И невозможное станет возможным. RGB интерфейс, реализованный на микроконтроллере ценой 1 доллар.

1. Методы управления LCD TFT индикатором.

Управление LCD TFT – это задача, которая потребует большой производительной мощности микроконтроллера. Однако разработчик приложения, успешно освоивший не очень мощный микроконтроллер, вполне может управлять LCD TFT через плату посредника, которой комплектуются LCD TFT непосредственно на заводе-изготовителе ЖК индикаторов. Плата-посредник – это изделие, собранное на базе мощного контроллера (как правило, с ядром ARM9 и мощнее) с оперативной памятью, которой достаточно для хранения нескольких кадров изображения. Данное изделие избавляет разработчика от множества вопросов вывода изображения на LCD TFT: вертикальная синхронизация формируемой развертки, горизонтальная синхронизация, сигнал синхронизации точки и одновременная выдача данных в RGB-интерфейс. Разработчик работает через определённый интерфейс с контекстом памяти, отображаемой на экране. Меняя контекст оперативной памяти платы-переходника можно сменить выводимое изображение. Требования к внешнему контроллеру – минимальны. Процесс вывода изображения на экран абсолютно не привязан к процессу формирования изображения в памяти. Низкая производительность управляющего контроллера практически не сказывается на качестве изображения (возможно снижение скорости прорисовки выводимых на экран объектов, но не частоты развёртки).

Упомянутое решение является классическим, самое главное, что дизайнеры приложений и по сей день голосуют за это решение рублём. Сколько стоит такое решение? Индикатор с RGB интерфейсом 5.7” 320*240*16 бит (с платой питания + сенсорная панель) – 1500 рублей, плата управления – мост I8080-RGB интерфейса – 1000 рублей, внешний контроллер – 30 рублей. Итого получается 2530 рублей за решение.

Компания Промэлектроника анонсирует своё решение, исключив из показанного выше дизайна мост I8080-RGB. Таким образом, стоимость решения отображения графической информации на экран 5,7” 320*240*16 снизится до 1530 рублей.

Итак, контроллер за 30 рублей должен формировать изображение и построчно выдавать его на экран. Специалисты, которые занимались вопросами отображения информации на графическом экране могут возразить: «Это не реально». Однако дизайн уже существует и работает на практике. Лишний повод дочитать этот текст ;)

2. Вопросы памяти.

В числе первых аргументов того, что невозможно сформировать изображение, а потом выдать его на экран – отсутствие ОЗУ, достаточной для хранения изображения кадра в контроллере за 30 рублей. Давайте вычислим требуемый размер ОЗУ, для хранения картинки размером 320*240*2, где 320*240 – разрешение в пикселях, 2 – глубина цвета (65536 цветов). Перемножив эти цифры получим 153600 байт. Память SRAM такого объёма будет стоить дороже 100 рублей. Что уж говорить о микроконтроллере, у которого на кристалле будет размещена память ОЗУ такого объёма.

На самом деле, для построения изображения, которое будет выдаваться на экран вполне достаточно двух строк. Первая уже построенная строка – выдаётся на экран по RGB интерфейсу, вторая – параллельно этому процессу строится. Далее строки меняются местами. Процессы идут параллельно. Таким образом, даже если выводится статическая картинка на экран – процессор работает так же, как если бы картинка динамически менялась, т.к. в процессе вывода изображения на экран каждая его строка заново формируется. Данный метод не является классическим на текущий момент времени,

однако вопросы требуемого объёма памяти мы уже сняли. Итак, нам надо хранить в памяти всего 2 строки. Значит требуемый объём при 16-битном цвете – 320*2*2 = 1280 байт.

3. Производительность микроконтроллера.

Вообще говоря, есть мнение, что для формирования изображения требуется высокая производительность микроконтроллера. По факту оказывается, что это требование является достаточным, но не необходимым. Даже если элементарно перемножить разрешение экрана на количество обновлений в секунду, допустим, 10 кадров, получим частоту, с которой должна поступать информация о точке всего 320*240*10 = 768кГц. Если взять какой-нибудь 8-ми битный контроллер, например, ATmega16, работающий на частоте 16МГц, то теоретически он сможет выполнить данную задачу. На практике, скорее всего частота обновления упадёт до единиц герц из-за неожиданных нюансов, которые возникают при более глубоком изучении предмета, при этом вряд ли данный контроллер сможет выполнять хоть сколько-то значимую работу. Программу однозначно придётся писать на ассемблере.

