYNC（也称为FRAME VALID）。

DCMI_PIXCLK和同步信号（DCMI_HSYNC和DCMI_VSYNC）的极性可设定。

数据将与DCMI_PIXCLK以及像素时钟上升或下降沿的变化同步，具体取决于配置的极性。

如果为DCMI_VSYNC和DCMI_HSYNC信号设定了有效电平（高电平有效或低电平有效），则当VSYNC或HSYNC处于该电平（高或低）时，并行接口中的数据无效。

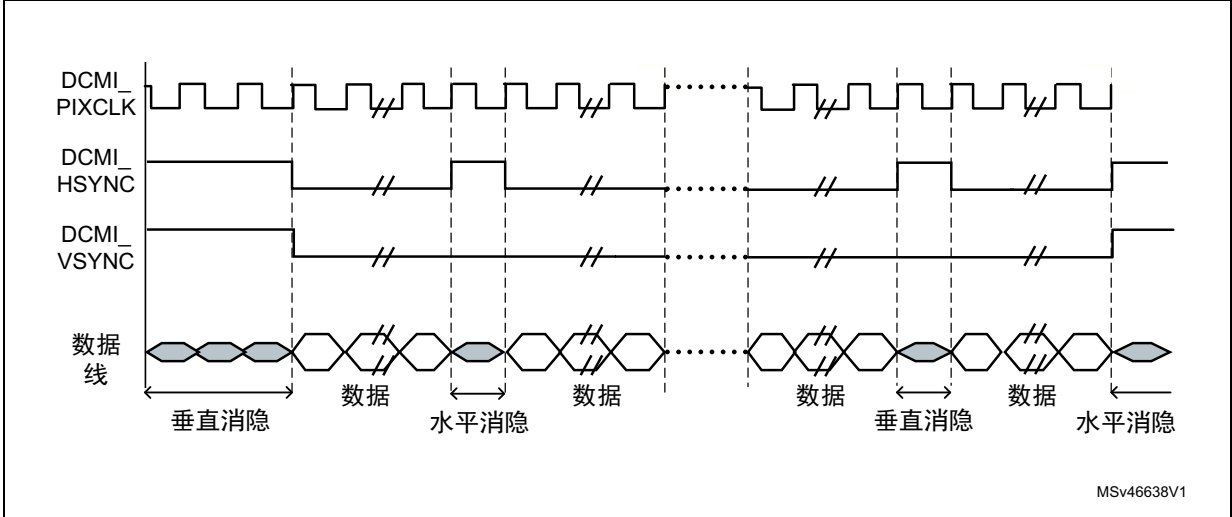
例如，如果VSYNC被设定为高电平有效：

- 当VSYNC处于低电平时，数据有效
- 当VSYNC处于高电平时，数据无效（垂直消隐）。

DCMI_HSYNC和DCMI_VSYNC信号类似于消隐信号，因为在DCMI_HSYNC / DCMI_VSYNC有效期间接收的所有数据都将被忽略。

图 18所示为DCMI_VSYNC和DCMI_HSYNC为高电平有效且DCMI_PIXCLK的捕获沿为上升沿时的数据传输示例。

图18. 硬件同步模式下的帧结构



MSv46638V1

压缩数据同步

对于压缩数据（JPEG），DCMI仅支持硬件同步。每个JPEG流被分成多个包。这些包的大小可设定。包的调度取决于图像内容，并在两个包之间的时间形成可变消隐。

DCMI_HSYNC用于指示包的起始/结束。

DCMI_VSYNC用于指示流起始/结束。

如果完整数据流结束但没有发生流结束检测（DCMI_VSYNC无变化），DCMI将通过插入零填充帧结束。

3.4.2 内嵌码（或内部）同步

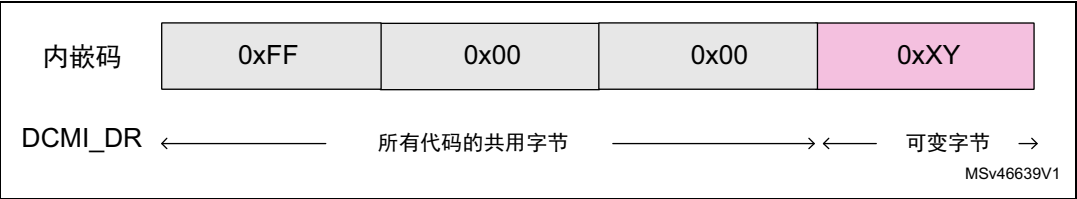
这种情况下，使用定界码进行同步。这些代码嵌入数据流，用于指示行起始/结束或帧起始/结束。

注： 仅当并行数据接口宽度为8位时，才支持这些代码。对于其它数据宽度，此模式将造成无法预知的结果，因此不得使用。

有了这些代码，不再需要DCMI_HSYNC和DCMI_VSYNC来指示行或帧的结束/起始。当使用该同步模式时，**有两个值不能用于数据：0和255（0x00和0xFF）**。这两个值被保留用于数据识别用途。由照相机模块负责控制数据值。因此，图像数据只能有254个可能的值（0x00 < 图像数据值 < 0xFF）。

每个同步码均包含4字节序列**0xFF 00 00 XY**，其中所有定界码的第一个3字节序列0xFF 00 00均相同。只将最后一个0xXY设定用于指示相应事件。

图19. 内嵌码字节



模式 1

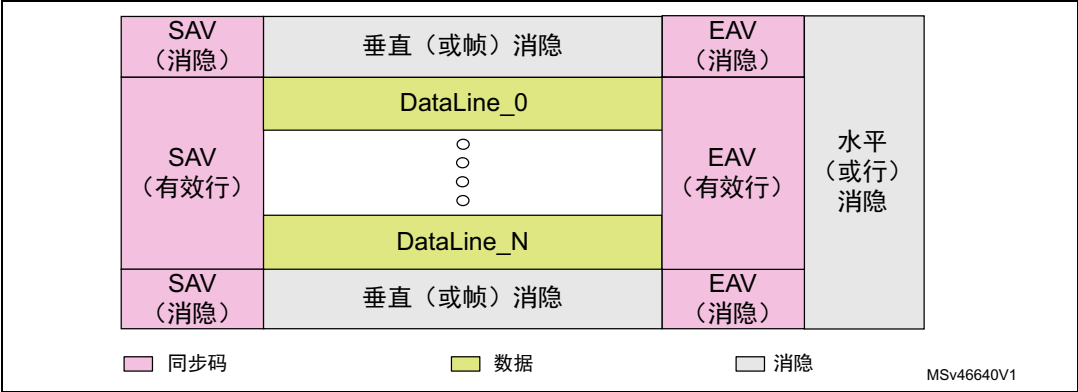
该模式与ITU656兼容（ITU656指数字视频协议ITU-R BT.656）。

有四个参考码，指示一组四个事件：

- **SAV（有效行）**：行起始
- **EAV（有效行）**：行结束
- **SAV（消隐）**：帧间消隐周期内行起始
- **EAV（消隐）**：帧间消隐周期内行结束。

图 20描述了使用该模式时的帧结构。

图20. 内嵌码同步模式1下的帧结构



模式 2

在该模式下，内嵌同步码指示另一组事件：

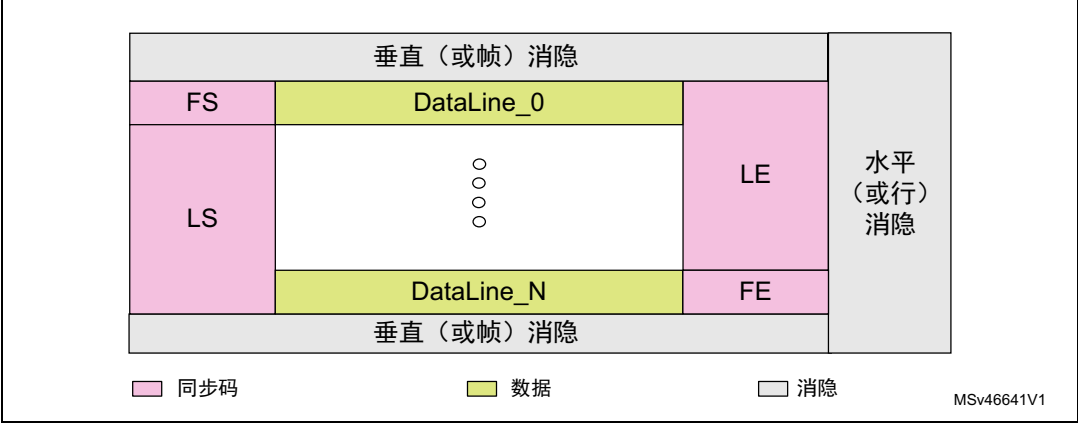
- 帧开始（FS）
- 帧结束（FE）
- 行开始（LS）
- 行结束（LE）

将0xFF值设定为帧结束（FE）意味着所有未使用的代码（除FS、LS和LE之外的可能代码值）都会被解读为有效FE码。

在该模式下，一旦使能照相机接口，将在首次出现FE码并且后接FS码之后开始捕获帧。

图 21描述了使用该模式时的帧结构。

图21. 内嵌码同步模式2下的帧结构



注：在隔行扫描模式下，照相机模块最多可以有8个同步码。因此，照相机接口不支持隔行扫描模式（否则，每帧数据的一半都会被丢弃）。
在使用内嵌码同步模式时，DCMI不支持压缩数据（JPEG）和裁剪功能。

内嵌码取消掩码

这些代码还用于指示行起始/结束或帧起始/结束。得益于这些代码，用户无需为了设置相应事件而将所有接收到的代码与设定码进行比较，而是可以只选择某些取消掩码的位与设定码的位置相同的位进行比较。

换句话说，用户通过配置DCMI内嵌码同步取消掩码寄存器（DCMI_ESUR）对相应代码应用掩码。该寄存器中的每个字节都是一个取消掩码码，对应于一个内嵌同步码：

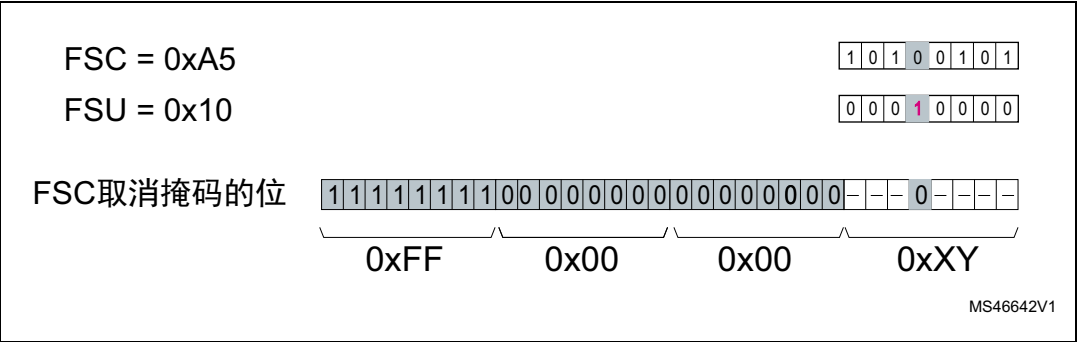
- 最高有效字节是帧结束定界符取消掩码（FEU）：将每一位置为1，表示必须将帧结束码中的这一位与接收到的数据进行比较，以便了解它是否是帧结束事件。
- 第二个字节是行结束定界符取消掩码（LEU）：将每一位置为1，表示必须将行结束码中的这一位与接收到的数据进行比较，以便了解它是否是行结束事件。
- 第三个字节是行起始定界符取消掩码（LSU）：将每一位置为1，表示必须将行起始码中的这一位与接收到的数据进行比较，以便了解它是否是行起始事件。
- 最低有效字节是帧起始定界符取消掩码（FSU）：将每一位置为1，表示必须将帧起始码中的这一位与接收到的数据进行比较，以便了解它是否是帧起始事件。

因此，每个事件可能有不同的代码（行起始或行结束或帧起始或帧结束），但它们（不同代码对应于一个事件）的取消掩码的位都在相同位置（相同取消掩码码）。

示例：FSC = 0xA5，取消掩码码FSU = 0x10。

这种情况下，帧开始信息被嵌入FS码的第4位。因此，用户必须只将所接收代码的第4位与设定码的第4位进行比较，以便了解它是否是帧开始事件。

图22. 内嵌码取消掩码



注：为避免同步错误，确保每个同步码具有不同的取消掩码码。

3.5 捕获模式

DCMI支持两种类型的捕获：**快照**（一帧）和**连续抓取**（连续多帧）。

根据DCMI_CR寄存器的配置，用户可以通过选择要捕获的字节、行和帧来控制捕获率。

这些功能用于转换图像的颜色格式和/或降低图像分辨率（通过每两行捕获一行，将垂直分辨率降低一半）。

有关详细信息，请参见第 3.7.2 节：图像大小调整（分辨率修改）。

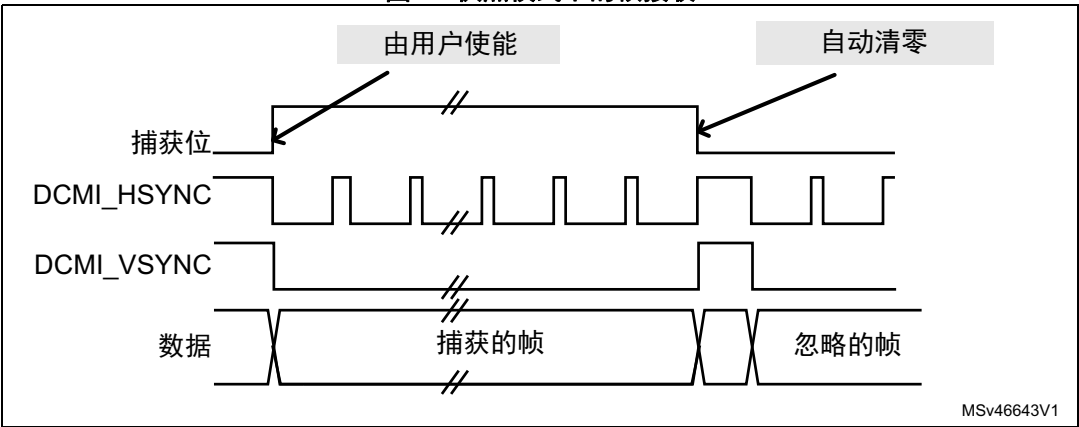
3.5.1 快照模式

在快照模式下，将捕获一个帧。在通过将DCMI_CR寄存器的CAPTURE位置1使能捕获后，接口在数据采样前等待帧起始（下一个DCMI_VSYNC或下一个内嵌帧起始码，取决于同步模式）检测。

在接收到第一个完整帧后，将自动禁用DCMI（CAPTURE位自动清零）并忽略所有其他帧。

如果发生溢出，该帧将丢失且照相机接口禁用。

图23. 快照模式下的帧接收



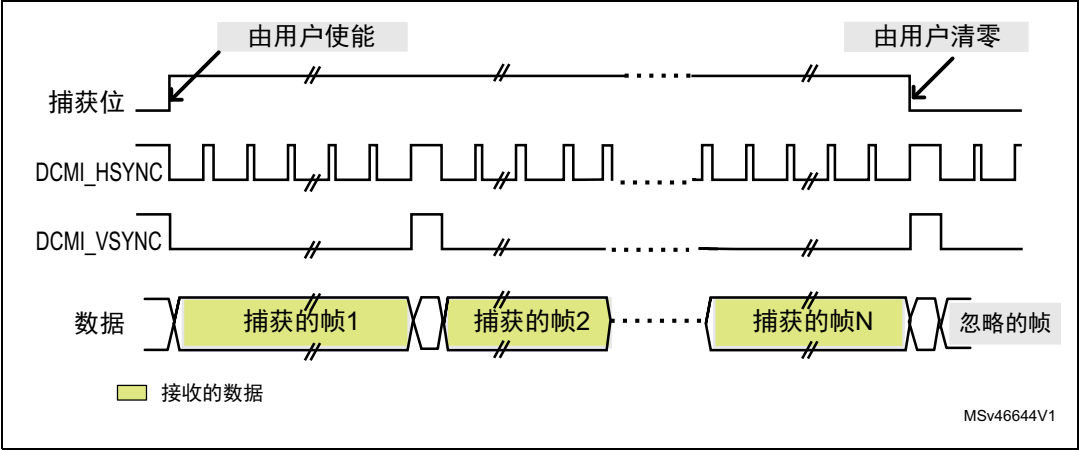
3.5.2 连续采集模式

在选择该模式并使能捕获（CAPTURE位置位）后，接口在数据采样前等待帧起始（下一个DCMI_VSYNC或下一个内嵌帧起始码，取决于同步模式）检测。

在该模式下，可以将DCMI配置为捕获所有帧、每隔一帧捕获一帧（减少50%带宽）或每四帧捕获一帧（减少75%带宽）。

在该模式下，照相机接口不自动禁用，但用户必须通过将CAPTURE位清零将其禁用。在被用户禁用后，DCMI继续抓取数据，直至当前帧结束。

图24. 连续抓取模式下的帧接收



3.6 数据格式和存储

DCMI支持以下数据格式：

- 8位逐行视频：单色或原始拜尔格式
- YCbCr 4:2:2 逐行视频
- RGB565逐行视频
- 压缩数据（JPEG）

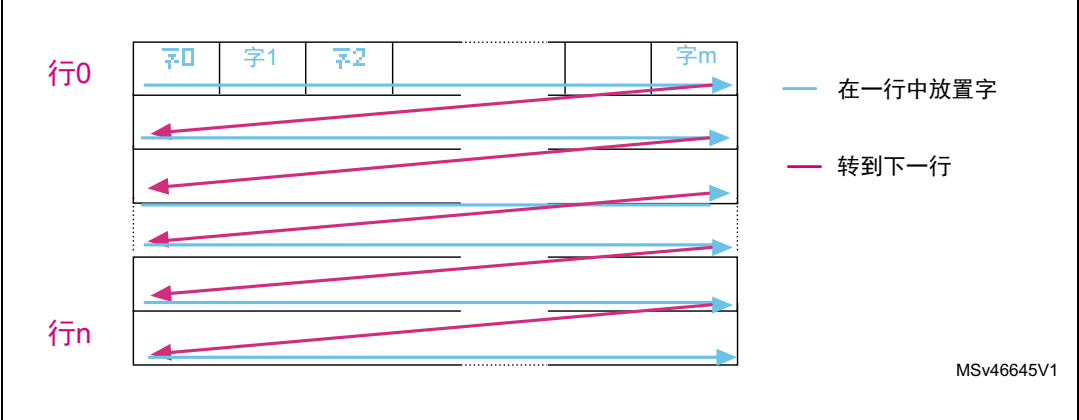
对于单色、RGB或YCbCr数据：

- 最大输入大小为2048 * 2048像素
- 以光栅模式保存帧缓冲区。

JPEG压缩数据没有大小限制。

对于单色、RGB和YCbCr格式，以光栅模式保存帧缓冲区，如图 25所示。

图25. 像素光栅扫描顺序



注： 仅使用32位字，并且仅支持小端序格式（最低有效字节保存在最小地址）。

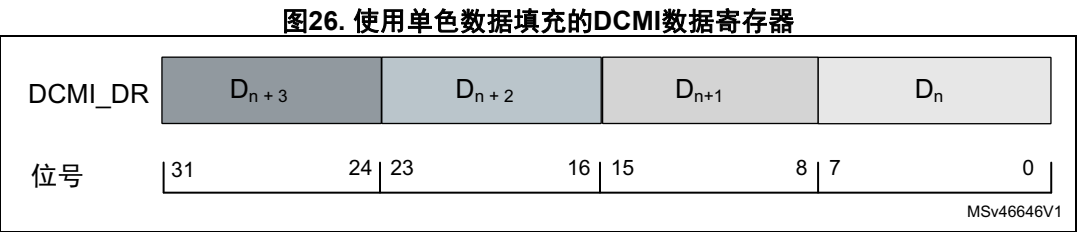
从照相机接收的数据可以按行/帧来组织（原始YUB/RGB/拜尔模式），也可以是一系列JPEG图像。

