life.augmented

AH5020

Примечание по применению

Интерфейс цифровой камеры (DCMI) для микроконтроллеров STM32

Введение

Поскольку спрос на все более и более высокое качество изображения растет, область визуализации постоянно развивается, что приводит к появлению множества технологий (таких как 3D, вычислительные, движущиеся и инфракрасные).

В настоящее время приложения для обработки изображений требуют высокого качества, простоты использования, энергоэффективности, высокого уровня интеграции, быстрого выхода на рынок и экономической эффективности.

Чтобы соответствовать этим требованиям, в микроконтроллеры STM32 встроен интерфейс цифровой камеры (DCMI), позволяющий подключаться к эффективным параллельным модулям камеры.

Кроме того, микроконтроллеры STM32 обеспечивают множество уровней производительности (ЦП, подсистема микроконтроллера, DSP и FPU). Они также обеспечивают различные режимы питания, обширный набор комбинаций периферийных устройств и интерфейсов (SPI, UART, I2C, SDIO, USB, ETHERNET, I2S...), богатое графическое портфолио (LTDC, QSPI, DMA2D,...) и ведущая в отрасли среда разработки, обеспечивающая сложные приложения и решения для подключения (IOT).

Это примечание по применению дает пользователям STM32 представление об основных концепциях с простыми для понимания объяснениями функций, архитектуры и конфигурации DCMI. Он поддерживается обширным набором подробных примеров.

Справочные документы

Это примечание по применению следует читать вместе со справочными руководствами серий STM32F2, STM32F4, STM32F7 и STM32L4x6, STM32H7x3:

- *STM32F205xx, STM32F207xx, STM32F215xx и STM32F217xx с расширенными возможностями ARM®- на базе 32-битных микроконтроллеров*(RM0033)
- *STM32F405/415, STM32F407/417, STM32F427/437 и STM32F429/439 с расширенными возможностями ARM®на базе 32-битных микроконтроллеров*(0090 рингитов)
- *STM32F446xx расширенный ARM®на базе 32-битных микроконтроллеров*(0390 рингитов)
- *STM32F469xx и STM32F479xx с расширенными возможностями ARM®на базе 32-битных микроконтроллеров*(0386 рингитов)
- *STM32F75xxx и STM32F74xxx расширенный ARM®на базе 32-битных микроконтроллеров*(0385 рингитов)
- *STM32F76xxx и STM32F77xxx расширенный ARM®на базе 32-битных микроконтроллеров*(RM0410)
- *STM32L4x5 и STM32L4x6 расширенный ARM®на базе 32-битных микроконтроллеров*(RM0351)
- *STM32H7x3 расширенный ARM®на базе 32-битных микроконтроллеров*(RM0433)

Таблица 1. Применимые продукты

Тип	Линии STM32
Серия STM32F2	CTM32F2x7
Серия STM32F4	CTM32Ф407/417, CTM32Ф427/437, CTM32Ф429/439, CTM32Ф446, CTM32Ф469/479

Таблица 1. Применимые продукты (продолжение)

Тип	Линии STM32
Серия STM32F7	STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8, STM32F7x9
Серия STM32L4	CTM32L4x6
Серия STM32H7	CTM32H7x3

Содержание

Список таблиц

Список рисунков

1 Обзор: модули камеры и основные концепции

В этом разделе представлено краткое описание модулей камеры и их основных компонентов. Он также выделяет внешний интерфейс, ориентированный на параллельные модули камер.

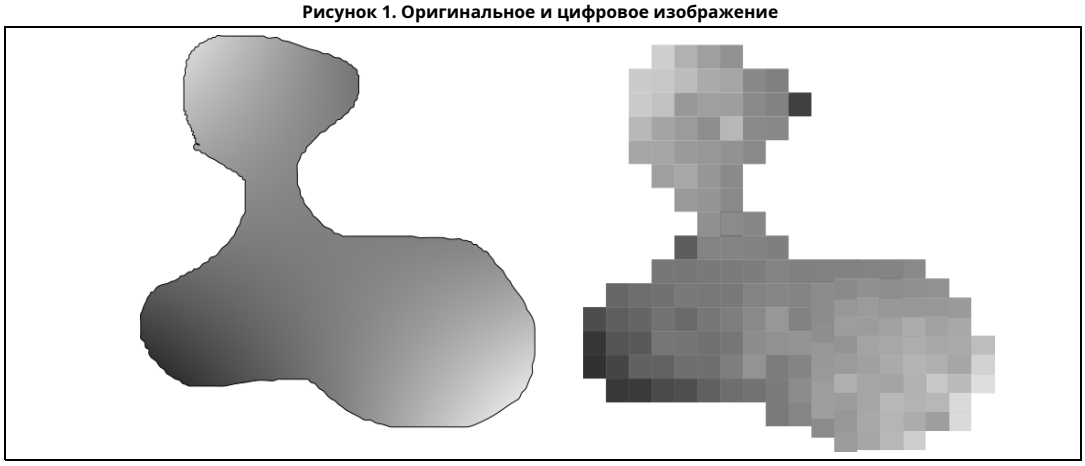
1.1 Основные понятия визуализации

В этом разделе представлено небольшое введение в область обработки изображений и дается обзор основных понятий и основ, таких как пиксели, разрешение, глубина цвета и гашение.

- **Пиксель**: каждая точка изображения представляет цвет для цветных изображений или шкалу серого для черно-белых фотографий. Цифровое приближение реконструируется, чтобы быть окончательным изображением. Это цифровое изображение представляет собой двумерный массив, состоящий из физических точек. Каждая точка называется пикселем (придумана из элементов изображения). Другими словами, пиксель — это наименьший управляемый элемент изображения. Каждый пиксель является адресуемым.

  *фигура 1*иллюстрирует разницу между исходным изображением и цифровой аппроксимацией.

Рисунок 1. Оригинальное и цифровое изображение



- **Разрешение**: количество пикселей в изображении. Чем больше увеличивается размер пикселя, тем больше увеличивается размер изображения. Для одного и того же размера изображения, чем больше количество пикселей, тем больше деталей содержит изображение.
- **Глубина цвета (разрядность)**: количество битов, используемых для указания цвета пикселя. Его также можно указать по битам на пиксель (bpp).

  Примеры:

  – Для двухтонального изображения каждый пиксель содержит один бит. Каждый пиксель либо черный, либо белый (0 или 1).

  – Для оттенков серого изображение в большинстве случаев состоит из 2 бит/пик (каждый пиксель может иметь один из четырех уровней серого) до 8 бит/пик (каждый пиксель может иметь один из 256 уровней серого).

  – Для цветных изображений количество битов на пиксель варьируется от 8 до 24 (каждый пиксель может иметь до 16777216 возможных цветов).
- **Частота кадров (для видео)**: количество кадров (или изображений), передаваемых каждую секунду, выраженное в кадрах в секунду (FPS).

- **Горизонтальное гашение**: игнорируются строки между концом одной строки и началом следующей.

Рис. 2. Иллюстрация горизонтального гашения

- **Вертикальное гашение**: игнорируются строки между концом последней строки кадра и началом первой строки в следующем кадре.

Рис. 3. Иллюстрация вертикального гашения

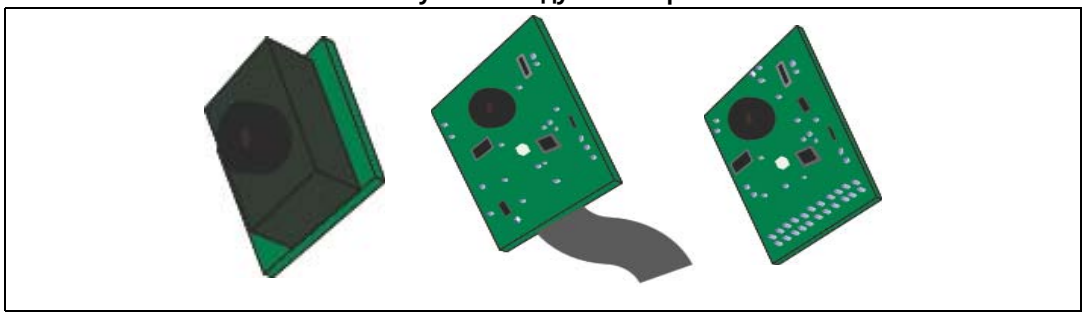
- **Прогрессивная развертка**: Это способ работы с движущимися изображениями. Он позволяет рисовать строки одну за другой последовательно, не отделяя нечетные строки от четных, как при чересстрочной развертке. Чтобы построить изображение:
  - в прогрессивной развертке рисуется первая строка, потом вторая, потом третья.
  - При чересстрочной развертке каждый кадр делится на два поля, нечетную и четную строки. Два поля отображаются попеременно.

1,2 Модуль камеры

Модуль камеры состоит из четырех частей: датчика изображения, объектива, печатной платы (PCB) и интерфейса.

*Рисунок 4*показаны некоторые распространенные примеры модулей камеры.

Рисунок 4. Модули камеры



1.2.1 Компоненты модуля камеры

Четыре компонента модуля камеры описаны ниже:

Датчик изображений

Это аналоговое устройство, позволяющее преобразовывать полученный свет в электронные сигналы. Эти сигналы передают информацию, которая составляет цифровое изображение.

В цифровых камерах можно использовать два типа датчиков:

- Датчики CCD (устройство с зарядовой связью)
- Датчики CMOS (комплементарные оксиды металлов и полупроводников).

Оба преобразуют свет в электронные сигналы, но у каждого свой метод преобразования. Поскольку их производительность постоянно растет, а стоимость снижается, КМОП-датчики изображений стали доминировать в сфере цифровой фотографии.

Объектив

Объектив представляет собой оптику, позволяющую точно воспроизводить реальное изображение, полученное на датчике изображения. Выбор подходящего объектива является частью творчества пользователя и значительно влияет на качество изображения.

Печатная плата (PCB)

Печатная плата представляет собой плату, состоящую из электронных компонентов, обеспечивающих хорошую поляризацию и защиту датчика изображения.

Печатная плата также поддерживает все остальные части модуля камеры.

Соединение модуля камеры

Интерфейс камеры представляет собой своего рода мост, позволяющий датчику изображения подключаться к встроенной системе и отправлять или получать сигналы. Сигналы, передаваемые между камерой и встроенной системой, в основном следующие:

- управляющие сигналы
- сигналы данных изображения
- сигналы питания
- сигналы конфигурации камеры.

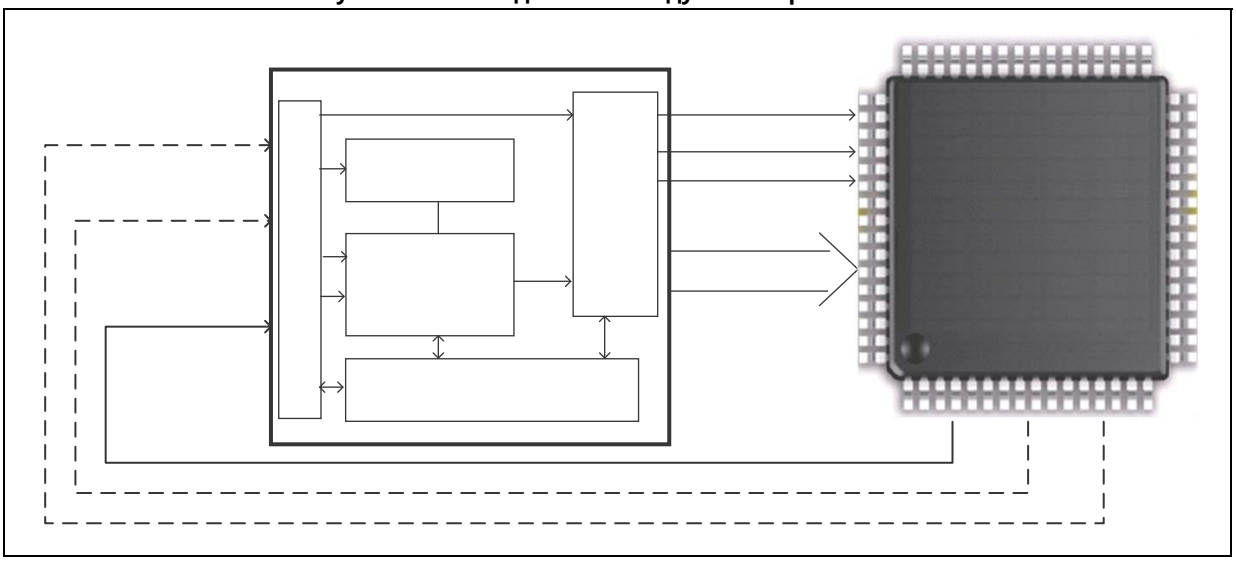
В зависимости от способа передачи сигналов данных интерфейсы камер делятся на два типа: **параллельно**а также**последовательные интерфейсы**.

1.2.2 Соединение модуля камеры (параллельный интерфейс)

Как упоминалось выше, для правильной передачи данных изображения модулю камеры требуются четыре основных типа сигналов: управляющие сигналы, сигналы данных изображения, сигналы питания и сигналы конфигурации камеры.

*Рисунок 5*иллюстрирует типичную блок-схему датчика CMOS и соединения с MCU.

Рисунок 5. Взаимодействие модуля камеры с MCU



Сигналы управления

Эти сигналы используются для генерации тактовых импульсов и синхронизации передачи данных. Часы камеры должны быть предоставлены в соответствии со спецификацией камеры.

Камера также обеспечивает сигналы синхронизации данных:

- **HSYNC,**используется для синхронизации линии
- **ВСИНХ,**используется для кадровой синхронизации.

Сигналы данных изображения

Каждый из этих сигналов передает бит данных изображения. Разрядность сигналов данных изображения представляет собой количество битов, которые должны быть переданы на каждой тактовой частоте пикселя. Это число зависит от параллельного интерфейса модуля камеры и интерфейса встроенной системы.

Сигналы питания

Как и любая встроенная электронная система, модуль камеры должен иметь источник питания. Рабочее напряжение модуля камеры указано в его паспорте.

Сигналы конфигурации

Эти сигналы используются для:

- настроить соответствующие функции изображения, такие как разрешение, формат и частота кадров
- настроить контрастность и яркость
- выберите тип интерфейса (модуль камеры может поддерживать более одного интерфейса: параллельный и последовательный интерфейс. Затем пользователь должен выбрать наиболее удобный для приложения).

Большинство модулей камер параметризуются через I2C коммуникационная шина.

2 Обзор интерфейса цифровой камеры STM32 (DCMI)

В этом разделе представлен общий предварительный обзор доступности интерфейса цифровой камеры (DCMI) на различных устройствах STM32, а также дано простое для понимания объяснение интеграции DCMI в архитектуру микроконтроллеров STM32.

2.1 Интерфейс цифровой камеры (DCMI)

Интерфейс цифровой камеры (DCMI) представляет собой синхронную параллельную шину данных. Это позволяет легко интегрировать и легко адаптироваться к конкретным требованиям приложения. DCMI соединяется с 8-, 10-, 12- и 14-битными модулями камер CMOS и поддерживает множество форматов данных.

2.2 Доступность и функции DCMI для микроконтроллеров STM32

*Таблица 2*суммирует устройства STM32 со встроенным DCMI; он также подчеркивает доступность других аппаратных ресурсов, которые облегчают работу DCMI или могут использоваться с DCMI в том же приложении.

Приложениям DCMI требуется буфер кадров для хранения захваченных изображений. Затем необходимо использовать место назначения памяти, которое зависит от размера изображения и скорости передачи.

В некоторых приложениях необходимо взаимодействовать с внешней памятью, которая предлагает большие размеры для хранения данных. По этой причине можно использовать Quad-SPI. Для получения более подробной информации см. примечания к применению*Интерфейс Quad-SPI на микроконтроллерах STM32*(AH4760).

DMA2D (контроллер Chrom-ART Accelerator™) полезен для преобразования цветовых пространств (например, из RGB565 в ARGB8888) или для передачи данных из одной памяти в другую.

Кодек JPEG позволяет сжимать данные (кодирование JPEG) или распаковывать (декодирование JPEG).

Табл. 2. DCMI и доступность связанных ресурсов

Линия СТМ32	Максимум Вспышка Память размер (байты)	Встроенный SRAM (байты)	КВАДРО СПИ	Максимум ФМС SRAM а также SDRAM частота (МГц)(1)	Максимум DCMI пиксель Часы вход (МГц) (2)	JPEG кодек	DMA2D	LCD_ TFT контроль-лер(3)	МИПИ- ДСИ хозяин(4)	Максимум АХБ частота (МГц)
СТМ32F2x7	1 М	128	Нет	60	48	Нет	Нет	Нет	Нет	120
СТМ32Ф407/417	1 М	192	Нет	60	54	Нет	Нет	Нет	Нет	168
СТМ32Ф427/437	2 м	256	Нет	90	54	Нет	Да	Нет	Нет	180
СТМ32Ф429/439	2 м	256	Нет	90	54	Нет	Да	Да	Нет	180
СТМ32Ф446	512 К	128	Да	90	54	Нет	Нет	Нет	Нет	180
СТМ32Ф469/479	2 м	384	Да	90	54	Нет	Да	Да	Да	180
СТМ32F7x5	2 м	512	Да	100	54	Нет	Да	Нет	Нет	216

Таблица 2. Доступность DCMI и связанных ресурсов (продолжение)

Линия CTM32	Максимум Вспышка Память размер (байты)	Встроенный SRAM (байты)	КВАДРО СПИ	Максимум ФМС SRAM а также SDRAM частота (МГц)(1)	Максимум DCMI пиксель Часы вход (МГц) (2)	JPEG кодек	DMA2D	LCD_ TFT контроль-лер(3)	МИПИ- ДСИ хозяин(4)	Максимум АХБ частота (МГц)
CTM32F7x6	1 М	320	Да	100	54	Нет	Да	Да	Нет	216
STM32F7x7	2 м	512	Да	100	54	Да	Да	Да	Нет	216
CTM32F7x8 STM32F7x9	2 м	512	Да	100	54	Да	Да	Да	Да	216
CTM32L4x6	1 М	320	Да	40	32	Нет	Да	Нет	Нет	80
CTM32H7x3	2 м	1000	Да	133	80	Да	Да	Да	Нет	400

1. FSMC для линеек STM32F2x7 и STM32F407/417.

2. Информацию о тактовой частоте пикселей (DCMI_PIXCLK) см. в техническом описании соответствующего устройства.

3. Дополнительные сведения о периферийном устройстве STM32 LTDC см. в примечаниях по применению AN4861.

4. Дополнительные сведения о хосте STM32 MIPI-DSI см. в примечаниях по применению AN4860.

2.3 DCMI в интеллектуальной архитектуре

DCMI подключается к матрице шины AHB через периферийную шину AHB2. К нему обращается DMA для передачи полученных данных изображения. Назначение полученных данных зависит от приложения.

Интеллектуальная архитектура микроконтроллеров STM32 позволяет:

- DMA, как мастер AHB, для автономного доступа к периферийным устройствам AHB2 и передачи полученных данных (номер изображения n+1) в память, в то время как ЦП обрабатывает ранее захваченное изображение (номер изображения n)
- DMA2D в качестве мастера AHB, который будет использоваться для передачи или изменения полученных данных и сохранения ресурсов ЦП для других задач.
- улучшение пропускной способности памяти и повышение производительности благодаря многослойной матрице шины.

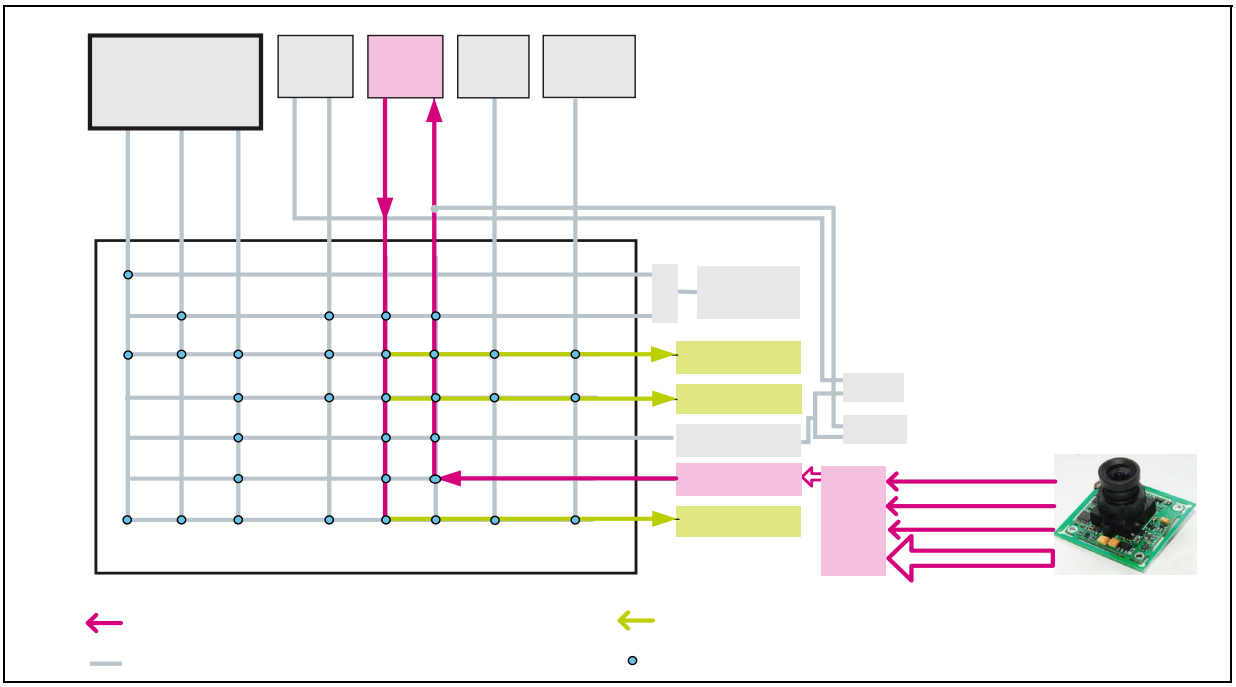
2.3.1 Системная архитектура линейки STM32F2x7

Линейные устройства STM32F2x7 основаны на 32-битной многоуровневой шинной матрице, обеспечивающей взаимосвязь между восемью ведущими и семью ведомыми устройствами.

DCMI является подчиненным периферийным устройством AHB2. DMA2 выполняет передачу данных из DCMI во внутреннюю SRAM или внешнюю память через FSMC.

*Рисунок 6*показывает соединение DCMI и путь данных в устройствах STM32F2x7xx.

Рис. 6. Ведомое периферийное устройство DCMI AHB2 в линейной интеллектуальной архитектуре STM32F2x7



2.3.2 Системная архитектура линеек STM32F407/417, STM32F427/437, STM32F429/439, STM32F446 и STM32F469/479

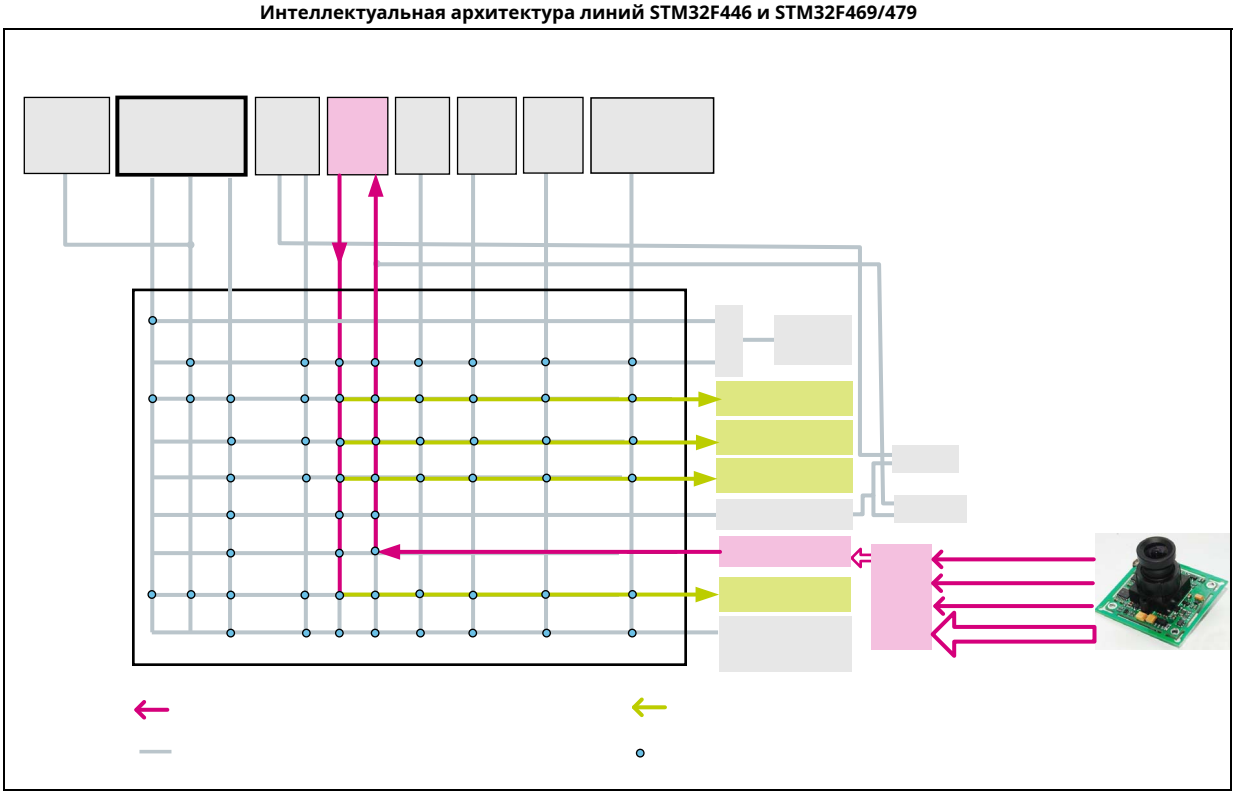
Устройства линеек STM32F407/417, STM32F427/437, STM32F429/439, STM32F446 и STM32F469/479 основаны на 32-битной многослойной шинной матрице, обеспечивающей взаимосвязь между:

- десять ведущих и восемь подчиненных для линии STM32F429/439
- десять ведущих и девять подчиненных для линии STM32F469/479
- семь мастеров и семь рабов для линии STM32F446
- восемь ведущих и семь подчиненных для STM32F407/417
- восемь ведущих и восемь подчиненных устройств для STM32F427/437.

DCMI является подчиненным периферийным устройством AHB2. DMA2 выполняет передачу данных из DCMI во внутреннюю SRAM или внешнюю память через FMC (FSMC для линии STM32F407/417).

*Рисунок 7*показано соединение DCMI и путь передачи данных в микроконтроллерах линеек STM32F407/417, STM32F427/437, STM32F429/439, STM32F446 и STM32F469/479.