Другое дело, контроллер может не иметь очень производительного ядра, но взамен этому он обязан иметь производительную и гибко настраиваемую периферию. Это, пожалуй, необходимое условие. Данному условию удовлетворяет STM32F100C4T6 – самый младший на данный момент в линейке 32-битных контроллеров в линейке STMicroelectronics. Цена такого контроллера – 30 рублей у официальных дистрибуторов STM в России. Именно на этом микроконтроллере компания Промэлектроника анонсирует своё решение данной задачи.

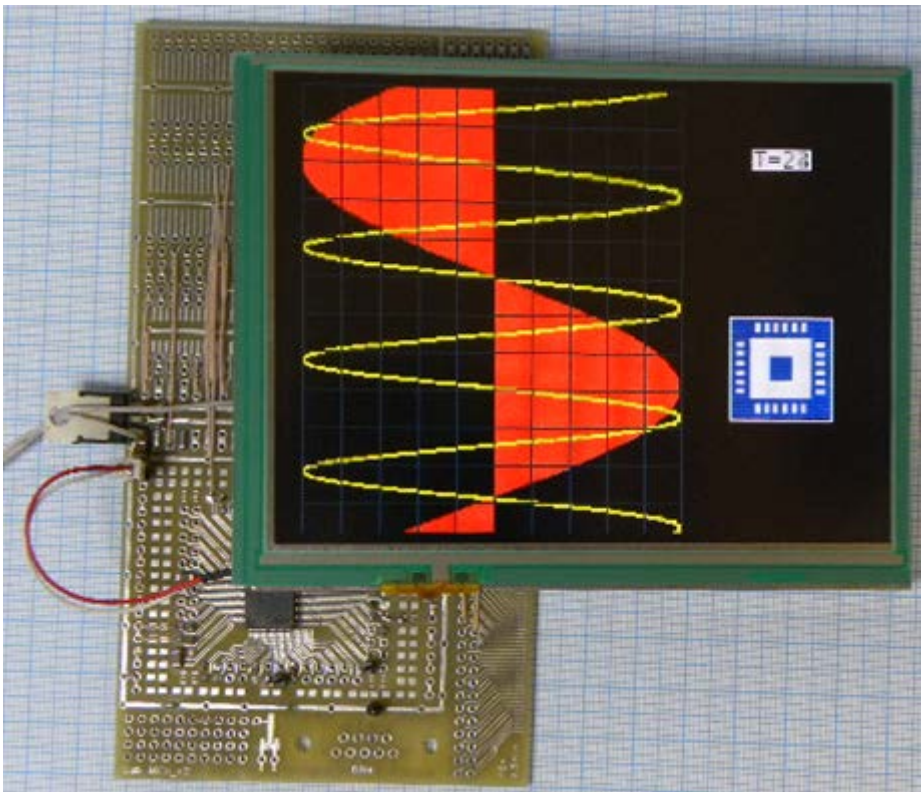

Рисунок. 1. RGB интерфейс на базе контроллера STM32F100C4T6.

Чем же, кроме цены, так хорош STM32F100C4T6? Если смотреть описание, то ничего сильно выдающегося можно не заметить:

- ядро ARM-Cortex M3 с частотой работы не более 24МГц
- Память ОЗУ – 4 кб

- Память Флэш – 16кб
- 6 таймеров (не считая Sys tick)
- 7 каналов обмена DMA
- несколько интерфейсов (SPI, I2C, USART, CEC)
- АЦП (10 каналов)
- ЦАП (2 канала)
- корпус LQFP48

По большому счёту весьма и весьма обычный контроллер по современным меркам. Однако, его уникальные свойства скрыты от тех кто поверхностно смотрит описания. Есть аналогичные по цене микроконтроллеры на базе ядра Cortex M0 других производителей, которые не обладают такими свойствами. Это обстоятельство лишает возможности элегантного решения задачи вывода изображения по RGB интерфейсу.

4. Замечательные свойства STM32.

Красивое решение поставленной задачи стало возможным благодаря двум свойствам микроконтроллеров STM32: возможность организации таймеров в сложные структуры и контроллеру DMA.