一行中的字节数不能是4的倍数。因此，用户在处理这种情况时应小心，因为每次从捕获的数据构建一个完整的32位字，就会生成一个DMA请求。当检测到帧结束并且尚未完整接收要传输的32位字时，将使用0填充其余数据，并生成DMA请求。

3.6.1 单色

DCMI支持每像素8位单色格式。

如果在配置DCMI时选择8位数据宽度，则数据寄存器的结构如图 26所示。



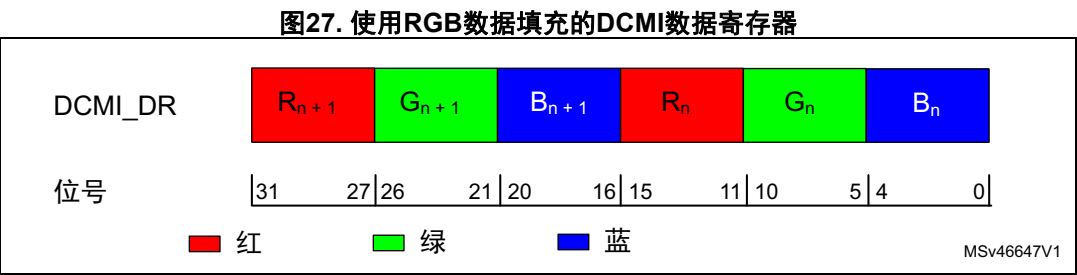
3.6.2 RGB565

RGB是指红色、绿色和蓝色，代表光的三种色调。任何颜色都是通过混合这三种颜色获得的。

565用于表示每个像素包含16位，其划分如下：

- 5位用于编码红色值（5个最高有效位）
- 6位用于编码绿色值
- 5位用于编码蓝色值（5个最低有效位）

每个分量具有相同的空间分辨率（4:4:4格式）。换句话说，每个样本都有一个红色（R）、一个绿色（G）和一个蓝色（B）分量。图 27所示为选择8位数据宽度时包含RGB数据的DCMI数据寄存器。



3.6.3 YCbCr

YCbCr是一系列将亮度与色度（色差）分离开来的色彩空间。

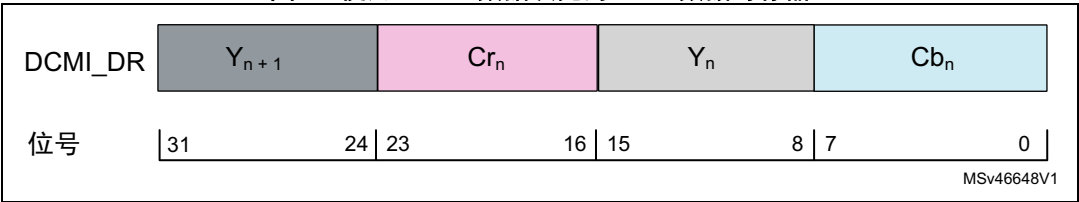
YCbCr包含三个分量：

- **Y**表示**亮度**（黑和白）
- **Cb**表示**蓝色差色度**
- **Cr**表示**红色差色度**。

YCbCr 4:2:2是下采样模式，水平方向需要一半的分辨率：在每两个水平Y样本中，有一个Cb或Cr样本。

每个分量（Y、Cb和Cr）编码为8位。[图 28](#)所示为选择8位数据宽度时包含YCbCr数据的DCMI数据寄存器。

图28. 使用YCbCr数据填充的DCMI数据寄存器



3.6.4 YCbCr，仅Y分量

注： 仅适用于STM32F446系列、STM32F469/479系列、STM32L496xx、STM32L4A6xx、STM32F7xxx器件和STM32H7x3系列。

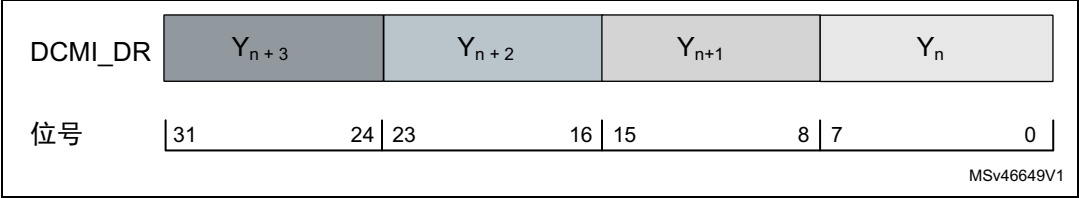
缓冲区仅包含Y分量信息 - 单色图像。

在此模式下，将丢弃色度信息。仅保存每个像素的亮度分量（编码为8位）。

因此，单色图像具有原始图像一半的水平分辨率（YCbCr数据）。

[图 29](#)所示为选择8位数据宽度时的DCMI寄存器。

图29. 仅使用Y分量数据填充的DCMI数据寄存器



3.6.5 JPEG

对于压缩数据（JPEG），DCMI仅支持硬件同步，并且不限制输入大小。

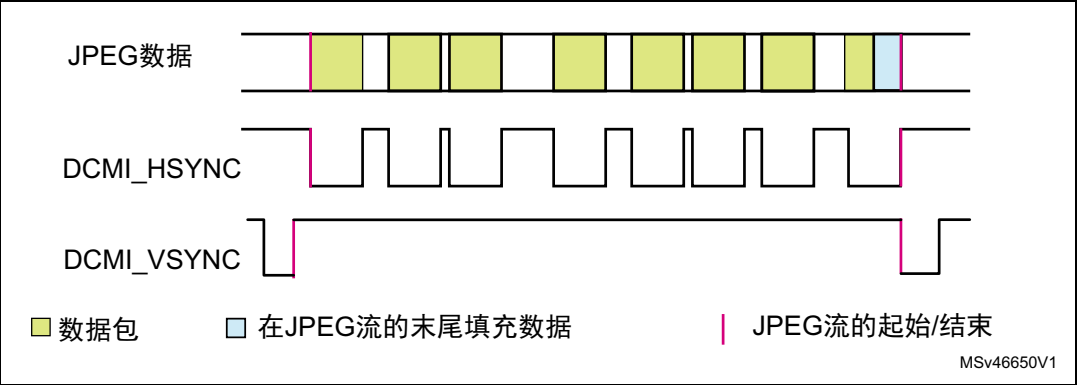
每个JPEG流分为多个包，其大小可设定。包的调度取决于图像内容，并在两个包之间的时间形成可变消隐。

要允许接收JPEG图像，必须将DCMI_CR寄存器中的JPEG位置1。JPEG图像不按行和帧存储，因此DCMI_VSYNC信号用于启动捕获，而DCMI_HSYNC则用作数据使能信号。

如果完整数据流结束但没有发生流结束检测（DCMI_VSYNC无变化），DCMI将通过插入零填充帧结束。换句话说，如果流的大小不是4的倍数，DCMI将在流的末尾用零填充剩余数据。

注：裁剪功能和内嵌码同步模式不能用于JPEG格式。

图30. JPEG数据接收



3.7 其他功能

3.7.1 裁剪功能

照相机接口可通过裁剪功能从收到的图像中选择一个矩形窗口。

在32位寄存器DCMI_CWSTRT中指定起始坐标（左上角）。

在32位寄存器DCMI_CWSIZE中，以像素时钟数（水平尺寸）和行数（垂直尺寸）指定窗口大小。

3.7.2 图像大小调整（分辨率修改）

注：仅STM32L496xx、STM32L4A6xx、STM32F446系列、STM32F469/479系列、STM32F7x5系列、STM32F7x6系列、STM32F7x7系列、STM32F7x8系列、STM32F7x9系列和STM32H7x3系列提供图像大小调整功能。

如第 3.5 节：捕获模式所述，DCMI捕获功能通过DCMI_CR寄存器进行设置。

DCMI可捕获：

- 所有接收的行
- 每两行中的一行（这种情况下，用户可以选择捕获奇数行或偶数行）。

该功能影响可通过DCMI接收的垂直分辨率，垂直分辨率可以从照相机模块接收或将图像分辨率除以2得到（只接收奇数行或偶数行）。

该接口还能捕获：

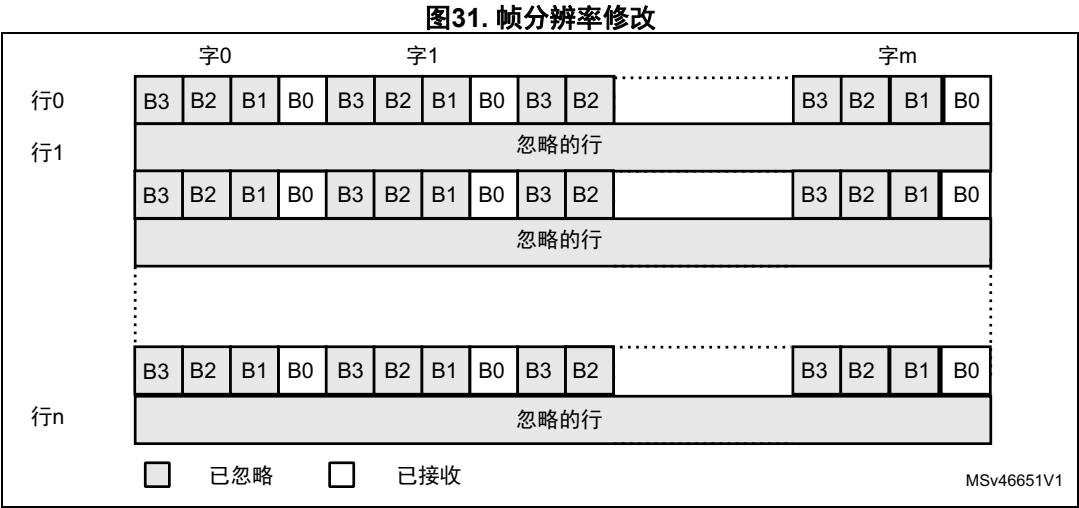
- 所有接收的数据
- 所接收数据中每隔一个字节（每两个字节中的一个字节。换句话说，只接收奇数或偶数字节）
- 四个中的一个字节
- 四个中的两个字节

该功能影响水平分辨率，允许用户选择以下分辨率之一：

- 完整的水平分辨率
- 水平分辨率的一半
- 水平分辨率的四分之一（该功能仅适用于每像素8位的数据格式）。

注： 使用该功能时需小心。对于某些数据格式（色彩空间），水平分辨率的修改会改变数据格式。
例如，当数据格式为YCbCr时，接收到的数据为隔行扫描数据（CbYCrYCbYCr）。当用户选择每隔一个字节接收一个字节时，DCMI只接收每个样本的Y分量，这意味着将YCbCr数据转换为仅Y分量数据。这种转换会影响水平分辨率（仅接收图像的一半数据）和数据格式。

图 31显示了只接收每四个字节中的一个字节和每两行中的一行时的一个帧。



3.8 DCMI 中断

可生成5个中断：

- **IT_LINE**表示行结束。
- **IT_FRAME**表示帧捕获结束。
- **IT_OVR**表示数据接收发生溢出。
- **IT_VSYNC**表示同步帧。
- **IT_ERR**表示在内嵌同步码顺序中检测到错误（仅内嵌码同步模式）。

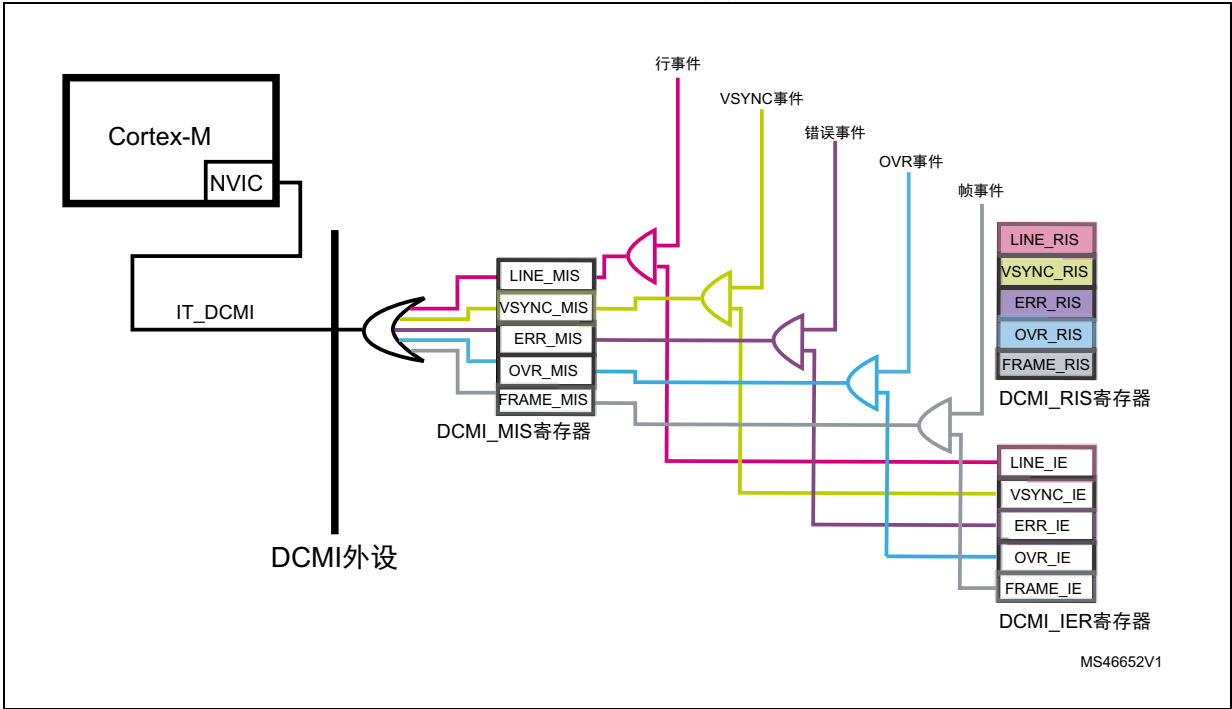
可通过软件屏蔽所有中断。全局中断`dcmi_it`是所有单个中断的逻辑或运算所得结果。

如 图 32 所示，通过三个寄存器处理DCMI中断：

- **DCMI_IER**：读/写寄存器，允许在发生相应事件时生成中断
- **DCMI_RIS**：只读寄存器，在通过DCMI_IER寄存器掩蔽相应中断之前给出该中断的当前状态（每一位给出可在DCMI_IER寄存器中使能或禁用的中断状态）。
- **DCMI_MIS**：只读寄存器，根据DCMI_IER和DCMI_RIS寄存器提供相应中断的当前掩蔽状态。

如果发生事件并使能了相应中断，将生成DCMI全局中断。

图32. DCMI中断和寄存器



3.9 低功耗模式

STM32电源模式对DCMI外设直接影响。因此，必须了解不同功耗模式下的DCMI外设操作。

在**运行模式**下，DCMI和所有外设均正常操作。

在**睡眠模式**下，DCMI和所有外设均正常操作，并生成中断以唤醒CPU。

在**停止模式**和**待机模式**下，DCMI不工作。

STM32L496xx和STM32L4A6xx器件有其他低功耗模式，这些模式下的DCMI状态各不相同：

- **低功耗运行模式**
- **低功耗睡眠模式**：外设中断导致器件退出该模式。
- **Stop 0、Stop 1和Stop 2模式**：外设寄存器内容被保留。
- **关断模式**：在退出关断模式时，必须重新初始化外设。

表 5总结了不同模式下的DCMI操作。

表5. 低功耗模式下的DCMI操作

模式	DCMI 操作
运行	有效
低功耗运行 ⁽¹⁾	
睡眠	
低功耗睡眠 ⁽¹⁾	
停止	冻结
Stop 0 ⁽¹⁾	
Stop 1 ⁽¹⁾	
Stop 2 ⁽¹⁾	
待机	掉电
关机 ⁽¹⁾	

1. 仅适用于STM32L496xx和STM32L4A6xx器件。

4 DCMI 配置

在选择与STM32MCU相连的照相机模块时，用户应考虑一些这样的参数：像素时钟、支持的数据格式和分辨率。

为了正确地实现其应用，用户需要执行以下配置：

- 配置GPIOs。
- 配置时序和时钟。
- 配置DCMI外设。
- 配置DMA。
- 配置照相机模块：
 - 配置I2C以便进行照相机模块的配置和控制
 - 设置参数，例如对比度、亮度、色彩效果、极性和数据格式。

注： *建议在开始配置前复位DCMI外设和照相机模块。DCMI可通过置位RCC_AHB2RSTR寄存器中的相应位来复位，这可以复位时钟域。*

4.1 GPIO配置

为了简化DCMI GPIO（例如数据引脚、控制信号引脚和照相机配置引脚）的配置并避免任何引脚冲突，建议使用STM32CubeMX配置和初始化代码生成器。

得益于STM32CubeMX，用户可以生成预配置了所有必要外设的项目。

根据通过配置DCMI_CR寄存器中的EDM位选择的扩展数据模式，DCMI可接收每像素时钟8、10、12或14位数据（DCMI_PIXCLK）。对于硬件同步，用户需要为DCMI配置11、13、15或17个GPIO。

对于内嵌码同步，必须配置9个GPIO（8个数据引脚和1个用于DCMI_PIXCLK的引脚）

用户还需要配置I2C，某些情况下还需要配置照相机电源引脚（如果照相机电源为STM32 MCU）

中断使能

为了能够使用DCMI中断，用户应该在NVIC端启用DCMI全局中断。然后，通过使能DCMI_IER寄存器中的相应使能位单独使能每种中断。

在硬件同步模式下，只能使用四种中断（IT_LINE、IT_FRAME、IT_OVR和IT_DCMI_VSYNC），而在内嵌码同步模式下，可以使用全部五种中断。

软件允许用户通过检查标志状态的方式检查指定的DCMI中断是否发生。

4.2 时钟和定时配置

本节描述了时序和时钟配置步骤。

4.2.1 系统时钟配置（HCLK）

建议使用最高的系统时钟，以获得最佳的性能。

此建议也适用于外部存储器的帧缓冲区。

如果对帧缓冲区使用外部存储器，则应将时钟设置为允许的最高速度以获得最佳存储器带宽。

示例：

- STM32F4x9xx器件：最大系统速度为180 MHz。如果外部SDRAM连接到FMC，则最大SDRAM时钟为90 MHz（HCLK/2）。
- STM32F7系列：最大系统速度为216 MHz。使用此速度和HCLK/2预分频器时，SDRAM速度超过最大允许速度（更多详细信息，请参见产品数据手册）。要获得最大的SDRAM，建议将HCLK配置为200 MHz，然后将SDRAM速度设置为100 MHz。