Рис. 7. Периферийное устройство AHB2 ведомого устройства DCMI в STM32F407/417, STM32F427/437, STM32F429/439, Интеллектуальная архитектура линий STM32F446 и STM32F469/479



1. Для получения дополнительной информации о SRAM1, SRAM2 и SRAM3 см. *Таблица 3*.

Таблица 3. Доступность SRAM в серии STM32F4

Линия STM32	SRAM1 (Кбайт)	SRAM2 (Кбайт)	SRAM3 (Кбайт)
STM32Ф407/417	112	16	Икс
STM32Ф427/437 - STM32Ф429/439	112	16	64
STM32Ф446	112	16	Икс
STM32Ф469/479	160	32	128

2. Интерфейс Dual Quad-SPI доступен только в линейках STM32F469/479 и STM32F446.

3. 64-килобайтное ОЗУ данных CCM недоступно в устройствах STM32F446xx.

4. Интерфейс Ethernet MAC недоступен в устройствах STM32F446xx.

5. Единственными линиями, включающими LTDC и DMA2D, являются STM32F429/439 и STM32F469/479.

6. Для линии STM32F407/417 нет взаимосвязи между
   - мастер Ethernet и шина DCode флэш-памяти
   - мастер USB и шина DCode флэш-памяти.
   Для линии STM32F446 отсутствует взаимосвязь между мастером USB и шиной DCODE флэш-памяти.

7. FSMC для линейки STM32F407/417.

2.3.3 Системная архитектура линеек STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9

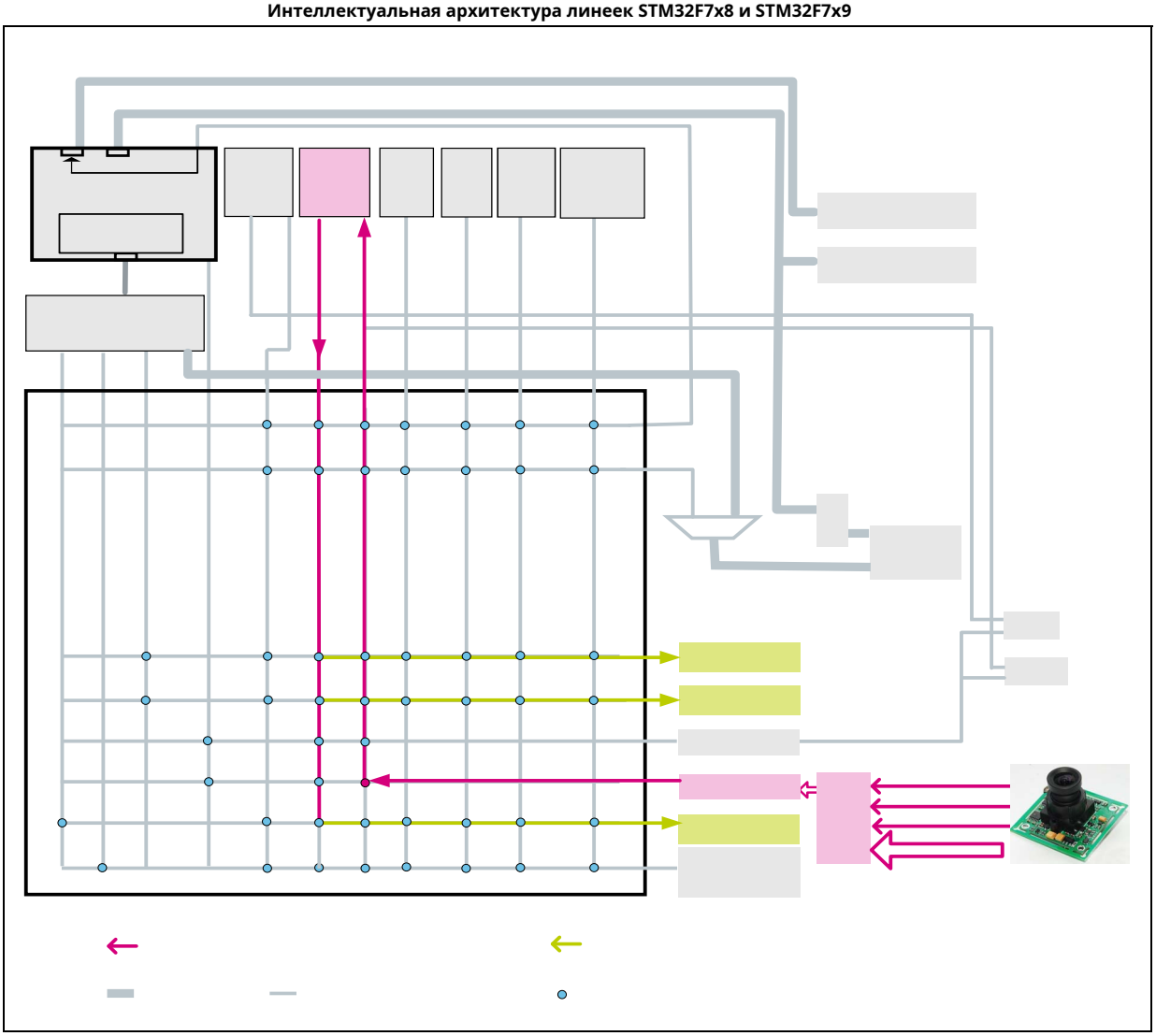
Устройства линеек STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9 основаны на 32-битной многослойной шинной матрице, позволяющей осуществлять взаимосвязь между:

- двенадцать ведущих и восемь подчиненных для линий STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9
- одиннадцать ведущих и восемь подчиненных для линии STM32F7x5.

DCMI является подчиненным периферийным устройством AHB2. DMA2 выполняет передачу данных из DCMI во внутреннюю SRAM или внешнюю память через FMC.

*Рисунок 8*показывает соединение DCMI и путь данных в линейных устройствах STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9.

Рисунок 8. Периферийное устройство AHB2 подчиненного устройства DCMI в STM32F7x5, STM32F7x6, STM32F7x7, Интеллектуальная архитектура линеек STM32F7x8 и STM32F7x9



1. Размер кэша I/D:
   - 4 Кбайт для линий STM32F7x5 и STM32F7x6
   - 16 Кбайт для линий STM32F7x7, STM32F7x8 и STM32F7x9.

2. LTDC (LCD-TFT контроллер) доступен только в линейках STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9.

3. Размер ОЗУ DTCM:
   - 64 Кбайт для линий STM32F7x5 и STM32F7x6
   - 128 Кбайт для линий STM32F7x7, STM32F7x8 и STM32F7x9.

4. Размер оперативной памяти ITCM составляет 16 Кбайт для линий STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9.

5. Размер SRAM1:
   - 240 Кбайт для линий STM32F7x5 и STM32F7x6
   - 368 Кбайт для линий STM32F7x7, STM32F7x8 и STM32F7x9.

6. Размер SRAM2 составляет 16 Кбайт для линий STM32F7x5, STM32F7x6, STM32F7x7, STM32F7x8 и STM32F7x9.

2.3.4 Системная архитектура устройств STM32L496 xx и STM32L4A6xx

Устройства STM32L496xx и STM32L4A6xx основаны на 32-битной многоуровневой шинной матрице, обеспечивающей взаимосвязь между шестью ведущими и восемью ведомыми устройствами.

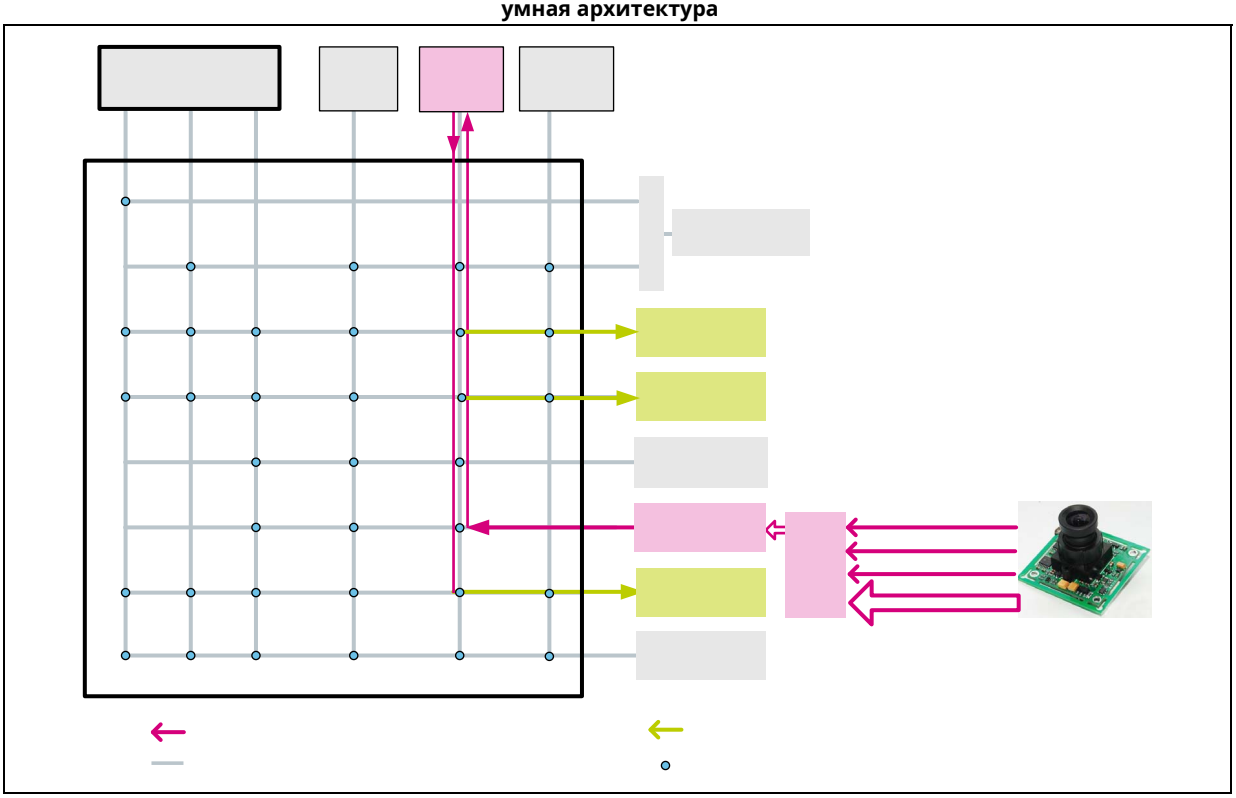
DCMI является подчиненным периферийным устройством AHB2. DMA2 выполняет передачу данных из DCMI во внутреннюю SRAM или внешнюю память через FMC.

В микроконтроллерах STM32L496xx и STM32L4A6xx DMA имеет только один порт (в отличие от серий STM32F2, STM32F4, STM32F7 и STM32H7, где периферийный порт отделен от порта памяти), но он поддерживает кольцевое управление буфером, периферийное устройство-память, память-периферийное устройство. и передачи от периферии к периферии.

Рисунок 9 показывает соединение DCMI и путь данных в устройствах STM32L496xx и STM32L4A6xx.

Рис. 9. Периферийное устройство AHB2 ведомого устройства DCMI в устройствах STM32L496xx и STM32L4A6xx

умная архитектура



2.3.5 Системная архитектура линейки STM32H7x3

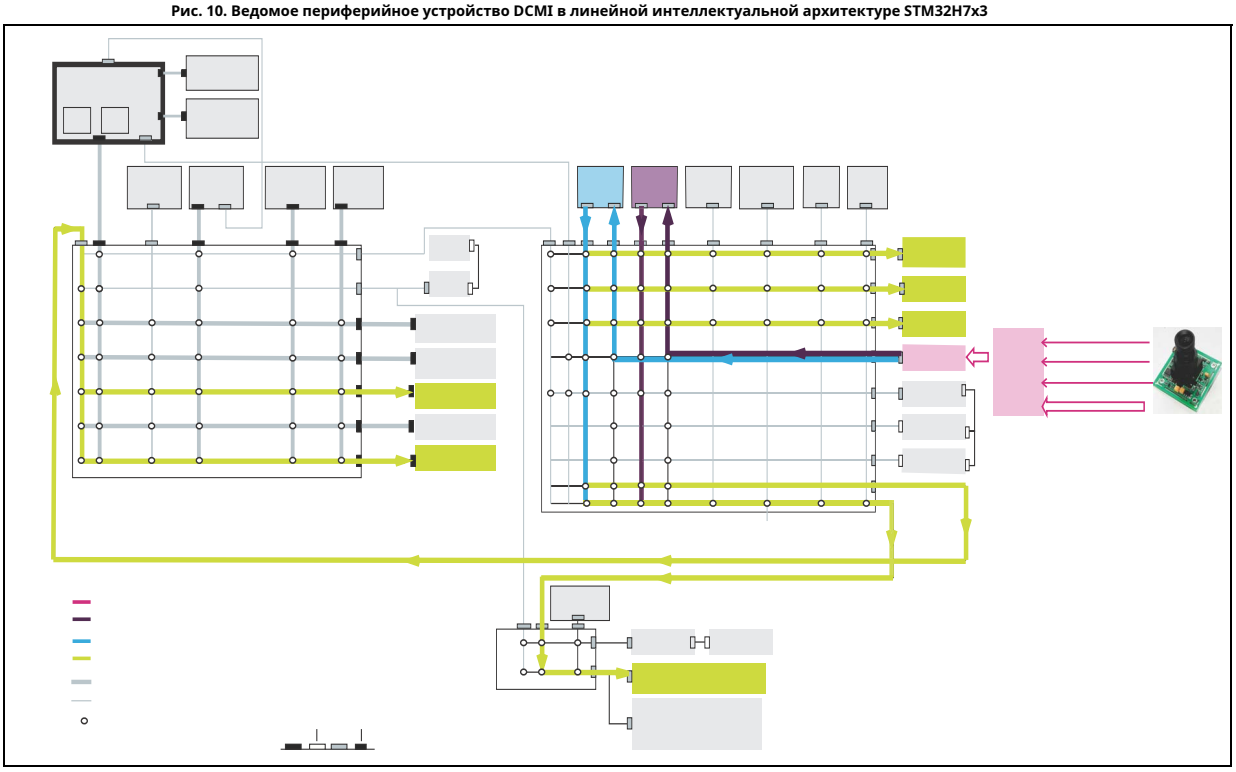
Устройства STM32H7x3xx основаны на матрице шин AXI, двух матрицах шин AHB и шинных мостах, обеспечивающих взаимосвязь между 18 ведущими и 20 ведомыми устройствами.

DCMI является подчиненным периферийным устройством AHB2. DMA1 или DMA2 могут выполнять передачу данных из DCMI во внутреннюю SRAM или внешнюю память через FMC.

DMA1 и DMA2 расположены в домене D2. Они могут получить доступ к ведомым устройствам в домене D1 и домене D3. В результате DMA1 или DMA2 могут передавать данные, полученные DCMI (расположенным в домене 2), в память, расположенную в домене 1 или домене 3.

*Рисунок 10*показывает соединение DCMI и путь данных в устройствах STM32H7x3xx.

Рис. 10. Ведомое периферийное устройство DCMI в линейной интеллектуальной архитектуре STM32H7x3



2,4 Эталонные платы с модулями DCMI и/или камеры

Доступно множество эталонных плат STM32, таких как платы NUCLEO, Discovery и EVAL. Большинство из них имеют встроенный DCMI, а некоторые из них имеют встроенный модуль камеры.

Выбор платы зависит от приложения и аппаратных ресурсов.

*Таблица 4*обобщает DCMI, модули камеры и доступность памяти на различных эталонных платах STM32.

Таблица 4. Модули DCMI и камеры на различных платах STM32(1)

Линия STM32	Доска	Камера модуль	КМОП датчик	Внутренний SRAM (Кбайт)	Внешний SDRAM ширина шины (биты)	Внешний Шина SRAM ширина (бит)
CTM32F2x7	STM3220G-EVAL	Да(2)	OV2640 или OV9655	132	нет данных	
	STM3221G-EVAL	Да(2)				
CTM32Ф407/417	STM32F4DISCOVERY	Да(3) или (4)	OV9655	196		
	STM3240G-EVAL	Да(2)				
	STM3241G-EVAL					
CTM32Ф429/439	32F429IDISCOVERY	нет данных(3)	нет данных	256	16	нет данных
	STM32429I-EVAL	Да(2)	OB264 0 или OV9655		32	16
	STM32439I-EVAL					
CTM32Ф446	STM32446E-EVAL	Да(2)	S5k5CAGA	128	16	нет данных
CTM32Ф469/479	32F469IDISCOVERY	нет данных(3)	нет данных	324	32	нет данных
	STM32469I-EVAL	Да(2)	S5k5CAGA			16
	STM32479I-EVAL					
CTM32F7x6	32F746GОБНАРУЖЕНИЕ	Да(4)	OV9655	320	16	нет данных
	STM32746G-EVAL	Да(2)	S5k5CAGA		32	16
	STM32756G-EVAL					
STM32F7x9	32F769IDISCOVERY	нет данных(3)	нет данных	512	32	нет данных
	CTM32Ф769И-ЭВАЛ	Да(2)	S5k5CAGA			16
	STM32F779I-EVAL					
CTM32L4x6	32L496GДИСКАВЕРИ	Да(4)	OV9655	320	нет данных	нет данных
CTM32H7x3	CTM32X743И-ЭВАЛ CTM32X753И-ЭВАЛ	нет данных(3)	нет данных	864	32	16

1. NA: недоступно. Пользователь должен использовать желаемый модуль камеры, совместимый с интерфейсом DCMI.

2. Для различных плат EVAL специальный разъем обеспечивает соединение между DCMI и модулем камеры.
   - Для STM3220G-EVAL, STM3221G-EVAL, STM32F40G-EVAL и STM32F41G-EVAL возможно подключение двух камер: модуль CN01302H1045-C (датчик CMOS OV9655, 1,3 мегапикселя) и модуль CN020VAH2554-C (датчик CMOS OV2640, 2 шт.) мегапикселей).
   - Для STM32429I-EVAL и STM32439I-EVAL подключена дочерняя плата модуля камеры MB1066.
   - Для STM32446E-EVAL, STM32469I-EVAL, STM32F479I-EVAL, STM32746G-EVAL, STM32756G-EVAL, STM32F769I-EVAL и STM32F779I-EVAL подключена дочерняя плата модуля камеры MB1183.

3. Модуль камеры можно подключить к DCMI через контакты GPIO.

4. Модуль камеры можно подключить к DCMI через FFC (гибкий плоский кабель):
   - Для STM32F4DISCOVERY для подключения модуля камеры STM32F4DIS-CAM следует использовать плату расширения STM32F4DIS-EXT.
   - Для 32F746IDISCOVERY и 32L496GDISCOVERY плата STM32F4DIS-CAM может быть подключена напрямую. Для получения более подробной информации о STM32F4DIS-EXT и STM32F4DIS-CAM посетите веб-сайт STMicroelectronics.

3 Описание DCMI

В этом разделе подробно описывается DCMI и его способ работы с данными изображения и сигналами синхронизации.

Примечание: *DCMI поддерживает только подчиненный режим ввода.*

3.1 Аппаратный интерфейс

ДКМИ состоит из:

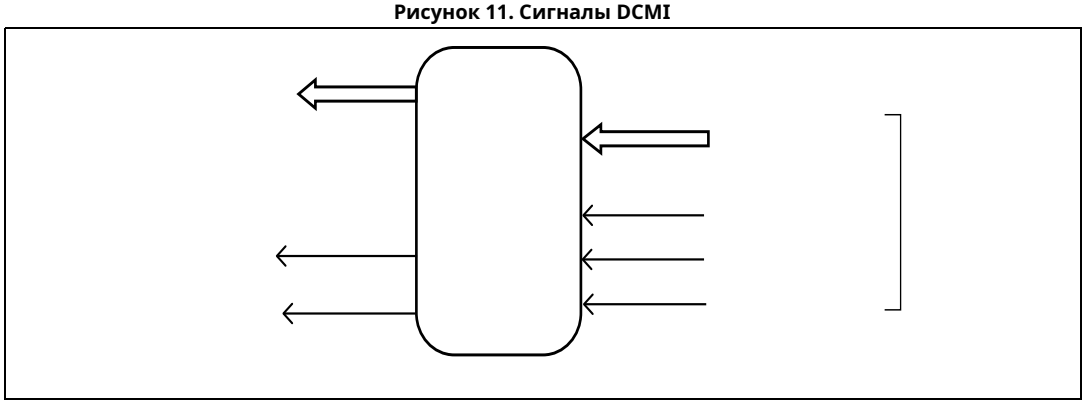
- до 14 строк данных (D13-D0)
- линия синхронизации пикселей DCMI_PIXCLK
- линия DCMI_HSYNC (горизонтальная синхронизация)
- линия DCMI_VSYNC (вертикальная синхронизация).

DCMI содержит до 17 входов. В зависимости от количества линий данных, включенных пользователем (8, 10, 12 или 14), количество входов DCMI различается (11, 13, 15 или 17 сигналов).

Если используется ширина данных менее 14 бит, неиспользуемые контакты не должны назначаться DCMI через альтернативную функцию GPIO. Неиспользуемые входные контакты могут быть назначены другим периферийным устройствам.

В случае встроенной синхронизации DCMI требуется только девять входов (восемь линий данных и DCMI_PIXCLK) для правильной работы. Восемь неиспользуемых контактов можно использовать для GPIO или других функций.

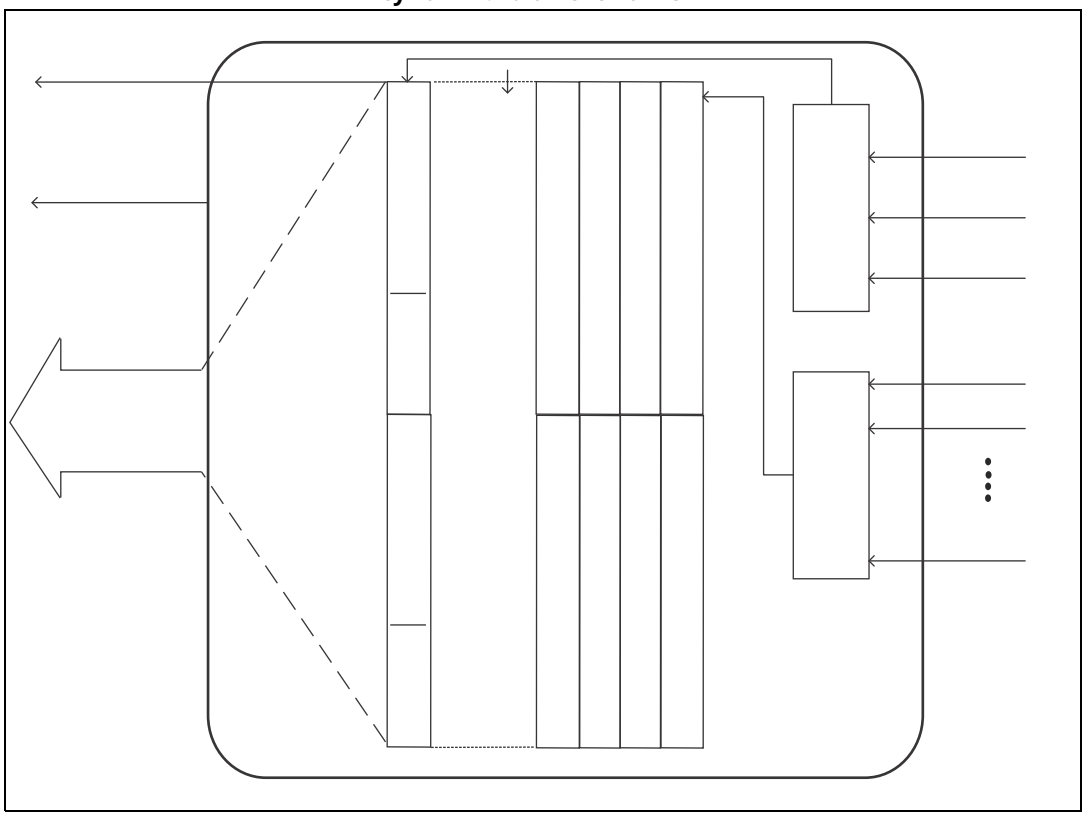
Рисунок 11. Сигналы DCMI



Если выбрана ширина данных x-бит (разрешено x строк данных и x равно 8, 10, 12 или 14), x бит данных изображения (или видео) передаются в каждом цикле DCMI_PIXCLK и упаковываются в 32-битный регистр.

Как показано в*Рисунок 12*, DCMI состоит из четырех основных компонентов:

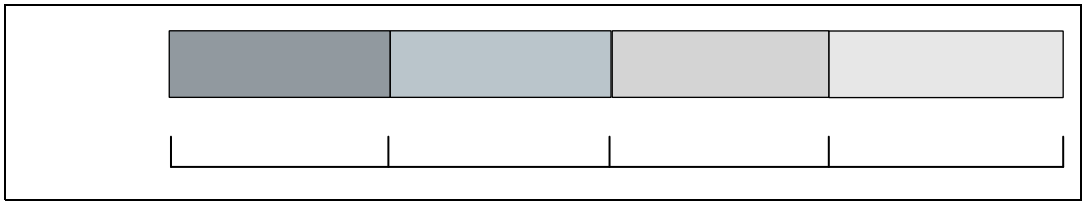
Рисунок 12. Блок-схема DCMI



- **Синхронизатор DCMI**: обеспечивает контроль упорядоченной последовательности потока данных через DCMI. Он управляет экстрактором данных, FIFO и 32-битным регистром.
- **Извлечение данных**: обеспечивает извлечение данных, полученных DCMI.
- **ФИФО**: этот FIFO из 4 слов реализован для адаптации скорости передачи данных к AHB. Нет защиты от переполнения для предотвращения перезаписи данных, если AHB не поддерживает скорость передачи данных. В случае переполнения или ошибок в сигналах синхронизации FIFO сбрасывается, и DCMI ожидает нового начала кадра.
- **32-битный регистр**: регистр данных, в котором биты данных упакованы для передачи по каналу DMA общего назначения. Размещение захваченных данных в 32-битном регистре зависит от разрядности данных:

  - За**8-битная ширина данных**, DCMI захватывает восемь LSB (шесть других входов D[13:8] игнорируются). Первый захваченный байт данных помещается в позицию LSB 32-битного слова, а четвертый захваченный байт данных помещается в позицию MSB.
    Итак, в этом случае 32-битное слово данных составляется каждые четыре такта пикселя.