Итак, возможность организации таймеров в сложные структуры. Таймер общего назначения имеет вид, как показано на рис.2. На существующий момент любой контроллер STM32 имеет как минимум 2 таких таймера. Кроме того, в каждом микроконтроллере STM32 имеется таймер с расширенными возможностями: добавлен интерфейс определения положения вала двигателя, комплементарные выходы сравнения и внешний сигнал торможения двигателя. В нашем случае, таймер расширенных возможностей можно использовать как таймер общего назначения.

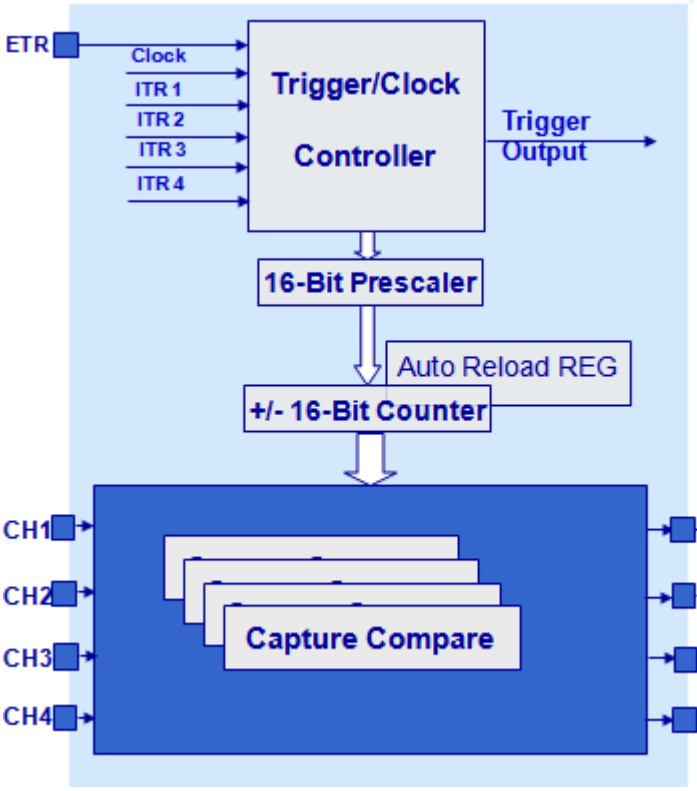

Рисунок. 2. Таймер общего назначения в STM32F1xx контроллере.

Блок-схема таймера общего назначения выглядит достаточно стандартно: Есть источник тактирования, который может быть внешним, есть делитель частоты счётного сигнала, есть 16-битный счётчик с регистром перезагрузки/сброса, 4 регистра сравнения/захвата. Мы не будем подробно останавливаться на свойствах таймеров, которые характерны для подавляющего большинства таймеров в контроллерах других

фирм-производителей: всевозможное направление счёта, выдача/захват ШИМ сигналов, однократная работа, счёт внешних/внутренних импульсов и т.д. Отметим, что все стандартные возможности в таймерах контроллеров STM32 представлены. Возможности, которых нет, в большинстве других контроллерах показаны на рисунке 2 надписями ITR1 – ITR4, а так же блоком «Timer/Clock controller». ITR – это внутренний триггер, который может быть не только счётным, но и управляющим. Управляющим – это значит, выходом, который может запустить таймер, сбросить таймер, открыть окно для счёта импульсов, закрыть окно счёта. Что генерирует сигнал ITR? Ответ – другой таймер. Это позволяет, например, жёстко синхронизировать несколько таймеров:

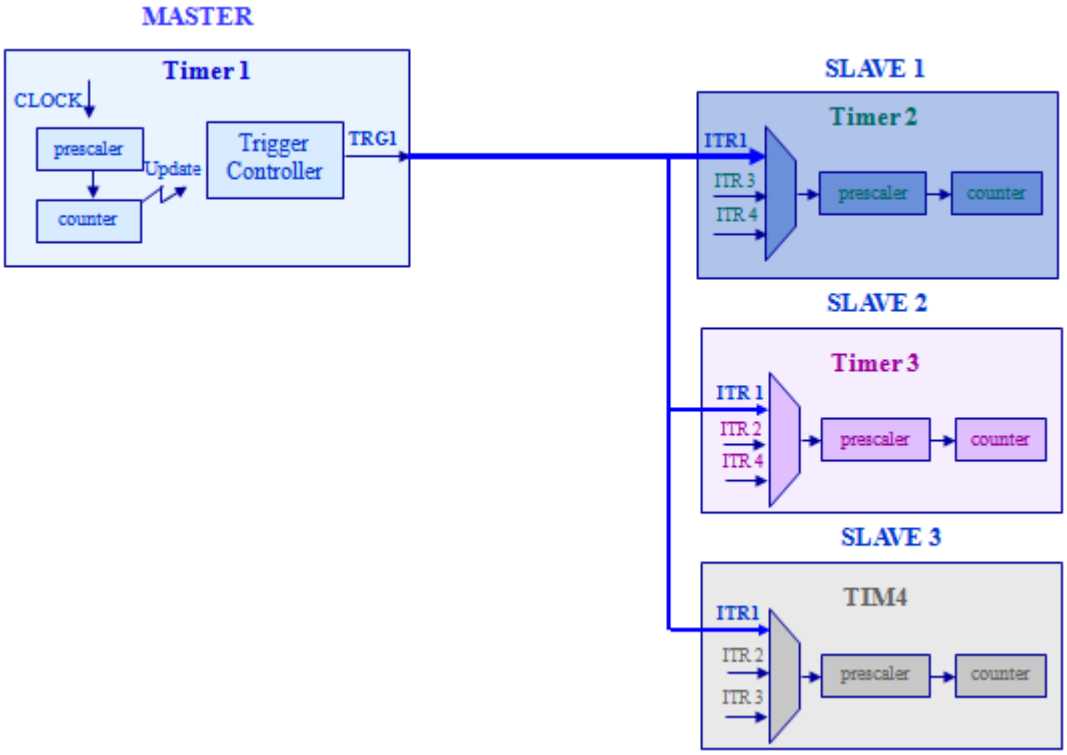

Рисунок. 3. Синхронизация нескольких таймеров

Таймеры 2, 3 и 4 настроены на запуск при срабатывании ITR1. Событие ITR1 возникает при переполнении Таймера 1. Таким образом, произойдёт одновременный запуск нескольких таймеров. Используя структуру, показанную на рисунке 3, можно индивидуально настроить реакцию каждого таймера, допустим: таймер2 будет сбрасываться, таймер 3 – менять состояние счёта/ожидания на противоположное, а таймер 4 – будет считать количество переполнений таймера 1. Событием, которое сформирует сигнал ITR может быть переполнение либо срабатывание одного из регистров сравнения.

Можно собрать из нескольких таймеров один, разрядность которого выше 16-ти. Например, 48-ми битный таймер:

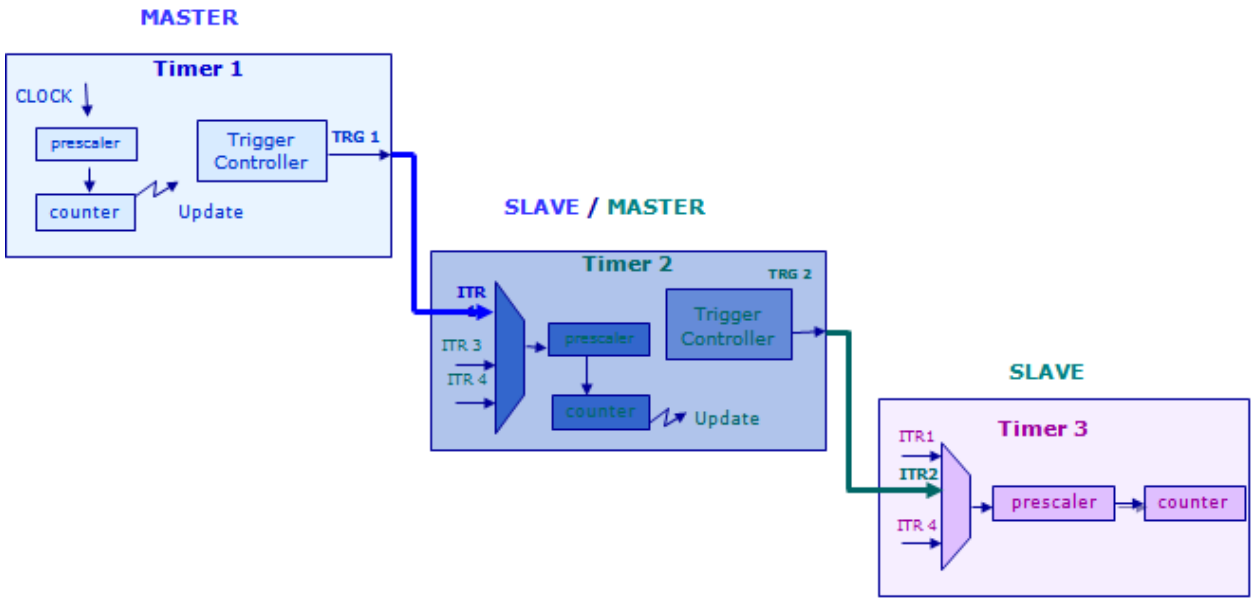

Рисунок 4. 48-битный таймер

Каждый из таймеров генерирует счётное событие для другого таймера при переполнении. Таким образом, из трёх 16-ти битных таймеров можно создать один – 48-битный. Эту же структуру можно использовать и для других целей: например, таймер 1 может предоставить окно счёта импульсов для 32-битного таймера, образованного таймерами 2 и 3. В данном случае изменится настройка таймера 2: именно он принимает решение о том, как использовать сигнал ITR1.

Автору неизвестен контроллер, который мог бы так гибко настраивать систему таймеров. Именно это свойство легло в основу проекта. Кроме этого, важной для проекта оказалась генерация прерываний при работе таймера и инициализация обмена по каналу DMA при переполнении таймера.

Второй периферией, которая легла в основу проекта, оказался контроллер прямого доступа к памяти. Именно с его помощью при переполнении одного из таймеров происходило копирование данных цвета рисуемой на экране точки. Копирование происходило из памяти в порт. Кроме того при построении строки, требовались копирования массивов из одной области памяти в другую. Эту работу так же выполнял контроллер DMA. Ядро всего лишь настраивало DMA на копирование новой области памяти при попадании в прерывание «Копирование по такому-то каналу завершено».

Несколько слов о возможностях контроллера прямого доступа к памяти в простейших микроконтроллерах STM32:

- 7 каналов обмена
- 5 уровней приоритета обмена
- возможность копирования память-память, память-периферия, периферия-память
- копирование 8-бит, 16-бит и 32-бит данных
- управляемый инкремент адреса источника/приёмника данных
- 3 прерывания: «копирование завершено», «половина копирования завершено», «ошибка копирования»
- копирование DMA инициируется программно и почти любой периферией контроллера (таймеры, АЦП, ЦАП, интерфейсы и т.д.)

Таким образом, мощная периферия – основа решения данной задачи.

5. RGB интерфейс.

До решения точно сформулируем задачу. Точную постановку задачи можно взять непосредственно из описания на индикатор. Итак, для реализации горизонтальной развёртки изображения потребуются сигналы (рисунок 5):