具有最高性能的时钟配置如下：

- 对于STM32F2x7系列，为HCLK @ 120 MHz和SRAM @ 60 MHz
- 对于STM32F407/417系列，为HCLK @ 168 MHz和SRAM @ 60 MHz
- 对于STM32L4x6系列，为HCLK @ 80 MHz和SRAM @ 40 MHz

4.2.2 DCMI时钟和定时配置（DCMI_PIXCLK）

DCMI像素时钟配置取决于照相机模块的像素时钟配置。用户必须确保DCMI和照相机模块端具有相同的像素时钟配置。

DCMI_PIXCLK是用于输入数据采样的DCMI的输入信号。用户通过配置DCMI_CR寄存器中的PCKPOL位来选择对上升沿或下降沿进行数据捕获。

如 [第 3.4节：数据同步](#) 所述，有两种类型的同步：内嵌码同步和硬件同步。为了给应用选择需要的同步模式，用户需要配置DCMI_CR寄存器中的ESS位。

硬件（外部）同步

使用DCMI_HSYNC和DCMI_VSYNC信号。通过选择DCMI_CR寄存器中VSPOL和HSPOL位的信号有效电平（高或低）来定义这两个信号的配置。

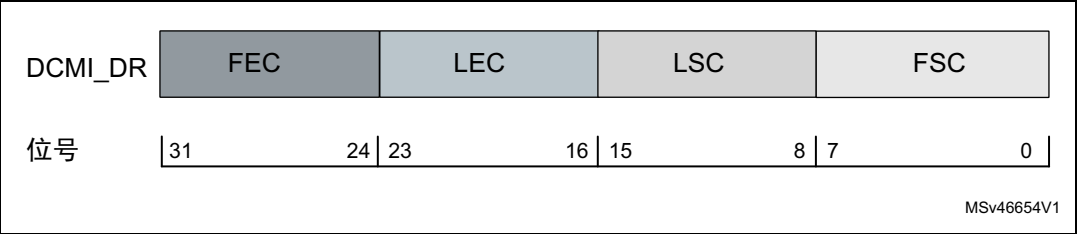
注：

用户必须确保DCMI_HSYNC和DCMI_VSYNC极性的设定符合照相机模块的配置。
在硬件同步模式（DCMI_CR寄存器的ESS位清零）下，将生成IT_VSYNC中断（如已使能），即使DCMI_CR寄存器的CAPTURE位清零。为了进一步降低帧捕获率，可结合快照模式，使用IT_VSYNC对两次捕获之间的帧数进行计数。这不适用于内嵌码同步模式。

内嵌码（内部）同步

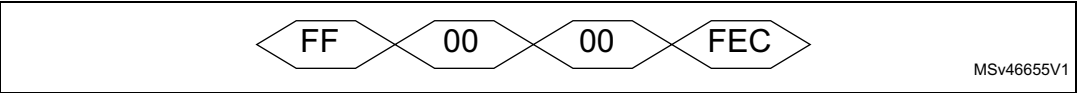
行起始或行结束和帧起始或帧结束由数据流中内嵌的代码或标记决定。仅当并行数据接口宽度为8位时，才支持内嵌同步码。同步码必须在DCMI_ESCR寄存器中设定，如 图 33所示。

图33. DCMI_ESCR寄存器字节



- FEC（帧结束码）：**最高有效字节指定帧结束定界符。照相机模块发送32位字，其中包含0xFF 00 00 XY（XY = FEC码），用于指示帧结束。接收到的代码如 图 34所示。

图34. FEC结构



在接收该FEC码之前，DCMI_SR寄存器中VSYNC位的值必须置1以指示有效帧。在接收FEC码之后，VSYNC位的值必须为0，以指示帧之间同步。该VSYNC位的值必须保持为0，直至收到下一个帧起始码。

如果FEC值等于0xFF（照相机模块发送0xFF 00 00 FF），则将所有未使用的代码解读为帧结束码。有253个值对应于帧结束定界符（0xFF0000FF和252个未使用代码）。

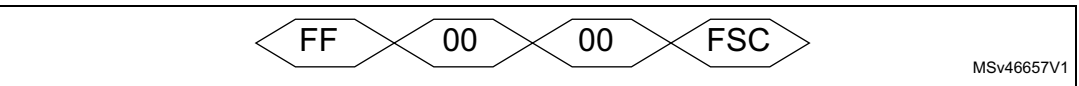
- LEC（行结束码）：**该字节指定行结束标记。从照相机接收到的指示行结束的代码为0xFF 00 00 XY，其中XY = LEC码。

图35. LEC结构



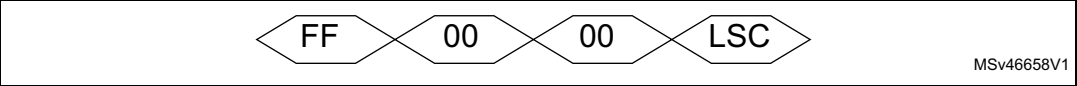
- FSC（帧起始码）：**该字节指定帧起始标记。从照相机接收到的指示新帧起始的代码为0xFF 00 00 XY，其中XY = FSC码。

图36. FSC结构



- **LSC（行起始码）**：该字节指定行起始标记。从照相机接收到的指示新行起始的代码为 0xFF 00 00 XY，其中XY = LSC码。
如果LSC设定为0xFF，则照相机模块不发送帧起始定界符。DCMI将FEC码后首次出现的LSC码解读为出现FSC码。

图37. LSC结构

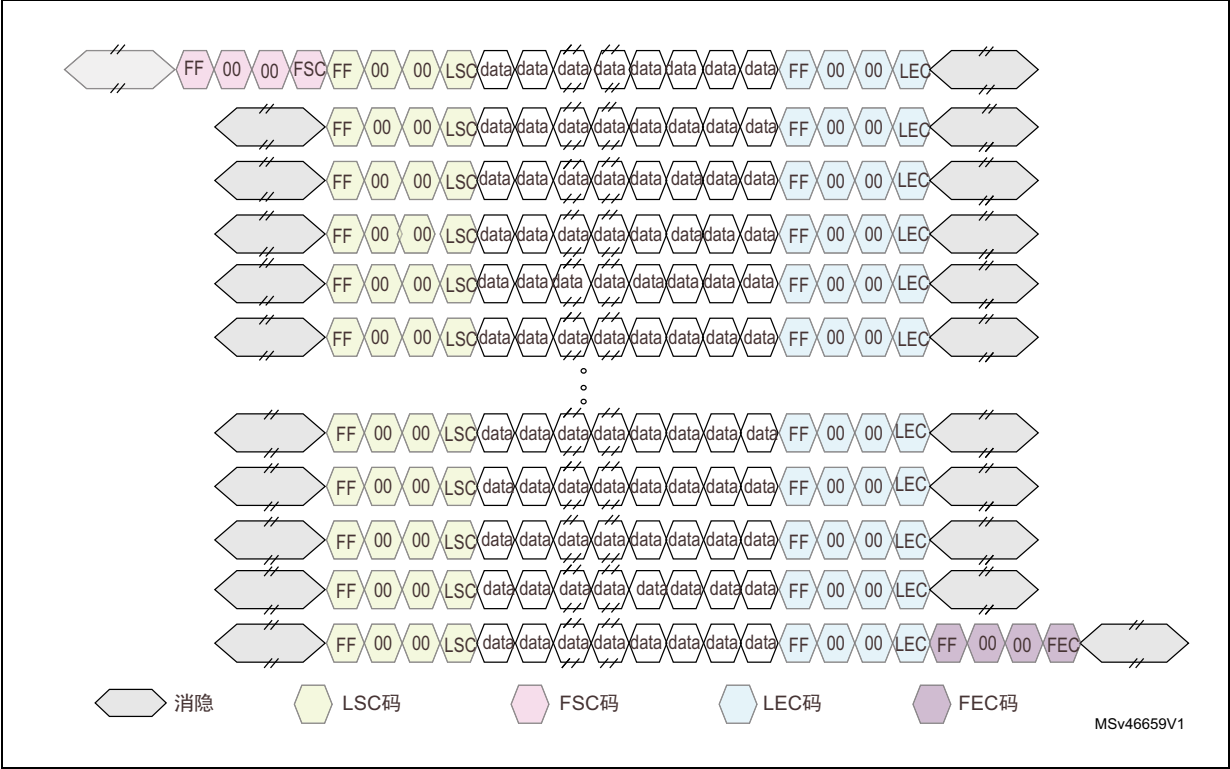


在该内嵌码同步模式下，HSPOL和VSPOL位均被忽略。当DCMI接收数据（DCMI_CR寄存器中的CAPTURE位置位）时，用户可以通过读取DCMI_SR寄存器中的VSYNC和HSYNC位监控数据流，以便了解它是有效行/帧还是行/帧之间的同步。

如果使能了DCMI_IER寄存器中的ERR_IE位，将在每次发生错误（例如，内嵌码同步字符的接收顺序不正确）时生成中断。

图 38所示为在内嵌码同步模式下接收的帧。

图38. 内嵌码同步模式下的帧结构



4.3 DCMI 配置

DCMI配置允许用户选择捕获模式、数据格式、图像大小和分辨率

4.3.1 捕获模式 (Capture mode)

用户可通过选择以下模式捕获图像或视频：

- 连续抓取模式，可连续捕获帧（图像）
- 快照模式，可捕获单一帧。

在快照或连续抓取模式下接收的数据由DMA传输至存储器帧缓冲区。缓冲区的位置和模式（线性或圆形缓冲区）由系统DMA控制。

4.3.2 数据格式

如前文所述，DCMI能够接收压缩数据（JPEG）或许多未压缩数据格式（例如单色、RGB和YCbCr）。

有关详细信息，请参见 [第 3.6节：数据格式和存储](#)。

4.3.3 图像分辨率和大小

DCMI能够接受各种分辨率（低、中和高）和图像大小，因为图像大小取决于图像分辨率和数据格式。DMA负责确保已接收图像的传输和在存储器帧缓冲区中的定位放置。

或者，用户可以配置字节、行和帧选择模式，以便修改图像分辨率和大小，某些情况下还可以修改数据格式。用户还可以配置和使能裁剪功能，以便在接收到的图像上选择一个矩形窗口。

有关这两种功能的更多详细信息，请参见 [第 3.7节：其他功能](#)。

注： *在使能DCMI_CR寄存器中的ENABLE位之前，应正确设定DCMI配置寄存器。
在使能DCMI_CR寄存器中的CAPTURE位之前，必须正确设定DMA控制器和所有DCMI配置寄存器。*

4.4 DMA 配置

DMA配置是保证应用成功的关键步骤。

如 [第 2.3节：智能架构中的DCMI](#)所述，对于所有内置DCMI的STM32器件，DMA2确保从DCMI至存储器（内部SRAM或外部SRAM/SDRAM）的传输，但STM32H7x3xx器件除外，其DMA1还可以访问AHB2外设并确保已接收数据从DCMI至存储器帧缓冲区的传输。

4.4.1 用于DCMI至存储器传输的DMA常用配置

对于DCMI至存储器的传输：

- 传输方向必须是外设至存储器，通过配置DMA_SxCR寄存器中的DIR[1:0]位来实现。在这种情况下：
 - 必须在DMA_SxPAR寄存器中写入源地址（DCMI数据寄存器地址）。
 - 必须在DMA_SxMAR寄存器中写入目标地址（内部SRAM或外部SRAM/SDRAM中的帧缓冲区地址）。
- 为确保从DCMI数据寄存器传输数据，DMA等待DCMI生成请求。因此，必须配置相关的流和通道。有关详细信息，请参见 [第 4.4.3 节：DCMI通道和流的配置](#)。
- 由于每次填充DCMI数据寄存器时都会生成DMA请求，从DCMI传输至DMA2（对于STM32H7x3xx器件，为DMA1）的数据必须具有32位宽度。因此，DMA_SxCR寄存器中PSIZE位设定的外设数据宽度必须为32位字。
- DMA是流控制器：要传输的32位数据字的数量可在DMA_SxNDTR寄存器（对于STM32L4x6系列，为DMA_CNDTRx）中进行软件设定，范围为1至65535。有关该寄存器的详细信息，请参见 [第 4.4.4 节：DMA_SxNDTR 寄存器](#)。

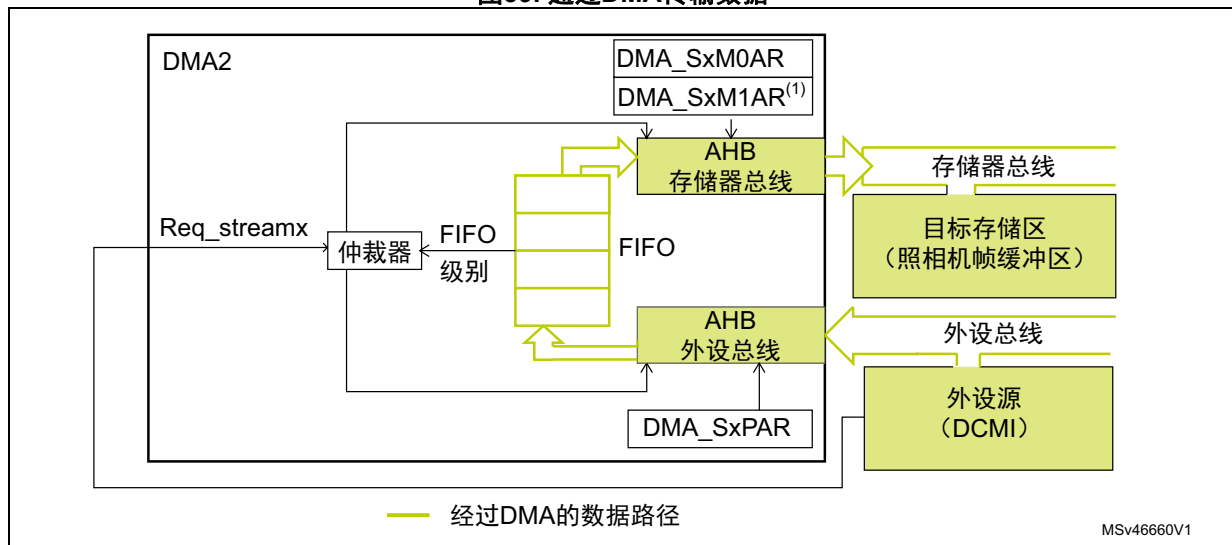
DMA 可在以下两种模式下工作：

- 直接模式：从DCMI接收的每个字被传输到存储器帧缓冲区。
- FIFO模式：DMA使用其内部FIFO确保批量传输（多个字从DMA FIFO传输至目标存储区）

有关DMA内部FIFO的详细信息，请参见 [第 4.4.5 节：FIFO和批量传输的配置](#)。

[图 39](#)所示为外设至存储器模式下的DMA2（对于STM32H7x3xx器件，为DMA1）操作（STM32L496xx和STM32L4A6xx器件除外，因为这些器件中的DMA2只有一个端口）。

图39. 通过DMA传输数据



1. 在双缓冲区模式下配置DMA_SxM1AR寄存器。

4.4.2 根据图像大小和捕获模式设置DMA

必须根据图像大小（色深和分辨率）和捕获模式配置DMA：

- 在**快照**模式下：DMA必须确保一个帧（图像）从DCMI到所需存储器的传输：
 - 如果以字计的图像大小不超过65535，则流可以配置为正常模式。有关该模式的更多详细描述，请参见第 4.4.6 节：快照捕获中用于低分辨率的正常模式。
 - 如果以字计的图像大小介于65535和131070之间，则流可以配置为双缓冲区模式。有关该模式的更多详细描述，请参见第 4.4.8 节：用于中等分辨率的双缓冲区模式（快照或连续捕获）。
 - 如果以字计的图像大小超过131070，则流不能配置为双缓冲区模式。有关必须使用的模式的更多详细描述，请参见第 4.4.9 节：用于更高分辨率的DMA配置。
- 在**连续**模式下：DMA必须确保连续多个帧（图像）从DCMI到所需存储器的传输。每次DMA完成一帧的传输时，将开始下一帧的传输：
 - 如果一幅图像以字计的图像大小不超过65535，则流可以配置为循环模式。有关该模式的更多详细描述，请参见第 4.4.7 节：连续捕获中用于低分辨率的循环模式。
 - 如果一幅以字计的图像大小介于65535和131070之间，则流可以配置为双缓冲区模式。有关该模式的更多详细描述，请参见第 4.4.8 节：用于中等分辨率的双缓冲区模式（快照或连续捕获）。
 - 如果一幅以字计的图像大小超过131070，则流不能配置为双缓冲区模式。有关必须使用的模式的更多详细描述，请参见第 4.4.9 节：用于更高分辨率的DMA配置。

4.4.3 DCMI通道和流的配置

用户还必须配置相应的DMA2（对于STM32H7x3xx器件，为DMA1）流和通道，以确保每次填充DCMI数据寄存器时的DMA确认。

表 6总结了来自DCMI的DMA通道使能DMA请求。

表6. 不同STM32器件的DMA流的选择

STM32系列	DMA数据流	通道
STM32F2	数据流 1	通道1或通道7
STM32F4		
STM32F7		
STM32L4	数据流 0	通道 6
	数据流 4	通道 5
STM32H7	数据流 0 数据流 1 数据流 2 数据流 3 数据流 4 数据流 5 数据流 6 数据流 7	复用器1请求74

注：有关流配置程序的分步描述，请参考相关的STM32参考手册。

4.4.4 DMA_SxNDTR 寄存器

注：在STM32L496xx和STM32L4A6xx器件中，该寄存器被称为DMA_CNDTRx。

由用户在该寄存器中设定从外设源（DCMI）传输至目标存储区的总字数。

当DMA开始从DCMI向存储器传输数据时，项目数从初始设定值开始下降，直至传输结束（达到零或在剩余数据的数量达到零之前通过软件禁用流）。

表 7总结了设定值和外设数据宽度（PSIZE位）对应的字节数：

表7. 一次DMA传输中传输的最大字节数

DMA_SxNDTR设定值	外设容量	字节数
65535	字	262140
否 ⁽¹⁾	字	4 * N

1. 0 < N < 65535.