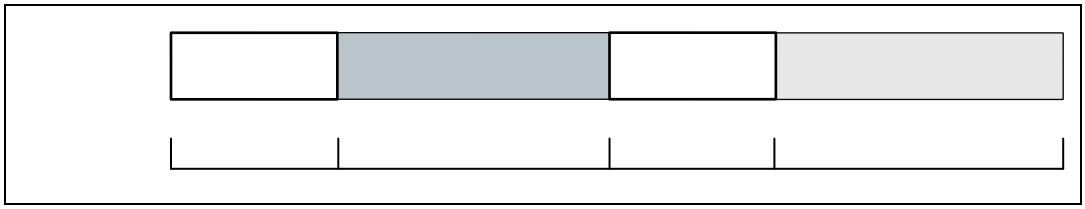
Рисунок 13. Регистр данных заполнен для 8-битной разрядности данных



для получения более подробной информации см.*Раздел 3.6: Форматы и хранение данных.*

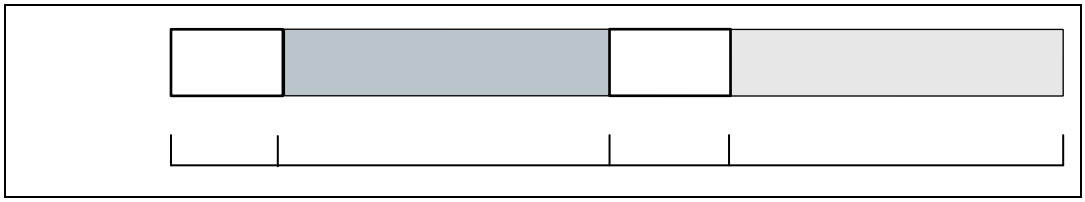
- За**10-битная ширина данных**, DCMI захватывает 10 LSB (четыре других входа D[13:10] игнорируются). Первые 10 захваченных битов помещаются как 10 младших разрядов 16-битного слова. Остальные старшие биты в 16-битном слове регистра DCMI_DR (биты с 10 по 15) очищаются.
  Итак, в этом случае 32-битное слово данных составляется каждые два такта пикселя.

Рис. 14. Регистр данных заполнен для разрядности данных 10 бит.



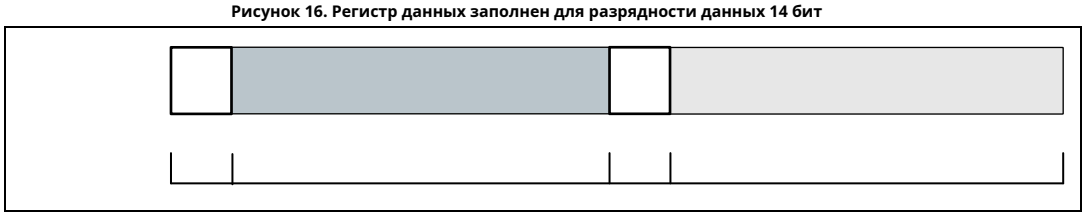
- За**12-битная ширина данных**, DCMI захватывает 12-битные младшие биты (два других входа D[13:12] игнорируются). Первые 12 захваченных битов помещаются как 12 младших разрядов 16-битного слова. Остальные старшие биты в 16-битном слове регистра DCMI_DR (биты с 12 по 15) очищаются.
  Итак, в этом случае 32-битное слово данных составляется каждые два такта пикселя.

Рис. 15. Регистр данных заполнен для разрядности данных 12 бит.



- За**14-битная ширина данных**, DCMI захватывает все полученные биты. Первые 14 захваченных битов помещаются как 14 младших разрядов 16-битного слова. Остальные старшие биты в 16-битном слове регистра DCMI_DR (биты 14 и 15) очищаются.
  Итак, в этом случае 32-битное слово данных составляется каждые два такта пикселя.

Рисунок 16. Регистр данных заполнен для разрядности данных 14 бит

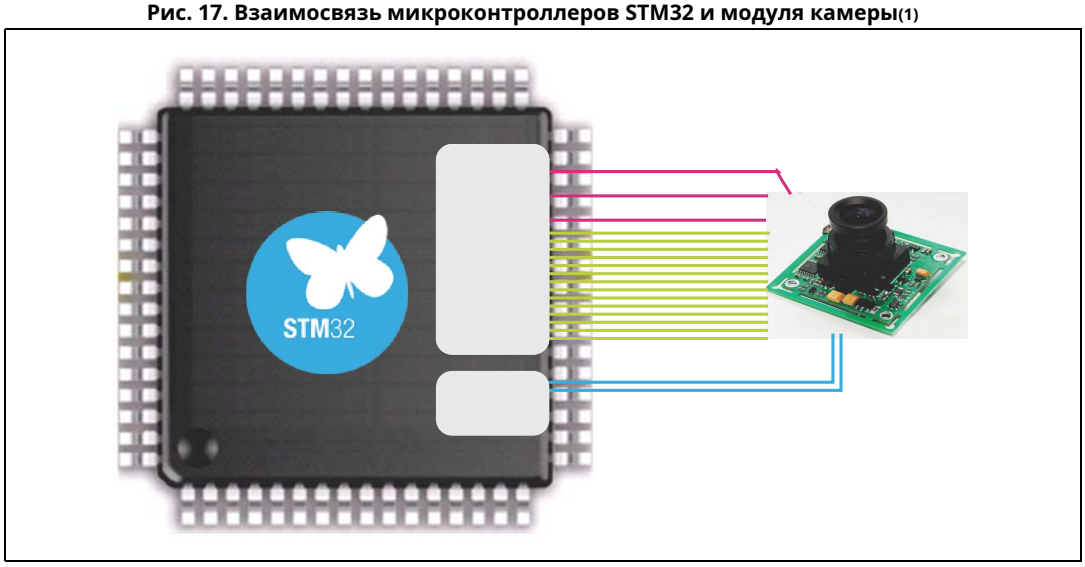


3.2 Модуль камеры и соединение DCMI

Как упоминалось в *Раздел 1.2.2: Соединение модуля камеры (параллельный интерфейс)*, модуль камеры подключается к DCMI через три типа сигналов:

- Часы DCMI и сигналы данных
- Сигналы конфигурации I2C

Рис. 17. Взаимосвязь микроконтроллеров STM32 и модуля камеры(1)



1. Для встроенной синхронизации сигналы DCMI_HSYNC и DCMI_VSYNC игнорируются и используются только 8 сигналов данных.

3.3 Функциональное описание DCMI

Следующие шаги обобщают работу внутренних компонентов DCMI и дают пример потока данных через матрицу системной шины:

- После получения различных сигналов синхронизатор управляет потоком данных через различные компоненты DCMI (извлекатель данных, FIFO и 32-битный регистр данных).
- После извлечения экстрактором данные упаковываются в FIFO из 4 слов, а затем упорядочиваются в 32-битном регистре.
- Как только 32-битный блок данных упакован в регистр, генерируется запрос DMA.
- DMA передает данные в соответствующее место назначения памяти.
- В зависимости от приложения данные, хранящиеся в памяти, могут обрабатываться по-разному.

Примечание: *Предполагается, что вся предварительная обработка изображения выполняется в модуле камеры.*

3.4 Синхронизация данных

Интерфейс камеры имеет настраиваемый параллельный интерфейс данных от 8 до 14 линий данных вместе с линией синхронизации пикселей DCMI_PIXCLK (нарастающий/задний фронт), линией горизонтальной синхронизации DCMI_HSYNC и линией вертикальной синхронизации DCMI_VSYNC с программируемой полярностью.

Такты DCMI_PIXCLK и AHB должны соблюдать минимальное соотношение AHB/DCMI_PIXCLK, равное 2,5.

Некоторые модули камеры поддерживают два типа синхронизации, в то время как другие поддерживают либо аппаратную, либо встроенную синхронизацию.

3.4.1 Аппаратная (или внешняя) синхронизация

В этом режиме для синхронизации используются два сигнала DCMI_VSYNC и DCMI_HSYNC:

- Синхронизация линии всегда обозначается как DCMI_HSYNC (также известная как LINE VALID).
- Кадровая синхронизация всегда называется DCMI_VSYNC (также известна как FRAME VALID).

Полярность DCMI_PIXCLK и сигналов синхронизации (DCMI_HSYNC и DCMI_VSYNC) программируется.

Данные синхронизируются с DCMI_PIXCLK и изменяются по переднему или заднему фронту тактовой частоты пикселей, в зависимости от настроенной полярности.

Если сигналы DCMI_VSYNC и DCMI_HSYNC запрограммированы на активный уровень (активный высокий или активный низкий), данные недействительны в параллельном интерфейсе, когда VSYNC или HSYNC находятся на этом уровне (высоком или низком).

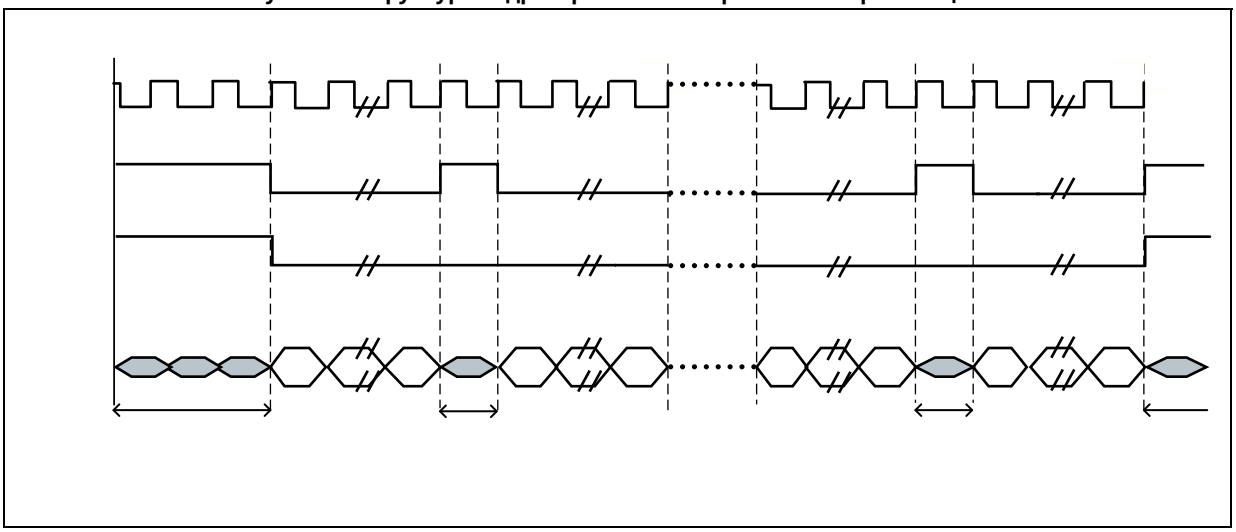
Например, если VSYNC запрограммирован на активный высокий уровень:

- когда VSYNC низкий, данные действительны
- когда VSYNC находится на высоком уровне, данные недействительны (вертикальное гашение).

Сигналы DCMI_HSYNC и DCMI_VSYNC действуют как сигналы гашения, поскольку все данные, полученные в течение активных периодов DCMI_HSYNC/DCMI_VSYNC, игнорируются.

Рисунок 18 показывает пример передачи данных, когда DCMI_VSYNC и DCMI_HSYNC имеют активный высокий уровень, а фронт захвата для DCMI_PIXCLK является нарастающим фронтом.

Рисунок 18. Структура кадра в режиме аппаратной синхронизации



Синхронизация сжатых данных

Для сжатых данных (JPEG) DCMI поддерживает только аппаратную синхронизацию. Каждый поток JPEG делится на пакеты. Эти пакеты имеют программируемый размер. Отправка пакетов зависит от содержимого изображения и приводит к переменной продолжительности гашения между двумя пакетами.

DCMI_HSYNC используется для сигнализации о начале/конце пакета.

DCMI_VSYNC используется для сигнализации о начале/конце потока.

Если полный поток данных завершается, а определение конца потока не происходит (DCMI_VSYNC не изменяется), DCMI дополняет конец кадра, вставляя нули.

3.4.2 Встроенная (или внутренняя) синхронизация

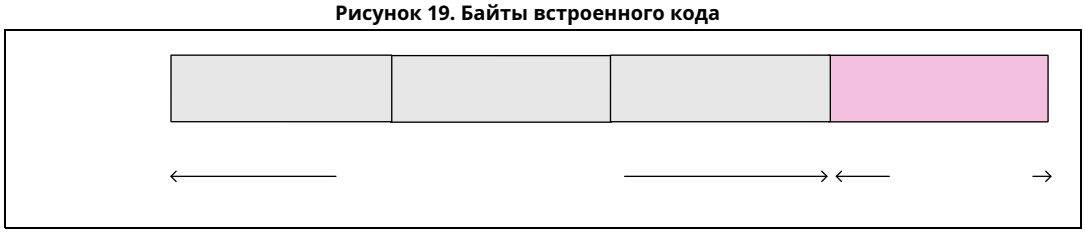
В этом случае для синхронизации используются коды-разделители. Эти коды встроены в поток данных для указания начала/конца строки или начала/конца кадра.

Примечание: *Эти коды поддерживаются только для 8-битной ширины интерфейса параллельных данных. Для другой ширины данных этот режим приводит к непредсказуемым результатам и не должен использоваться.*

Эти коды устраняют необходимость в DCMI_HSYNC и DCMI_VSYNC для сигнализации конца/начала строки или кадра. Когда используется этот режим синхронизации, есть два значения, которые должны быть**не использовать для данных: 0 и 255 (0x00 и 0xFF)**. Эти два значения зарезервированы для идентификации данных. Модуль камеры должен контролировать значения данных. По этой причине данные изображения могут иметь только 254 возможных значения (0x00 < значение данных изображения < 0xFF).

Каждый код синхронизации состоит из 4-байтовой последовательности**0xFF 00 00 XY,**где все коды-разделители имеют одинаковую первую 3-байтовую последовательность 0xFF 00 00. Только последний 0xXY запрограммирован для обозначения соответствующего события.

Рисунок 19. Байты встроенного кода



Режим 1

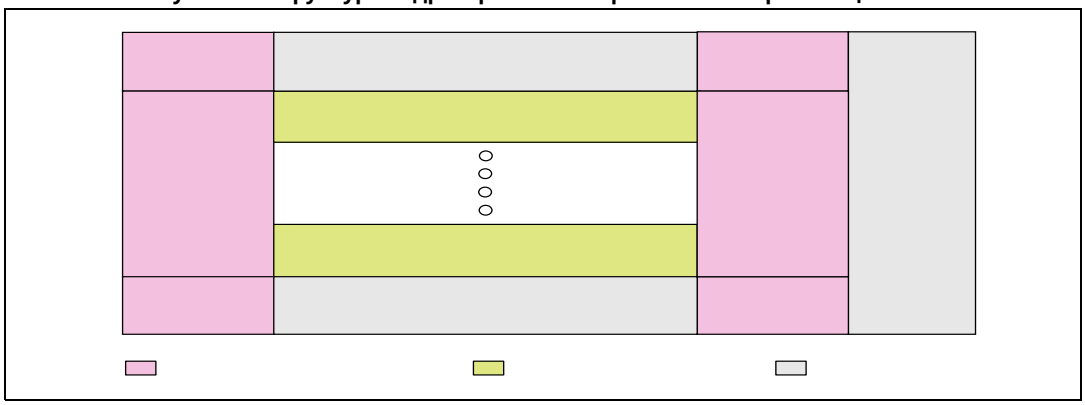
Этот режим совместим с ITU656 (ITU656 — это цифровой видеопротокол ITU-R BT.656).

Существует четыре справочных кода, обозначающих набор из четырех событий:

- **SAV (активная линия)**: начало строки
- **EAV (активная линия)**: конец строки
- **CAB (гашение)**: начало строки во время периода гашения между кадрами
- **EAV (гашение)**: конец строки во время периода гашения между кадрами.

*Рисунок 20*иллюстрирует структуру кадра, использующую этот режим.

Рисунок 20. Структура кадра в режиме встроенной синхронизации 1



Режим 2

В этом режиме встроенные коды синхронизации сигнализируют о другом наборе событий:

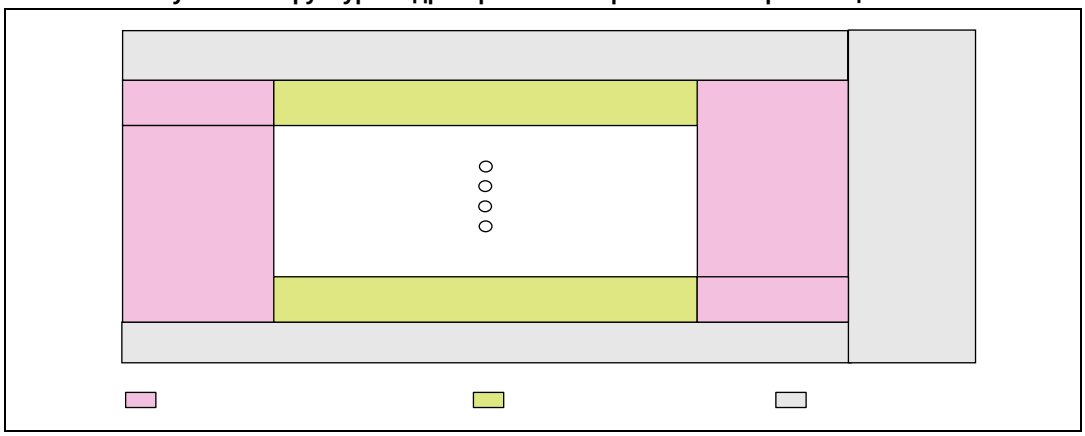
- начало кадра (**ФС**)
- конец кадра (**КЭ**)
- начало строки (**ЛС**)
- конец строки (**ЛЭ**)

Значение 0xFF, запрограммированное как конец кадра (FE), означает, что все неиспользуемые коды (возможные значения кодов, кроме FS, LS, LE) интерпретируются как действительные коды FE.

В этом режиме после включения интерфейса камеры захват кадра начинается после первого появления кода FE, за которым следует код FS.

*Рисунок 21*иллюстрирует структуру кадра при использовании этого режима.

Рисунок 21. Структура кадра в режиме встроенной синхронизации 2



Примечание:

Модули камеры могут иметь до восьми кодов синхронизации в чередующемся режиме. По этой причине этот чередующийся режим не поддерживается интерфейсом камеры (иначе каждая вторая половина кадра будет отброшена).
При использовании встроенного режима синхронизации DCMI не поддерживает сжатые данные (JPEG) и функцию кадрирования.

Встроенные демаскирующие коды

Эти коды также используются для обозначения начала/конца строки или начала/конца кадра. Благодаря этим кодам вместо того, чтобы сравнивать весь полученный код с запрограммированным для установки соответствующего события, пользователь может выбрать только некоторые немаскированные биты для сравнения с битами запрограммированного кода, имеющими ту же позицию.

Другими словами, пользователь применяет маску к соответствующему коду, настраивая регистр демаскирования встроенной синхронизации DCMI (DCMI_ESUR). Каждый байт в этом регистре представляет собой демаскирующий код, соответствующий встроенному коду синхронизации:

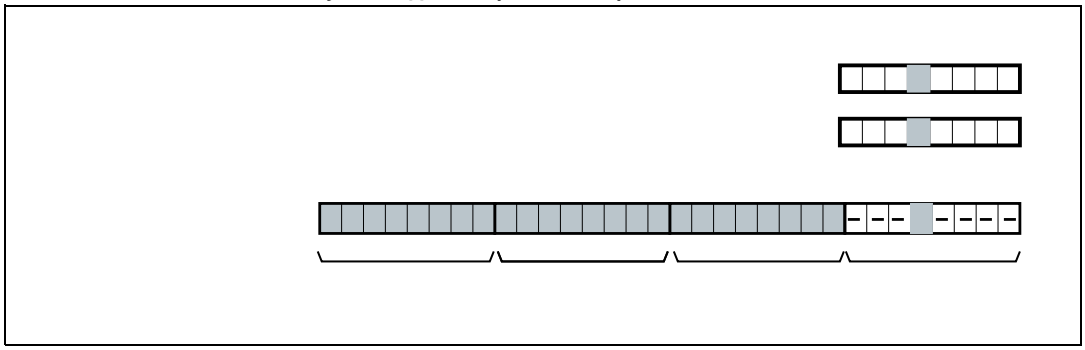
- Самый значащий байт — это демаскирующий разделитель конца кадра (FEU): каждый бит, установленный в 1, подразумевает, что этот бит в коде конца кадра необходимо сравнить с полученными данными, чтобы узнать, является ли это событием конца кадра. или нет.
- Второй байт представляет собой демаскировку ограничителя конца строки (LEU): каждый бит, установленный в 1, подразумевает, что этот бит в коде конца строки необходимо сравнить с полученными данными, чтобы узнать, является ли это событием конца строки или нет.
- Третий байт представляет собой демаскировку разделителя начала строки (LSU): каждый бит, установленный в 1, подразумевает, что этот бит в коде начала строки необходимо сравнить с полученными данными, чтобы узнать, является ли это событием начала строки или нет.
- Меньший значащий байт представляет собой демаскировку разделителя начала кадра (FSU): каждый бит, установленный в 1, означает, что этот бит в коде начала кадра необходимо сравнить с полученными данными, чтобы узнать, является ли это событием начала кадра. или нет.

В результате для каждого события могут быть разные коды (начало строки или конец строки, начало или конец кадра), но все они (разные коды, соответствующие одному событию) имеют немаскированные биты в одной и той же позиции. (тот же демаскирующий код).

Пример: FSC = 0xA5 и код размаскировки FSU = 0x10.

В этом случае информация о начале кадра встраивается в бит номер 4 кода FS. В результате пользователь должен сравнить только бит номер 4 полученного кода с битом номер 4 запрограммированного кода, чтобы узнать, является ли это событием начала кадра или нет.

Рисунок 22. Демаскирование встроенных кодов



Примечание: *Убедитесь, что каждый код синхронизации имеет другой код демаскирования, чтобы избежать ошибок синхронизации.*

3,5 Режимы захвата

DCMI поддерживает два типа захвата:**снимок**(один кадр) и**непрерывный**захват (последовательность кадров).

В зависимости от конфигурации регистра DCMI_CR пользователь может управлять скоростью захвата, выбирая байты, строки и кадры для захвата.

Эти функции используются для преобразования цветового формата изображения и/или уменьшения разрешения изображения (при захвате одной строки из двух разрешение по вертикали будет делиться на 2).

Для получения более подробной информации см. *Раздел 3.7.2: Изменение размера изображения (изменение разрешения)*.

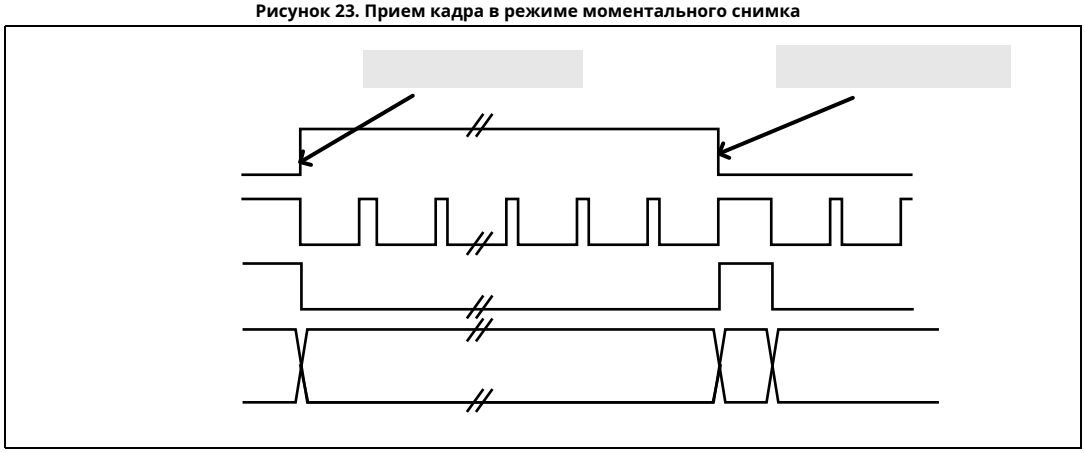
3.5.1 Режим моментального снимка

В режиме моментального снимка захватывается один кадр. После включения захвата путем установки бита CAPTURE в регистре DCMI_CR интерфейс ожидает обнаружения начала кадра (следующего DCMI_VSYNC или следующего внедренного кода начала кадра, в зависимости от режима синхронизации) перед выборкой данных.

После получения первого полного кадра DCMI автоматически отключается (бит CAPTURE автоматически сбрасывается), а все остальные кадры игнорируются.

В случае переполнения кадр теряется и интерфейс камеры отключается.

Рисунок 23. Прием кадра в режиме моментального снимка



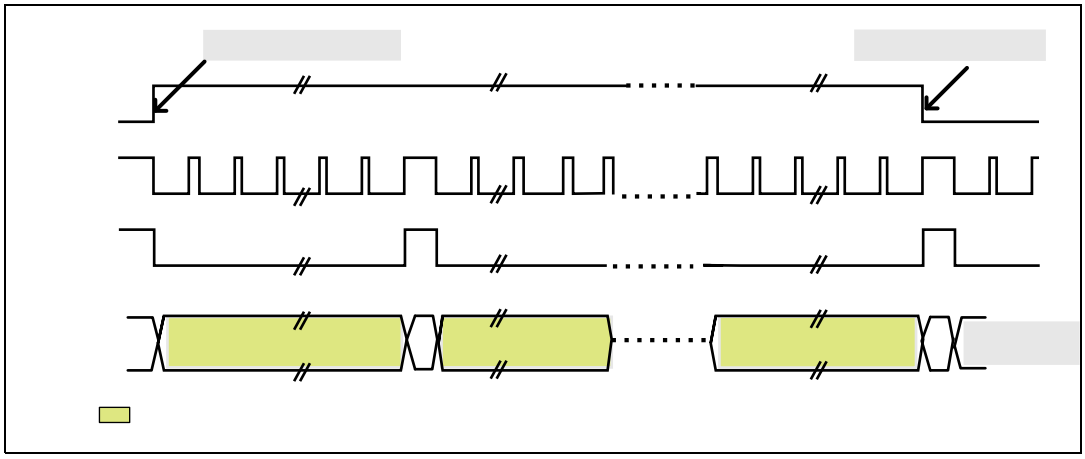
3.5.2 Непрерывный режим захвата

После выбора этого режима и включения захвата (установлен бит CAPTURE) интерфейс ожидает обнаружения начала кадра (следующего DCMI_VSYNC или следующего встроенного кода начала кадра, в зависимости от режима синхронизации) перед выборкой данных. .

В этом режиме DCMI можно настроить на захват всех кадров, каждого чередующегося кадра (уменьшение полосы пропускания на 50%) или одного кадра из четырех (уменьшение полосы пропускания на 75%).

В этом случае интерфейс камеры не отключается автоматически, но пользователь должен отключить его, установив бит CAPTURE в ноль. После отключения пользователем DCMI продолжает захват данных до конца текущего кадра.

Рисунок 24. Прием кадров в режиме непрерывного захвата



3,6 Форматы данных и хранение

DCMI поддерживает следующие форматы данных:

- 8-битное прогрессивное видео: либо монохромное, либо необработанное Байер
- Прогрессивное видео YCbCr 4:2:2
- Прогрессивное видео RGB565
- сжатые данные (JPEG).