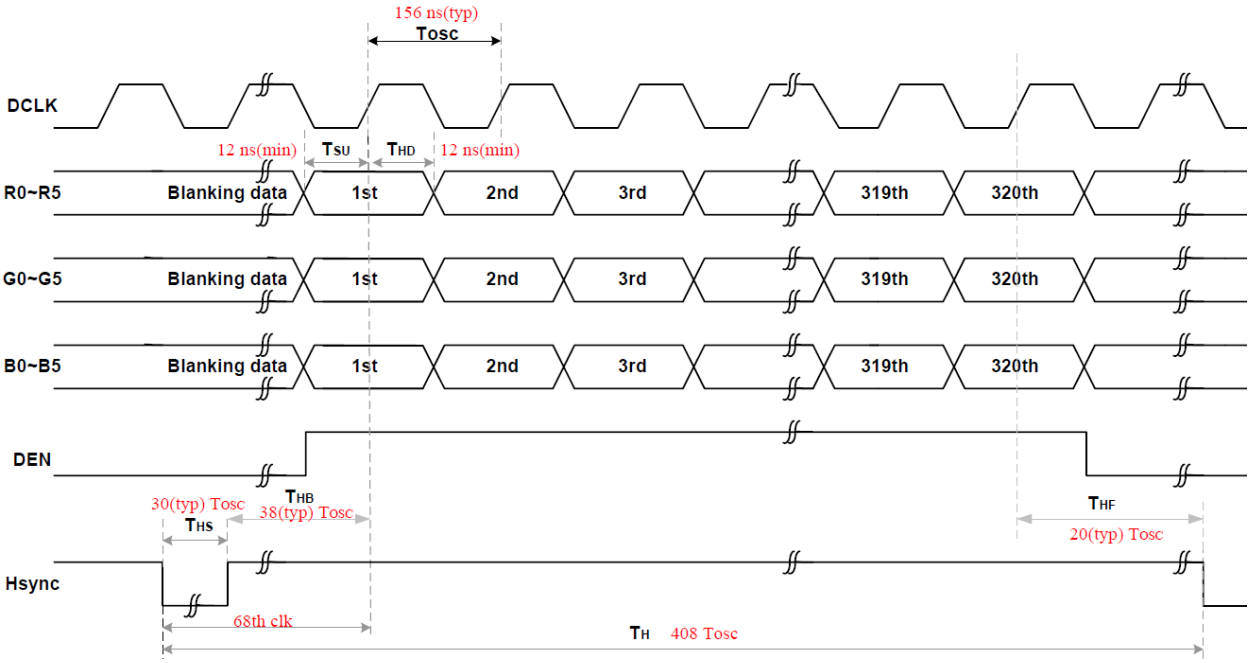

Рисунок 5. Сигналы горизонтальной развёртки

Начало запуска прорисовки новой строки формируется переходом уровня сигнала Hsync из логической «1» в «0». При этом сигнал DCLK, по переднему фронту которого индикатор считывает данные по цвету точки с линий R0-R5, G0-G5 и B0-B5, выдаётся постоянно. Однако, данные по цвету прорисовываемой точки в строке воспринимаются индикатором при условии, что линия DEN находится в состоянии логической «1». Момент появления и длительность сигнала DEN, как и Hsync стандартизован (должно пройти определённое число импульсов DCLK). Для прорисовки строки из 320 точек требуется 408 тактов DCLK.

Начало прорисовки кадра производится переходом уровня сигнала Vsync из логической «1» в «0» (рисунок 6).

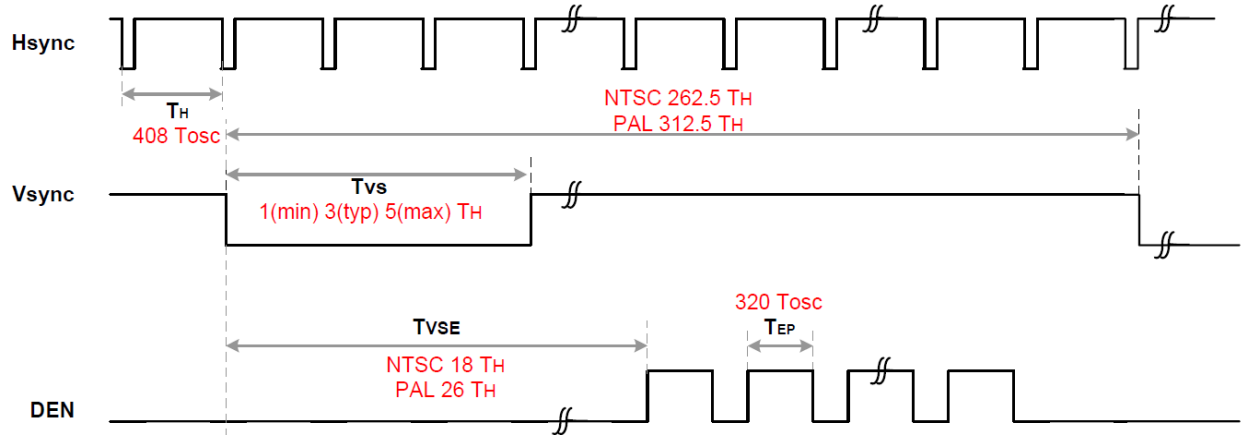

Рисунок 6. Вертикальная синхронизация.

Отметим, что на время кадровой синхронизации приостанавливается генерация выдачи сигнала DEN. Длительность сигнала вертикальной синхронизации так же нормируется и измеряется в длительностях прорисовки строки. До начала прорисовки изображения согласно документации индикатора должно быть выдано 18 – 26 Hsync (строк, не содержащих никакой информации изображения). Забегая вперёд, отмечу, что в реализованном проекте две таких строки.