注： 为避免数据损坏，在DMA_SxNDTR中设定的值必须是MSIZE值/PSIZE值的倍数。

4.4.5 FIFO和批量传输的配置

DMA在4字FIFO使能或没有使能时执行传输。如前文所述，当FIFO使能时，源数据宽度（在PSIZE位设定）可能不同于目标数据宽度（在MSIZE位设定）。这种情况下，用户必须注意将DMA_SxPAR和DMA_SxM0AR（对于双缓冲区模式配置，还有DMA_SxM1AR）中要写入的地址修改为在DMA_SxCR寄存器的PSIZE和MSIZE位设定的数据宽度。为了获得更好的性能，建议使用FIFO。

当FIFO模式使能时，用户可以配置MBURST位，以使DMA从其内部FIFO向目标存储区执行批量传输（最多4个字），从而保证更好的性能。

4.4.6 快照捕获中用于低分辨率的正常模式

低分辨率图像是指大小（以32位字计）小于65535的图像。

在快照模式下，可以使用正常模式确保低分辨率帧的传输（参见表 7）。

表 8总结了使用正常模式可以传输的最大图像大小。

表8. 正常模式下的最高图像分辨率

项目	最大字节数	位深（每像素字节数） ⁽¹⁾	最大像素数	最高分辨率
字	262140	1	262140	720x364
		2	131070	480x272

1. 最大像素数取决于图像的位深（每个像素的字节数）。
DCMI支持两种可能的位深：
- 单色或仅Y分量格式下，每个像素1个字节
- RGB565或YCbCr格式下，每个像素2个字节。

4.4.7 连续捕获中用于低分辨率的循环模式

低分辨率图像是指大小（以32位字计）小于65536的图像。

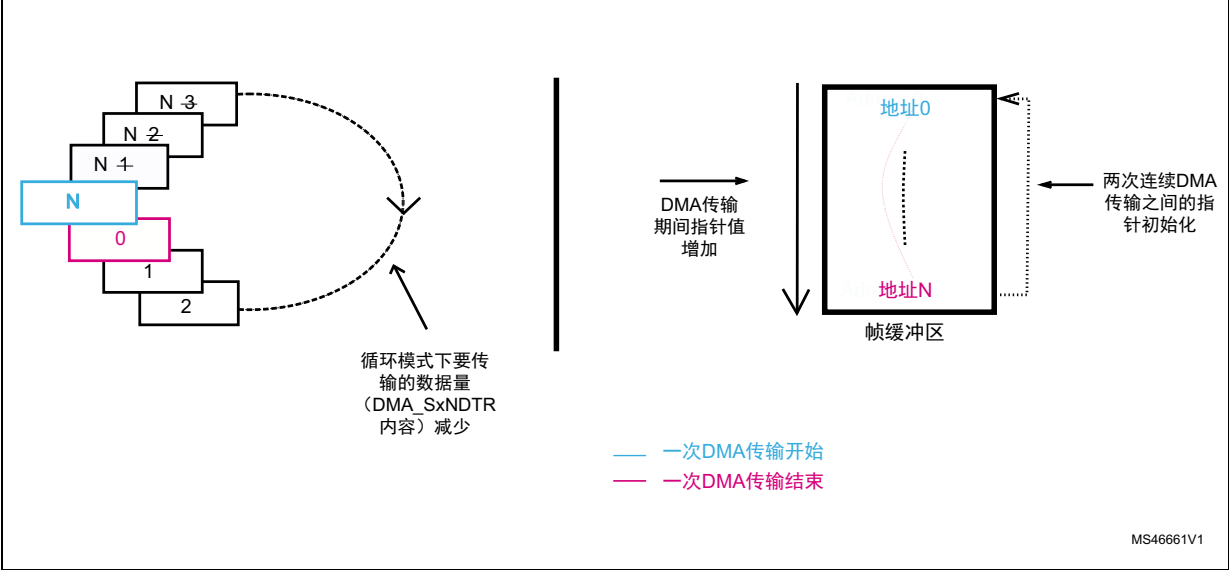
该循环模式能够处理连续帧（连续数据流），前提是一个帧的大小（DMA_SxNDTR寄存器（STM32L4系列为DMA_CNDTR）中的初始设定值）小于65535。

每当数据的数量递减至零时，数据字的数量自动重载为初始值。并且，每当DMA指针达到帧缓冲区末尾时，它将重新初始化（返回DMA_SxM0AR中的设定地址），同时DMA确保下一帧的传输。

表 8中列出的分辨率对连续模式下的低分辨率也有效。

图 40显示DMA传输期间和两次连续的DMA传输之间的DMA_SxNDTR值和帧缓冲区指针修改。

图40. 循环模式下的帧缓冲区和DMA_SxNDTR寄存器



4.4.8 用于中等分辨率的双缓冲区模式
(快照或连续捕获)

注：STM32L4A6xx和STM32L496xx器件不提供该模式。

中等分辨率图像是指大小（以32位字计）介于65536和131070之间的图像。

使能双缓冲区模式时，自动使能循环模式。

如果图像大小超过（以字计）表 8所示快照或连续捕获模式下的最大大小，则必须在快照或连续模式下使用双缓冲区模式。这种情况下，允许的每帧像素数将翻倍，因为接收的数据保存在两个缓冲区中，每一个的最大大小（以32位字计）为65535（最大帧大小为131070字或524280字节）。因此，允许DCMI接收和DMA传输的图像大小和分辨率翻倍，如表 9所示。

表9. 双缓冲区模式下的最高图像分辨率

项目	最大字节数	位深（每像素字节数）	SxNDTR寄存器中的设定值	像素数	最高分辨率
字	524280	1	65535	524280	960x544
		1	否 ⁽¹⁾	8 * N	960x544
		2	65535	262140	720x364
		2	否 ⁽¹⁾	4 * N	720x364

1. 0 < N < 65536。

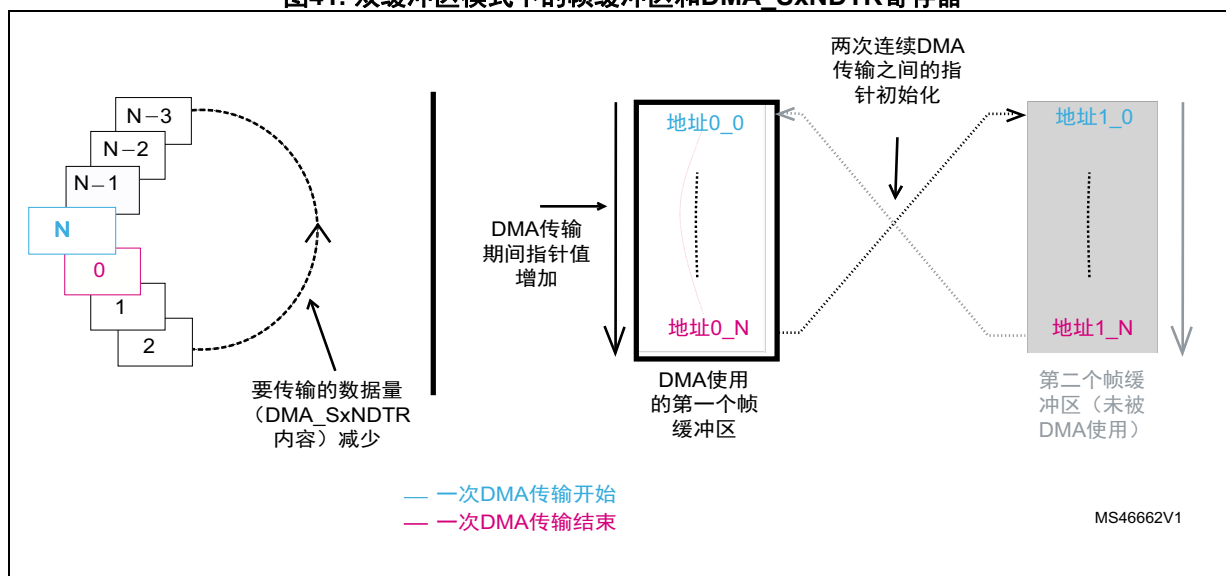
在该模式下，双缓冲区流有两个指针（两个用于存储数据的缓冲区），在每次事务结束时切换：

- 在快照模式下，DMA控制器将数据写入第一个帧缓冲区。当第一个帧缓冲区装满（达到该水平时，SxNDTR寄存器重新初始化为设定值且DMA指针切换至第二个帧缓冲区）时，数据将传输至第二个缓冲区。也就是说，将帧总大小（以字计）除以2并在SxNDTR寄存器中设定，将图像保存在两个大小相等的缓冲区中。
- 在连续模式下，每次接收一个帧（图像）并保存在两个缓冲区中，由于使能了循环模式，SxNDTR寄存器被重新初始化为设定值（帧总大小除以2），并且DMA指针切换至第一个帧缓冲区以便接收下一帧。

通过将 DMA_SxCR 寄存器中的 DBM 位置1, 即可使能双缓冲区模式。

图 41所示为DMA传输期间的两个指针和DMA SxNDTR值修改。

图41. 双缓冲区模式下的帧缓冲区和DMA SxNDTR寄存器



4.4.9 用于更高分辨率的DMA配置

当快照或连续模式下一帧（图像）的字数超过131070时，以及图像分辨率超过表 9所示值时，DMA双缓冲区模式无法确保已接收数据的传输。

注：本节只强调了高分辨率时的DMA操作。第 6.3.6 节：SxGA 分辨率的捕获（YCbCr 数据格式）中描述了使用该DMA配置开发的示例。

STM32F2、STM32F4、STM32F7和STM32H7系列的双缓冲区模式内置一项十分重要的功能：能够在流使能时实时更新AHB存储器端口的设定地址（DMA_SxM0AR或DMA_SxM1AR中）。必须考虑以下条件：

- 当DMA_SxCR寄存器中的**CT**位置为**0**（当前目标存储区为存储区0）时，DMA_Sx**M1**AR寄存器可以写入。
在CT置为1时尝试写入该寄存器，将生成错误标志（TEIF）并自动禁用流。

- 当DMA_SxCR寄存器中的**CT**位置为**1**（当前目标存储区为存储区1）时，DMA_SxM0AR寄存器可以写入。
在CT置为0时尝试写入该寄存器，将生成错误标志（TEIF）并自动禁用流。

为避免任何错误情况，建议在断言TCIF标志后立即更改设定地址。这时，必须将目标存储区从存储区0更改为存储区1（或从1更改为0），具体取决于DMA_SxCR寄存器中CT位的值。

注： 对于所有其他模式（双缓冲区模式除外），一旦使能流，存储器地址寄存器即被写保护。

然后，DMA可以进行两个以上缓冲区的管理：

- 在第一个周期中，当DMA使用通过**指针0**寻址（DMA_SxM0AR寄存器中的存储区0地址）**缓冲区0**时，**缓冲区1**可以通过**指针1**寻址（DMA_SxM1AR寄存器中的存储区1地址）。
- 在第二个周期中，当DMA使用通过**指针1**寻址的**缓冲区1**时，可以更改缓冲区0的地址并且**帧缓冲区2**可以通过**指针0**寻址。
- 在第二个周期中，当DMA使用通过**指针0**寻址的**缓冲区2**时，可以更改帧缓冲区1的地址并且**缓冲区3**可以通过**指针1**寻址。

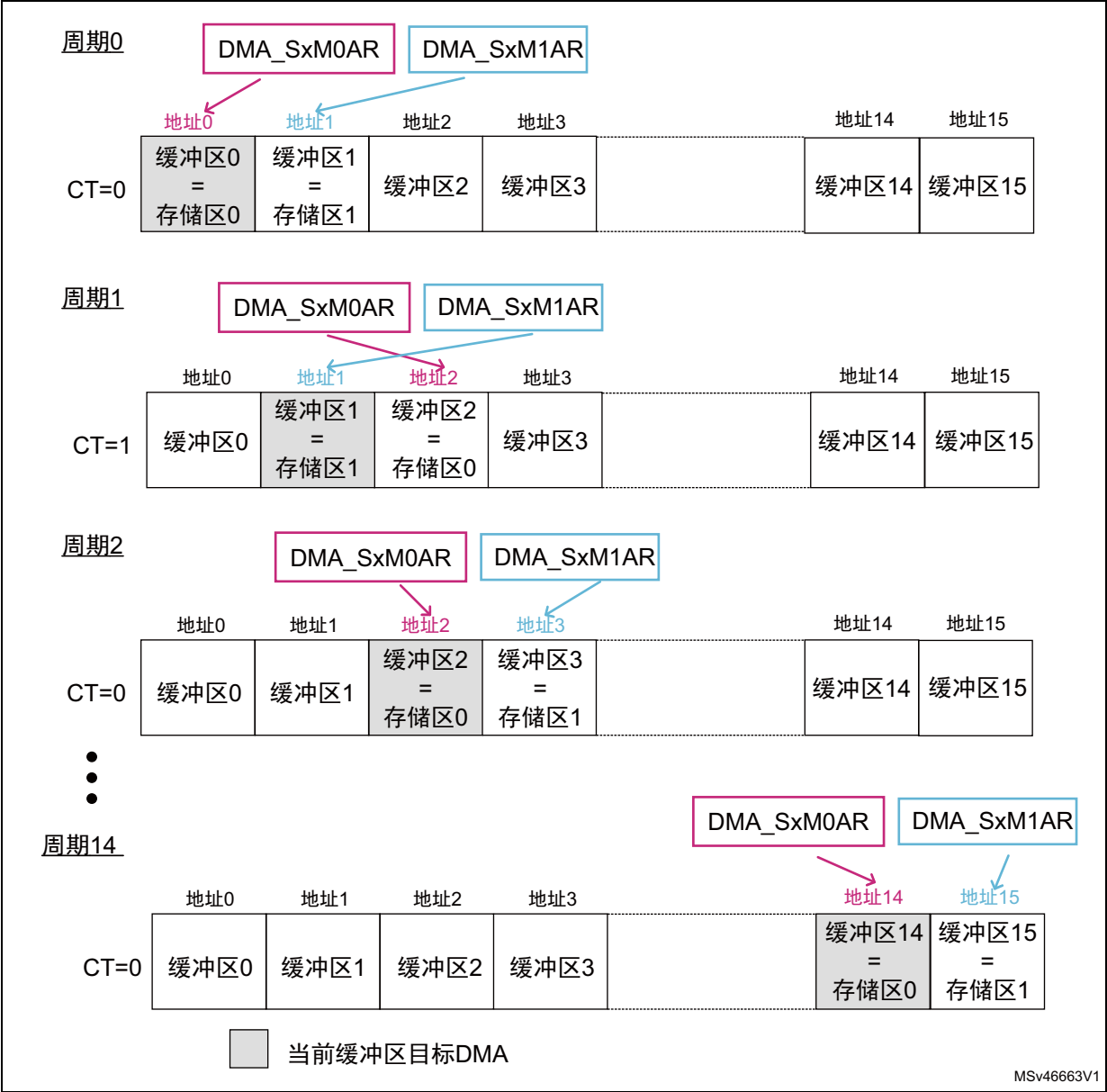
然后，DMA可以使用它的两个寄存器DMA_SxM0AR和DMA_SxM1AR对许多缓冲区进行寻址，确保高分辨率图像的传输。

注： 为了简化该特殊功能的使用，建议将图像分给相等的缓冲区。
在捕获高分辨率图像时，用户必须确保目标存储区具有足够的容量。

示例：对于SxGA分辨率（1280x1024），图像大小为655360字（32位）。必须将图像大小分成相等的缓冲区，每一个的最大容量为65535。然后，为了正确接收，必须将图像分给16个帧缓冲区，每个帧缓冲区的容量为40960（小于65535）。

 **图 42**所示为DMA传输期间DMA_SxM0AR和DMA_SxM1AR寄存器的更新：

图42. 高分辨率时的DMA操作



4.5 相机模块配置

为了正确配置照相机模块，用户需参考模块的数据手册。

按照以下步骤正确配置照相机模块：

- 配置照相机配置引脚的输入/输出功能，以便能够修改其寄存器（串行通信，通常为I²C）。
- 在照相机模块上应用硬件复位。
- 通过以下方式初始化照相机模块：
 - 配置图像分辨率
 - 配置对比度和亮度
 - 配置照相机的白平衡（例如黑白、白色负片和白色正常）
 - 选择照相机接口（某些照相机模块具有串行和并行接口）
 - 如果照相机模块支持多种模式，选择同步模式
 - 配置时钟信号频率
 - 选择输出数据格式。

5 功耗和性能考虑

5.1 功耗

为了在应用处于低功耗模式时节省更多能量，建议在STM32进入低功耗模式之前使照相机模块进入低功耗模式。

使照相机模块进入低功耗模式可显著降低功耗。

以OV9655 CMOS传感器为例：

- 在激活模式下，工作电流为**20 mA**。
- 在待机模式下，对于I2C发起的待机，电流需求降至**1 mA**（内部电路活动暂停但时钟不停止），而对于引脚发起的待机，则降至**10 μ A**（内部器件时钟停止且所有内部计数器复位）。有关详细信息，请参见相应的照相机数据手册。

5.2 性能考虑

对于所有STM32 MCU而言，每个像素时钟要传输的字节数取决于扩展数据模式：

- 如果将DCMI配置为接收**8位**数据，照相机接口将用**4**个像素时钟周期捕获一个32位数据字。
- 如果将DCMI配置为接收**10、12或14位**数据，照相机接口将用**2**个像素时钟周期捕获一个32位数据字。

[表 10](#)总结了使用不同数据宽度配置时的最大数据流。

表10. 最大DCMI_PIXCLK时的最大数据流⁽¹⁾

STM32系列		扩展数据模式 (Extended data mode)			
		8位	10位	12位	14位
每PICXCLK字节数		1	1.25	1.5	1.75
数据流 (最大 Mb/s)	STM32F2	46.875	58.594	70.312	82.031
	STM32F4	52.734	65.918	79.101	92.285
	STM32F7	52.734	65.918	79.101	92.285
	STM32H7	78.125	97.656	117.187	136.718
	STM32L4	31.25	39.062	46.875	54.687

1. 这些值均按第 表2. 节: DCMI和相关资源可用性所述最大DCMI_PIXCLK计算得出。
- 在某些应用中，DMA2（对于STM32H7x3器件，为DMA1）被配置为在服务于DCMI请求的同时服务于其他请求。这种情况下，如果DMA在服务于DCMI的同时服务于其他流，则用户必须注意流优先级配置，并考虑性能影响。
 - 为了获得更好的性能，在同时使用DCMI和其他外设（发出请求且可以连接到DMA1或DMA2）时，最好将这些流配置为由不服务于DCMI的DMA提供服务。
 - 为避免溢出，用户必须确保在照相机模块端配置的像素时钟为STM32 DCMI所支持。
 - 为了获得更好的性能，建议使用最高系统速度HCLK，但用户必须考虑使用的所有外设速度（例如，外部存储器速度），以避免溢出并保证其应用的成功。
 - DCMI不是唯一的AHB2外设，还有许多其他外设，并且DMA不是可以访问AHB2外设的唯一主设备。使用许多AHB2外设或有其他主设备访问AHB2外设会在AHB2上造成并发，用户必须考虑这种情况对性能的影响。

6 DCMI应用示例

本章提供大量与DCMI的使用有关的信息，以及示例实现的分步描述。

6.1 DCMI应用场景示例

使用DCMI和其他STM32外设可实现多种成像应用。下面是一些应用示例：

- 机器视觉
- 玩具
- 生物统计
- 安全和视频监视
- 门铃和家居自动化
- 工业监控系统和自动化检查
- 系统控制
- 访问控制系统
- 条形码扫描
- 视频会议
- 无人机
- 实时视频数据流和电池供电摄影机。