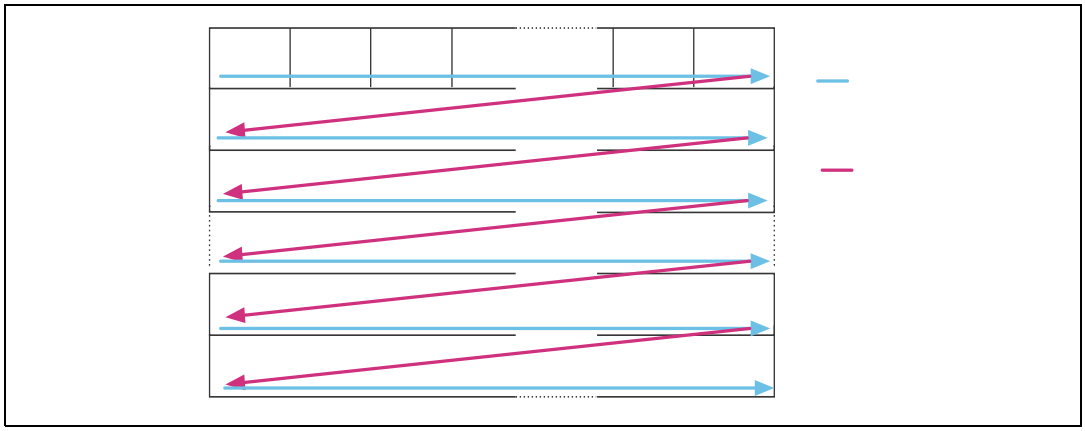
Для монохромных данных, данных RGB или YCbCr:

- максимальный размер ввода 2048 * 2048 пикселей
- буфер кадров хранится в растровом режиме.

Для сжатых данных JPEG нет ограничений по размеру.

Для монохромного, RGB и YCbCr буфер кадров хранится в растровом режиме, как показано на рис. *Рисунок 25*.

Рисунок 25. Порядок сканирования пиксельного растра



Примечание: Используются только 32-битные слова и поддерживается только формат с прямым порядком байтов (младший байт хранится в наименьшем адресе).

Данные, полученные от камеры, могут быть организованы в строки, кадры (необработанные режимы YUV/RGB/Bayer) или могут представлять собой последовательность изображений JPEG.

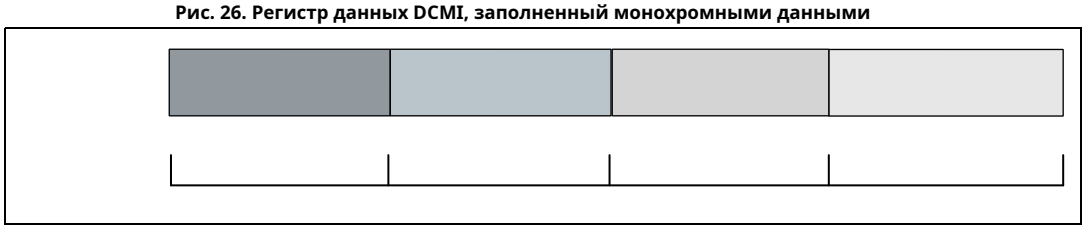
Количество байтов в строке не может быть кратно четырем. Поэтому пользователь должен быть осторожен при обработке этого случая, поскольку запрос DMA генерируется каждый раз, когда из захваченных данных создается полное 32-битное слово. Когда обнаруживается конец кадра, а 32-битное слово, которое должно быть передано, не было полностью получено, оставшиеся данные дополняются нулями и генерируется запрос DMA.

3.6.1 Монохромный

DCMI поддерживает монохромный формат 8 бит на пиксель.

В случае выбора 8-битной ширины данных при настройке DCMI регистр данных имеет структуру, показанную на рис.*Рисунок 26*.

Рис. 26. Регистр данных DCMI, заполненный монохромными данными



3.6.2 RGB565

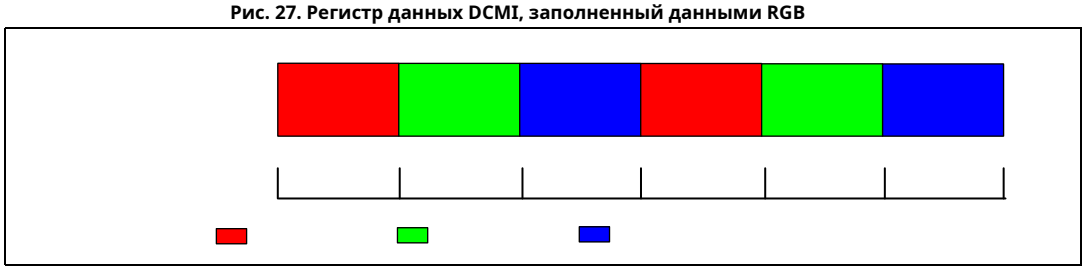
RGB относится к**красный**,**зеленый**а также**синий**, которые представляют три оттенка света. Любой цвет получается при смешивании этих трех цветов.

565 используется для обозначения того, что каждый пиксель состоит из 16 бит, разделенных на:

- **5**биты для кодирования**красный**значение (старшие 5 бит)
- **6**биты для кодирования**зеленый**ценность
- **5**биты для кодирования**синий**значение (менее значащие 5 бит)

Каждый компонент имеет одинаковое пространственное разрешение (формат 4:4:4). Другими словами, каждый образец имеет красный (R), зеленый (G) и синий (B) компоненты.*Рисунок 27*показывает регистр данных DCMI, содержащий данные RGB, когда выбрана разрядность данных 8 бит.

Рис. 27. Регистр данных DCMI, заполненный данными RGB



3.6.3 YCbCr

YCbCr — это семейство цветовых пространств, которое отделяет светимость или яркость (яркость) от цветности или цветности (цветовые различия).

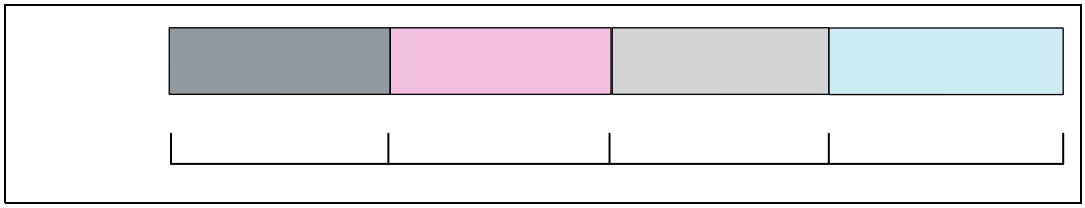
YCbCr состоит из трех компонентов:

- **Д**относится к яркости или**яркость**(черное и белое)
- **Кб**относится к**синий**разница**цветность**
- **Кр**относится к**красный**разница**цветность**.

YCbCr 4:2:2 — это схема подвыборки, требующая половинного разрешения в горизонтальном направлении: на каждые две горизонтальные выборки Y приходится одна выборка Cb или Cr.

Каждый компонент (Y, Cb и Cr) кодируется 8 битами.*Рисунок 28*показывает регистр данных DCMI, содержащий данные YCbCr, когда выбрана разрядность данных 8 бит.

Рис. 28. Регистр данных DCMI, заполненный данными YCbCr



3.6.4 YCbCr, только Y

Примечание: *только для линейки STM32F446, линейки STM32F469/479, устройств STM32L496xx, STM32L4A6xx, STM32F7xxxx и линейки STM32H7x3.*

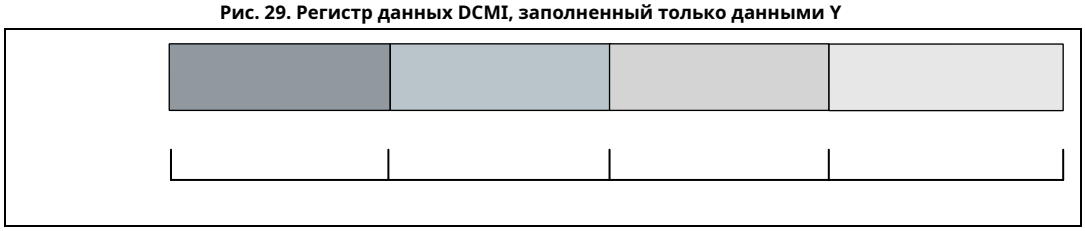
Буфер содержит только информацию Y — монохромное изображение.

В этом режиме информация о цветности отбрасывается. Сохраняется только компонент яркости каждого пикселя, закодированный в 8 битах.

В результате получается монохромное изображение, имеющее половину горизонтального разрешения исходного изображения (данные YCbCr).

*Рисунок 29*показывает регистр DCMI, когда выбрана 8-битная ширина данных.

Рис. 29. Регистр данных DCMI, заполненный только данными Y



3.6.5 JPEG

Для сжатых данных (JPEG) DCMI поддерживает только аппаратную синхронизацию, а размер входных данных не ограничен.

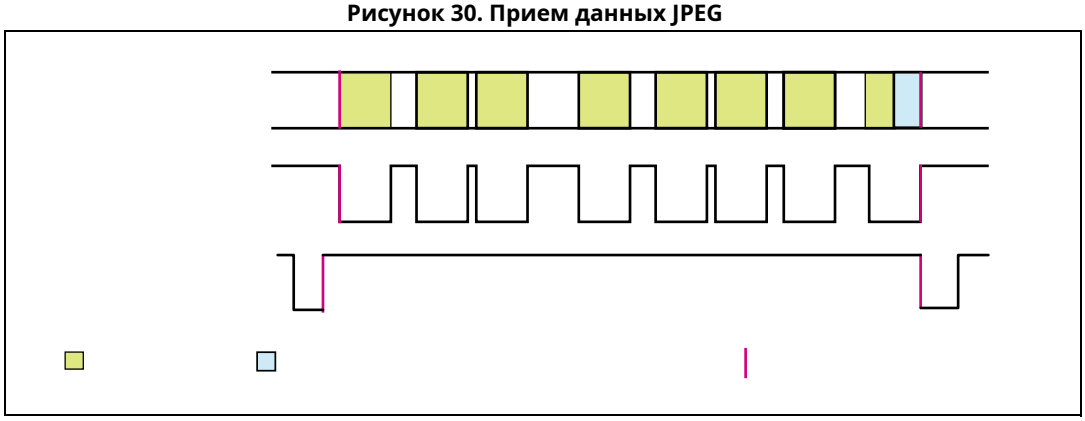
Каждый поток JPEG делится на пакеты, размер которых можно запрограммировать. Отправка пакетов зависит от содержимого изображения и приводит к переменной продолжительности гашения между двумя пакетами.

Чтобы разрешить прием изображений в формате JPEG, необходимо установить бит JPEG в регистре DCMI_CR. Изображения JPEG не сохраняются в виде строк и кадров, поэтому сигнал DCMI_VSYNC используется для запуска захвата, а DCMI_HSYNC служит сигналом включения данных.

Если полный поток данных завершается, а определение конца потока не происходит (DCMI_VSYNC не изменяется), DCMI дополняет конец кадра, вставляя нули. Другими словами, если размер потока не кратен четырем, в конце потока DCMI дополняет оставшиеся данные нулями.

Примечание: *Функция обрезки и встроенный режим синхронизации не могут использоваться в формате JPEG.*

Рисунок 30. Прием данных JPEG



3,7 Другие особенности

3.7.1 Функция обрезки

С функцией обрезки интерфейс камеры выбирает прямоугольное окно из полученного изображения.

Начальные координаты (левый верхний угол) задаются в 32-битном регистре DCMI_CWSTRT.

Размер окна указывается в количестве тактов пикселей (горизонтальное измерение) и в количестве строк (вертикальное измерение) в 32-битном регистре DCMI_CWSIZE.

3.7.2 Изменение размера изображения (изменение разрешения)

Примечание: *Функция изменения размера изображения доступна только в линейке STM32L496xx, STM32L4A6xx, STM32F446, линейке STM32F469/479, линейке STM32F7x5, линейке STM32F7x6, линейке STM32F7x7, линейке STM32F7x8, линейке STM32F7x9 и линейке STM32H7x3.*

Как описано в*Раздел 3.5: Режимы захвата*, функции захвата DCMI задаются через регистр DCMI_CR.

DCMI может захватывать:

- все полученные строки
- одну строку из двух (в этом случае пользователь может выбрать захват четных или нечетных строк).

Эта функция влияет на разрешение по вертикали, которое может быть получено DCMI, как отправленное с модуля камеры или разделенное на два (принимаются только нечетные или четные строки).

Этот интерфейс также позволяет захватывать:

- все полученные данные
- каждый второй байт из полученных данных (один байт из двух. Другими словами, принимаются только нечетные или четные байты)
- один байт из четырех
- два байта из четырех

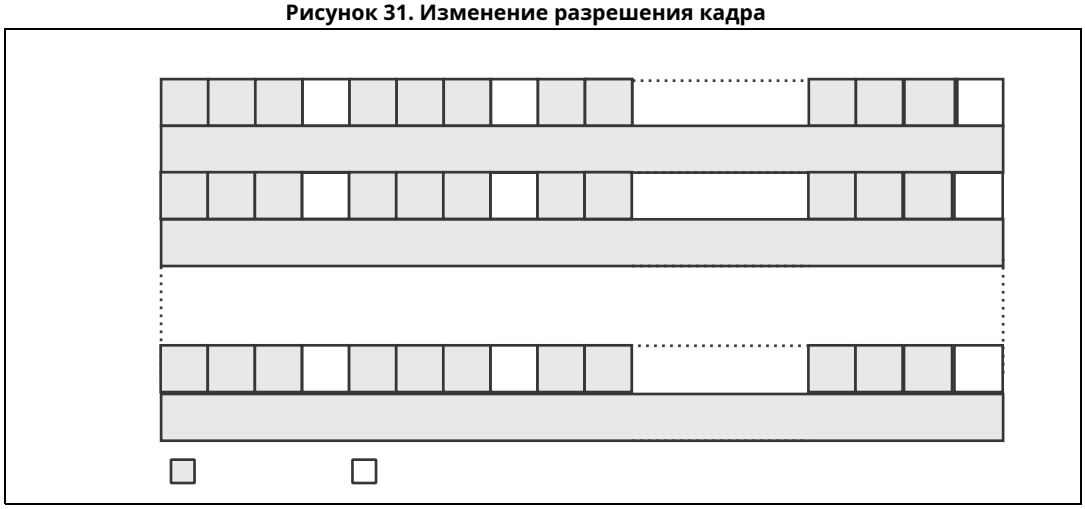
Эта функция влияет на разрешение по горизонтали, позволяя пользователю выбрать одно из следующих разрешений:

- полное горизонтальное разрешение
- половина горизонтального разрешения
- четверть горизонтального разрешения (эта функция доступна только для форматов данных восемь бит на пиксель).

Примечание: *При использовании этой функции требуется осторожность. Для некоторых форматов данных (цветовые пространства) изменение разрешения по горизонтали позволяет изменить формат данных.*
Например, если формат данных YCbCr, данные принимаются с чередованием (CbYCrYCbYCr). Когда пользователь выбирает получение каждого второго байта, DCMI получает только компонент Y каждой выборки, что означает преобразование данных YCbCr в данные только Y. Это преобразование влияет как на разрешение по горизонтали (получается только половина изображения), так и на формат данных.

*Рисунок 31*показывает один кадр при приеме только одного байта из четырех и одной строки из двух.

Рисунок 31. Изменение разрешения кадра



3,8 Прерывания DCMI

Могут быть сгенерированы пять прерываний:

- **IT_LINE**указывает на конец строки.
- **IT_FRAME**указывает на окончание захвата кадра.
- **IT_OVR**указывает на превышение приема данных.
- **IT_VSYNC**указывает кадр синхронизации.
- **IT_ERR**указывает на обнаружение ошибки в порядке встроенных кодов синхронизации (только в режиме встроенной синхронизации).

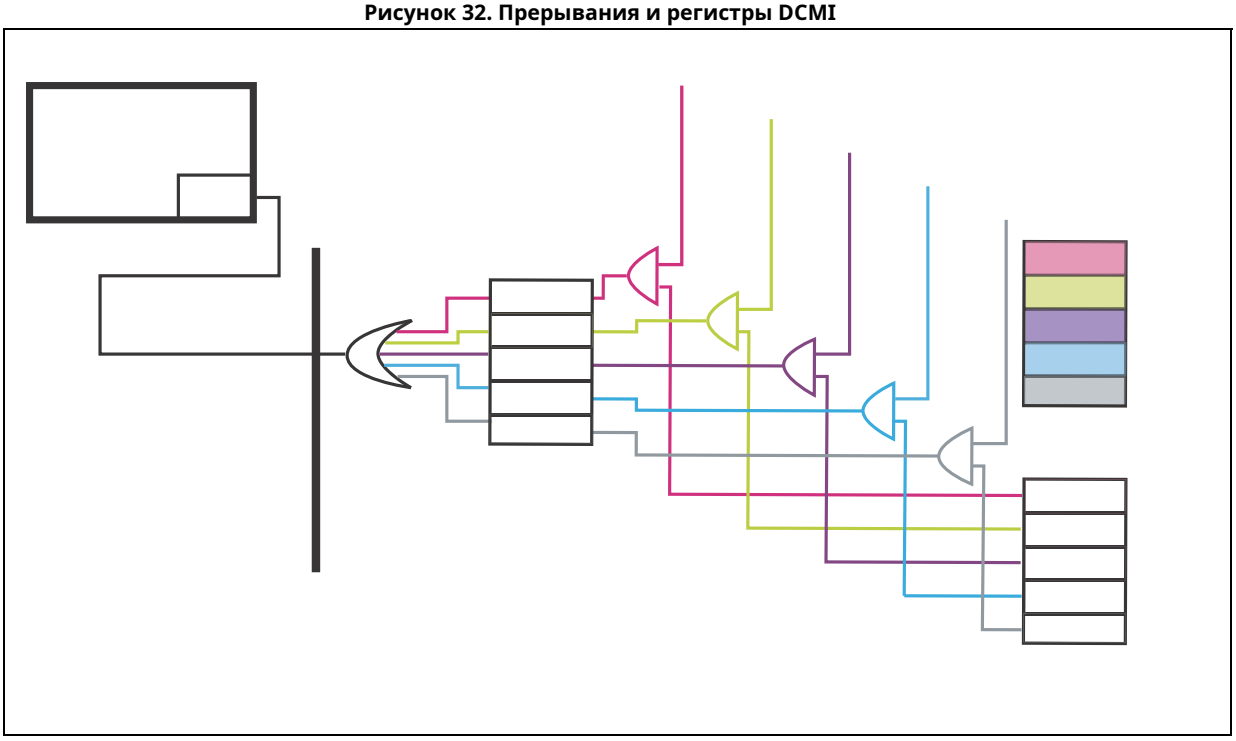
Все прерывания могут быть замаскированы программным обеспечением. Глобальное прерывание**dcmi_it**является логическим ИЛИ всех отдельных прерываний.

Как показано в*Рисунок 32*, прерывания DCMI обрабатываются тремя регистрами:

- **DCMI_IER**: регистр чтения/записи, позволяющий генерировать прерывания при возникновении соответствующего события
- **DCMI_RIS**: регистр только для чтения, указывающий текущий статус соответствующего прерывания перед маскировкой этого прерывания регистром DCMI_IER (каждый бит указывает статус прерывания, который может быть разрешен или запрещен в регистре DCMI_IER).
- **DCMI_MIS**: регистр только для чтения, предоставляющий текущий маскированный статус соответствующего прерывания, в зависимости от регистров DCMI_IER и DCMI_RIS.

Если происходит событие и разрешено соответствующее прерывание, генерируется глобальное прерывание DCMI.

Рисунок 32. Прерывания и регистры DCMI



3,9 Режимы пониженного энергопотребления

Режим питания STM32 напрямую влияет на периферийное устройство DCMI. По этой причине важно знать работу периферийного устройства DCMI в различных режимах питания.

В**Бежать**режиме DCMI и все периферийные устройства работают нормально.

В**Спать**режиме DCMI и все периферийные устройства работают нормально и генерируют прерывания для пробуждения ЦП.

В**Останавливаться**режим и**Стоять рядом с**режиме, DCMI не работает.

Для устройств STM32L496xx и STM32L4A6xx существуют другие режимы с низким энергопотреблением, в которых состояние DCMI меняется от одного к другому:

- **Бег с низким энергопотреблением**Режим
- **Сон с низким энергопотреблением**режим: прерывания от периферийных устройств заставляют устройство выйти из этого режима.
- **Стоп 0, Стоп 1, Стоп 2**режим: содержимое периферийных регистров сохраняется.
- **Неисправность**режим: периферийное устройство должно быть повторно инициализировано при выходе из режима отключения.

*Таблица 5*суммирует работу DCMI в различных режимах.

Табл. 5. Работа DCMI в режимах пониженного энергопотребления

Режим	Операция DCMI
Бежать	Активный
Бег с низким энергопотреблением(1)	
Спать	
Сон с низким энергопотреблением(1)	
Останавливаться	замороженный
Стоп 0(1)	
Остановка 1(1)	
Остановка 2(1)	
Стоять рядом с	Выключен
Неисправность(1)	

1. Только для устройств STM32L496xx и STM32L4A6xx.

4 Конфигурация DCMI

При выборе модуля камеры для взаимодействия с микроконтроллерами STM32 пользователь должен учитывать некоторые параметры, такие как: частота пикселей, поддерживаемый формат данных и разрешения.

Для корректной реализации своего приложения пользователю необходимо выполнить следующие настройки:

- Настройте GPIO.
- Настройте тайминги и часы.
- Настройте периферийное устройство DCMI.
- Настройте прямой доступ к памяти.
- Настройте модуль камеры:
  – настроить I2C, чтобы разрешить настройку и управление модулем камеры
  – установить такие параметры, как контрастность, яркость, цветовой эффект, полярность, формат данных.

Примечание: *Перед началом настройки рекомендуется сбросить периферийное устройство DCMI и модуль камеры. DCMI можно сбросить, установив соответствующий бит в регистре RCC_AHB2RSTR, который сбрасывает домены часов.*

4.1 Конфигурация GPIO

Чтобы легко настроить DCMI GPIO (например, контакты данных, контакты сигналов управления, контакты конфигурации камеры) и избежать конфликтов контактов, рекомендуется использовать STM32CubeMX, генератор кода конфигурации и инициализации.

Благодаря STM32CubeMX пользователь создает проект со всеми предварительно сконфигурированными необходимыми периферийными устройствами.

В зависимости от режима расширенных данных, выбранного путем настройки битов EDM в регистре DCMI_CR, DCMI получает 8, 10, 12 или 14 бит на пиксельный такт (DCMI_PIXCLK). Пользователю необходимо настроить 11, 13, 15 или 17 GPIO для DCMI в случае аппаратной синхронизации.

В случае встроенной синхронизации необходимо настроить только девять GPIO (восемь контактов для данных и один контакт для DCMI_PIXCLK).

Пользователю также необходимо настроить I2C и, в некоторых случаях, контакт питания камеры (если источником питания камеры является микроконтроллер STM32).

Прерывания, позволяющие

Чтобы иметь возможность использовать прерывания DCMI, пользователь должен включить глобальные прерывания DCMI на стороне NVIC. Затем каждое прерывание разрешается отдельно путем установки соответствующего бита разрешения в регистре DCMI_IER.

В режиме аппаратной синхронизации можно использовать только четыре прерывания (IT_LINE, IT_FRAME, IT_OVR и IT_DCMI_VSYNC), но в режиме встроенной синхронизации можно использовать все пять прерываний.

Программное обеспечение позволяет пользователю проверить, произошло ли указанное прерывание DCMI, путем проверки состояния флагов.

4.2 Настройка часов и таймингов

В этом разделе описываются этапы настройки таймингов и часов.

4.2.1 Конфигурация системных часов (HCLK)

Для достижения наилучшей производительности рекомендуется использовать самые высокие системные часы.

Эта рекомендация относится и к кадровому буферу внешней памяти.

Если для кадрового буфера используется внешняя память, часы должны быть установлены на максимально допустимую скорость, чтобы получить наилучшую пропускную способность памяти.

Примеры :

- Устройства STM32F4x9xx: максимальная системная частота 180 МГц. Если к FMC подключена внешняя SDRAM, максимальная тактовая частота SDRAM составляет 90 МГц (HCLK/2).
- Серия STM32F7: максимальная системная частота 216 МГц. С этой скоростью и предделителем HCLK/2 скорость SDRAM превышает максимально допустимую скорость (дополнительную информацию см. в техническом описании продукта). Чтобы получить максимальную SDRAM, рекомендуется настроить HCLK @ 200 МГц, тогда скорость SDRAM устанавливается на 100 МГц.

Конфигурации часов, обеспечивающие наивысшую производительность, следующие:

- для линии STM32F2x7, HCLK @ 120 МГц и SRAM @ 60 МГц
- для линии STM32F407/417, HCLK @ 168 МГц и SRAM @ 60 МГц
- для линии STM32L4x6, HCLK @ 80 МГц и SRAM @ 40 МГц

4.2.2 Конфигурация часов и таймингов DCMI (DCMI_PIXCLK)

Конфигурация пиксельных часов DCMI зависит от конфигурации пиксельных часов модуля камеры. Пользователь должен убедиться, что тактовая частота пикселей имеет одинаковую конфигурацию на стороне DCMI и модуля камеры.

DCMI_PIXCLK — это входной сигнал для DCMI, используемый для выборки входных данных. Пользователь выбирает нарастающий или спадающий фронт для захвата данных путем настройки бита PCKPOL в регистре DCMI_CR.

Как поясняется в *Раздел 3.4: Синхронизация данных*, существует два типа синхронизации: встроенная и аппаратная. Чтобы выбрать желаемый режим синхронизации для своего приложения, пользователю необходимо настроить бит ESS в регистре DCMI_CR.

Аппаратная (внешняя) синхронизация

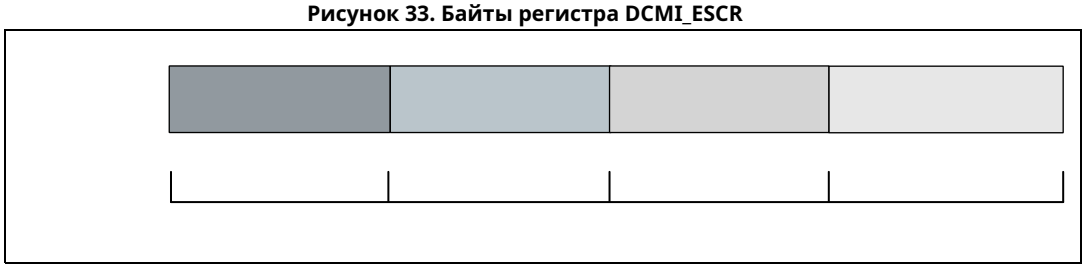
Используются сигналы DCMI_HSYNC и DCMI_VSYNC. Конфигурация этих двух сигналов определяется выбором активного уровня каждого сигнала (высокий или низкий) в битах VSPOL и HSPOL в регистре DCMI_CR.