Итак, задача: формирование сигналов Hsync, Vsync, DEN и DCLK с синхронно выдаваемыми данными по цвету рисуемой точки R0-R5, G0-G5, B0-B5 из буфера рисуемой строки. Параллельно этому процессу должен идти процесс формирования каждой строки в одном из буферов, подлежащей отображению.

6. Решение задачи.

Настроим таймеры (Рисунок 7).

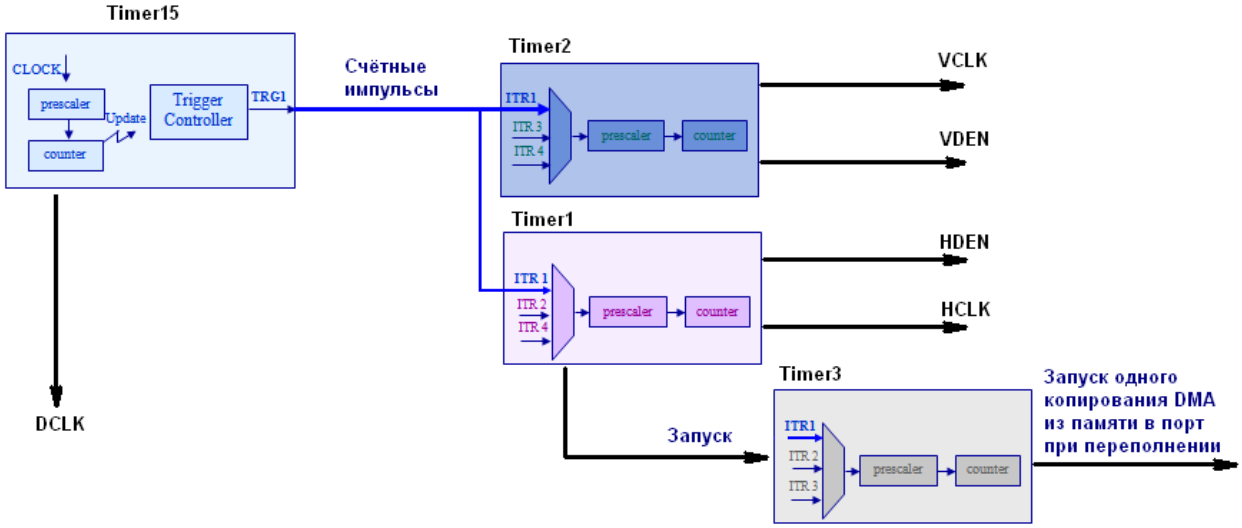

Рисунок 7. Структура таймеров, решающая задачу выдачи данных на RGB интерфейс.

Именно такая структура применена автором при построении RGB интерфейса. Автор уверен, что данная структура таймеров не оптимальна, однако её работа проверена на практике, поэтому будем описывать именно её. Таймер 15 задаёт временную базу для работы всей системы, а так же постоянно выдаёт сигнал DCLK в индикатор. Если мы останавливаем таймер 15, то останавливаются все процессы в системе (в части формирования экранной развёртки). Это очень удобно: при входе в прерывание, при необходимости, можно одной командой остановить процесс. Таймер 2, по факту тактируемый сигналом DCLK формирует сигнал вертикальной развёртки VCLK и сигнал VDEN (как показано выше в структуре таймера общего назначения, возможно до 4-х выходов сравнения = 4 формируемых ШИМ сигнала). Сигналы VDEN и HDEN – должны объединиться на внешней логической микросхеме «2И» в сигнал DEN. В этом случае можно будет исключить появление сигнала DEN во время вертикальной синхронизации. Timer1 формирует сигналы HDEN и HCLK. Кроме того, синхронно с выдачей переднего фронта HDEN, запускается Timer3, который имеет одинаковую частоту с Timer15. При переполнении Timer3 – формируется запрос контроллеру прямого доступа к памяти на копирование данных цвета выдаваемого пикселя. Таким образом, данные R1-R5, G0-G5, B1-B5 попадают в порт контроллера.

После 320 копирований DMA выдаёт прерывание «копирование завершено». В теле этого прерывание происходит останов Timer3, а так же настройка DMA на следующие 320 копирований из следующего буфера.