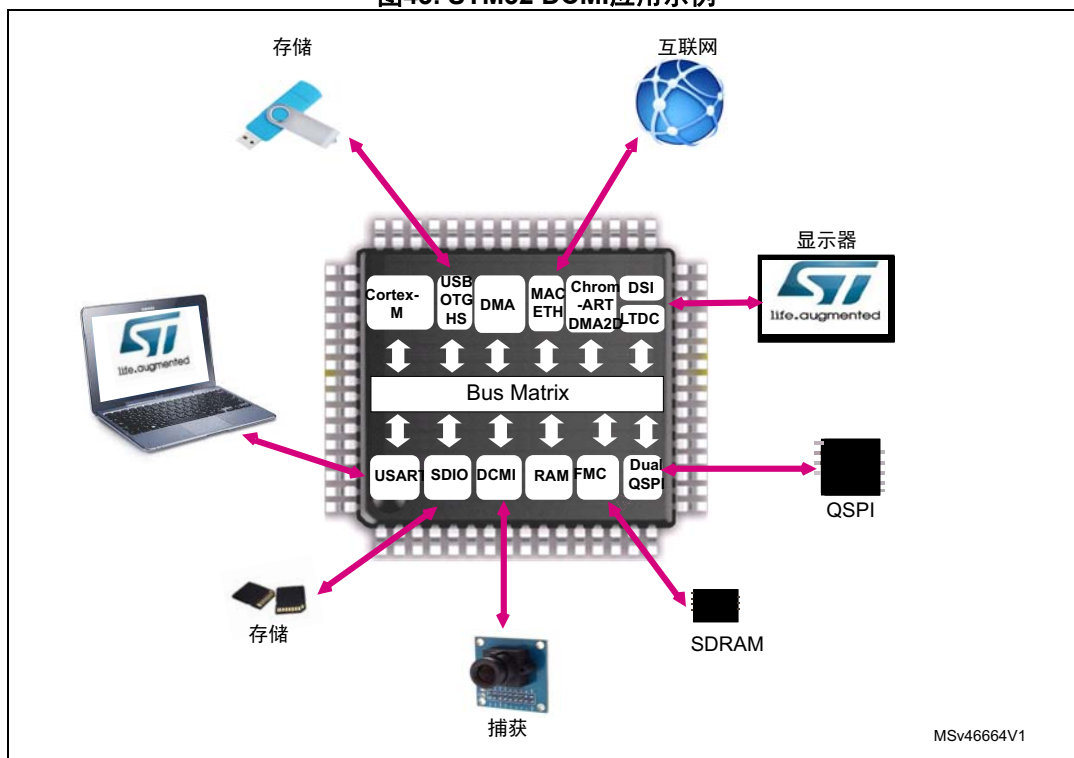
 **43**提供了使用STM32 MCU的应用示例，它允许用户捕获数据、将数据保存在内部或外部存储器中、显示数据、通过互联网共享数据以及与人类通信。

图43. STM32 DCMI应用示例



6.2 STM32Cube固件示例

STM32CubeF2、STM32CubeF4、STM32CubeF7和STM32CubeL4固件包提供大量在相应的板上实现并测试的示例。表 11提供了不同STM32Cube固件的DCMI示例和应用的总览。

表11. STM32Cube DCMI示例

固件包	项目名称 ⁽¹⁾	板
STM32CubeF2	DCMI_CaptureMode	STM3220G-EVAL STM3221G-EVAL
	SnapshotMode	
	Camera_To_USBDisk	
STM32CubeF4	DCMI_CaptureMode	STM32446E-EVAL
	SnapshotMode	STM324x9I-EVAL STM324xG-EVAL
	Camera_To_USBDisk	STM32446E-EVAL
STM32CubeF7	DCMI_CaptureMode	STM32756G-EVAL STM32F769I-EVAL
	SnapshotMode	
	Camera_To_USBDisk	
STM32CubeL4	DCMI_CaptureMode	32L496GDISCOVERY
	SnapshotMode	

1. 所有示例均开发用于捕获RGB数据。对于大多数示例，用户可以选择以下分辨率之一：QQVGA 160x120、QVGA 320x240、480x272或VGA 640x480。

6.3 基于STM32CubeMX的DCMI示例

本节描述了五个使用DCMI的典型示例：

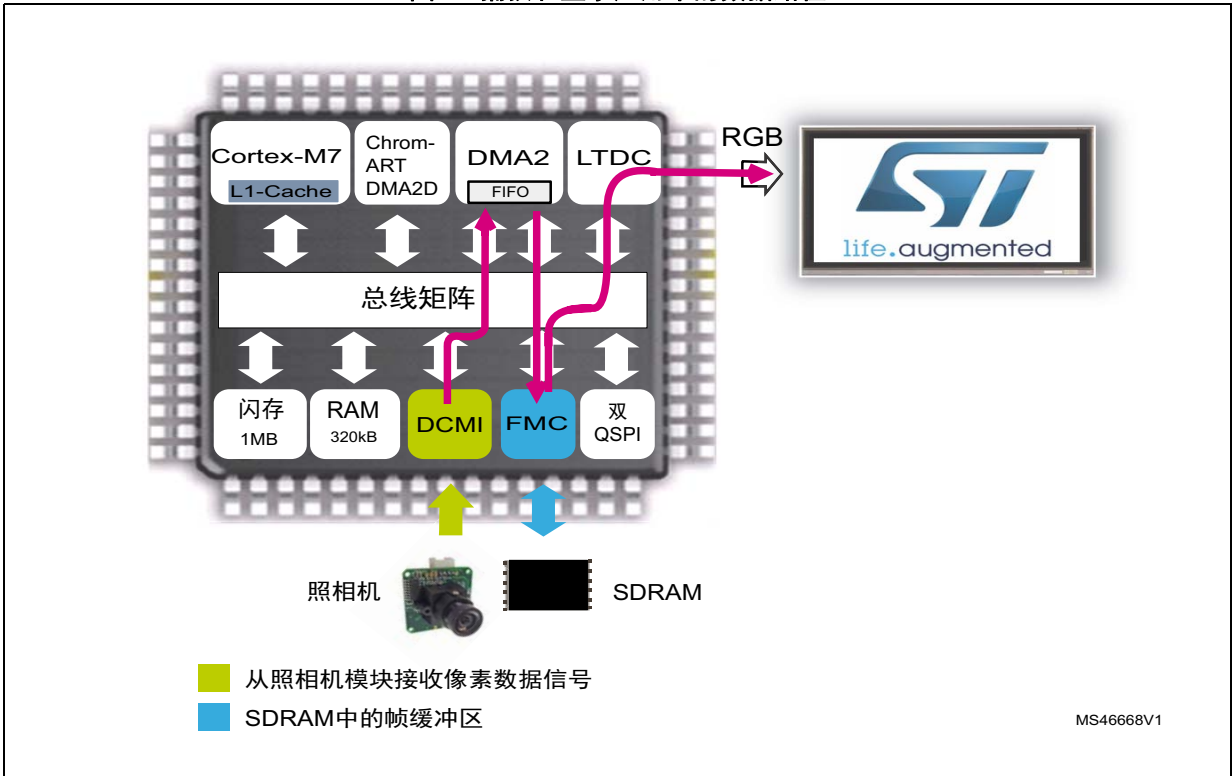
- RGB数据的捕获和显示：捕获的数据为RGB565格式，QVGA（320x240）分辨率，保存在SDRAM中并显示在LCD-TFT上。
- YCbCr数据的捕获：捕获的数据为YCbCr格式，QVGA（320x240）分辨率，保存在SDRAM中。
- 仅Y分量数据的捕获：DCMI被配置为接收仅Y分量数据，数据将保存在SDRAM中。
- SxGA分辨率捕获（YCbCr数据格式）：捕获的数据为YCbCr格式，SxGA（1280x1024）分辨率，保存在SDRAM中。
- JPEG数据的捕获：捕获的数据为JPEG格式，将保存在SDRAM中。

所有这些示例均在32F746GDISCOVERY上使用STM32F4DIS-CAM（OV9655 CMOS传感器）实现，只有JPEG数据捕获是在STM324x9I-EVAL（OV2640 CMOS传感器）上实现

如图 44所示，应用包含三个主要步骤：

- 将接收到的数据通过其外设端口从DCMI导入到DMA（将暂时保存在FIFO中）。
- 将数据从FIFO传输到SDRAM
- 从SDRAM导入要在LCD-TFT上显示的数据，仅适用于RGB数据格式。对于YCbCr或JPEG数据格式，用户必须将接收到的数据转换为要显示的RGB格式。

图44. 捕获和显示应用中的数据路径



用户需要为这些示例配置DCMI、DMA2、LTDC（就RGB数据捕获和显示示例而言）和SDRAM。

下面几节中描述的五个子例具有一些基于STM32CubeMX的共有配置：

- GPIO配置
- DMA配置
- 时钟配置

仅Y分量和JPEG捕获示例需要以下特定配置：

- DCMI外设配置
- 相机模块配置

下面几节将提供硬件描述、使用STM32CubeMX的常用配置以及必须添加到STM32CubeMX生成的项目的常见修改。

6.3.1 硬件说明

下面的示例（JPEG捕获示例除外）在32F746GDISCOVERY上使用照相机板STM32F4DIS-CAM实现。

图45. 32F746GDISCOVERY和STM32F4DIS-CAM的互联



1. 图片不属于合同范围。

STM32F4DIS-CAM板包含一个130万像素的Omnivision CMOS传感器（ov9655）。分辨率可达到1280x1024。该照相机模块通过一个30引脚FFC连接到DCMI。

32F746GDISCOVERY板以具有电容式触摸屏的4.3英寸彩色LCD-TFT为特色，第一个示例中使用它来显示捕获的图像。

如 [图 46](#)所示，照相机模块通过以下方式连接到STM32F7：

- 控制信号DCMI_PIXCLK、DCMI_VSYNC和DCMI_HSYNC
- 图像数据信号DCMI_D[0..7]

更多信号通过30引脚FFC提供给照相机模块：

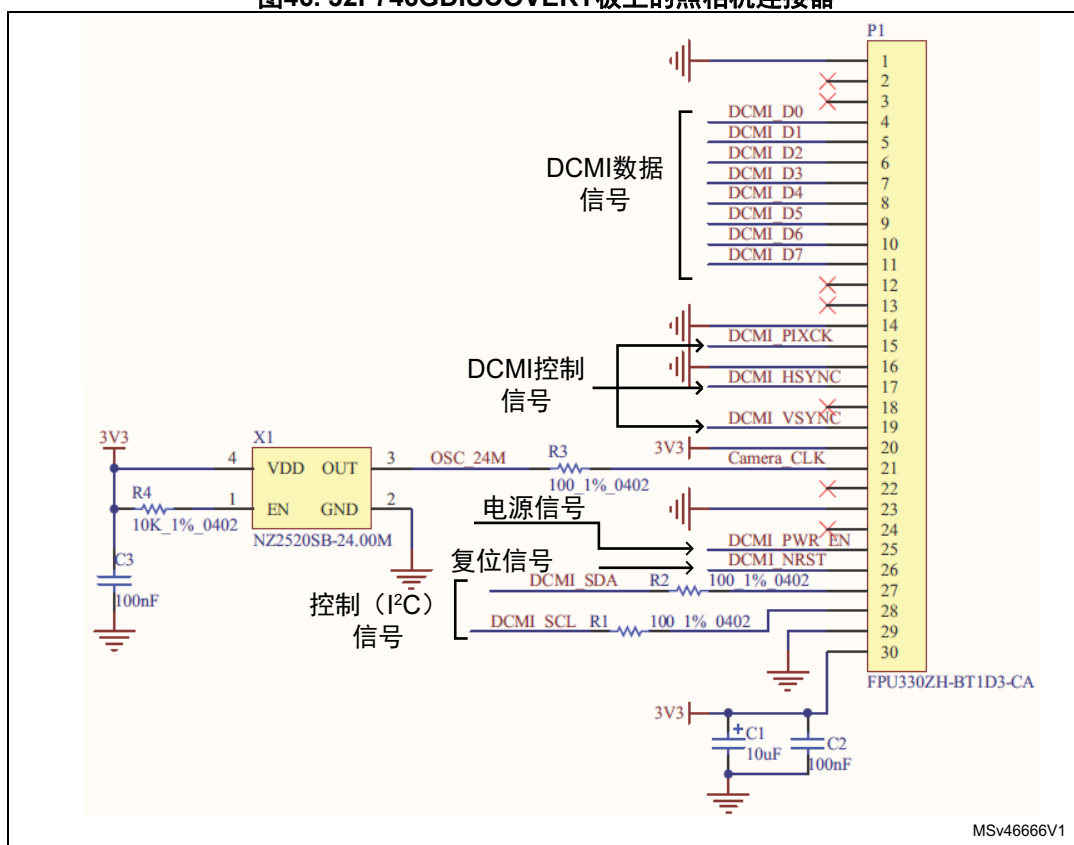
- 电源信号（DCMI_PWR_EN）
- 照相机模块的时钟（Camera_CLK）
- 配置信号（I2C）
- 复位信号（DCMI_NRST）

有关这些信号的更多详细信息，请参见 [第 1.2.2 节：照相机模块互联（并行接口）](#)。

照相机时钟由32F746GDISCOVERY板上的内置NZ2520SB晶体时钟振荡器（X1）通过Camera_CLK引脚提供给照相机模块。照相机时钟的频率等于24 MHz。

DCMI复位引脚（DCMI_NRST）可以复位照相机模块，它连接到全局MCU复位引脚（NRST）。

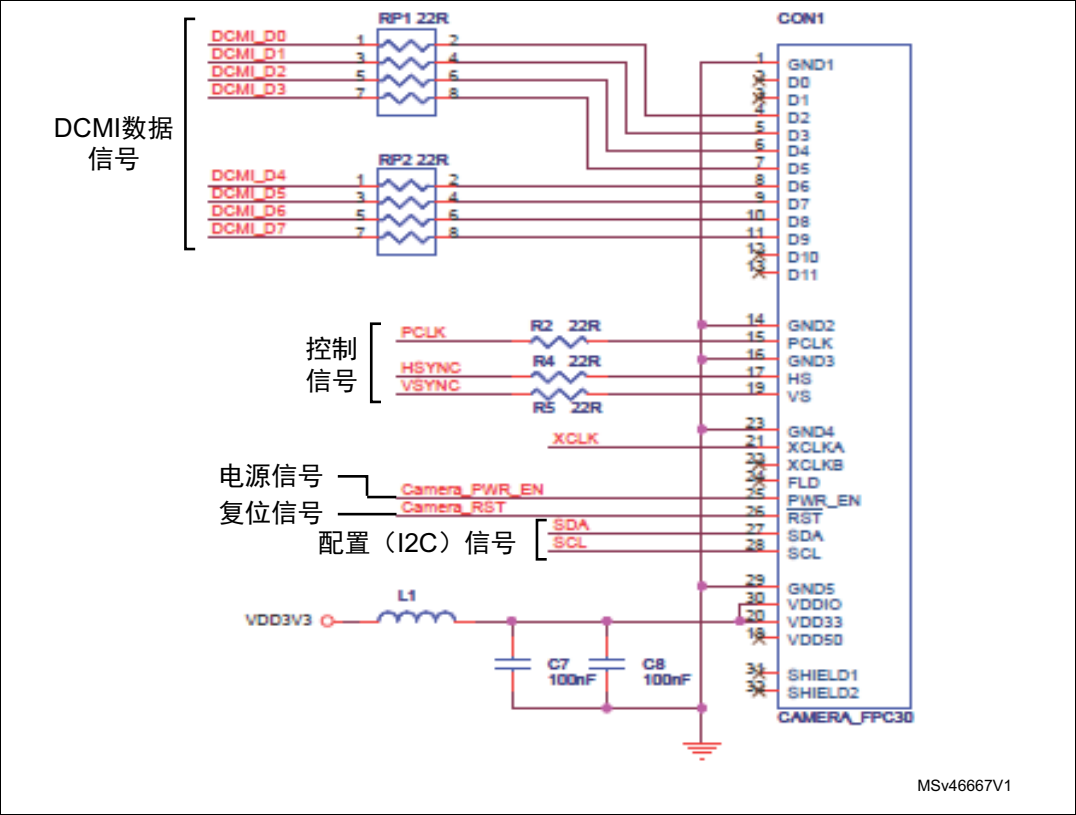
图46. 32F746GDISCOVERY板上的照相机连接器



有关32F746GDISCOVERY板的更多详细信息，请参考意法半导体网站上的用户手册 *带STM32F746NG MCU的STM32F7系列产品探索套件* (UM1907)。

图 47描述了STM32F4DIS-CAM上实现的照相机模块连接器。

图47. STM32F4DIS-CAM上的照相机连接器



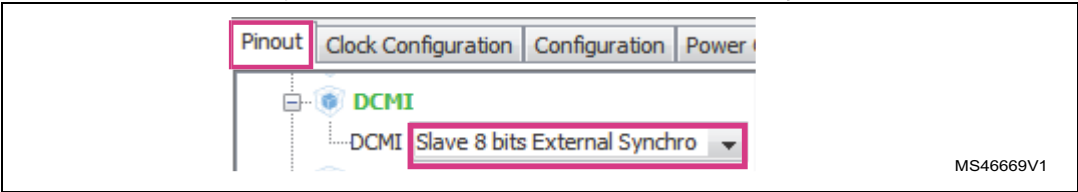
6.3.2 常用配置示例

使用STM32CubeMX时，第一步是配置项目位置和相应的工具链或IDE（菜单“项目/设置”）。

STM32CubeMX：DCMI GPIO配置

1. 选择DCMI，然后在“引脚排列”选项卡中选择“从设备8位外部同步”，以便将DCMI配置为从设备8位外部（硬件）同步（图 48）。

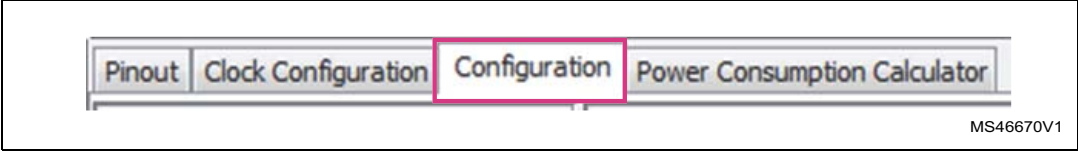
图48. STM32CubeMX - DCMI同步模式选择



如果在选择一种硬件配置（从设备8位外部同步）后，使用的GPIO与硬件不匹配，则用户可以更改所需的GPIO并直接在引脚上配置复用功能。另一种方法包括通过为每个GPIO选择合适的复用功能来手动配置GPIO引脚。有关必须配置的GPIO的更多详细信息，请参见图 52：STM32CubeMX - 选择DCMI引脚。在这一步后，必须以绿色突出显示11个引脚（D[0..7]、DCMI_VSYNC、DCMI_HSYNC和DCMI_PIXCLK）。

2. 选择“配置”选项卡配置GPIO模式和速度，如图 51所示。

图49. STM32CubeMX - 选择“配置选项卡”



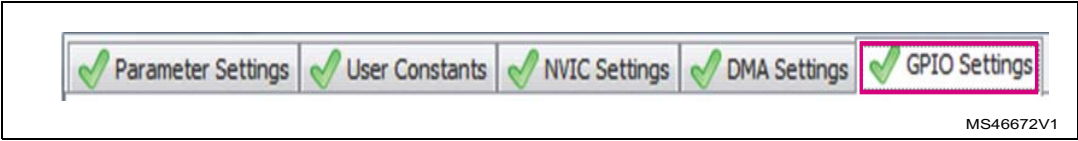
3. 点击“配置”选项卡上的DCMI按钮，如图 50所示。

图50. STM32CubeMX - “配置”选项卡上的DCMI按钮



4. 在显示DCMI配置窗口后，选择“GPIO设置”选项卡，如图 51所示。

图51. STM32CubeMX - 选择“GPIO设置”