Примечание: *Пользователь должен убедиться, что полярность DCMI_HSYNC и DCMI_VSYNC запрограммирована в соответствии с конфигурацией модуля камеры.*
В режиме аппаратной синхронизации (бит ESS регистра DCMI_CR сброшен в ноль) генерируется прерывание IT_VSYNC (если разрешено), даже если бит CAPTURE регистра DCMI_CR сброшен в ноль. Чтобы еще больше снизить скорость захвата кадров, можно использовать прерывание IT_VSYNC для подсчета количества кадров между двумя захватами в сочетании с режимом моментальных снимков. Это не разрешено встроенным режимом синхронизации.

Встроенная (внутренняя) синхронизация

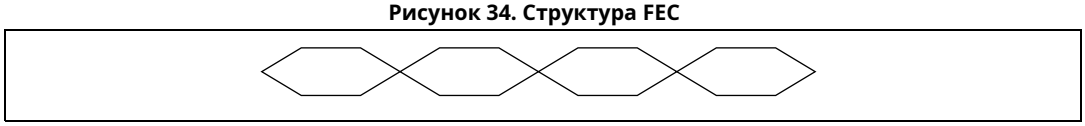
Начало или конец строки и начало или конец кадра определяются кодами или маркерами, встроенными в поток данных. Встроенные коды синхронизации поддерживаются только для 8-битной ширины интерфейса параллельных данных. Коды синхронизации должны быть запрограммированы в регистре DCMI_ESCR, как определено в *Рисунок 33*.

Рисунок 33. Байты регистра DCMI_ESCR



- **FEC (код конца кадра)**: старший байт определяет разделитель конца кадра. Модуль камеры отправляет 32-битное слово, содержащее 0xFF 00 00 XY с кодом XY = FEC, чтобы сигнализировать о конце кадра. Код получен, как указано в *Рисунок 34*.

Рисунок 34. Структура FEC

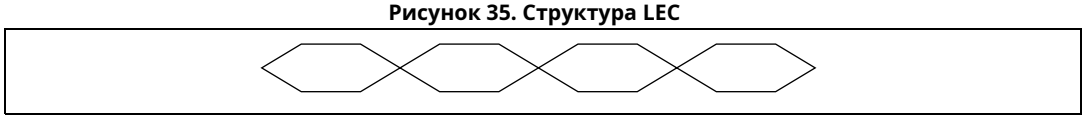


Перед приемом этого кода FEC значение бита VSYNC в регистре DCMI_SR должно быть установлено в 1, чтобы указать действительный кадр. После приема FEC значение бита VSYNC должно быть равно 0, чтобы указать, что это синхронизация между кадрами. Значение этого бита VSYNC должно оставаться равным 0 до приема следующего кода начала кадра.

Если значение FEC равно 0xFF (модуль камеры отправляет 0xFF 00 00 FF), все неиспользуемые коды интерпретируются как коды конца кадра. Есть 253 значения, соответствующие разделителю конца кадра (0xFF0000FF и 252 неиспользуемых кода).

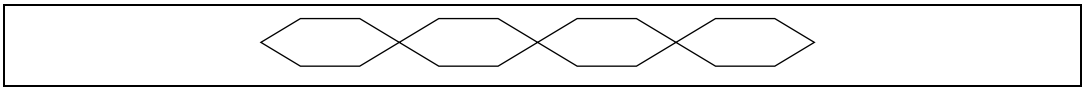
- **LEC (код конца строки)**: этот байт определяет маркер конца строки. Код, полученный от камеры для указания конца строки: 0xFF 00 00 XY, где XY = код LEC.

Рисунок 35. Структура LEC



- **FSC (код начала кадра)**: этот байт определяет маркер начала кадра. Код, полученный от камеры для указания начала нового кадра, равен 0xFF 00 00 XY, где XY = код FSC.

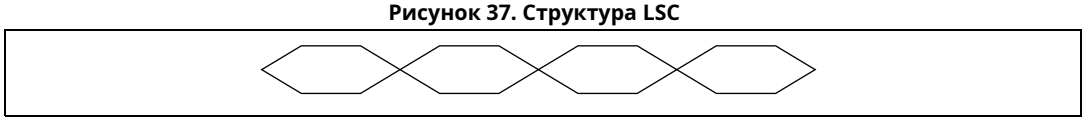
Рисунок 36. Структура FSC



- **LSC (код начала строки)**: этот байт определяет маркер начала строки. Код, полученный от камеры для указания начала новой строки: 0xFF 00 00 XY, где XY = код LSC.

  Если LSC запрограммирован на 0xFF, модуль камеры не отправляет разделитель начала кадра. DCMI интерпретирует первое появление кода LSC после кода FEC как появление кода FSC.

Рисунок 37. Структура LSC

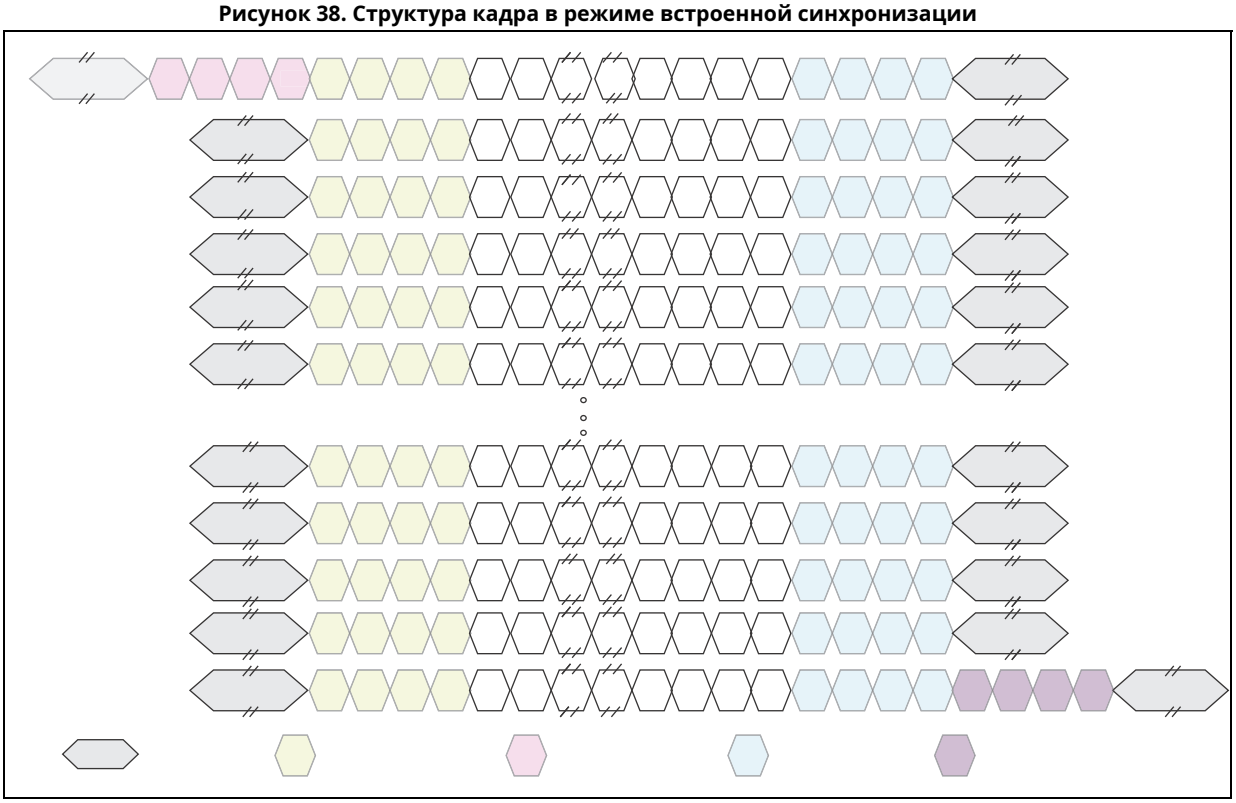


В этом встроенном режиме синхронизации биты HSPOL и VSPOL игнорируются. Пока DCMI получает данные (бит CAPTURE установлен в регистре DCMI_CR), пользователь может контролировать поток данных, чтобы узнать, является ли это активной строкой/кадром или синхронизацией между строками/кадрами, считывая биты VSYNC и HSYNC в регистре DCMI_CR. Регистр DCMI_SR.

Если бит ERR_IE в регистре DCMI_IER включен, прерывание генерируется каждый раз, когда возникает ошибка (например, встроенные символы синхронизации не принимаются в правильном порядке).

*Рисунок 38*показывает кадр, полученный в режиме встроенной синхронизации.

Рисунок 38. Структура кадра в режиме встроенной синхронизации



4.3 Конфигурация DCMI

Конфигурация DCMI позволяет пользователю выбирать режим захвата, формат данных, размер и разрешение изображения.

4.3.1 Режим захвата

Пользователь может захватить изображение или видео, выбрав:

- режим непрерывного захвата, позволяющий непрерывно захватывать кадры (изображения)
- режим моментального снимка, позволяющий захватить один кадр.

Полученные данные в режиме моментального снимка или непрерывного захвата передаются в кадровый буфер памяти с помощью DMA. Расположение и режим буфера (линейный или циклический буфер) контролируются через системный DMA.

4.3.2 Формат данных

Как упоминалось ранее, DCMI позволяет принимать сжатые данные (JPEG) или многие несжатые форматы данных (например, монохромный, RGB, YCbCr).

Для получения более подробной информации см.*Раздел 3.6: Форматы и хранение данных*.

4.3.3 Разрешение и размер изображения

DCMI позволяет принимать широкий диапазон разрешений (низкое, среднее, высокое) и размеров изображения, поскольку размер изображения зависит от разрешения изображения и формата данных. Прямой доступ к памяти должен обеспечить передачу и размещение полученных изображений в кадровом буфере памяти.

При желании пользователь может настроить режим выбора байтов, строк и кадров, чтобы изменить разрешение и размер изображения, а в некоторых случаях и формат данных. Пользователь также может настроить и включить функцию обрезки, чтобы выбрать прямоугольное окно из полученного изображения.

Дополнительные сведения об этих двух функциях см.*Раздел 3.7: Другие функции*.

Примечание: *Регистры конфигурации DCMI должны быть правильно запрограммированы перед включением бита ENABLE в регистре DCMI_CR.*
Контроллер прямого доступа к памяти и все регистры конфигурации DCMI должны быть правильно запрограммированы перед включением бита CAPTURE в регистре DCMI_CR.

4.4 Конфигурация прямого доступа к памяти

Конфигурация прямого доступа к памяти — важный шаг, гарантирующий успех приложения.

Как упоминалось в*Раздел 2.3: DCMI в интеллектуальной архитектуре*, DMA2 обеспечивает передачу из DCMI в память (внутреннюю SRAM или внешнюю SRAM/SDRAM) для всех устройств STM32 со встроенным DCMI, за исключением устройств STM32H7x3xx, где DMA1 также может получить доступ к периферийным устройствам AHB2 и обеспечить передачу полученных данных из DCMI в кадровый буфер памяти.

4.4.1 Общая конфигурация DMA для передачи DCMI в память

В случае передачи DCMI в память:

- Направление передачи должно быть от периферии к памяти, путем настройки битов DIR[1:0] в регистре DMA_SxCR. В таком случае:
  - Адрес источника (адрес регистра данных DCMI) должен быть записан в регистр DMA_SxPAR.
  - Адрес назначения (адрес кадрового буфера во внутренней SRAM или внешней SRAM/SDRAM) должен быть записан в регистр DMA_SxMAR.
- Чтобы обеспечить передачу данных из регистра данных DCMI, DMA ожидает генерации запроса от DCMI. Поэтому необходимо настроить соответствующий поток и канал. Для получения более подробной информации см.*Раздел 4.4.3: Конфигурация каналов и потоков DCMI*.
- Поскольку запрос DMA генерируется каждый раз при заполнении регистра данных DCMI, данные, передаваемые из DCMI в DMA2 (или DMA1 для устройств STM32H7x3xx), должны иметь 32-битную ширину. Таким образом, ширина периферийных данных, запрограммированная в битах PSIZE в регистре DMA_SxCR должны быть 32-битные слова.
- DMA является контроллером потока: количество 32-битных слов данных, которые должны быть переданы, программируется программно от 1 до 65535 в регистре DMA_SxNDTR (называемый DMA_CNDTRx в строках STM32L4x6). Дополнительные сведения об этом регистре см. *Раздел 4.4.4: Регистр DMA_SxNDTR*.

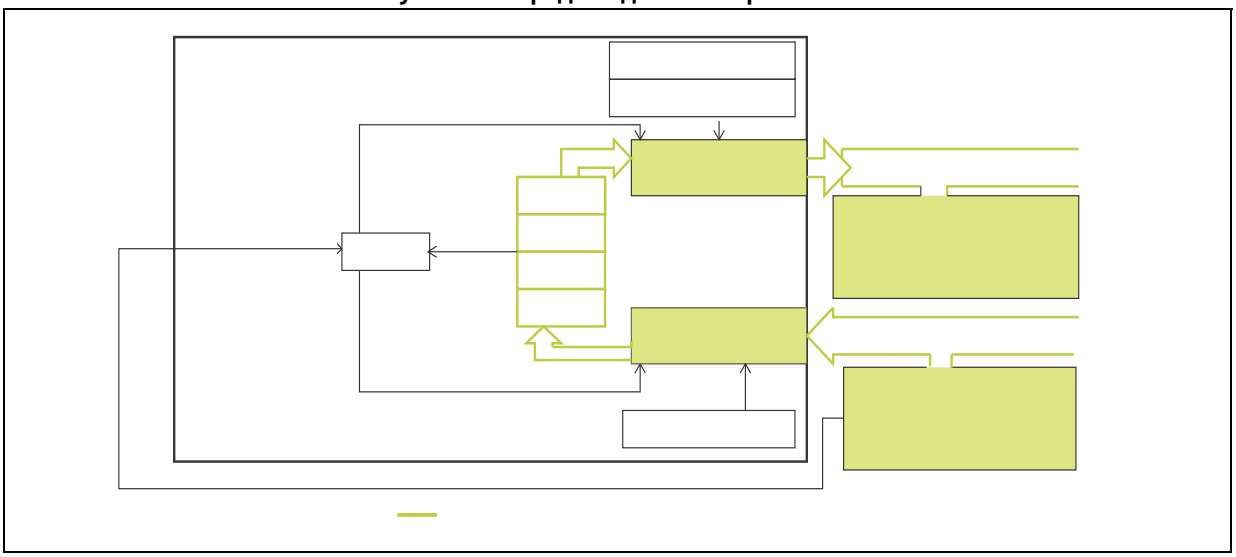
DMA может работать в двух режимах:

- прямой режим: каждое слово, полученное от DCMI, передается в кадровый буфер памяти.
- Режим FIFO: DMA использует свой внутренний FIFO для обеспечения пакетной передачи (более одного слова из DMA FIFO в место назначения памяти)

Дополнительные сведения о внутреннем FIFO DMA см.*Раздел 4.4.5: FIFO и конфигурация пакетной передачи*.

*Рисунок 39*показывает работу DMA2 (или DMA1 для устройств STM32H7x3xx) в режиме периферийных устройств в память (за исключением устройств STM32L496xx и STM32L4A6xx, поскольку DMA2 в этих устройствах имеет только один порт).

Рисунок 39. Передача данных через DMA



1. Регистр DMA_SxM1AR конфигурируется в случае режима двойной буферизации.

4.4.2 Настройка DMA в зависимости от размера изображения и режима захвата

DMA необходимо настроить в соответствии с размером изображения (глубина цвета и разрешение) и режимом захвата:

- В**снимок**режим: DMA должен обеспечить передачу одного кадра (изображения) из DCMI в нужную память:
  - Если размер изображения в словах не превышает 65535, поток можно настроить в обычном режиме. Более подробное описание этого режима см. *Раздел 4.4.6: Нормальный режим для низкого разрешения при захвате моментальных снимков*.
  - Если размер изображения в словах находится в диапазоне от 65 535 до 131 070, поток можно настроить в режиме двойной буферизации. Более подробное описание этого режима см. *Раздел 4.4.8: Режим двойной буферизации для средних разрешений (моментальный снимок или непрерывный захват)*.
  - Если размер изображения в словах превышает 131070, поток нельзя настроить в режиме двойной буферизации. Более подробное описание режима, который необходимо использовать, см. *Раздел 4.4.9: Конфигурация DMA для более высоких разрешений*.
- в**непрерывный**режим: DMA должен обеспечить передачу последовательных кадров (изображений) из DCMI в требуемую память. Каждый раз, когда DMA заканчивает передачу одного кадра, он начинает передачу следующего кадра:
  - Если размер одного изображения в словах не превышает 65535, поток можно настроить в циклическом режиме. Более подробное описание этого режима см. *Раздел 4.4.7: Круговой режим для низкого разрешения при непрерывной съемке*.
  - Если размер одного изображения в словах находится между 65535 и 131070, поток можно настроить в режиме двойной буферизации. Более подробное описание этого режима см. *Раздел 4.4.8: Режим двойной буферизации для средних разрешений (моментальный снимок или непрерывный захват)*.
  - Если размер одного изображения в словах превышает 131070, поток нельзя настроить в режиме двойной буферизации. Более подробное описание режима, который необходимо использовать, см. *Раздел 4.4.9: Конфигурация DMA для более высоких разрешений*.

4.4.3 Конфигурация каналов и потоков DCMI

Пользователь также должен настроить соответствующий поток и канал DMA2 (или DMA1 для устройств STM32H7x3xx), чтобы гарантировать подтверждение DMA каждый раз при заполнении регистра данных DCMI.

*Таблица 6*суммирует каналы DMA, разрешающие запрос DMA от DCMI.

Таблица 6. Выбор потока DMA для устройств STM32

Серия СТМ32	DMA-поток	Канал
СТМ32Ф2	Поток 1	Канал 1 или канал 7
СТМ32Ф4		
СТМ32Ф7		
СТМ32Л4	Поток 0	Канал 6
	Поток 4	Канал 5
СТМ32Х7	Поток 0 Поток 1 Поток 2 Поток 3 Поток 4 Поток 5 Поток 6 Поток 7	Мультиплексор1 запрос 74

Примечание: Пошаговое описание процедуры настройки потока см. в соответствующем справочном руководстве по STM32.

4.4.4 Регистр DMA_SxNDTR

*Примечание: Этот регистр называется**DMA_CNDTRx**в устройствах STM32L496xx и STM32L4A6xx.*

Общее количество слов для передачи из периферийного источника (DCMI) в место назначения памяти программируется в этом регистре пользователем.

Когда DMA начинает передачу из DCMI в память, количество элементов уменьшается от начального запрограммированного значения до конца передачи (достижение нуля или отключение потока программно до того, как количество оставшихся данных достигнет нуля).

*Таблица 7*восстанавливает количество байтов, соответствующее запрограммированному значению и ширине периферийных данных (биты PSIZE):

Таблица 7. Максимальное количество байтов, передаваемых за одну передачу DMA

DMA_SxNDTR запрограммирован ценность	Периферийный размер	Количество байтов
65535	Слова	262140
H(1)	Слова	4 * H

1. 0 < N < 65535.

Примечание: *Чтобы избежать повреждения данных, значение, запрограммированное в DMA_SxNDTR, должно быть кратно значению MSIZE / значению PSIZE.*

4.4.5 Конфигурация FIFO и пакетной передачи

DMA выполняет передачу с включением или без включения FIFO из 4 слов. Как упоминалось ранее, когда FIFO включен, ширина данных источника (запрограммированная в битах PSIZE) может отличаться от ширины данных назначения (запрограммирована в битах MSIZE). В этом случае пользователь должен обратить внимание на то, чтобы адаптировать адрес для записи в DMA_SxPAR и DMA_SxM0AR (и DMA_SxM1AR в случае конфигурации режима двойного буфера) к ширине данных, запрограммированной в битах PSIZE и MSIZE в регистре DMA_SxCR. Для лучшей производительности рекомендуется использовать FIFO.

Когда режим FIFO включен, пользователь может настроить биты MBURST, чтобы заставить DMA выполнять пакетную передачу (до четырех слов) из своего внутреннего FIFO в память назначения, чтобы гарантировать лучшую производительность.

4.4.6 Нормальный режим для низкого разрешения при захвате моментальных снимков

Изображения с низким разрешением — это изображения, размер которых (в 32-битном слове) меньше 65535.

В режиме моментальных снимков можно использовать обычный режим для обеспечения передачи кадров с низким разрешением (см. *Таблица 7*).

Таблица 8 суммирует максимальные размеры изображений, которые могут быть переданы в обычном режиме.

Таблица 8. Максимальное разрешение изображения в обычном режиме

Вещь	Максимум количество байтов	Разрядность (байты на пиксель)(1)	Максимум количество пикселей	Максимум разрешающая способность
Слово	262140	1	262140	720x364
		2	131070	480x272

1. Максимальное количество пикселей зависит от разрядности изображения (количество байтов на пиксель). DCMI поддерживает две возможные разрядности:
   - 1 байт на пиксель в монохромном или Y-формате
   - 2 байта на пиксель в случае формата RGB565 или YCbCr.

4.4.7 Круговой режим для низкого разрешения при непрерывной съемке

Изображения с низким разрешением — это изображения, размер которых (в 32-битном слове) меньше 65536.

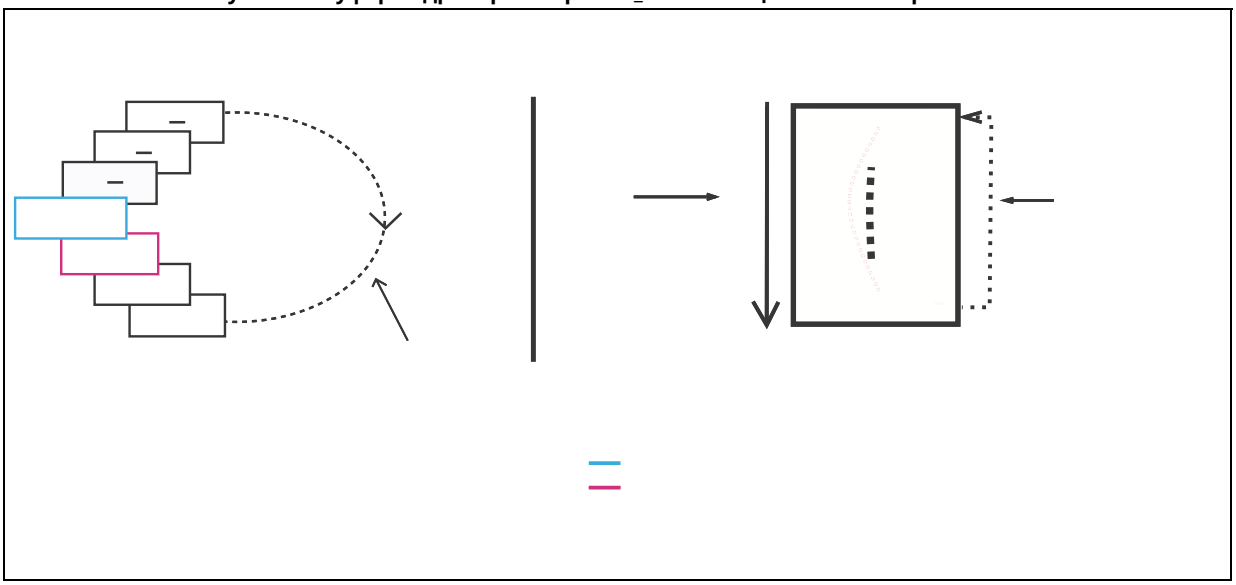
Этот круговой режим позволяет обрабатывать последовательные кадры (непрерывные потоки данных), при условии, что размер одного кадра (начальное запрограммированное значение в регистре DMA_SxNDTR (DMA_CNDTR для серии STM32L4)) меньше 65535.

Каждый раз, когда количество декрементируемых данных достигает нуля, количество слов данных автоматически перезагружается до начального значения. И каждый раз, когда указатель DMA достигает конца буфера кадров, он повторно инициализируется (возвращается к запрограммированному адресу в DMA_SxM0AR), и DMA обеспечивает передачу следующего кадра.

Резолюции, перечисленные в *Таблица 8* справедливы и для низкого разрешения в непрерывном режиме.

Рисунок 40 показывает значение DMA_SxNDTR и изменения указателя кадрового буфера во время передачи DMA и между двумя последовательными передачами DMA.

Рисунок 40. Буфер кадра и регистр DMA_SxNDTR в циклическом режиме



4.4.8 Режим двойной буферизации для средних разрешений (моментальный или непрерывный захват)

Примечание: *Этот режим недоступен в устройствах STM32L4A6xx и STM32L496xx.*

Изображения среднего разрешения имеют размер (в 32-битном слове) от 65536 до 131070.

Когда включен режим двойной буферизации, круговой режим включается автоматически.

Если размер изображения превышает (в словах) максимальные размеры, указанные в *Таблица 8*в моментальном или непрерывном захвате режим двойной буферизации должен использоваться в моментальном или непрерывном режиме. В этом случае допустимое количество пикселей на кадр удваивается, поскольку полученные данные хранятся в двух буферах, максимальный размер каждого из которых (в 32-битных словах) равен 65535 (максимальный размер кадра составляет 131070 слов или 524280 байт). В результате размеры и разрешения изображений, которые могут быть получены DCMI и переданы DMA, удваиваются, как показано на рис. *Таблица 9*.

Табл. 9. Максимальное разрешение изображения в режиме двойной буферизации

Вещь	Максимум количество байтов	Разрядность (байты на пиксель)	запрограммировано стоимость в SxNDTR регистр	Количество пикселей	Максимум разрешающая способность
Слово	524280	1	65535	524280	960x544
		1	H(1)	8 * H	960x544
		2	65535	262140	720x364
		2	H(1)	4 * H	720x364

1. 0 < N < 65536.