После переполнения Timer1, ответственного за формирование горизонтальной развёртки происходит проверка готовности формируемой строки к выдаче в RGB интерфейс. Если такая готовность есть, то включается Timer15, отключенный при входе в прерывание, что автоматически запускает процесс выдачи видеоинформации в следующую строку экрана. Если следующая строка ещё не готова, то Timer15 будет запущен сразу по окончанию формирования следующей строки.

По окончанию прорисовки кадра формируется прерывание по переполнению Timer2. В нём происходит обновление видеоинформации в целом: изменение положения, размеров, цвета, области памяти расположения объектов и т.д. Словом, если должна произойти смена вида картинки при смене кадра, то это лучше сделать именно тут.

Параллельно отображению одной строки идёт процесс построения следующей. Оба процесса выполняются периферией при минимальном участии ядра. Для примера, допустим, идёт отображение белого квадрата на чёрном фоне. Сначала мы настраиваем DMA на копирование видеоинформации заднего плана. Т.е., в нашем случае, заполняем буфер формируемой строки нулями. Когда DMA закончит копирование возникнет

прерывание «копирование DMA такого-то канала завершено». Далее, если в будущей строке должен отобразиться белый квадрат, будет запущен процесс копирования единиц в буфер строящейся строки. По окончанию копирования будет проверка возможности построения строки в новом буфере и, в случае необходимости, запущен Timer15.

Итак, силами периферии мы построили 2 процесса: формирование изображения будущей строки и выдача изображения в RGB интерфейс. Ядро принимает участие исключительно в прерываниях.

7. Параметры получившегося проекта.

Автору удалось добиться стабильной синхронизации и выдачи изображения на экран. Качество изображения при этом оказалось более чем достаточным для построения серийных приборов на базе этого приложения. Итак

Объём кода: 15кб, в том числе объём картинок – 8кб.
Язык программирования – Си
Параметры дисплея: 320*240 с глубиной цвета 16 бит
Частота обновления: 10 кадров/сек
Количество отображаемых объектов – 9 (см. Рисунок 1):

- фон
- красный график (динамически меняется)
- жёлтый график (динамически меняется)
- сетка вертикальная
- сетка горизонтальная
- логотип «Промэлектроника»
- надпись «T=»
- старший разряд цифровых показаний (динамически меняется)
- младший разряд цифровых показаний (динамически меняется)

Загрузка ядра процессора – не измерялось (ожидается 50%)
Используется таймеров – 4 из 7 (7 – включая SysTick)
Используемое количество каналов обмена DMA – 2 из 7
Используется одна микросхема внешней логики «2ИЛИ-НЕ»

Безусловно получившийся проект является базой для построения законченных приложений. Однако, по мнению автора, архитектура проекта не является оптимальной. Например, как оказалось, можно было бы отказаться от микросхемы внешней логики, которая предотвращала появление сигнала DEN во время процедуры вертикальной синхронизации. В программе оказалось востребованным определение момента окончания процедуры вертикальной синхронизации, соответственно в этот момент разрешать выдачу DEN сигнала, запрещать – по окончанию прорисовки экрана. Вполне разумным представляется отказ от 16-ти битного цвета в пользу 8-ми битного. Во-первых, это сэкономит память, во-вторых, почти в 2 раза ускорит внутренний обмен данных изображения при построении строки.

Почему в проекте реализована частота обновления всего 10 герц? Изначально расчёт показывал возможность обновления изображения указанного экрана более 30Гц. Проблема оказалась в скорости реакции DMA на запрос копирования. Данные по цвету рисуемой точки не успевали за тактами DCLK. Тем не менее, частота 30 кадров в секунду вполне достижима, если применить контроллер STM32F103C4T6, который полностью совместим по выводам с STM32F100C4T6, но имеет тактовую частоту в 3 раза выше (72МГц), и Timer4 который потребуется использовать вместо Timer15. Цена такого контроллера примерно в 2.5 раза выше, чем у применённого в проекте.

В заключение, хочется отметить, что на сегодняшний момент контроллеры фирмы STMicroelectronics одни из лучших по сочетанию цена - возможности. Софт, созданный компанией Промэлектроника – тому подтверждение.

Более подробную информацию можно получить обратившись в техническую поддержку компании Промэлектроника support@promelec.ru

Обсудить статью можно на форуме компании Промэлектроника в разделе «Обсуждение микроконтроллеров и средств отладки» тут: http://forum.promelec.ru/index.php/board,8.0.html