5. 如 图 52所示，选择所有DCMI引脚。

图52. STM32CubeMX - 选择DCMI引脚

PA4	DCMI_HSYNC	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_HSYNC	<input checked="" type="checkbox"/>
PA6	DCMI_PIXCLK	n/a	Alternate Fu...	No pull-up a...	Low		<input type="checkbox"/>
PD3	DCMI_D5	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D5	<input checked="" type="checkbox"/>
PE5	DCMI_D6	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D6	<input checked="" type="checkbox"/>
PE6	DCMI_D7	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D7	<input checked="" type="checkbox"/>
PG9	DCMI_VSYNC	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_VSYNC	<input checked="" type="checkbox"/>
PH9	DCMI_D0	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D0	<input checked="" type="checkbox"/>
PH10	DCMI_D1	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D1	<input checked="" type="checkbox"/>
PH11	DCMI_D2	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D2	<input checked="" type="checkbox"/>
PH12	DCMI_D3	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D3	<input checked="" type="checkbox"/>
PH14	DCMI_D4	n/a	Alternate Fu...	No pull-up a...	Low	DCMI_D4	<input checked="" type="checkbox"/>

6. 设置“GPIO上拉/下拉”，如 图 53所示。

图53. STM32CubeMX - 选择“GPIO无上拉且无下拉”

GPIO Pull-up/Pull-down

No pull-up and no pull-down

MS46674V1

7. 点击“应用”，然后点击“确定”。

STMCubeMX - DCMI控制信号和捕获模式配置

1. 点击“DCMI配置”窗口中的“参数设置”选项卡，然后选择“参数设置”选项卡，如 图 54所示。

图54. STM32CubeMX - 选择“参数设置”选项卡

DCMI Configuration

☒ Parameter Settings

☒ User Constants

☒ NVIC Settings

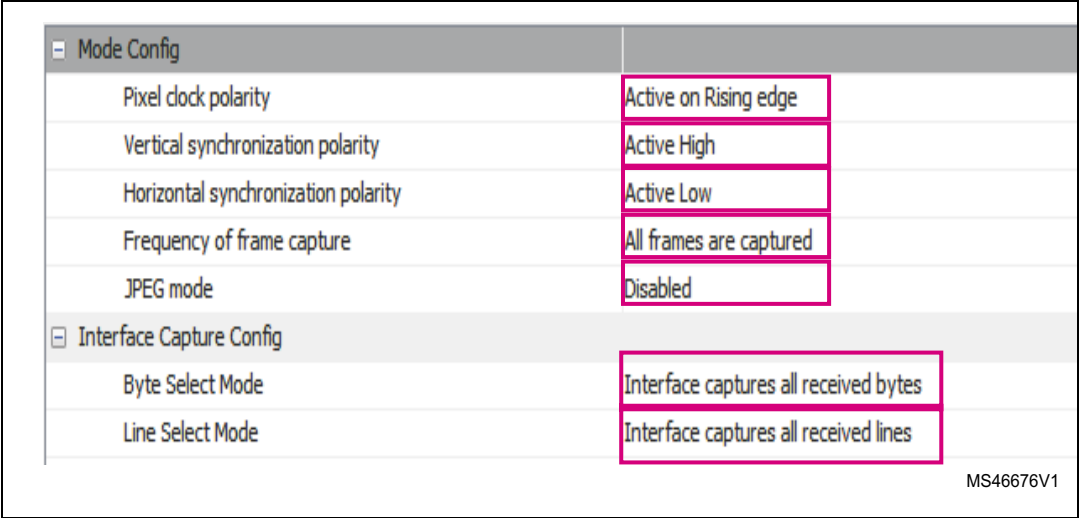
☒ DMA Settings

☒ GPIO Settings

MS46675V1

2. 设置不同的参数，如 图 55所示。必须根据照相机模块的配置进行垂直同步、水平同步和像素时钟极性的设定。

图55. STM32CubeMX - DCMI控制信号和捕获模式配置



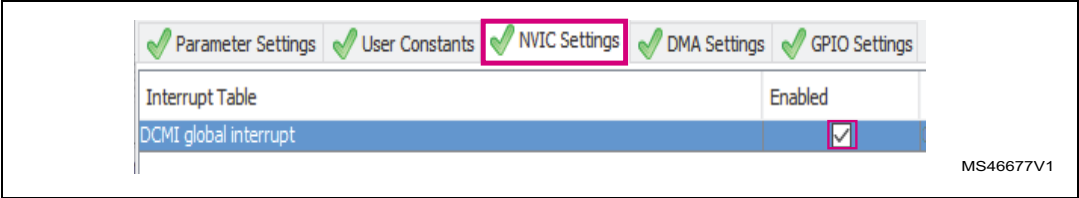
3. 点击“应用”，然后点击“确定”。

注：垂直同步极性必须为高电平有效，而水平同步极性必须为低电平有效。就该相机模块配置而言，不能将它们颠倒过来。

STM32CubeMX - 使能DCMI中断

1. 选择“DCMI配置”窗口中的“NVIC设置”选项卡，然后选中“DCMI全局中断”，如图 56所示。

图56. STM32CubeMX - 配置DCMI中断



2. 点击“应用”，然后点击“确定”。

STM32CubeMX：DMA配置

该配置旨在接收RGB565数据（每像素2个字节），图像分辨率为QVGA（320x240）。因此，图像大小为320 * 240 * 2 = 153600字节。

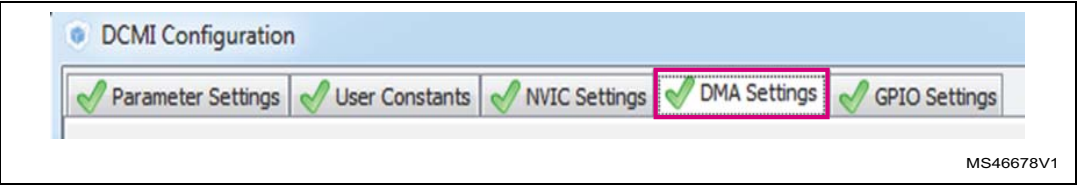
由于从DCMI发送的数据宽度为4字节（从DCMI中的数据寄存器发送32位字），因此，DMA_SxNDTR寄存器中数据项的数量为要传输的字的数量。因此，字的数量为38400（153600 / 4），小于65535。

在快照模式下，用户可以配置正常模式下的DMA。

在连续模式下，用户可以配置循环模式下的DMA。

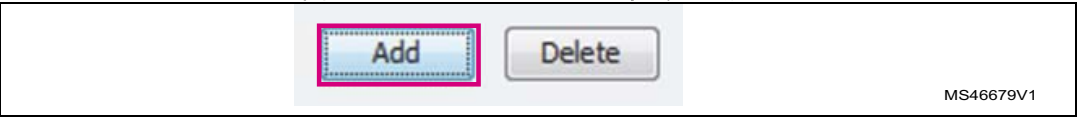
1. 选择“DCMI配置”窗口中的“DMA设置”选项卡，如 图 57所示。

图57. STM32CubeMX - 选择“DMA设置”选项卡



2. 点击 图 58所示的“添加”按钮。

图58. STM32CubeMX - 选择“添加”按钮



3. 点击“在DMA请求下选择”并选择DCMI。DMA请求的配置如 图 59所示。DMA2流1通道1被配置为在每次其时间寄存器装满时传输DCMI请求。

图59. STM32CubeMX - DMA流配置

DMA Request	Stream	Direction	Priority
DCMI	DMA2 Stream 1	Peripheral To Memory	High

MSv46680V1

4. 修改“DMA请求设置”，如 图 60所示。

图60. STM32CubeMX：DMA配置

DMA Request Settings

Mode		Peripheral		Memory	
Circular		Increment Address		<input checked="" type="checkbox"/>	
Use Fifo	Threshold	Data Width	Burst Size	Word	4 Increment
<input checked="" type="checkbox"/>	Full	Word	Single	Word	4 Increment

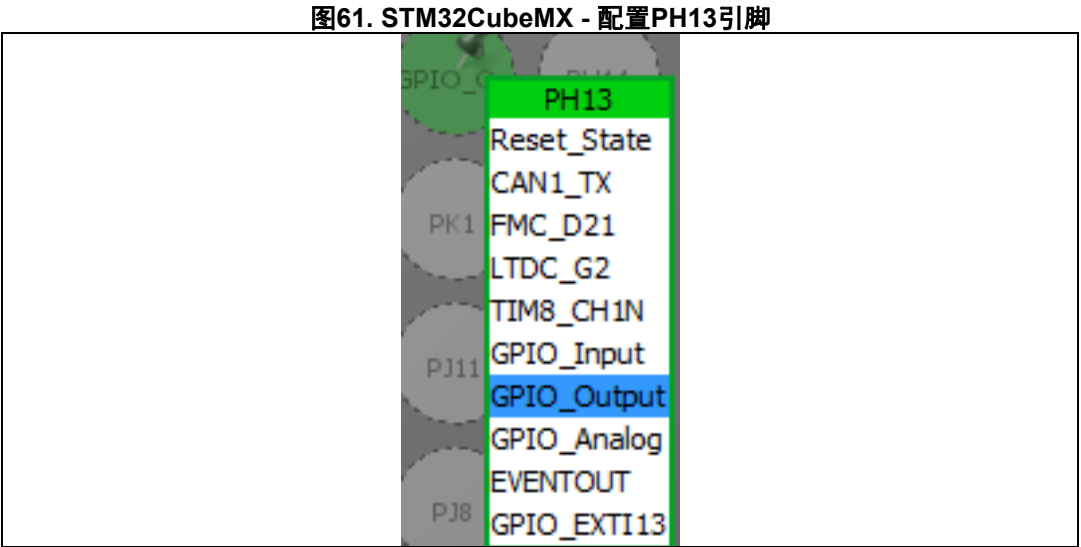
MS46681V1

5. 点击“应用”，然后点击“确定”。

STM32CubeMX - 照相机模块上电引脚

要为照相机模块上电，必须为32F746GDISCOVERY配置PH13引脚。

- 1. 在“引脚排列”选项卡上点击PH13引脚并选择GPIO_Output，如 图 61所示。

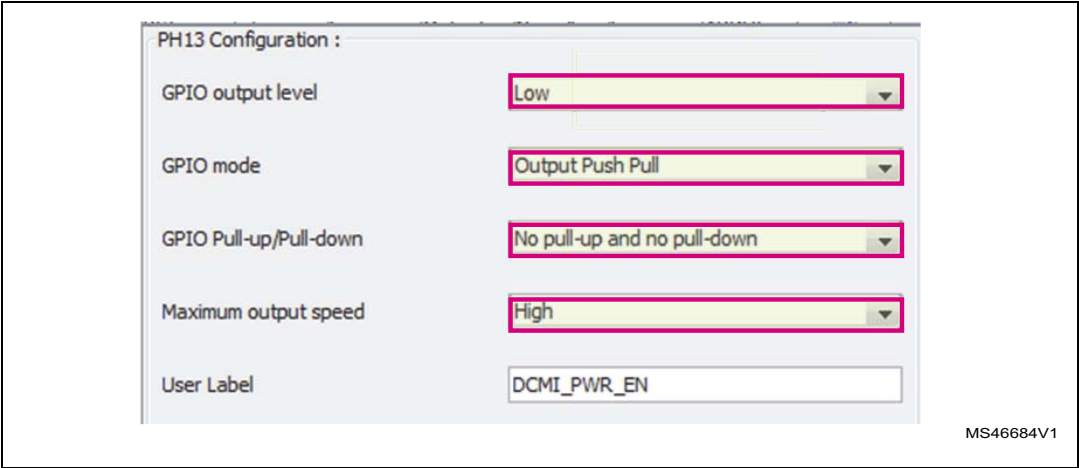


- 2. 在“配置”选项卡上，点击 图 62所示的“GPIO”按钮。



3. 设置参数，如图 63所示。

图63. STM32CubeMX - 配置DCMI电源引脚



STM32CubeMX - 系统时钟配置

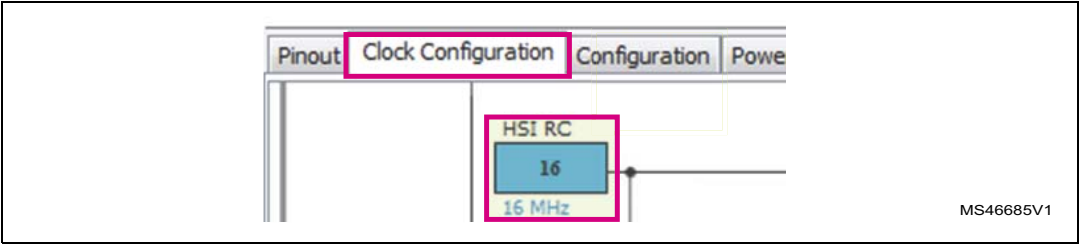
在本例中，系统时钟的配置如下：

- 使用内部HSI RC，其中主PLL用作系统源时钟。
- HCLK @ 200 MHz，因此Cortex®-M7和LTDC均运行于200 MHz。

注：HCLK设置为200MHz而非216MHz，这是为了使用HCLK2预分频器将SDRAM_FMC设置为其最大速度100 MHz。

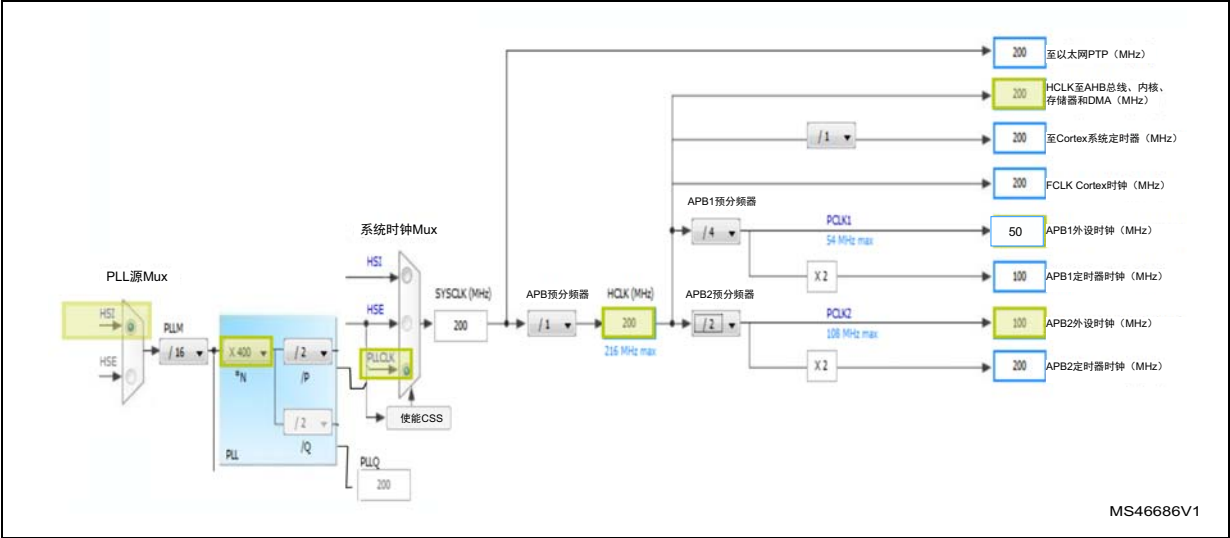
1. 选择“时钟配置”选项卡，如图 64所示。

图64. STM32CubeMX - HSI配置



2. 在“时钟配置”选项卡上设置PLL和预分频器，以获取系统时钟HCLK @ 200 MHz，如 [图 65](#)所示。

图65. STM32CubeMX - 时钟配置



在该阶段，用户可以生成项目。

向项目中添加文件

使用首选工具链，按照以下步骤生成代码并打开生成的项目：

1. 右键单击“Drivers/STM32F7xx_HAL_Driver”。
2. 选择“将现有文件添加到组'Drivers/STM32F7xx_HAL_Driver...'”。
3. 在“Drivers/STM32F7xx_HAL_Driver/Src”中选择以下文件：
 - **stm32f7xx_hal_dma2d.c**
 - **stm32f7xx_hal ltdc.c**
 - **stm32f7xx_hal ltdc_ex.c**
 - **stm32f7xx_hal_sdram.c**
 - **stm32f7xx_hal_uart.c**
 - **stm32f7xx_ll_fmc.c**
4. 在**stm32f7xx_hal_conf.h**中取消模块DMA2D、LTDC、SDRAM和UART的注释。
5. 新建名为Imported_Drivers的组。
6. 将以下文件从C:目录下的STM32746G_Discovery文件夹复制到项目的**Src**文件夹：
 - **stm32746g_discovery.c**
 - **stm32746g_discovery_sdram.c**
7. 将以下文件从C:目录下的STM32746G_Discovery文件夹复制到项目的**Src**文件夹：
 - **stm32746g_discovery.h**
 - **stm32746g_discovery_sdram.h**
8. 将**ov9655.c**从Components文件夹复制到Src文件夹。
9. 将**ov9655.h**从Components文件夹复制到Inc文件夹。
10. 将**camera.h**从Component/Common文件夹复制到Inc文件夹。
11. 在新组（本例中为Imported_Drivers）中添加以下文件：
 - **stm32746g_discovery.h**
 - **stm32746g_discovery_sdram.h**
 - **ov9655.c**
12. 通过以下操作允许修改**ov9655.h**和**camera.h**（默认只读）：
 - 右键单击该文件
 - 取消选中只读
 - 点击“应用”，然后点击“确定”。
13. 修改**ov9655.h**文件，将#include "../Common/camera.h"替换为#include "camera.h。”
14. 将以下文件复制到Inc文件夹：
 - Components文件夹中的**rk043fn48h.h**
 - Utilities/Fonts文件夹中的**fonts.h**和**fonts24.h**。
15. 通过重建所有文件检查并确认没有发生问题。必须无错误且无警告。

main.c文件的修改

1. 通过插入一些说明更新**main.c**，以便在空间充足处包含需要的文件，如下面的**绿色粗体**部分所示。该任务提供项目修改和再生，而不会丢失用户代码：

```
/* 用户代码开始Includes */
#include "stm32746g_discovery.h"
#include "stm32746g_discovery_sdram.h"
#include "ov9655.h"
#include "rk043fn48h.h"
#include "fonts.h"
#include "font24.c"
/* 用户代码结束Includes */
```

然后，必须在空间充足处插入一些变量声明，如下面的**绿色粗体**部分所示。

```
/* 用户代码开始 PV */
/* 私有变量 -----*/
typedef enum
{
    CAMERA_OK          = 0x00,
    CAMERA_ERROR       = 0x01,
    CAMERA_TIMEOUT     = 0x02,
    CAMERA_NOT_DETECTED = 0x03,
    CAMERA_NOT_SUPPORTED = 0x04
} Camera_StatusTypeDef;
typedef struct
{
    uint32_t TextColor;
    uint32_t BackColor;
    sFONT *pFont;
}LCD_DrawPropTypeDef;
typedef struct
{
    int16_t X;
    int16_t Y;
}Point, * pPoint;
static LCD_DrawPropTypeDef DrawProp[2];
LTDC_HandleTypeDef hltdc;
LTDC_LayerCfgTypeDef layer_cfg;
static RCC_PeriphCLKInitTypeDef periph_clk_init_struct;
CAMERA_DrvTypeDef *camera_driv;
/* 照相机模块I2C硬件地址 */
static uint32_t CameraHwAddress;
```

```
/* 图像大小 */
uint32_t Im_size = 0;
/* 用户代码结束 PV */
```