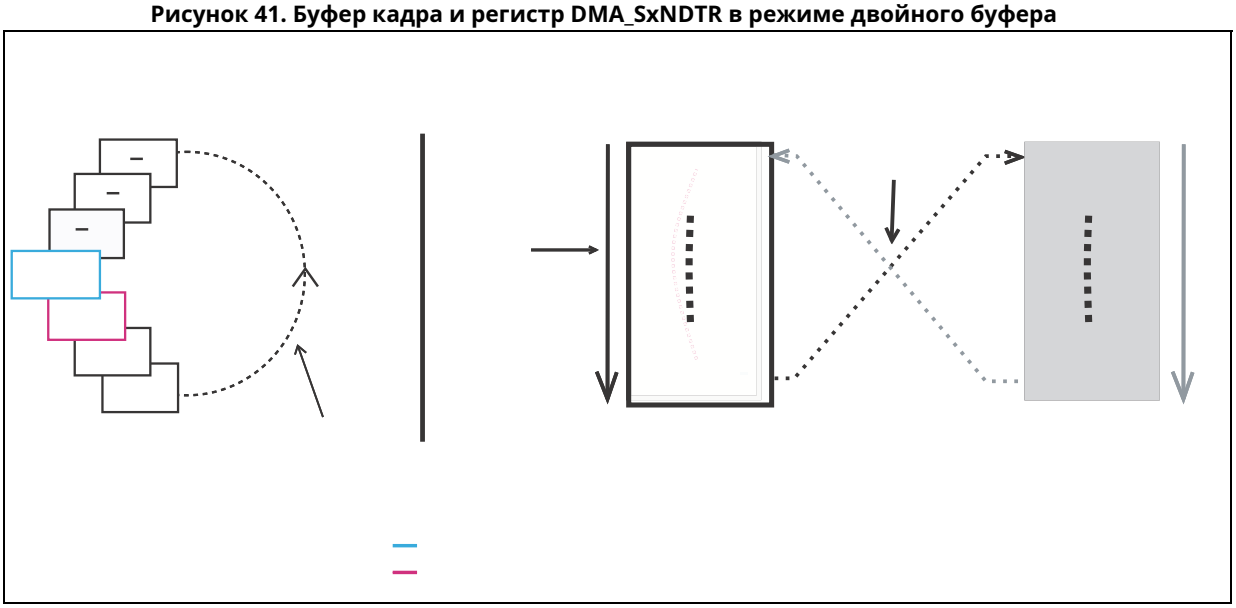
В этом режиме поток с двойным буфером имеет два указателя (два буфера для хранения данных), переключаемых каждым концом транзакции:

- В режиме моментального снимка контроллер DMA записывает данные в первый кадровый буфер. После заполнения этого первого кадрового буфера (на этом уровне регистр SxNDTR повторно инициализируется до запрограммированного значения и указатель DMA переключается на второй кадровый буфер), данные передаются во второй буфер. Фактически общий размер кадра (в словах) делится на два и программируется в регистре SxNDTR, а изображение хранится в двух буферах одинакового размера.
- В непрерывном режиме каждый раз, когда один кадр (изображение) принимается и сохраняется в двух буферах, поскольку включен циклический режим, регистр SxNDTR повторно инициализируется до запрограммированного значения (общий размер кадра, разделенный на два), а указатель DMA переключается на буфер первого кадра для приема следующего кадра.

Режим двойного буфера включается установкой бита DBM в регистре DMA_SxCR.

*Рисунок 41*показывает два указателя и изменения значения DMA_SxNDTR во время передачи DMA.

Рисунок 41. Буфер кадра и регистр DMA_SxNDTR в режиме двойного буфера



4.4.9 Конфигурация DMA для более высоких разрешений

При количестве слов в одном кадре (изображении) в режиме моментального или непрерывного снимка более 131070, а также при разрешении изображения, превышающем указанные в*Таблица 9*, режим двойного буфера DMA не может обеспечить передачу полученных данных.

Примечание: *В этом разделе освещается только операция прямого доступа к памяти в случае высокого разрешения. Пример разработан и описан с использованием этой конфигурации прямого доступа к памяти в*Раздел 6.3.6: Запись с разрешением SxGA (формат данных YCbCr)*.*

Серии STM32F2, STM32F4, STM32F7 и STM32H7 включают очень важную функцию в **режим двойной буферизации**:**возможность обновить запрограммированный адрес порта памяти AHB**на лету (в DMA_SxM0AR или DMA_SxM1AR)**когда поток включен**. Необходимо соблюдать следующие условия:

- Когда**бит КТ**установлен на**нуль**в регистре DMA_SxCR (текущая целевая память — память 0), DMA_Sx**M1**Регистр AR может быть записан.

Попытка записи в этот регистр, когда CT установлен в единицу, генерирует флаг ошибки (TEIF), и поток автоматически отключается.

- Когда**бит КТ**установлен на**один**в регистре DMA_SxCR (текущая целевая память — память 1), DMA_Sx**M0**Регистр AR может быть записан.
  Попытка записи в этот регистр при нулевом значении CT генерирует флаг ошибки (TEIF) и поток автоматически отключается.

Во избежание любого состояния ошибки рекомендуется изменить запрограммированный адрес, как только будет установлен флаг TCIF. В этот момент целевая память должна измениться с памяти 0 на память 1 (или с 1 на 0), в зависимости от значения бита CT в регистре DMA_SxCR.

Примечание: *Для всех остальных режимов, кроме режима с двойной буферизацией, регистры адреса памяти защищены от записи, как только поток активируется.*

DMA позволяет управлять более чем двумя буферами:

- В первом цикле, пока DMA использует**буфер 0**обратился**указатель 0**(адрес памяти 0 в регистре DMA_SxM0AR),**буфер 1**обращается к**указатель 1** (адрес памяти 1 в регистре DMA_SxM1AR).
- Во втором цикле, пока DMA использует**буфер 1**обратился**указатель 1**, адрес буфера 0 можно изменить и кадр**буфер 2**может быть адресовано**указатель 0**.
- Во втором цикле, пока DMA использует**буфер 2**обратился**указатель 0**, адрес кадрового буфера 1 можно изменить и**буфер 3**может быть адресовано **указатель 1**.

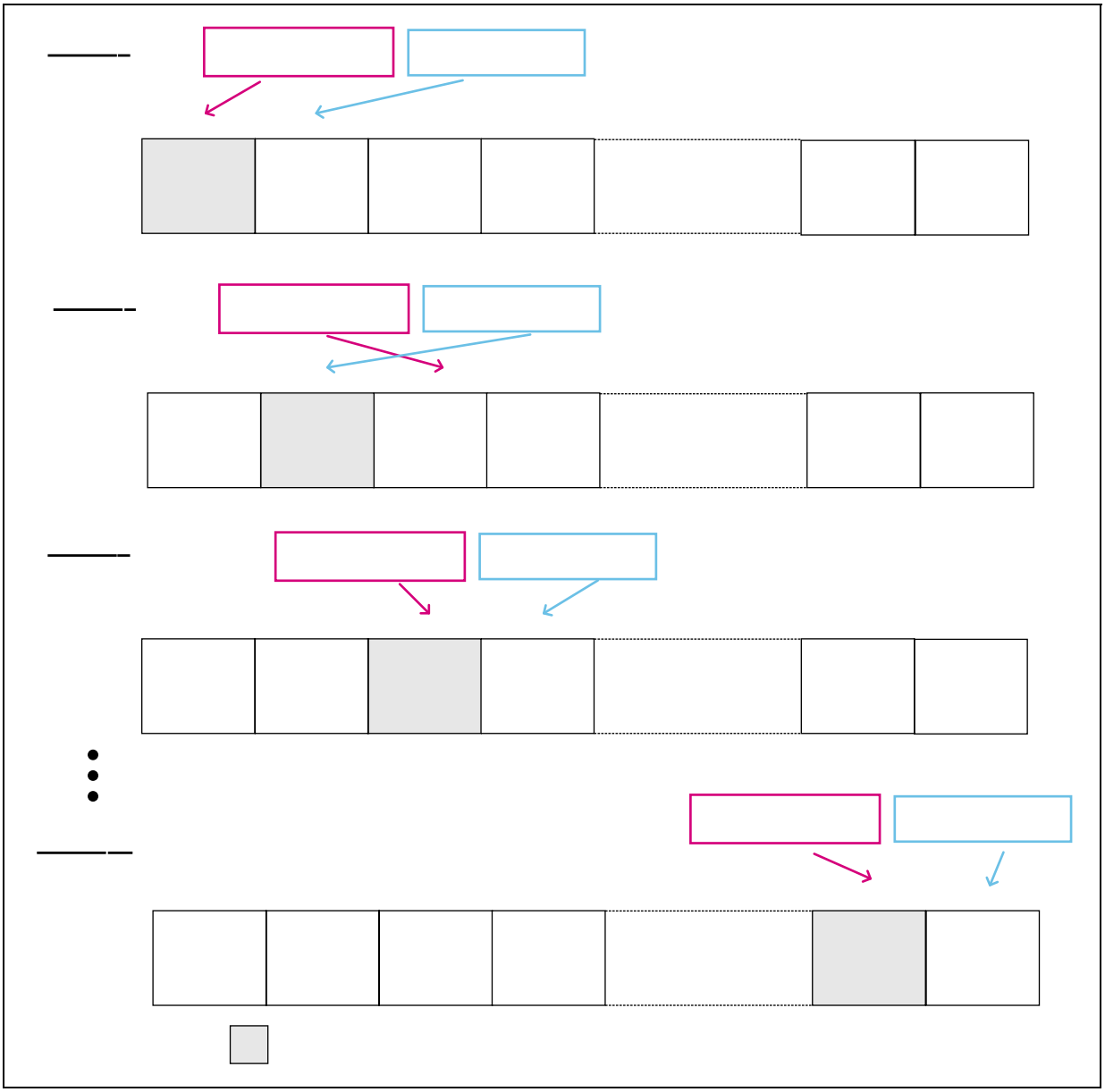
Затем DMA позволяет использовать свои два регистра DMA_SxM0AR и DMA_SxM1AR для адресации многих буферов, обеспечивая передачу изображений с высоким разрешением.

Примечание: *Для упрощения использования этой специфической функции рекомендуется разделить изображение на равные буферы.*
При захвате изображений с высоким разрешением пользователь должен убедиться, что место назначения памяти имеет достаточный размер.

Пример: В случае разрешения SxGA (1280x1024) размер изображения составляет 655360 слов (32 бита). Этот размер необходимо разделить на равные буферы с максимальным размером 65535 для каждого из них. Для корректного приема изображение должно быть разделено на 16 кадровых буферов, размер каждого кадрового буфера равен 40960 (меньше 65535).

*Рисунок 42*иллюстрирует обновление регистров DMA_SxM0AR и DMA_SxM1AR во время передачи DMA:

Рисунок 42. Работа прямого доступа к памяти в случае высокого разрешения



4,5 Конфигурация модуля камеры

Чтобы правильно настроить модуль камеры, пользователю необходимо обратиться к его техническому описанию.

Следующие шаги позволяют правильно настроить модуль камеры:

- Настройте функции ввода / вывода для контактов конфигурации камеры, чтобы иметь возможность изменять ее регистры (последовательная связь, в основном I2C).
- Примените аппаратный сброс на модуле камеры.
- Инициализируйте модуль камеры,

  – настройка разрешения изображения

  – настройка контрастности и яркости

  – настройка баланса белого камеры (например, черно-белый, белый негатив, белый нормальный)

  – выбор интерфейса камеры (некоторые модули камеры имеют последовательный и параллельный интерфейс)

  – выбор режима синхронизации, если модуль камеры поддерживает более одного

  – настроить частоты тактовых сигналов

  – выберите формат выходных данных.

5 Соображения по мощности и производительности

5.1 Потребляемая мощность

Чтобы сэкономить больше энергии, когда приложение находится в режиме пониженного энергопотребления, рекомендуется перевести модуль камеры в режим пониженного энергопотребления перед входом в STM32 в режиме пониженного энергопотребления.

Перевод модуля камеры в режим пониженного энергопотребления обеспечивает значительный выигрыш в энергопотреблении.

Пример для датчика CMOS OV9655:

- В активном режиме рабочий ток**20 мА**.
- В режиме ожидания текущие требования падают до**1 мА**в случае ожидания, инициированного I2C (активность внутренней цепи приостанавливается, но часы не останавливаются) и для **10 мкА**в случае инициированного пином режима ожидания (внутренние часы устройства останавливаются и все внутренние счетчики сбрасываются). Для получения более подробной информации обратитесь к соответствующему техническому описанию камеры.

5.2 Соображения производительности

Для всех микроконтроллеров STM32 количество байтов, передаваемых за каждый пиксельный такт, зависит от режима расширенных данных:

- когда DCMI настроен на получение**8-битный**данные, интерфейс камеры принимает**четыре** тактовых циклов пикселей для захвата 32-битного слова данных.
- когда DCMI настроен на получение**10-, 12- или 14-битный**данные, интерфейс камеры принимает**два** тактовых циклов пикселей для захвата 32-битного слова данных.

*Таблица 10*суммирует максимальный поток данных в зависимости от конфигурации ширины данных.

Таблица 10. Максимальный поток данных при максимальном значении DCMI_PIXCLK(1)

Серия CTM32		Расширенный режим данных			
		8-битный	10-битный	12-битный	14-битный
Байтов на PICXCLK		**1**	**1,25**	**1,5**	**1,75**
Поток данных (макс. Мбайт/с)	**CTM32Ф2**	46.875	58.594	70.312	82.031
	CTM32Ф4	52.734	65,918	79.101	92,285
	CTM32Ф7	52.734	65,918	79.101	92,285
	CTM32X7	78,125	97,656	117,187	136.718
	CTM32Л4	31.25	39.062	46.875	54,687

1. Эти значения рассчитаны для максимального DCMI_PIXCLK, описанного в*Раздел Таблица 2.: DCMI и доступность связанных ресурсов*.

- В некоторых приложениях DMA2 (или DMA1 для устройств STM32H7x3) настроен на параллельное обслуживание других запросов вместе с запросом DCMI. В этом случае пользователь

следует обратить внимание на настройки приоритетов потоков и учитывать влияние на производительность, когда DMA обслуживает другие потоки параллельно с DCMI.

- Для повышения производительности при использовании DCMI параллельно с другими периферийными устройствами, имеющими запросы, которые могут быть подключены либо к DMA1, либо к DMA2, лучше настроить эти потоки для обслуживания DMA, который не обслуживает DCMI.
- Пользователь должен убедиться, что частота пикселей, настроенная на стороне модуля камеры, поддерживается STM32 DCMI, чтобы избежать переполнения.
- Рекомендуется использовать самую высокую скорость системы HCLK для лучшей производительности, но пользователь должен учитывать скорость всех используемых периферийных устройств (например, скорость внешней памяти), чтобы избежать переполнения и гарантировать успех своего приложения.
- DCMI — не единственное периферийное устройство AHB2, но существует множество других периферийных устройств, и DMA — не единственный мастер, который может получить доступ к периферийным устройствам AHB2. Использование большого количества периферийных устройств AHB2 или другого мастера, обращающегося к периферийным устройствам AHB2, приводит к параллелизму на AHB2, и пользователь должен учитывать его влияние на производительность.

6 Примеры приложений DCMI

В этом разделе представлена информация, связанная с использованием DCMI, и представлены пошаговые примеры реализации.

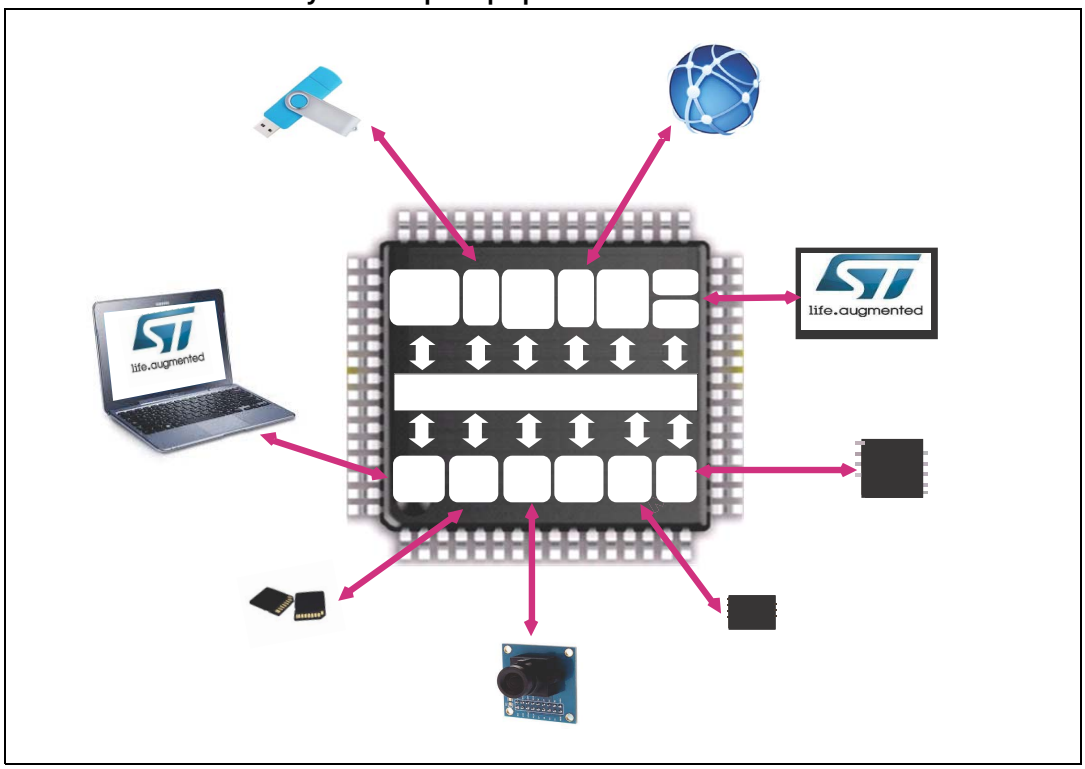
6.1 Примеры использования DCMI

Существует несколько приложений обработки изображений, которые можно реализовать с помощью DCMI и других периферийных устройств STM32. Вот несколько примеров приложений:

- машинное зрение
- игрушки
- биометрия
- охрана и видеонаблюдение
- домофон и домашняя автоматизация
- промышленные системы мониторинга и автоматизированный контроль
- система управления
- системы контроля доступа
- сканирование штрих-кода
- видео-конференция
- дроны
- потоковое видео в реальном времени и видеокамера с питанием от батареи.

*Рисунок 43*предоставляет примеры приложений с использованием микроконтроллера STM32, который позволяет пользователю собирать данные, сохранять их во внутренней или внешней памяти, отображать их, делиться ими через Интернет и общаться с людьми.

Рисунок 43. Пример приложения STM32 DCMI



6.2 Примеры прошивки STM32Cube

Пакеты прошивки STM32CubeF2, STM32CubeF4, STM32CubeF7 и STM32CubeL4 предлагают большой набор примеров, реализованных и протестированных на соответствующих платах. *Таблица 11* предлагает обзор примеров и приложений DCMI в различных прошивках STM32Cube.

Таблица 11. Примеры STM32Cube DCMI

Пакет прошивки	Название проекта(1)	Доска
STM32CubeF2	DCMI_CaptureMode	STM3220G-EVAL STM3221G-EVAL
	Режим моментального снимка	
	Camera_To_USBDisk	
STM32CubeF4	DCMI_CaptureMode	STM32446E-EVAL STM324x9I-EVAL STM324xG-EVAL STM32446E-EVAL
	Режим моментального снимка	
	Camera_To_USBDisk	
STM32CubeF7	DCMI_CaptureMode	STM32756G-EVAL СТМ32Ф769И-ЭВАЛ
	Режим моментального снимка	
	Camera_To_USBDisk	
STM32CubeL4	DCMI_CaptureMode	32L496GДИСКАВЕРИ
	Режим моментального снимка	

1. Все примеры разработаны для захвата данных RGB. Для большинства примеров пользователь может выбрать одно из следующих разрешений: QQVGA 160x120, QVGA 320x240, 480x272, VGA 640x480.

6.3 Примеры DCMI на основе STM32CubeMX

В этом разделе представлено описание пяти типичных примеров использования DCMI:

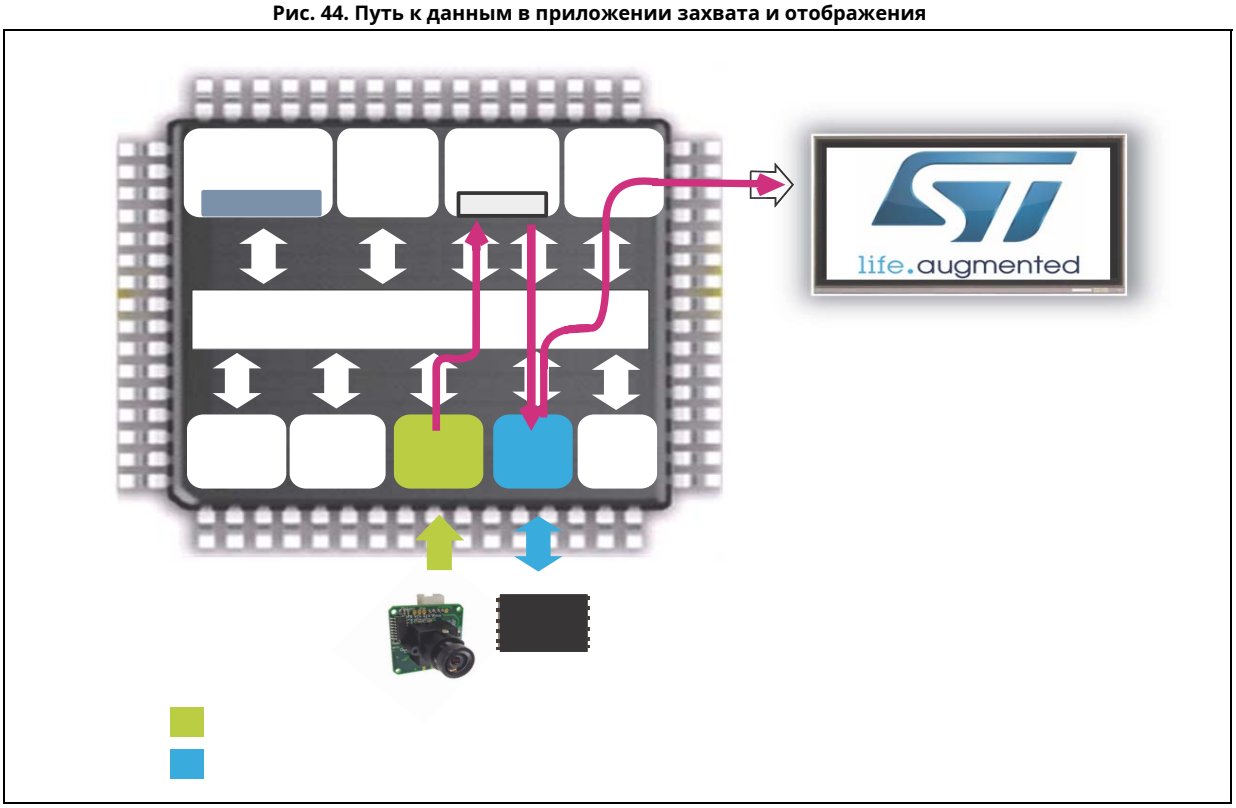
- Захват и отображение данных RGB: данные захватываются в формате RGB565 с разрешением QVGA (320x240), сохраняются в SDRAM и отображаются на LCD-TFT.
- Захват данных YCbCr: данные записываются в формате YCbCr с разрешением QVGA (320x240) и сохраняются в SDRAM.
- Захват данных только Y: DCMI настроен на получение данных только Y для сохранения в SDRAM.
- Захват с разрешением SxGA (формат данных YCbCr): данные записываются в формате YCbCr с разрешением SxGA (1280x1024) и сохраняются в SDRAM.
- Захват данных JPEG: данные захватываются в формате JPEG для сохранения в SDRAM.

Все эти примеры были реализованы на 32F746GDISCOVERY с использованием STM32F4DIS-CAM (датчик CMOS OV9655), за исключением захвата данных JPEG, реализованного на STM324x9I-EVAL (датчик CMOS OV2640).

Как показано в*Рисунок 44*, приложение состоит из трех основных шагов:

- импорт полученных данных из DCMI в DMA (для временного хранения в FIFO) через свой периферийный порт.
- перенос данных из FIFO в SDRAM
- импорт данных из SDRAM для отображения на LCD-TFT, только для формата данных RGB. Для формата данных YCbCr или JPEG пользователь должен преобразовать полученные данные в формат RGB для отображения.

Рис. 44. Путь к данным в приложении захвата и отображения



Для этих примеров пользователю необходимо настроить DCMI, DMA2, LTDC (для примера захвата и отображения данных RGB) и SDRAM.

Пять примеров, описанных в следующих разделах, имеют некоторые общие конфигурации на основе STM32CubeMX:

- Конфигурация GPIO
- Конфигурация прямого доступа к памяти
- Конфигурация часов

Для примеров захвата только Y и JPEG необходимы следующие конкретные конфигурации:

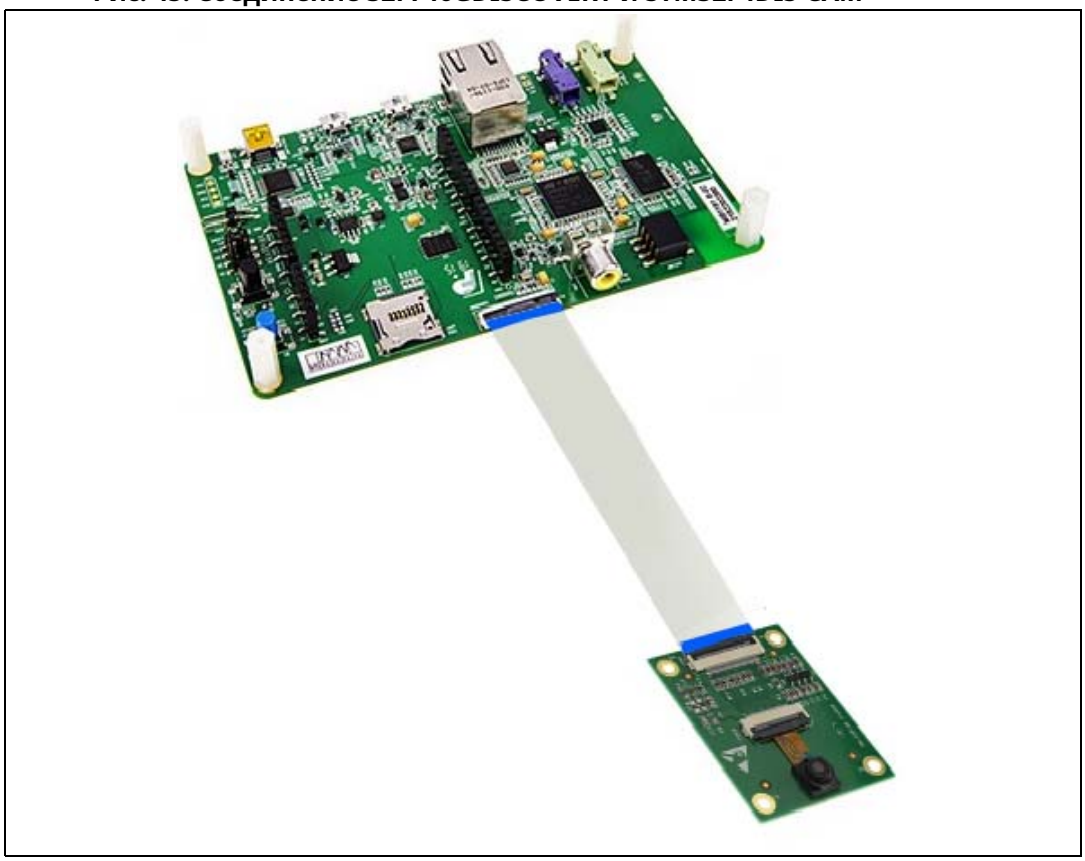
- Конфигурация периферийных устройств DCMI
- Конфигурация модуля камеры

В следующих разделах представлено описание оборудования, общая конфигурация с использованием STM32CubeMX и общие модификации, которые необходимо добавить в проект, сгенерированный STM32CubeMX.

6.3.1 Описание оборудования

Следующие примеры (кроме примера захвата JPEG) были реализованы на 32F746GDISCOVERY с использованием платы камеры STM32F4DIS-CAM.

Рис. 45. Соединение 32F746GDISCOVERY и STM32F4DIS-CAM



1. Изображение не является договорным.

Плата STM32F4DIS-CAM включает CMOS-сенсор Omnivision (ov9655), 1,3 мегапикселя. Разрешение может достигать 1280x1024. Этот модуль камеры подключается к DCMI через 30-контактный разъем FFC.

Плата 32F746GDISCOVERY оснащена 4,3-дюймовым цветным ЖК-дисплеем TFT с емкостным сенсорным экраном, который используется в первом примере для отображения захваченных изображений.

Как показано в*Рисунок 46*, модуль камеры подключается к STM32F7 через:

- управляющие сигналы DCMI_PIXCLK, DCMI_VSYNC, DCMI_HSYNC
- сигналы данных изображения DCMI_D[0..7]

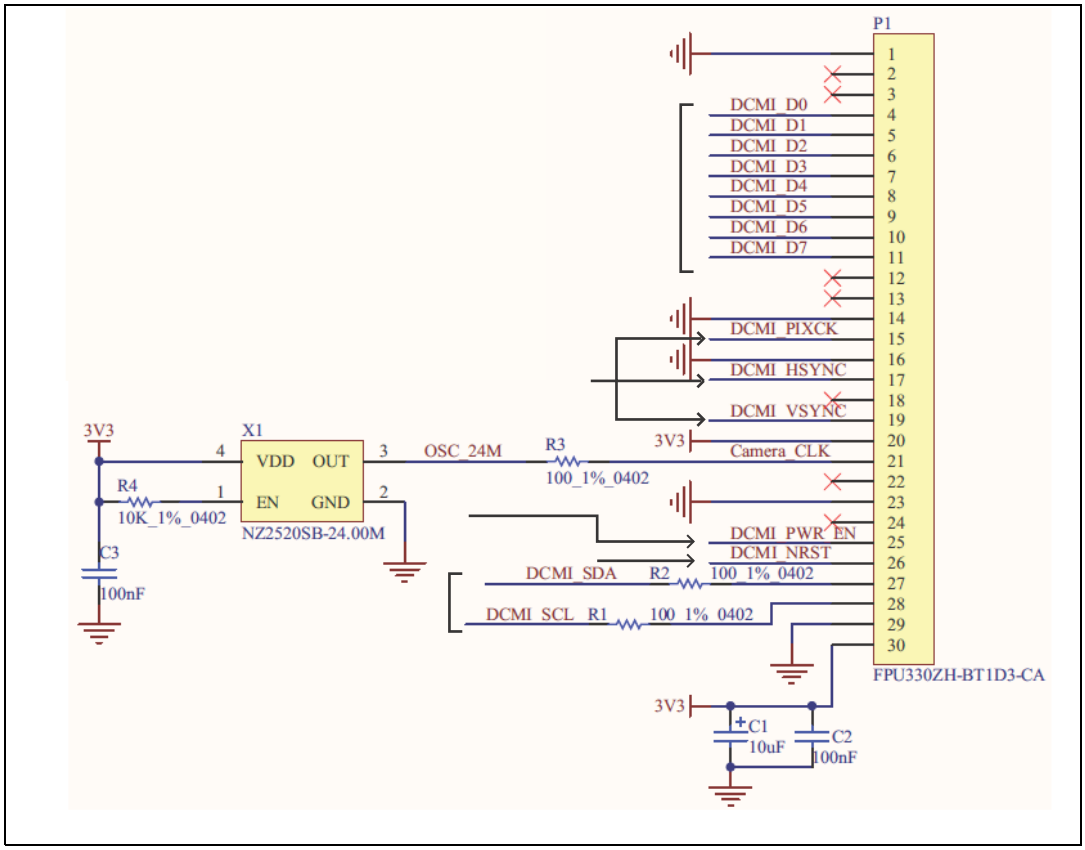
Дополнительные сигналы подаются на модуль камеры через 30-контактный FFC:

- сигналы питания (DCMI_PWR_EN)
- часы для модуля камеры (Camera_CLK)
- сигналы конфигурации (I2C)
- сигнал сброса (DCMI_NRST)

Подробнее об этих сигналах см.*Раздел 1.2.2: Соединение модуля камеры (параллельный интерфейс)*.

Часы камеры подаются на модуль камеры через вывод Camera_CLK от кварцевого генератора тактовых импульсов NZ2520SB (X1), встроенного в плату 32F746GDISCOVERY. Частота часов камеры равна 24 МГц.

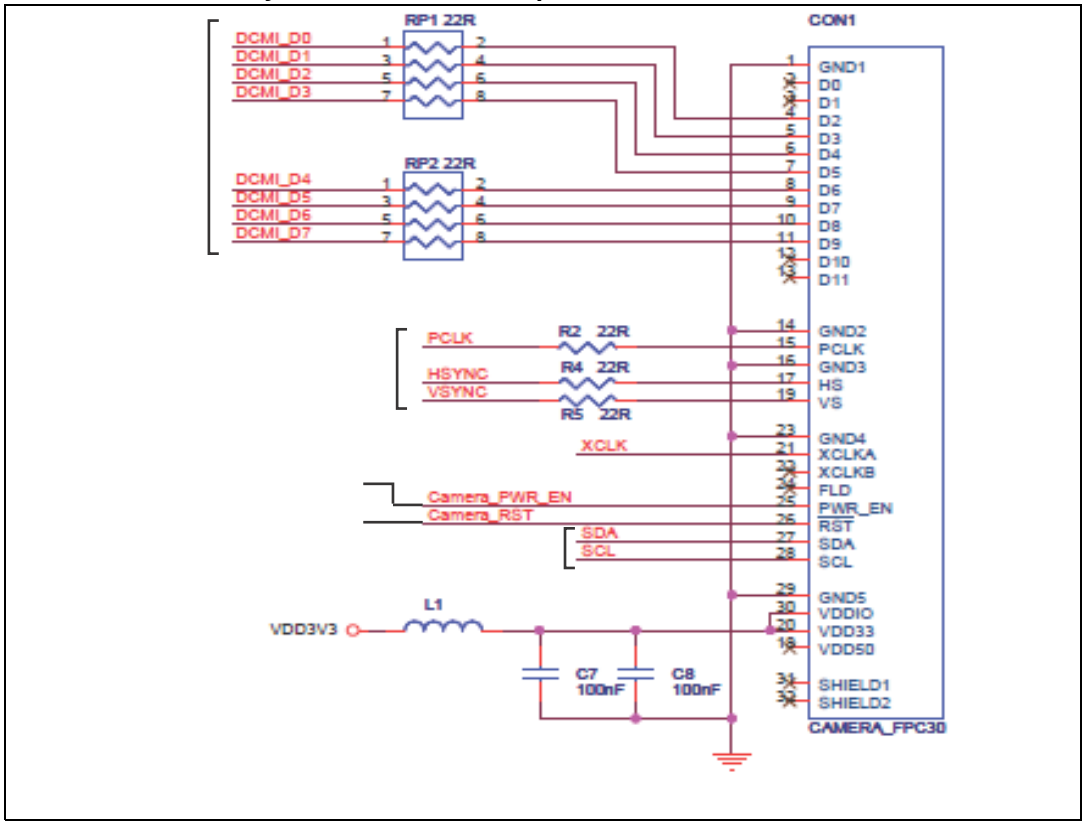
Контакт сброса DCMI (DCMI_NRST), позволяющий сбросить модуль камеры, подключен к глобальному контакту сброса MCU (NRST).



Дополнительные сведения о плате 32F746GDISCOVERY см. в руководстве пользователя. *Комплект Discovery для серии STM32F7 с микроконтроллером STM32F746NG*(UM1907), доступный на веб-сайте STMicroelectronics.

Разъем модуля камеры, реализованный на STM32F4DIS-CAM, показан на *Рисунок 47*.

Рисунок 47. Разъем камеры на STM32F4DIS-CAM



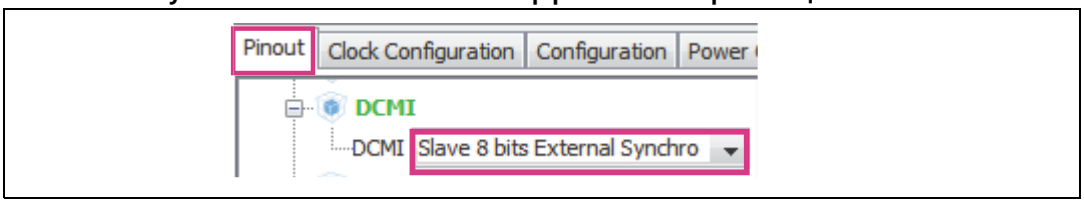
6.3.2 Общие примеры конфигурации

При запуске с STM32CubeMX первым шагом является настройка местоположения проекта и соответствующей цепочки инструментов или IDE (меню Project/Settings).

STM32CubeMX — конфигурация DCMI GPIO

1. Выберите DCMI и выберите «Slave 8 bit External Synchro» на вкладке Pinout, чтобы настроить DCMI на подчиненную 8-битную внешнюю (аппаратную) синхронизацию (*Рисунок 48*).

Рисунок 48. STM32CubeMX — выбор режима синхронизации DCMI



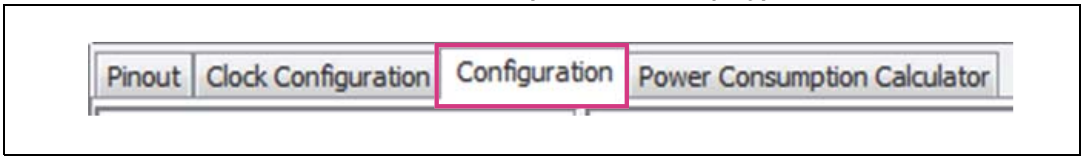
Если после выбора одной аппаратной конфигурации (Вспомогательная 8-битная внешняя синхронизация) GPIO не соответствует оборудованию, пользователь может изменить желаемый GPIO и настроить альтернативную функцию непосредственно на выводе.

Другой метод состоит в ручной настройке контактов GPIO путем выбора правильной альтернативной функции для каждого из них. Дополнительные сведения о GPIO, которые необходимо настроить, см. *Рисунок 52: STM32CubeMX — выбор контактов DCMI*.

После этого шага 11 контактов должны быть выделены зеленым цветом (D[0..7], DCMI_VSYNC, DCMI_HSYNC и DCMI_PIXCLK).

2. Выберите вкладку «Конфигурация», чтобы настроить режим и скорость GPIO, как показано на *Рисунок 51*.

Рис. 49. STM32CubeMX — выбор вкладки «Конфигурация»

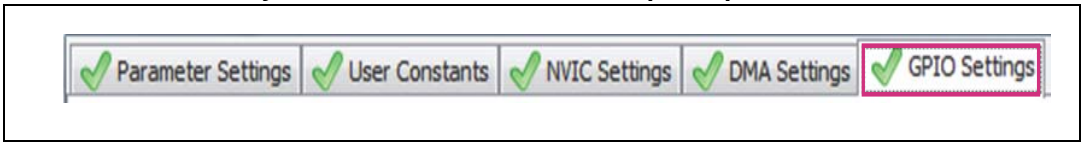


3. Нажмите кнопку DCMI на вкладке конфигурации, как показано на *Рисунок 50*.

Рис. 50. STM32CubeMX — кнопка DCMI на вкладке «Конфигурация»

4. Когда появится окно конфигурации DCMI, выберите вкладку настроек GPIO, как показано на *Рисунок 51*.

Рисунок 51. STM32CubeMX — выбор настроек GPIO

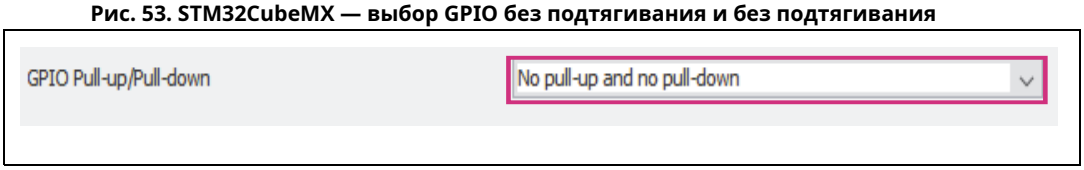


5. Выберите все контакты DCMI, как показано на *Рисунок 52*.

Рисунок 52. STM32CubeMX — выбор выводов DCMI

6. Установите подтяжку/подтягивание GPIO, как показано на *Рисунок 53*.

Рис. 53. STM32CubeMX — выбор GPIO без подтягивания и без подтягивания

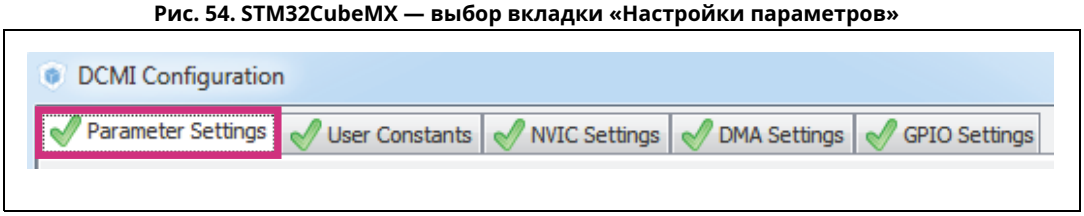


7. Нажмите «Применить» и «OK».

STMCubeMX — управляющие сигналы DCMI и конфигурация режима захвата

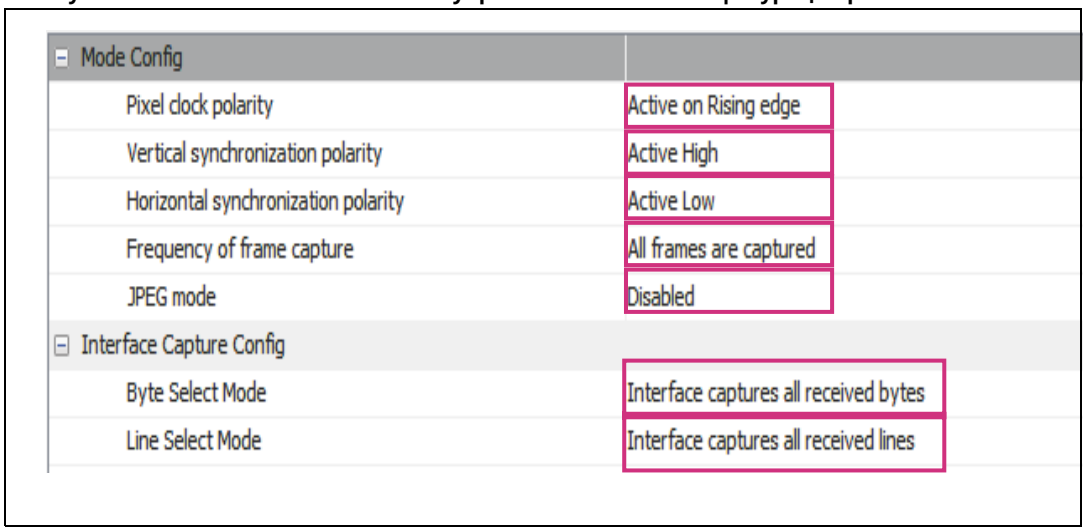
1. Щелкните вкладку «Параметры параметров» в окне «Конфигурация DCMI», затем выберите вкладку «Параметры параметров», как показано на *Рисунок 54*.

Рис. 54. STM32CubeMX — выбор вкладки «Настройки параметров»



2. Установите различные параметры, как показано на *Рисунок 55*. Вертикальная синхронизация, горизонтальная синхронизация и полярность синхронизации пикселей должны быть запрограммированы в соответствии с конфигурацией модуля камеры.

Рисунок 55. STM32CubeMX — сигналы управления DCMI и конфигурация режима захвата



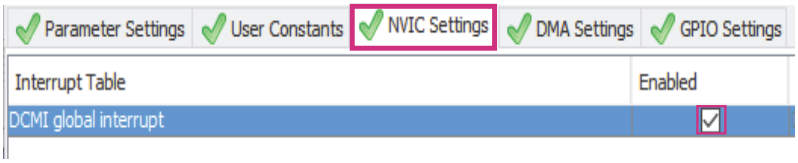
3. Нажмите «Применить» и «OK».

Примечание: *Полярность вертикальной синхронизации должна быть активной высокой, а полярность горизонтальной синхронизации должна быть активной низкой. Для данной конфигурации модуля камеры их нельзя переворачивать.*

STM32CubeMX — включение прерываний DCMI

1. Выберите вкладку «Настройки NVIC» в окне «Конфигурация DCMI» и проверьте глобальное прерывание DCMI, как показано на *Рисунок 56*.

Рисунок 56. STM32CubeMX — конфигурация прерываний DCMI



2. Нажмите «Применить» и «OK».

STM32CubeMX — конфигурация прямого доступа к памяти

Эта конфигурация предназначена для приема данных RGB565 (2 байта на пиксель), а разрешение изображения — QVGA (320x240). Тогда размер изображения будет 320 * 240 * 2 = 153600 байт.

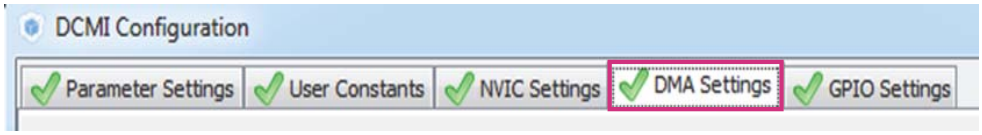
Поскольку ширина данных, отправляемых из DCMI, составляет 4 байта (32-битные слова, отправленные из регистра данных в DCMI), количество элементов данных в регистре DMA_SxNDTR равно количеству слов для передачи. Тогда количество слов равно 38400 (153600/4), что меньше 65535.

В режиме моментальных снимков пользователь может настроить DMA в обычном режиме.

В непрерывном режиме пользователь может настроить DMA в круговом режиме.

1. Выберите вкладку «Настройки DMA» в окне «Конфигурация DCMI», как показано на *Рисунок 57*.

Рис. 57. STM32CubeMX — выбор вкладки DMA Settings

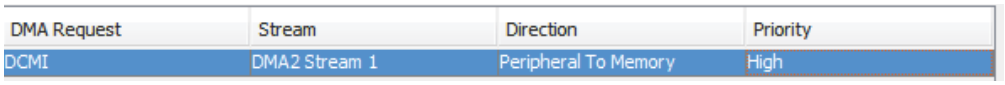


2. Нажмите кнопку «Добавить», показанную на *Рисунок 58*.

Рис. 58. STM32CubeMX — выбор кнопки «Добавить»

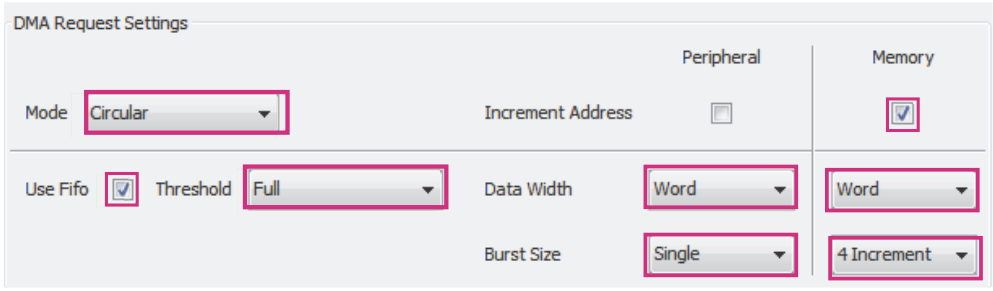
3. Нажмите «Выбрать» в разделе «Запрос DMA» и выберите «DCMI». Запрос DMA настроен, как показано на *Рисунок 59*. Канал 1 потока 1 DMA2 настроен на передачу запроса DCMI каждый раз, когда заполняется его временной регистр.

Рисунок 59. STM32CubeMX — конфигурация потока DMA



4. Измените настройки запроса DMA, как показано на *Рисунок 60*.

Рисунок 60. STM32CubeMX — конфигурация DMA



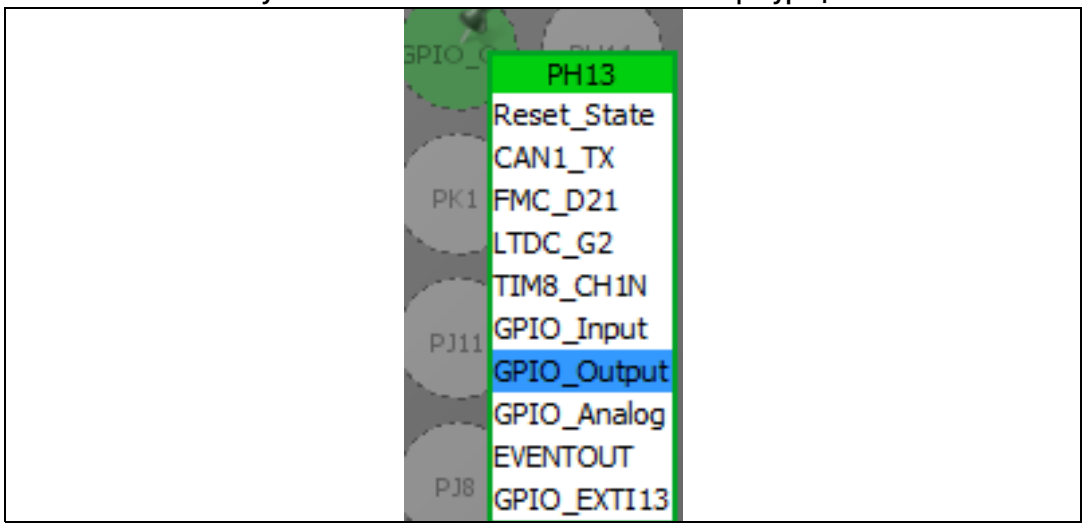
5. Нажмите «Применить» и «OK».

STM32CubeMX — контакты включения модуля камеры

Для включения модуля камеры контакт PH13 должен быть настроен на 32F746GDISCOVERY.

1. Нажмите на контакт PH13 и выберите GPIO_Output на вкладке Pinout, как показано на *Рисунок 61*.

Рисунок 61. CTM п конфигурация

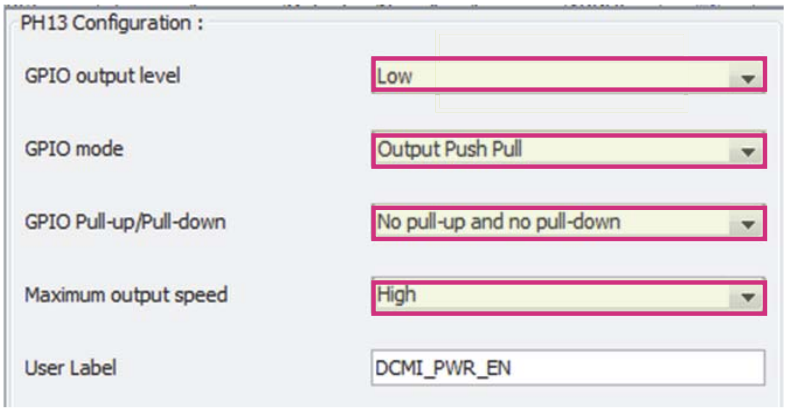


2. На вкладке «Конфигурация» нажмите кнопку GPIO, показанную на рис.*Рисунок 62*.

Рисунок 62. STM32CubeMX — кнопка GPIO во вкладке конфигурации

3. Установите параметры, как показано на *Рисунок 63*.

Рис. 63. STM32CubeMX — конфигурация выводов питания DCMI



STM32CubeMX — конфигурация системных часов

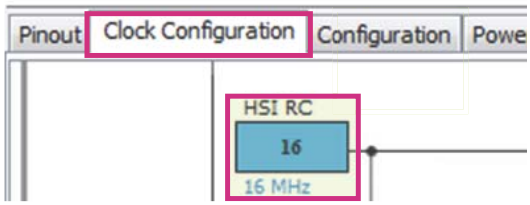
В этом примере системные часы настроены следующим образом:

- использование внутреннего HSI RC, где основной PLL используется в качестве системного источника тактового сигнала.
- HCLK @ 200 МГц, поэтому Cortex®-M7 и LTDC работают на частоте 200 МГц.

Примечание: *HCLK настроен на 200 МГц, но не на 216 МГц, чтобы установить SDRAM_FMC на максимальную скорость 100 МГц с предварительным делителем HCLK/2.*

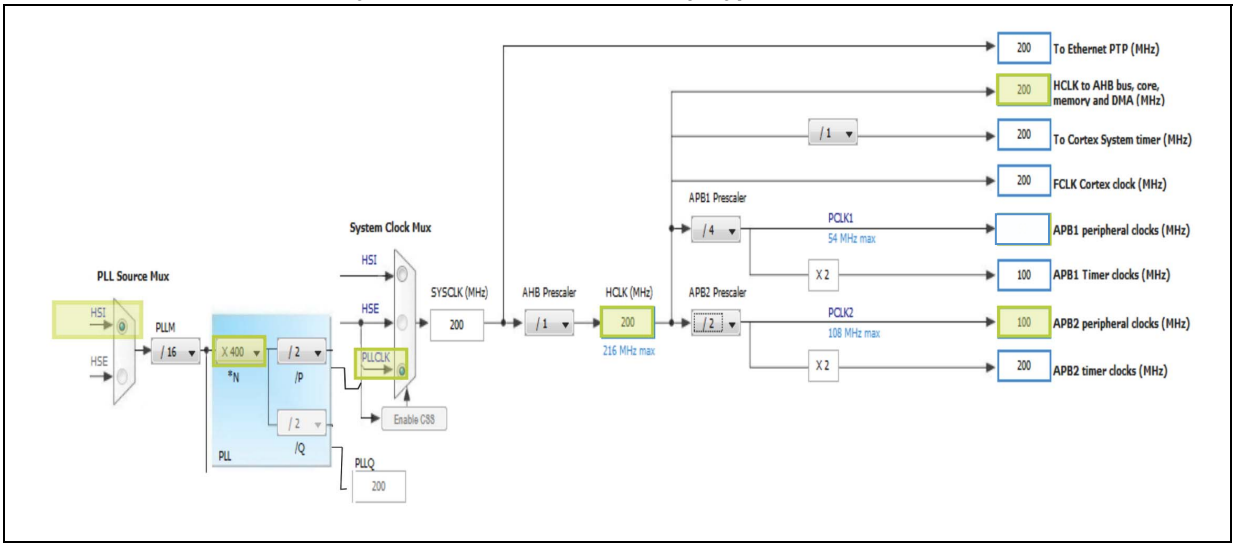
1. Выберите вкладку «Конфигурация часов», как показано на *Рисунок 64*.