之后，必须在空间充足处插入函数原型，如下面的**绿色粗体**部分所示。

```
/* 用户代码开始 PFP */
/* 私有函数原型 -----*/
uint8_t CAMERA_Init(uint32_t );
static void LTDC_Init(uint32_t , uint16_t , uint16_t , uint16_t , uint16_t );
void LCD_GPIO_Init(LTDC_HandleTypeDef *, void *);
/* 用户代码结束 PFP */
```

2. 通过在空间充足处插入一些说明更新**main()**函数，如下面的**绿色粗体**部分所示。
LTDC_Init函数用于配置和初始化LCD。BSP_SDRAM_Init函数用于配置和初始化SDRAM。
CAMERA_Init函数用于配置照相机模块以及DCMI寄存器和参数。对于在快照或连续模式
下用于配置DCMI的两个函数之一HAL_DCMI_Start_DMA，必须取消注释。

```
/* 用户代码开始 2 */
LTDC_Init(FRAME_BUFFER, 0, 0, 320, 240);
BSP_SDRAM_Init();
CAMERA_Init(CAMERA_R320x240);
HAL_Delay(1000); //Delay for the camera to output correct data
Im_size = 0x9600; //size=320*240*2/4
/* 对于快照模式，取消以下行的注释 */
//HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size);
/* 对于连续模式，取消以下行的注释 */
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS , (uint32_t)FRAME_BUFFER, Im_size);
/* 用户代码结束 2 */
```

3. 在空间充足处插入新函数（在main()函数中调用）的实现，如下面的**绿色粗体**部分所示。

```
/* 用户代码开始 4 */
void LCD_GPIO_Init(LTDC_HandleTypeDef *hltdc, void *Params)
{
    GPIO_InitTypeDef gpio_init_structure;
    /* 使能LTDC和DMA2D时钟 */
    __HAL_RCC_LTDC_CLK_ENABLE();
    __HAL_RCC_DMA2D_CLK_ENABLE();
    /* 使能GPIO时钟 */
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOG_CLK_ENABLE();
}
```

```

__HAL_RCC_GPIOI_CLK_ENABLE();
__HAL_RCC_GPIOJ_CLK_ENABLE();
__HAL_RCC_GPIOK_CLK_ENABLE();

/** LTDC引脚配置 */
/* GPIOE配置 */
gpio_init_structure.Pin    = GPIO_PIN_4;
gpio_init_structure.Mode   = GPIO_MODE_AF_PP;
gpio_init_structure.Pull   = GPIO_NOPULL;
gpio_init_structure.Speed  = GPIO_SPEED_FAST;
gpio_init_structure.Alternate = GPIO_AF14_LTDC;
HAL_GPIO_Init(GPIOE, &gpio_init_structure);

/* GPIOG配置 */
gpio_init_structure.Pin    = GPIO_PIN_12;
gpio_init_structure.Mode   = GPIO_MODE_AF_PP;
gpio_init_structure.Alternate = GPIO_AF9_LTDC;
HAL_GPIO_Init(GPIOG, &gpio_init_structure);

/* GPIOI LTDC替代配置 */
gpio_init_structure.Pin    = GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_13 | GPIO_PIN_14 |
GPIO_PIN_15;
gpio_init_structure.Mode   = GPIO_MODE_AF_PP;
gpio_init_structure.Alternate = GPIO_AF14_LTDC;
HAL_GPIO_Init(GPIOI, &gpio_init_structure);

/* GPIOJ配置 */
gpio_init_structure.Pin    = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 |
GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7 | GPIO_PIN_8 | GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_13 |
GPIO_PIN_14 | GPIO_PIN_15;
gpio_init_structure.Mode   = GPIO_MODE_AF_PP;
gpio_init_structure.Alternate = GPIO_AF14_LTDC;
HAL_GPIO_Init(GPIOJ, &gpio_init_structure);

/* GPIOK配置 */
gpio_init_structure.Pin    = GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_4 |
GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7;
gpio_init_structure.Mode   = GPIO_MODE_AF_PP;
gpio_init_structure.Alternate = GPIO_AF14_LTDC;
HAL_GPIO_Init(GPIOK, &gpio_init_structure);

/* LCD_DISP GPIO配置 */
gpio_init_structure.Pin    = GPIO_PIN_12; /* LCD_DISP pin has to be manually controlled */
gpio_init_structure.Mode   = GPIO_MODE_OUTPUT_PP;
HAL_GPIO_Init(GPIOI, &gpio_init_structure);

/* LCD_BL_CTRL GPIO配置 */
gpio_init_structure.Pin    = GPIO_PIN_3; /* LCD_BL_CTRL pin has to be manually controlled */
gpio_init_structure.Mode   = GPIO_MODE_OUTPUT_PP;
HAL_GPIO_Init(GPIOK, &gpio_init_structure);

```

```

}

static void LTDC_Init(uint32_t FB_Address, uint16_t Xpos, uint16_t Ypos, uint16_t Width, uint16_t
Height)
{
    /* 时序配置 */
    hltdc.Init.HorizontalSync = (RK043FN48H_HSYNC - 1);
    hltdc.Init.VerticalSync = (RK043FN48H_VSYNC - 1);
    hltdc.Init.AccumulatedHBP = (RK043FN48H_HSYNC + RK043FN48H_HBP - 1);
    hltdc.Init.AccumulatedVBP = (RK043FN48H_VSYNC + RK043FN48H_VBP - 1);
    hltdc.Init.AccumulatedActiveH = (RK043FN48H_HEIGHT + RK043FN48H_VSYNC +
RK043FN48H_VBP - 1);
    hltdc.Init.AccumulatedActiveW = (RK043FN48H_WIDTH + RK043FN48H_HSYNC +
RK043FN48H_HBP - 1);
    hltdc.Init.TotalHeigh = (RK043FN48H_HEIGHT + RK043FN48H_VSYNC + RK043FN48H_VBP +
RK043FN48H_VFP - 1);
    hltdc.Init.TotalWidth = (RK043FN48H_WIDTH + RK043FN48H_HSYNC + RK043FN48H_HBP +
RK043FN48H_HFP - 1);

    /* LCD时钟配置 */
    periph_clk_init_struct.PeriphClockSelection = RCC_PERIPHCLK_LTDC;
    periph_clk_init_struct.PLLSAI.PLLSAIN = 192;
    periph_clk_init_struct.PLLSAI.PLLSAIR = RK043FN48H_FREQUENCY_DIVIDER;
    periph_clk_init_struct.PLLSAIDivR = RCC_PLLSAIDIVR_4;
    HAL_RCCEx_PeriphCLKConfig(&periph_clk_init_struct);

    /* 初始化LCD像素宽度和像素高度 */
    hltdc.LayerCfg->ImageWidth = RK043FN48H_WIDTH;
    hltdc.LayerCfg->ImageHeight = RK043FN48H_HEIGHT;
    hltdc.Init.Backcolor.Blue = 0; /* Background value */
    hltdc.Init.Backcolor.Green = 0;
    hltdc.Init.Backcolor.Red = 0;

    /* 极性 */
    hltdc.Init.HSPolarity = LTDC_HSPOLARITY_AL;
    hltdc.Init.VSPolarity = LTDC_VSPOLARITY_AL;
    hltdc.Init.DEPolarity = LTDC_DEPOLARITY_AL;
    hltdc.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
    hltdc.Instance = LTDC;
    if(HAL_LTDC_GetState(&hltdc) == HAL_LTDC_STATE_RESET)
    {
        LCD_GPIO_Init(&hltdc, NULL);
    }
    HAL_LTDC_Init(&hltdc);

    /* 断言显示使能LCD_DISP引脚 */
    HAL_GPIO_WritePin(GPIOI, GPIO_PIN_12, GPIO_PIN_SET);

    /* 断言背光LCD_BL_CTRL引脚 */
    HAL_GPIO_WritePin(GPIOK, GPIO_PIN_3, GPIO_PIN_SET);
    DrawProp[0].pFont = &Font24 ;

```



```

/* 层初始化 */
layer_cfg.WindowX0 = Xpos;
layer_cfg.WindowX1 = Width;
layer_cfg.WindowY0 = Ypos;
layer_cfg.WindowY1 = Height;
layer_cfg.PixelFormat = LTDC_PIXEL_FORMAT_RGB565;
layer_cfg.FBStartAddress = FB_Address;
layer_cfg.Alpha = 255;
layer_cfg.Alpha0 = 0;
layer_cfg.Backcolor.Blue = 0;
layer_cfg.Backcolor.Green = 0;
layer_cfg.Backcolor.Red = 0;
layer_cfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_PAxCA;
layer_cfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_PAxCA;
layer_cfg.ImageWidth = Width;
layer_cfg.ImageHeight = Height;
HAL_LTDC_ConfigLayer(&hltdc, &layer_cfg, 1);
DrawProp[1].BackColor = ((uint32_t)0xFFFFFFFF);
DrawProp[1].pFont = &Font24;
DrawProp[1].TextColor = ((uint32_t)0xFF000000);
}

uint8_t CAMERA_Init(uint32_t Resolution) /*Camera initialization*/
{
    uint8_t status = CAMERA_ERROR;
    /* Read ID of Camera module via I2C */
    if(ov9655_ReadID(CAMERA_I2C_ADDRESS) == OV9655_ID)
    {
        camera_driv = &ov9655_drv; /* Initialize the camera driver structure */
        CameraHwAddress = CAMERA_I2C_ADDRESS;
        if (Resolution == CAMERA_R320x240)
        {
            camera_driv->Init(CameraHwAddress, Resolution);
            HAL_DCMI_DisableCROP(&hdcmi);
        }
        status = CAMERA_OK; /* Return CAMERA_OK status */
    }
    else
    {
        status = CAMERA_NOT_SUPPORTED; /* Return CAMERA_NOT_SUPPORTED status */
    }
    return status;
}
/* 用户代码结束 4 */

```

Modifications in main.h file

通过在空间充足处插入帧缓冲区地址声明更新main.h，如下面的绿色部分所示。

```
/* 用户代码开始Private defines */
#define FRAME_BUFFER 0xC0000000
/* 用户代码结束Private defines */
```

在该阶段，用户可以构建、调试和运行项目。

6.3.3 RGB数据的捕获和显示

为了简化本例，以RGB565格式（2 bpp）捕获和显示数据。图像分辨率为320x240（QVGA）。帧缓冲区位于SDRAM中。照相机和LCD数据位于同一帧缓冲区。然后，LCD直接显示通过DCMI捕获的数据，不进行任何处理。随即将照相机模块配置为输出RGB565数据，分辨率QVGA（320x240）。

可按照 [第 6.3.2 节：常用配置示例](#) 所述步骤完成该示例的配置。

6.3.4 YCbCr数据的捕获

说明

该示例实现旨在从照相机模块接收YCbCr数据并传输至SDRAM。

以YCbCr格式接收的数据在LCD（在之前的配置中被配置用于显示RGB565数据）上显示不正确但可以用于验证。

为了正确显示图像，必须将YCbCr数据转换为RGB565数据（或RGB888或ARGB8888，具体取决于应用的需要）。

必须遵循 [第 6.3.2 节：常用配置示例](#) 所述的所有配置步骤，下面是为了获取YCbCr数据要添加的一些说明。必须更新的只有照相机配置。

照相机模块的配置：

通过添加以下内容完成新照相机模块的配置：

- 用于照相机模块寄存器配置的常数表
 - 用于照相机模块配置（通过I2C发送寄存器配置）的新函数。
1. 必须在main.c文件中“/* 私有变量 -----
-----*/”的下方添加包含照相机模块寄存器配置的表声明。

```
const unsigned char OV9655_YUV_QVGA [][2]=
{ { 0x12, 0x80 }, { 0x00, 0x00 }, { 0x01, 0x80 }, { 0x02, 0x80 }, { 0x03, 0x02 }, { 0x04, 0x03 }, { 0x0e, 0x61 },
  { 0x0f, 0x42 }, { 0x11, 0x01 }, { 0x12, 0x62 }, { 0x13, 0xe7 }, { 0x14, 0x3a }, { 0x16, 0x24 }, { 0x17, 0x18 },
  { 0x18, 0x04 }, { 0x19, 0x01 }, { 0x1a, 0x81 }, { 0x1e, 0x04 }, { 0x24, 0x3c }, { 0x25, 0x36 }, { 0x26, 0x72 },
  { 0x27, 0x08 }, { 0x28, 0x08 }, { 0x29, 0x15 }, { 0x2a, 0x00 }, { 0x2b, 0x00 }, { 0x2c, 0x08 }, { 0x32, 0x24 },
  { 0x33, 0x00 }, { 0x34, 0x3f }, { 0x35, 0x00 }, { 0x36, 0x3a }, { 0x38, 0x72 }, { 0x39, 0x57 }, { 0x3a, 0x0c },
  { 0x3b, 0x04 }, { 0x3d, 0x99 }, { 0x3e, 0x0e }, { 0x3f, 0xc1 }, { 0x40, 0xc0 }, { 0x41, 0x01 }, { 0x42, 0xc0 },
  { 0x43, 0x0a }, { 0x44, 0xf0 }, { 0x45, 0x46 }, { 0x46, 0x62 }, { 0x47, 0x2a }, { 0x48, 0x3c }, { 0x4a, 0xfc },
  { 0x4b, 0xfc }, { 0x4c, 0x7f }, { 0x4d, 0x7f }, { 0x4e, 0x7f }, { 0x52, 0x28 }, { 0x53, 0x88 }, { 0x54, 0xb0 },
  { 0x4f, 0x98 }, { 0x50, 0x98 }, { 0x51, 0x00 }, { 0x58, 0x1a }, { 0x59, 0x85 }, { 0x5a, 0xa9 }, { 0x5b, 0x64 },
  { 0x5c, 0x84 }, { 0x5d, 0x53 }, { 0x5e, 0x0e }, { 0x5f, 0xf0 }, { 0x60, 0xf0 }, { 0x61, 0xf0 }, { 0x62, 0x00 },
  { 0x63, 0x00 }, { 0x64, 0x02 }, { 0x65, 0x20 }, { 0x66, 0x00 }, { 0x69, 0x0a }, { 0x6b, 0x5a }, { 0x6c, 0x04 },
  { 0x6d, 0x55 }, { 0x6e, 0x00 }, { 0x6f, 0x9d }, { 0x70, 0x21 }, { 0x71, 0x78 }, { 0x72, 0x11 }, { 0x73, 0x01 },
  { 0x74, 0x10 }, { 0x75, 0x10 }, { 0x76, 0x01 }, { 0x77, 0x02 }, { 0x7a, 0x12 }, { 0x7b, 0x08 }, { 0x7c, 0x15 }, }
```

```
0x7d, 0x24 }, { 0x7e, 0x45 }, { 0x7f, 0x55 }, { 0x80, 0x6a }, { 0x81, 0x78 }, { 0x82, 0x87 }, { 0x83, 0x96 }, {
0x84, 0xa3 }, { 0x85, 0xb4 }, { 0x86, 0xc3 }, { 0x87, 0xd6 }, { 0x88, 0xe6 }, { 0x89, 0xf2 }, { 0x8a, 0x24 }, {
0x8c, 0x80 }, { 0x90, 0x7d }, { 0x91, 0x7b }, { 0x9d, 0x02 }, { 0x9e, 0x02 }, { 0x9f, 0x7a }, { 0xa0, 0x79 }, {
0xa1, 0x40 }, { 0xa4, 0x50 }, { 0xa5, 0x68 }, { 0xa6, 0x4a }, { 0xa8, 0xc1 }, { 0xa9, 0xef }, { 0xaa, 0x92 }, {
0xab, 0x04 }, { 0xac, 0x80 }, { 0xad, 0x80 }, { 0xae, 0x80 }, { 0xaf, 0x80 }, { 0xb2, 0xf2 }, { 0xb3, 0x20 }, {
0xb4, 0x20 }, { 0xb5, 0x00 }, { 0xb6, 0xaf }, { 0xbb, 0xae }, { 0xbc, 0x7f }, { 0xbd, 0x7f }, { 0xbe, 0x7f }, {
0xbf, 0x7f }, { 0xc0, 0xaa }, { 0xc1, 0xc0 }, { 0xc2, 0x01 }, { 0xc3, 0x4e }, { 0xc6, 0x05 }, { 0xc7, 0x81 }, {
0xc9, 0xe0 }, { 0xca, 0xe8 }, { 0xcb, 0xf0 }, { 0xcc, 0xd8 }, { 0xcd, 0x93 }, { 0xcd, 0x93 }, { 0xff, 0xff } };
```

2. 必须在以下内容的下方插入新函数原型

```
/* 私有函数原型 ----- */
```

```
void OV9655_YUV_Init (uint16_t);
```

3. 必须更新 [第 6.3.2 节：常用配置示例](#) 中 **main.c** 文件的修改的第二步。通过在空间充足处插入以下函数修改 **main()** 函数，如下面的 **绿色粗体** 部分所示。对于在快照或连续模式下用于配置 DCMI 的两个函数之一，必须取消注释。

```
/* 用户代码开始 2 */
```

```
BSP_SDRAM_Init();
```

```
CAMERA_Init(CameraHwAddress);
```

```
OV9655_YUV_Init(CameraHwAddress);
```

```
HAL_Delay(1000); //Delay for the camera to output correct data
```

```
Im_size = 0x9600; //size=320*240*2/4
```

```
/* 对于快照模式，取消以下行的注释 */
```

```
//HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size);
```

```
/* 对于连续模式，取消以下行的注释 */
```

```
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, (uint32_t)FRAME_BUFFER, Im_size);
```

```
/* 用户代码结束 2 */
```

4. 必须通过添加新函数实现来更新 [第 6.3.2 节：常用配置示例](#) 中 **main.c** 文件的修改的第三步。

```
void OV9655_YUV_Init(uint16_t DeviceAddr)
```

```
{ uint32_t index;
  for(index=0; index<(sizeof(OV9655_YUV_QVGA)/2); index++)
  { CAMERA_IO_Write(DeviceAddr, OV9655_YUV_QVGA[index][0],
    OV9655_YUV_QVGA[index][1]);
    CAMERA_Delay(1);
  } }
```

6.3.5 仅Y分量数据的捕获

说明

在本例中，将照相机模块配置为输出YCbCr数据格式。通过在DCMI端使用字节选择功能，忽略色度分量（Cb和Cr）并且只将Y分量传输至SDRAM中的帧缓冲区。

必须遵循 [第 6.3.2 节：常用配置示例](#) 所述的所有配置步骤，下面是为了获取仅Y分量数据要添加的一些说明。必须更新的只有照相机和DCMI配置。

为了简化该任务，必须按照 [第 6.3.4 节：YCbCr数据的捕获](#) 中的描述修改main.c文件，但必须将**STM32CubeMX - DCMI控制信号和捕获模式配置**的第二步或**static void MX_DCMI_Init(void)** 函数（在main.c文件中实现该函数）修改为如下配置：

```
hdcmi.Instance = DCMI;
hdcmi.Init.SynchroMode = DCMI_SYNCHRO_HARDWARE;
hdcmi.Init.PCKPolarity = DCMI_PCKPOLARITY_RISING;
hdcmi.Init.VSPolarity = DCMI_VSPOLARITY_HIGH;
hdcmi.Init.HSPolarity = DCMI_HSPOLARITY_LOW;
hdcmi.Init.CaptureRate = DCMI_CR_ALL_FRAME;
hdcmi.Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
hdcmi.Init.ByteSelectMode = DCMI_BSM_OTHER;
hdcmi.Init.ByteSelectStart = DCMI_OEBS_EVEN;
hdcmi.Init.LineSelectMode = DCMI_LSM_ALL;
hdcmi.Init.LineSelectStart = DCMI_OELS_ODD;
```