Рис. 64. STM32CubeMX — конфигурация HSI



2. Установите PLL и предварительные делители на вкладке Clock Configuration, чтобы получить системные часы HCLK @ 200 МГц, как показано на рис.*Рисунок 65*.

Рисунок 65. STM32CubeMX — конфигурация часов



На этом уровне пользователь может сгенерировать проект.

Добавление файлов в проект

Сгенерируйте код и откройте сгенерированный проект, используя предпочитаемую цепочку инструментов, и выполните следующие действия:

1. Щелкните правой кнопкой мыши «Драйверы/STM32F7xx_HAL_Driver».

2. Выберите «Добавить существующие файлы в группу «Драйверы/STM32F7xx_HAL_Driver...».

3. Выберите следующие файлы в «Drivers/STM32F7xx_HAL_Driver/**Источник**":

   – 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. Раскомментировать модули DMA2D, LTDC, SDRAM, UART в**stm32f7xx_hal_conf.h.**

5. Создайте новую группу с именем, например, Imported_Drivers.

6. Скопируйте следующие файлы из папки STM32746G_Discovery в каталоге C: в папку **Источник**папка проекта:

   – stm32746g_discovery.c

   – stm32746g_discovery_sdram.c

7. Скопируйте следующие файлы из папки STM32746G_Discovery в каталоге C: в папку **Источник**папка проекта:

   – stm32746g_discovery.h

   – stm32746g_discovery_sdram.h

8. Копировать**ov9655.c**из папки Components в папку Src.

9. Копировать**ov9655.h**из папки Components в папку Inc.

10. Копировать**камера.ч**из папки Component/Common в папку Inc.

11. Добавьте следующие файлы в новую группу (называемую в данном примере Imported_Drivers):

    – stm32746g_discovery.h

    – stm32746g_discovery_sdram.h

    – ov9655.c.

12. Разрешить модификации**ov9655.h**а также**камера.ч**(только для чтения по умолчанию), по:

    - щелчок прямо на файле

    - снять галочку только для чтения

    – нажав «Применить» и «ОК».

13. Измените**ov9655.h**файл, заменив #включить "../Common/camera.hпо # включить "camera.h."

14. Скопируйте следующие файлы в папку Inc:

    – **rk043fn48h.h**из папки Компоненты

    – **шрифты.h**а также**шрифты24.h**из папки Утилиты/Шрифты.

15. Убедитесь, что проблем не возникло, восстановив все файлы. Не должно быть ни ошибки, ни предупреждения.

Изменения в файле main.c

1. Обновить**main.c**вставив некоторые инструкции, чтобы включить необходимые файлы в достаточное пространство, указанное в**зеленый полужирный**ниже. Эта задача обеспечивает модификацию и перегенерацию проекта без потери пользовательского кода:

```
/* НАЧАЛО КОДА ПОЛЬЗОВАТЕЛЯ Включает */
# включить "stm32746g_discovery.h"
# включить "stm32746g_discovery_sdram.h"
# включить "ov9655.h"
# включить "rk043fn48h.h"
# включить "fonts.h"
# включить "font24.c"
/* КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ Включает */
```

Затем необходимо вставить некоторые объявления переменных в соответствующее место, указанное в**зеленый полужирный**ниже.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО PV */
/* Частные переменные ----------------------------------------------------------- ------*/ перечисление typedef

{
  КАМЕРА_OK             = 0x00,
  КАМЕРА_ОШИБКА         = 0x01,
  КАМЕРА_TIMEOUT        = 0x02,
  КАМЕРА_НЕ ОБНАРУЖЕНА  = 0x03,
  CAMERA_NOT_SUPPORTED = 0x04 }
Camera_StatusTypeDef; структура
typedef
{
  uint32_t Цвет текста;
  uint32_t Цвет фона;
  sFONT * pШрифт;
}LCD_DrawPropTypeDef;
структура typedef
{
  int16_t X;
  int16_t Y;
} Точка, * pPoint;
статический LCD_DrawPropTypeDef DrawProp[2];
LTDC_HandleTypeDef hltdc;
LTDC_LayerCfgTypeDef layer_cfg;
статический RCC_PeriphCLKInitTypeDef periph_clk_init_struct;
CAMERA_DrvTypeDef * camera_drive;
/* Аппаратный адрес I2C модуля камеры */
статический uint32_t CameraHwAddress;
```

```
/* Размер изображения */
uint32_t Im_size = 0; /* КОД
ПОЛЬЗОВАТЕЛЯ КОНЕЦ PV */
```

После этого необходимо вставить прототипы функций в соответствующее место, указанное в**зеленый полужирный**ниже.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО PFP */
/* Прототипы закрытых функций ----------------------------------------------------------*/ uint8_t CAMERA_Init
(uint32_t);
static void LTDC_Init(uint32_t, uint16_t, uint16_t, uint16_t, uint16_t);

недействительным LCD_GPIO_Init (LTDC_HandleTypeDef *, недействительным *); /*
КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ PFP */
```

2. Обновить**главный()**функцию, вставив некоторые функции в соответствующее место, указанное в **зеленый полужирный**ниже. Функция LTDC_Init позволяет настраивать и инициализировать ЖК-дисплей. Функция BSP_SDRAM_Init позволяет настраивать и инициализировать SDRAM. Функция CAMERA_Init позволяет настраивать модуль камеры, а также регистры и параметры DCMI. Одну из двух функций HAL_DCMI_Start_DMA, позволяющую настраивать DCMI в моментальном снимке или в непрерывном режиме, необходимо раскомментировать.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО 2 */
LTDC_Init(FRAME_BUFFER, 0, 0, 320, 240);
BSP_SDRAM_Init();
CAMERA_Init(CAMERA_R320x240);
HAL_Delay (1000);          //Задержка для камеры для вывода правильных данных
Im_size = 0x9600;                  //размер=320*240*2/4
/* раскомментируйте следующую строку в случае режима снимка */ //
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size); /*
раскомментировать следующую строку в случае непрерывного режима */
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, (uint32_t)FRAME_BUFFER, Im_size); /*
КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ 2 */
```

3. Вставьте реализацию новых функций (вызываемых в функции main()) вне основной функции в соответствующем месте, указанном в**зеленый полужирный**ниже.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО 4 */
void LCD_GPIO_Init(LTDC_HandleTypeDef *hltdc, void *Params) {

  GPIO_InitTypeDef gpio_init_structure; /* Включить
  часы LTDC и DMA2D */
  __HAL_RCC_LTDC_CLK_ENABLE();
  __HAL_RCC_DMA2D_CLK_ENABLE(); /* Включить
  часы GPIO */ __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(); /***
Конфигурация контактов LTDC ***/ /*
Конфигурация GPIOE */
gpio_init_structure.Pin gpio_init_structure.Mode GPIO_PIN_4;
gpio_init_structure.Pull gpio_init_structure.Speed = GPIO_MODE_AF_PP;
gpio_init_structure.Alternate = GPIO_AF14_LTDC; = GPIO_NOPULL;
HAL_GPIO_Init(GPIOE, &gpio_init_structure); = GPIO_SPEED_FAST; /*
GPIOG */ gpio_init_structure.Pin gpio_init_structure.Mode
gpio_init_structure.Alternate = GPIO_AF9_LTDC;
HAL_GPIO_Init(GPIOG, &gpio_init_structure); /*
Альтернативная конфигурация GPIOI LTDC */ GPIO_PIN_12;
                                          = GPIO_MODE_AF_PP;

gpio_init_structure.Pin                   = GPIO_PIN_9 | GPIO_PIN_10 | GPIO_PIN_13 |
GPIO_PIN_14 | GPIO_PIN_15;
gpio_init_structure.Mode gpio_init_structure.Alternate = GPIO_MODE_AF_PP;
GPIO_AF14_LTDC; HAL_GPIO_Init(GPIOI, &gpio_init_structure); /
* Конфигурация GPIOJ */

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); /* Конфигурация
GPIOK */
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); /* Конфигурация
LCD_DISP GPIO */
gpio_init_structure.Pin ручное            = GPIO_PIN_12;          /* Вывод LCD_DISP должен быть
управление */
gpio_init_structure.Mode HAL_GPIO_Init(GPIOK, = GPIO_MODE_OUTPUT_PP;
&gpio_init_structure); /* LCD_BL_CTRL Конфигурация
GPIO */
gpio_init_structure.Pin ручное            = GPIO_PIN_3;       /* вывод LCD_BL_CTRL должен быть
управление */
gpio_init_structure.Mode HAL_GPIO_Init(GPIOK, = GPIO_MODE_OUTPUT_PP;
&gpio_init_structure);
```

```
}

static void LTDC_Init (uint32_t FB_Address, uint16_t Xpos, uint16_t Ypos, uint16_t Width, uint16_t
Height)
{
/* Конфигурация синхронизации */
   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);
   /* Конфигурация ЖК-часов */ 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); /*
   Инициализировать ширину пикселя LCD и высоту пикселя */
   hltdc.LayerCfg->ImageWidth = RK043FN48H_WIDTH; hltdc.LayerCfg-
   >ImageHeight = RK043FN48H_HEIGHT; hltdc.Init.Backcolor.Blue = 0;/*
   Фоновое значение */ hltdc.Init.Backcolor.Green = 0;
   hltdc.Init.Backcolor.Red = 0; /* Полярность */


   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;
   если (HAL_LTDC_GetState (& hltdc) == HAL_LTDC_STATE_RESET) {

      LCD_GPIO_Init(&hltdc, NULL);
   }
   HAL_LTDC_Init(&hltdc);
   /* Подтверждение включения дисплея на выводе LCD_DISP */
   HAL_GPIO_WritePin(GPIOI, GPIO_PIN_12, GPIO_PIN_SET); /*
   Утверждаем вывод подсветки LCD_BL_CTRL */
   HAL_GPIO_WritePin(GPIOK, GPIO_PIN_3, GPIO_PIN_SET);
 DrawProp[0].pFont = &Font24 ;
```

```
    /* Инициализация слоя */ layer_cfg.WindowX0 = Xpos;
    layer_cfg.WindowX1 = Ширина; layer_cfg.WindowY0 = Ypos;
    layer_cfg.WindowY1 = высота; layer_cfg.PixelFormat =
    LTDC_PIXEL_FORMAT_RGB565; layer_cfg.FBStartAdress =
    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 = Ширина;


    layer_cfg.ImageHeight = высота;
    HAL_LTDC_ConfigLayer(&hltdc, &layer_cfg, 1);
    DrawProp[1].BackColor = ((uint32_t)0xFFFFFFFF);
    DrawProp[1].pFont                = &Шрифт24;
    DrawProp[1].TextColor = ((uint32_t)0xFF000000);
}

uint8_t CAMERA_Init (разрешение uint32_t)/*Инициализация камеры*/ {

    uint8_t статус = CAMERA_ERROR;
    /* Чтение идентификатора модуля камеры через I2C */ если
    (ov9655_ReadID (КАМЕРА_I2C_ADDRESS) == OV9655_ID) {

        camera_drive = &ov9655_drv;/* Инициализировать структуру драйвера камеры */
        CameraHwAddress = CAMERA_I2C_ADDRESS; если (Разрешение == CAMERA_R320x240)

        {
            camera_drive->Init(CameraHwAddress, Resolution);
            HAL_DCMI_DisableCROP(&hdcmi);
        }
        статус = КАМЕРА_ОК;/* Возвращаем статус CAMERA_OK */
    }
    еще
    {
        статус = КАМЕРА_НЕ ПОДДЕРЖИВАЕТСЯ;/* Возвращаем статус CAMERA_NOT_SUPPORTED */
    }
    статус возврата;
}
/* КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ 4 */
```

Изменения в файле main.h

Обновлять**main.h**путем вставки объявления адреса буфера кадра в соответствующее место, указанное взеленыйниже.

```
/* USER CODE BEGIN Private определяет */
# определить FRAME_BUFFER 0xC0000000
/* КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ Приватные определения */
```

На этом этапе пользователь может собрать, отладить и запустить проект.

6.3.3 Захват и отображение данных RGB

Чтобы упростить этот пример, данные захватываются и отображаются в формате RGB565 (2 бита на пиксель). Разрешение изображения 320x240 (QVGA). Буфер кадра помещается в SDRAM. Данные камеры и ЖК-дисплея находятся в одном и том же кадровом буфере. Затем на ЖК-дисплее отображаются данные, полученные через DCMI, без какой-либо обработки. Затем модуль камеры настроен на вывод данных RGB565, QVGA (320x240).

Конфигурацию этого примера можно выполнить, выполнив шаги, описанные в *Раздел 6.3.2: Общие примеры конфигурации*.

6.3.4 Сбор данных YCbCr

Описание

Этот пример реализации направлен на получение данных YCbCr от модуля камеры и передачу их в SDRAM.

Отображение полученных данных YCbCr на ЖК-дисплее (настроенном для отображения данных RGB565 в предыдущей конфигурации) некорректно, но может использоваться для проверки.

Для правильного отображения изображений данные YCbCr необходимо преобразовать в данные RGB565 (или RGB888 или ARGB8888, в зависимости от потребностей приложения).

Все этапы конфигурации обозначены в*Раздел 6.3.2: Общие примеры конфигурации*необходимо соблюдать, и вот некоторые инструкции, которые необходимо добавить для получения данных YCbCr. Необходимо обновить только конфигурацию камеры.

Конфигурация модуля камеры:

Конфигурация нового модуля камеры выполняется путем добавления:

- таблица констант, позволяющая конфигурировать регистры модуля камеры
- новая функция, позволяющая настраивать модуль камеры путем отправки конфигурации регистров через I2C.

1. Объявление таблицы, содержащей конфигурации регистров модуля камеры, должно быть добавлено в файл main.c ниже "/* Частные переменные ---------- ---------- ---------- */".

```
константный символ без знака 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. Новый прототип функции должен быть вставлен ниже

"/* Прототипы закрытых функций ---------- ---*/".

недействительным OV9655_YUV_Init (uint16_t);

3. Второй шаг**изменения в файле main.c**описано в*Раздел 6.3.2: Общие примеры конфигурации* должен быть обновлен. Изменить**главный()**функцию, вставив следующие функции в соответствующее место, указанное в**зеленый полужирный**ниже. Одну из двух функций, позволяющих настраивать DCMI в моментальном снимке или в непрерывном режиме, необходимо раскомментировать.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО
2 */ BSP_SDRAM_Init();
  CAMERA_Init(CameraHwAddress);
OV9655_YUV_Init (КамераHwAddress);
HAL_Delay (1000); //Задержка для камеры для вывода правильных данных
  Im_size = 0x9600;//размер=320*240*2/4
/* раскомментируйте следующую строку в случае режима снимка */ //
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size); /*
раскомментировать следующую строку в случае непрерывного режима */
  HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, (uint32_t)FRAME_BUFFER, Im_size); /* КОД
ПОЛЬЗОВАТЕЛЯ КОНЕЦ 2 */
```

4. Третий шаг**изменения в main.c**описано в*Раздел 6.3.2: Общие примеры конфигурации*необходимо обновить, добавив новую реализацию функции

пустота OV9655_YUV_Init (uint16_t DeviceAddr)

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

6.3.5 Захват только формата данных Y

Описание

В этом примере модуль камеры настроен на вывод данных в формате YCbCr. При использовании функции выбора байта на стороне DCMI компоненты цветности (Cb и Cr) игнорируются, и только компонент Y передается в кадровый буфер в SDRAM.

Все этапы конфигурации обозначены в*Раздел 6.3.2: Общие примеры конфигурации*необходимо соблюдать, и вот некоторые инструкции, которые необходимо добавить для получения данных только Y. Необходимо обновить только камеру и конфигурацию DCMI.

Чтобы упростить эту задачу,**файл main.c**необходимо изменить, как описано в*Раздел 6.3.4: Сбор данных YCbCr*но второй шаг**STM32CubeMX — управляющие сигналы DCMI и конфигурация режима захвата**или**статическая пустота MX_DCMI_Init (пустая)**Функция (эта функция реализована в файле main.c) должна быть изменена, чтобы иметь следующую конфигурацию:

```
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 Захват с разрешением SxGA (формат данных YCbCr)

Описание

Этот пример реализации направлен на получение данных YCbCr от модуля камеры и передачу их в SDRAM. Разрешение захваченных изображений SxGA (1280x1024).

Отображение полученных данных YCbCr на ЖК-дисплее (настроенном для отображения данных RGB565) некорректно.

Для правильного отображения изображений данные YCbCr необходимо преобразовать в данные RGB565 (или RGB888 или ARGB8888, в зависимости от потребностей приложения).

Все этапы конфигурации обозначены в*Раздел 6.3.2: Общие примеры конфигурации*необходимо соблюдать, и вот некоторые инструкции, которые необходимо добавить для получения данных YCbCr. Необходимо обновить только камеру и конфигурацию DMA.

Конфигурация прямого доступа к памяти

DMA настраивается, как описано в*Раздел 4.4.9: Конфигурация DMA для более высоких разрешений*и функция HAL_DMA_START при вызове обеспечивает эту конфигурацию, поскольку размер изображения превышает максимально допустимый размер для режима двойного буфера.

Фактически при вызове функции HAL_DMA_START она обеспечивает разделение полученных кадров на равные части и размещение каждой части в одном кадровом буфере. Как пояснено, для разрешения SxGA каждый кадр разбивается на 16 кадровых буферов. Размер каждого буфера равен 40960 слов.

Для адресов буферов функция HAL_DMA_START обеспечивает размещение 16 кадровых буферов в памяти. В этом случае адрес первого кадрового буфера равен 0xC0000000, второго адреса — 0xC0163840 (0xC0000000 + (40960 * 4)) и 16йадрес буфера кадров (0xC0000000 + 16 * (40960 * 4)).

В конце передачи DMA заполняет один кадр, генерируется прерывание, вычисляется адрес следующего буфера и модифицируется один указатель, как показано на*Рисунок 42: Работа прямого доступа к памяти в случае высокого разрешения*.

Конфигурация модуля камеры:

Конфигурация нового модуля камеры выполняется путем добавления:

- таблица констант, позволяющая конфигурировать регистры модуля камеры
- новая функция, позволяющая настраивать модуль камеры путем отправки конфигурации регистров через I2C.

Чтобы убедиться, что модуль камеры отправляет изображение с разрешением SxGA и форматом YCbCr, регистры датчика CMOS должны быть настроены, как показано ниже:

1. Объявление таблицы, содержащей конфигурации регистров модуля камеры, должно быть добавлено в файл main.c ниже "/* Частные переменные ---------- -----*/".

константный символ без знака 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},{{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}, {{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}, {0x87},{0x83, 0x96},{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. Новый прототип функции должен быть вставлен ниже

"/* Прототипы закрытых функций ---------- ---*/ ":

недействительным OV9655_YUV_Init (uint16_t);

3. Второй шаг**изменения в папке main.c**в этом примере — обновить **главный()**функцию, вставив следующие функции в соответствующее место, указанное в **зеленый полужирный**ниже. Одну из двух функций, позволяющих настраивать DCMI в моментальном снимке или в непрерывном режиме, необходимо раскомментировать.

```
/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО
2 */ BSP_SDRAM_Init();
CAMERA_Init(CameraHwAddress);
OV9655_YUV_Init (КамераHwAddress);
HAL_Delay (1000); //Задержка для вывода корректных данных камерой Im_size =
0xA0000; //размер=1280*1024*2/4
/* раскомментируйте следующую строку в случае режима моментального снимка */
//HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_SNAPSHOT, (uint32_t)FRAME_BUFFER, Im_size);

/* раскомментировать следующую строку в случае непрерывного режима */
HAL_DCMI_Start_DMA(&hdcmi, DCMI_MODE_CONTINUOUS, (uint32_t)FRAME_BUFFER, Im_size);

/* КОД ПОЛЬЗОВАТЕЛЯ КОНЕЦ 2 */
```

4. Третий шаг**изменения в main.c**описано в*Раздел 6.3.2: Общие примеры конфигурации*необходимо обновить, добавив новую реализацию функции ниже

/* КОД ПОЛЬЗОВАТЕЛЯ НАЧАЛО 4 */ .

```
пустота OV9655_YUV_Init (uint16_t DeviceAddr)
 {
 индекс uint32_t;
for(index=0; index<(sizeof(ov9655_yuv_sxga)/2); index++)
 {
 CAMERA_IO_Write(DeviceAddr, ov9655_yuv_sxga[index][0],
ov9655_yuv_sxga[index][1]);
 КАМЕРА_Задержка (1);
 }
}
```

Примечание: *В случае кадра SxGA с форматом данных RGB пользователь может уменьшить разрешение для отображения полученных изображений на TFT-LCD, используя функцию изменения размера DCMI.*

6.3.7 Захват формата JPEG

Описание

Датчик CMOS OV9655, встроенный в плату STM32F4DIS-Cam, не поддерживает сжатые выходные данные. Затем этот пример реализуется с использованием КМОП-сенсора OV2640, поддерживающего сжатые данные 8-битного формата.

Итак, этот пример основан на плате STM324x9I-EVAL (REV B), в которую встроен датчик CMOS OV2640 (MB1066).

Сжатые данные (JPEG) должны быть несжатыми, чтобы иметь данные YCbCr, и преобразованы в RGB для отображения, например, но эта реализация нацелена только на получение данных JPEG через DCMI и их сохранение в SDRAM.

Этот пример разработан на основе примера DCMI (SnapshotMode), предоставленного в прошивке STM32CubeF4, расположенной в Проекты\STM324x9I_EVAL\Примеры\DCMI\DCMI_SnapshotMode. Приведенный пример предназначен для захвата одного кадра RGB (разрешение QVGA) и его отображения на LCD-TFT со следующей конфигурацией:

- DCMI и I2C GPIO настраиваются, как описано в*Раздел 6.3.2: Общие примеры конфигурации*.
- Системные часы работают на частоте 180 МГц.
- Тактовая частота SDRAM работает на частоте 90 МГц.
- DCMI настроен на захват 8-битных данных при аппаратной синхронизации (несжатые данные).
- Модуль камеры сконфигурирован для вывода изображений данных RGB с разрешением QVGA.

На основе этого примера, чтобы иметь возможность захвата данных JPEG, пользователю необходимо изменить DCMI и конфигурацию модуля камеры.

Конфигурация DCMI

DCMI необходимо настроить для получения сжатых данных (JPEG), установив бит JPEG в регистре DCMI_CR. Чтобы установить этот бит, пользователь должен просто в stm324x9i_eval_camera.c в функцию "uint8_t BSP_CAMERA_Init(uint32_t Resolution)", где настраивается DCMI (эта функция вызывается в функции main() для настройки DCMI и модуля камеры), добавьте инструкцию, написанную в**смелый**ниже и сохраните предыдущую конфигурацию DCMI, как показано ниже:

```
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;
```

Конфигурация модуля камеры

Конфигурация регистров датчика CMOS (ov2640) должна быть вставлена в файл ov2640.c, как показано ниже:

константный символ без знака 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, 0x02}, 0x00},{0x6d, 0x80}, {0x6e, 0x00},{0x70, {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} };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} };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} };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} };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} };

Для модификации модуль камеры регистрируется, предыдущая таблица должна быть отправлена на камеру через I2C; В том же файле (ov2640.c) в функции "void ov2640_Init(uint16_t DeviceAddr, разрешение uint32_t)" заменить:

чехол CAMERA_R320x240:

```
{
    for(index=0; index<(sizeof(OV2640_QVGA)/2); index++) {

      CAMERA_IO_Write(DeviceAddr, OV2640_QVGA[индекс][0],
OV2640_QVGA[индекс][1]);
      КАМЕРА_Задержка (1);
    }
    ломать;
  }
```

по:

чехол CAMERA_R320x240:

```
{
```

```
for(index=0; index<(sizeof(OV2640_JPEG)/2); index++) {

  CAMERA_IO_Write(DeviceAddr, OV2640_JPEG[index][0],
OV2640_JPEG[index][1]);
  КАМЕРА_Задержка (1);
}
ломать;
}
```

7 Поддерживаемые устройства

Чтобы узнать, совместим ли датчик CMOS (модуль камеры) с DCMI, пользователь должен проверить следующие пункты в технических характеристиках датчика CMOS:

- параллельный интерфейс (8-, 10-, 12- или 14-битный)
- управляющие сигналы (VSYNC, HSYNC и PIXCLK)
- поддерживаемая частота тактовой частоты пикселей
- поддерживаемый вывод данных.

Существует широкий спектр модулей камер и датчиков CMOS, совместимых с STM32 DCMI. в *Таблица 12*, упоминаются некоторые модули камеры.

Табл. 12. Примеры поддерживаемых модулей камеры

КМОП-сенсор	Модуль камеры	Форматы	Параллельный интерфейс
OV9655	STM32F4DIS-CAM	– RGB – YCbCr	– 8-битный – 10-битный
OB7740	TD7740-ФБАК	– RGB – YCbCr	– 8-битный – 10-битный
MT9M001	АрдуКАМ	– RGB	– 8-битный – 10-битный
OV5642	АрдуКАМ 5 мегапикселей	– RGB – YCbCr	– 8-битный – 10-битный
MT9M111	КМОП-камера	– RGB – YCbCr	– 8-битный
MT9P031	HDCAM	– RGB	– 8-битный – 10-битный – 12-битный
OB3640	3 мегапикселя	– RGB – YCbCr – JPEG	– 8-битный – 10-битный

8 Вывод

Периферийное устройство DCMI представляет собой эффективный интерфейс для подключения модулей камеры к микроконтроллерам STM32, поддерживающим высокую скорость, высокое разрешение, различные форматы данных и ширину данных.

Вместе с разнообразными периферийными устройствами и интерфейсами, встроенными в микроконтроллеры STM32, и преимуществами интеллектуальной архитектуры STM32, DCMI можно использовать в больших и сложных приложениях обработки изображений.

В этом примечании по применению рассматривается периферийное устройство DCMI для микроконтроллеров STM32, и предоставляется вся необходимая информация для правильного использования DCMI и успешной реализации приложений, начиная с выбора совместимого модуля камеры и заканчивая подробными примерами реализации.

9 Лист регистраций изменений

Таблица 13. История изменений документа

Свидание	Редакция	Изменения
3 августа 2017 г.	1	Первый выпуск.