6.3.6 SxGA分辨率的捕获（YCbCr数据格式）

说明

该示例实现旨在从照相机模块接收YCbCr数据并传输至SDRAM。捕获图像的分辨率为SxGA（1280x1024）。

以YCbCr格式接收的数据在LCD（配置为显示RGB565数据）上显示不正确。

为了正确显示图像，必须将YCbCr数据转换为RGB565数据（或RGB888或ARGB8888，具体取决于应用的需要）。

必须遵循 [第 6.3.2 节：常用配置示例](#) 所述的所有配置步骤，下面是为了获取 YCbCr 数据要添加的一些说明。必须更新的只有照相机和 DMA 配置。

DMA 配置

DMA 的配置如 [第 4.4.9 节：用于更高分辨率的 DMA 配置](#) 所述，调用 HAL_DMA_START 函数可确保该配置，因为图像大小超出双缓冲区模式的最大允许大小。

事实上，在调用 HAL_DMA_START 函数时，它确保所接收的帧被分成相等的几部分且每部分放置在一个帧缓冲区中。如前文所述，对于 SxGA 分辨率，将每一帧分给 16 个帧缓冲区。每个缓冲区的大小等于 40960 字。

对于缓冲区地址，HAL_DMA_START 功能确保存储器中 16 帧缓冲区的放置。这种情况下，第一个帧缓冲区的地址为 0xC0000000，第二个帧缓冲区的地址为 0xC0163840 (0xC0000000 + (40960 * 4))，第 16 个帧缓冲区的地址为 (0xC0000000 + 16 * (40960 * 4))。

每次传输结束时，DMA 已填充一个帧，生成中断，计算下一个缓冲区的地址，并将一个指针修改为如 [图 42：高分辨率时的 DMA 操作](#) 所示。

照相机模块的配置：

通过添加以下内容完成新照相机模块的配置：

- 用于照相机模块寄存器配置的常数表
- 用于照相机模块配置（通过 I2C 发送寄存器配置）的新函数。

为确保照相机模块发送具有 SxGA 分辨率和 YCbCr 格式的图像，必须按如下方式配置 CMOS 传感器寄存器：

1. 必须在 main.c 文件中“/* 私有变量 -----
*/”的下方添加包含照相机模块寄存器配置的表声明。

```
const unsigned char ov9655_yuv_sxga[][2]= {
{ 0x12, 0x80 }, { 0x00, 0x00 }, { 0x01, 0x80 }, { 0x02, 0x80 }, { 0x03, 0x1b }, { 0x04, 0x03 }, { 0x0e, 0x61 }, { 0x0f, 0x42 }, { 0x11, 0x00 }, { 0x12, 0x02 }, { 0x13, 0xe7 }, { 0x14, 0x3a }, { 0x16, 0x24 }, { 0x17, 0x1d }, { 0x18, 0xbd }, { 0x19, 0x01 }, { 0x1a, 0x81 }, { 0x1e, 0x04 }, { 0x24, 0x3c }, { 0x25, 0x36 }, { 0x26, 0x72 }, { 0x27, 0x08 }, { 0x28, 0x08 }, { 0x29, 0x15 }, { 0x2a, 0x00 }, { 0x2b, 0x00 }, { 0x2c, 0x08 }, { 0x32, 0xff }, { 0x33, 0x00 }, { 0x34, 0x3d }, { 0x35, 0x00 }, { 0x36, 0xf8 }, { 0x38, 0x72 }, { 0x39, 0x57 }, { 0x3a, 0x0c }, { 0x3b, 0x04 }, { 0x3d, 0x99 }, { 0x3e, 0x0c }, { 0x3f, 0xc1 }, { 0x40, 0xd0 }, { 0x41, 0x00 }, { 0x42, 0xc0 }, { 0x43, 0x0a }, { 0x44, 0xf0 }, { 0x45, 0x46 }, { 0x46, 0x62 }, { 0x47, 0x2a }, { 0x48, 0x3c }, { 0x4a, 0xfc }, { 0x4b, 0xfc }, { 0x4c, 0x7f }, { 0x4d, 0x7f }, { 0x4e, 0x7f }, { 0x52, 0x28 }, { 0x53, 0x88 }, { 0x54, 0xb0 }, { 0x4f, 0x98 }, { 0x50, 0x98 }, { 0x51, 0x00 }, { 0x58, 0x1a }, { 0x58, 0x1a }, { 0x59, 0x85 }, { 0x5a, 0xa9 }, { 0x5b, 0x64 }, { 0x5c, 0x84 }, { 0x5d, 0x53 }, { 0x5e, 0x0e }, { 0x5f, 0xf0 }, { 0x60, 0xf0 }, { 0x61, 0xf0 }, { 0x62, 0x00 }, { 0x63, 0x00 }, { 0x64, 0x02 }, { 0x65, 0x16 }, { 0x66, 0x01 }, { 0x69, 0x02 }, { 0x6b, 0x5a }, { 0x6c, 0x04 }, { 0x6d, 0x55 }, { 0x6e, 0x00 }, { 0x6f, 0x9d }, { 0x70, 0x21 }, { 0x71, 0x78 }, { 0x72, 0x00 }, { 0x73, 0x01 }, { 0x74, 0x3a }, { 0x75, 0x35 }, { 0x76, 0x01 }, { 0x77, 0x02 }, { 0x7a, 0x12 }, { 0x7b, 0x08 }, { 0x7c, 0x15 }, { 0x7d, 0x24 }, { 0x7e, 0x45 }, { 0x7f, 0x55 }, { 0x80, 0x6a }, { 0x81, 0x78 }, { 0x82, 0x87 }, { 0x83, 0x96 }, { 0x84, 0xa3 }, { 0x85, 0xb4 }, { 0x86, 0xc3 }, { 0x87, 0xd6 }, { 0x88, 0xe6 }, { 0x89, 0xf2 }, { 0x8a, 0x03 }, { 0x8c, 0x0d }, { 0x90, 0x7d }, { 0x91, 0x7b }, { 0x9d, 0x03 }, { 0x9e, 0x04 }, { 0x9f, 0x7a }, { 0xa0, 0x79 }, { 0xa1, 0x40 }, { 0xa4, 0x50 }, { 0xa5, 0x68 }, { 0xa6, 0x4a }, { 0xa8, 0xc1 }, { 0xa9, 0xef }, { 0xaa, 0x92 }, { 0xab, 0x04 }, { 0xac, 0x80 }, { 0xad, 0x80 }, { 0xae, 0x80 }, { 0xaf, 0x80 }, { 0xb2, 0xf2 }, { 0xb3, 0x20 }, { 0xb4, 0x20 }, { 0xb5, 0x00 }, { 0xb6, 0xaf }, { 0xbb, 0xae }, { 0xbc, 0x7f }, { 0xbd, 0x7f }, { 0xbe, 0x7f }, { 0xbf, 0x7f }, { 0xc0, 0xe2 }, { 0xc1, 0xc0 }, { 0xc2, 0x01 }, { 0xc3, 0x4e }, { 0xc6, 0x05 }, { 0xc7, 0x80 }, { 0xc9, 0xe0 }, { 0xca, 0xe8 }, { 0xcb, 0xf0 }, { 0xcc, 0xd8 }, { 0xcd, 0x93 }, { 0xff, 0xff } };
```

2. 必须在以下内容的下方插入新函数原型

```
/* 私有函数原型 ----- */
```

```
void OV9655_YUV_Init (uint16_t);
```

3. 本例中**main.c**文件夹的修改的第二步是通过在空间充足处插入以下函数来更新**main()**函数，如下面的**绿色粗体**部分所示。对于在快照或连续模式下用于配置DCMI的两个函数之一，必须取消注释。

```
/* 用户代码开始 2 */
```

```
BSP_SDRAM_Init();
```

```
CAMERA_Init(CameraHwAddress);
```

```
OV9655_YUV_Init(CameraHwAddress);
```

```
HAL_Delay(1000); //Delay for the camera to output correct data
```

```
Im_size = 0xA0000; //size=1280*1024*2/4
```

```
/* 对于快照模式，取消以下行的注释 */
```

```
//HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size);
```

```
/* 对于连续模式，取消以下行的注释 */
```

```
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, (uint32_t)FRAME_BUFFER, Im_size);
```

```
/* 用户代码结束 2 */
```

4. 必须通过添加新函数实现来更新**第 6.3.2 节：常用配置示例**中**main.c**文件的修改的第三步

```
/* 用户代码开始 4 */
```

```
void OV9655_YUV_Init(uint16_t DeviceAddr)
```

```
{
```

```
uint32_t index;
```

```
for(index=0; index<(sizeof(ov9655_yuv_sxga)/2); index++)
```

```
{
```

```
CAMERA_IO_Write(DeviceAddr, ov9655_yuv_sxga[index][0], ov9655_yuv_sxga[index][1]);
```

```
CAMERA_Delay(1);
```

```
}
```

```
}
```

注：对于具有RGB数据格式的SxGA帧，用户可以降低分辨率，以便使用DCMI的调整大小功能在TFT-LCD上显示接收的图像。

6.3.7 JPEG格式的捕获

说明

STM32F4DIS-Cam板中内置的OV9655 CMOS传感器不支持压缩输出数据。因此，本例使用OV2640 CMOS传感器实现，该传感器支持8位格式压缩数据。

也就是说，本例基于内置OV2640 CMOS传感器（MB1066）的STM324x9I-EVAL（REV B）板。

必须将压缩数据（JPEG）解压缩以获得YCbCr数据，并转换为可显示的RGB（举例来说），但这类实现仅用于通过DCMI接收JPEG数据并保存到SDRAM中。

本例以STM32CubeF4固件中提供的DCMI示例（SnapshotMode）（位于Projects\STM324x9I_EVAL\Examples\DCMI\DCMI_SnapshotMode中）为基础进行开发。提供的示例旨在捕获一个RGB帧（QVGA分辨率）并显示在LCD-TFT上，其配置如下：

- DCMI和I2C GPIO的配置如 [第 6.3.2 节：常用配置示例](#) 所述。
- 系统时钟的运行频率为180 MHz。
- SDRAM时钟的运行频率为90 MHz
- 将DCMI配置为在硬件同步模式下捕获8位数据宽度（未压缩数据）。
- 将相机模块配置为输出具有QVGA分辨率的RGB数据图像。

在本例的基础上，为了能够捕获JPEG数据，用户需修改DCMI和相机模块的配置。

DCMI 配置

需将DCMI配置为通过置位DCMI_CR寄存器中的JPEG位来接收压缩数据（JPEG）。为了置位该位，用户只需在stm324x9i_eval_camera.c文件中配置DCMI的“uint8_t BSP_CAMERA_Init(uint32_t Resolution)”函数（main()函数调用该函数配置DCMI和相机模块）中添加下方以**粗体**书写的说明，并保留之前的DCMI配置，如下文所示：

```
phdcmi->Init.CaptureRate = DCMI_CR_ALL_FRAME;
phdcmi->Init.HSPolarity = DCMI_HSPOLARITY_LOW;
phdcmi->Init.SynchroMode = DCMI_SYNCHRO_HARDWARE;
phdcmi->Init.VSPolarity = DCMI_VSPOLARITY_LOW;
phdcmi->Init.ExtendedDataMode = DCMI_EXTEND_DATA_8B;
phdcmi->Init.PCKPolarity = DCMI_PCKPOLARITY_RISING;
phdcmi->Init.JPEGMode = DCMI_JPEG_ENABLE;
```

相机模块配置

必须在ov2640.c文件中插入CMOS传感器（ov2640）寄存器的配置，如下文所示：

```
const unsigned char OV2640_JPEG[][2]=
{ {0xff, 0x00},{0x2c, 0xff},{0x2e, 0xdf},{0xff, 0x01},{0x12, 0x80},{0x3c, 0x32},{0x11,
0x00},{0x09,0x02},{0x04, 0x28},{0x13, 0xe5},{0x14, 0x48},{0x2c, 0x0c},{0x33, 0x78},{0x3a,
0x33},{0x3b, 0xfb},{0x3e, 0x00},{0x43, 0x11},{0x16, 0x10},{0x39, 0x02},{0x35, 0x88},{0x22,
0x0a},{0x37, 0x40},{0x23, 0x00},{0x34, 0xa0},{0x36,0x1a},{0x06, 0x02},{0x07, 0xc0},{0x0d,
0xb7},{0x0e, 0x01},{0x4c, 0x00},{0x4a, 0x81},{0x21, 0x99},{0x24, 0x40},{0x25, 0x38},{0x26,
0x82},{0x5c, 0x00},{0x63, 0x00},{0x46, 0x3f},{0x61, 0x70},{0x62, 0x80},{0x7c, 0x05},{0x20,
0x80},{0x28, 0x30},{0x6c, 0x00},{0x6d, 0x80},{0x6e, 0x00},{0x70, 0x02},{0x71,0x94},{0x73,
0xc1},{0x3d, 0x34},{0x5a, 0x57},{0x4f, 0xbb},{0x50, 0x9c},{0xff, 0x00},{0xe5, 0x7f},{0xf9,
0xc0},{0x41, 0x24},{0xe0, 0x14},{0x76, 0xff},{0x33, 0xa0},{0x42, 0x20},{0x43, 0x18},{0x4c,
```



```

0x00},{0x87, 0xd0},{0x88, 0x3f},{0xd7, 0x03},{0xd9, 0x10},{0xd3, 0x82},{0xc8, 0x08},{0xc9,
0x80},{0x7c, 0x00}, {0x7d, 0x00},{0x7c, 0x03},{0x7d, 0x48},{0x7d, 0x48},{0x7c,0x08},{0x7d,
0x20},{0x7d, 0x10},{0x7d, 0x0e},{0x90, 0x00},{0x91, 0x0e},{0x91, 0x1a},{0x91, 0x31},{0x91,
0x5a},{0x91, 0x69},{0x91, 0x75},{0x91, 0x7e},{0x91, 0x88},{0x91, 0x8f},{0x91, 0x96}, {0x91,
0xa3},{0x91, 0xaf},{0x91, 0xc4},{0x91, 0xd7},{0x91, 0xe8},{0x91, 0x20},{0x92, 0x00},{0x93,
0x06},{0x93, 0xe3},{0x93, 0x05},{0x93, 0x05},{0x93, 0x00},{0x93, 0x04},{0x93, 0x00},{0x93,
0x00},{0x93, 0x00},{0x93, 0x00},{0x93, 0x00},{0x93, 0x00},{0x93, 0x00},{0x96, 0x00},{0x97,
0x08},{0x97, 0x19},{0x97, 0x02},{0x97, 0x0c},{0x97, 0x24},{0x97, 0x30},{0x97, 0x28},{0x97,
0x26},{0x97, 0x02},{0x97, 0x98},{0x97, 0x80},{0x97, 0x00},{0x97, 0x00},{0xc3, 0xed},{0xc5,
0x11},{0xc6, 0x51},{0xbf, 0x80},{0xc7, 0x00},{0xb6, 0x66},{0xb8, 0xA5},{0xb7, 0x64},{0xb9,
0x7C},{0xb3, 0xaf},{0xb4, 0x97},{0xb5, 0xFF},{0xb0, 0xC5},{0xb1, 0x94},{0xb2, 0x0f},{0xc4,
0x5c},{0xc0, 0xc8},{0xc1, 0x96},{0x86, 0x1d},{0x50, 0x00},{0x51, 0x90},{0x52, 0x18}, {0x53,
0x00},{0x54, 0x00},{0x55, 0x88},{0x57, 0x00},{0x5a, 0x90},{0x5b, 0x18}, {0x5c, 0x05},{0xc3,
0xed},{0x7f, 0x00},{0xda, 0x00},{0xe5, 0x1f},{0xe1, 0x77},{0xe0, 0x00},{0xdd, 0x7f},{0x05,
0x00},{0xFF, 0x00},{0x05, 0x00},{0xDA, 0x10},{0xD7, 0x03},{0xDF, 0x00},{0x33, 0x80},{0x3C,
0x40}, {0xe1, 0x77}, {0x00, 0x00} };

```

为了修改照相机模块寄存器，必须将之前的表格通过I2C发送至照相机；在同一个文件（ov2640.c）的函数“void ov2640_Init(uint16_t DeviceAddr, uint32_t resolution)”中，将：

```
case CAMERA_R320x240:
```

```

{
    for(index=0; index<(sizeof(OV2640_QVGA)/2); index++)
    {
        CAMERA_IO_Write(DeviceAddr, OV2640_QVGA[index][0], OV2640_QVGA[index][1]);
        CAMERA_Delay(1);
    }
    break;
}

```

替换为：

```
case CAMERA_R320x240:
```

```
{
```



```
for(index=0; index<(sizeof( OV2640_JPEG)/2); index++)  
{  
    CAMERA_IO_Write(DeviceAddr, OV2640_JPEG[index][0], OV2640_JPEG[index][1]);  
    CAMERA_Delay(1);  
}  
break;  
}
```

7 支持的设备

要知道CMOS传感器（照相机模块）与DCMI是否兼容，用户必须检查CMOS传感器规格中的以下几点：

- 并行接口（8、10、12或14位）
- 控制信号（VSYNC、HSYNC和PIXCLK）
- 支持的像素时钟频率输出
- 支持的数据输出。

与STM32DCMI兼容的照相机模块和CMOS传感器有许多。表 12中提到了一些照相机模块。

表12. 支持的照相机模块示例

CMOS传感器	照相机模块	格式	并行接口
OV9655	STM32F4DIS-CAM	– RGB – YCbCr	– 8 位 – 10位
OV7740	TD7740-FBAC	– RGB – YCbCr	– 8 位 – 10位
MT9M001	ArduCAM	– RGB	– 8位 – 10位
OV5642	ArduCAM 5百万像素	– RGB – YCbCr	– 8位 – 10位
MT9M111	CMOS照相机	– RGB – YCbCr	– 8位
MT9P031	HDCAM	– RGB	– 8位 – 10位 – 12位
OV3640	3百万像素	– RGB – YCbCr – JPEG	– 8位 – 10位

8 结论

DCMI外设提供了一种将照相机模块连接到STM32 MCU的高效接口，该接口支持高速度、高分辨率、各种数据格式和数据宽度。

与各种外设和STM32 MCU中集成的接口一起使用时，得益于STM32智能架构，DCMI可以用在大型的复杂成像应用中。

本应用笔记涵盖不同STM32 MCU的DCMI外设，提供正确使用DCMI和成功实现应用所必需的所有信息，从兼容照相机模块的选择到具体示例的实现。

9 版本历史

表13. 文档版本历史

日期	版本	变更
2017年8月3日	1	初始版本。

表14. 中文文档版本历史

日期	版本	变更
2018年9月15日	1	中文初始版本。



重要通知 - 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对 ST 产品和 / 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用，ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。本文档的中文版本为英文版本的翻译件，仅供参考之用；若中文版本与英文版本有任何冲突或不一致，则以英文版本为准。

© 2018 STMicroelectronics - 保留所有权利