Построение обмена по сети Ethernet на контроллерах STM32F107.

1. Почему так популярны стандарты связи?

Надо признать, что мы живём в эпоху глобального разделения труда. Одни организации производят решения в железе, другие пишут программное обеспечение, третьи – создают стандарты, удовлетворяющие самым последним достижениям науки и техники. Представим себе ситуацию, когда требуется организовать систему сбора данных по определённому алгоритму. В этом случае, безусловно, лучше обратиться к поиску подходящего существующего стандарта связи, чем выдумывать свой (даже если задача весьма специфична). Причина – проста. Над разработкой стандарта связи работали целые институты, группы разработчиков, которые всю жизнь занимались исключительно связью. Во всех стандартах решены технические задачи, о которых рядовой разработчик может даже не догадываться. Например, электро-магнитная совместимость (ЭМС). Насколько будет устойчив канал связи к нано или микросекундным помехам? В конечном счёте, окажется ли пригоден такой канал связи для решения предложенной задачи? В случае построения канала связи на базе собственных алгоритмов на эти и другие вопросы можно будет ответить исключительно после построения всей системы. При этом, если в результате испытаний канал связи окажется непригодным для решения задач связи, то не факт, что его можно будет доработать. Зато всех этих трудностей не возникнет при использовании готового стандарта связи. Таким образом, даже рядовой разработчик сможет решить проблему связи двух и более объектов, интегрировав определённый стандарт связи в своё приложение.

Чем должен руководствоваться разработчик при выборе стандарта связи? В первую очередь, стандарт связи определяет пропускную способность и дальность передачи информации. Эти два параметра практически определяют помехоустойчивость канала связи. Тем не менее, помехоустойчивость определяется так же скрытыми от пользователя параметрами (на физическом уровне), поэтому при выборе стандарта передачи данных стоит обратить внимание на параметр помехоустойчивости.

Кроме решения задачи связи перед разработчиком предстанет множество электронных компонент, удовлетворяющих выбранному стандарту от множества производителей элементной базы. Речь идёт не только о стандартных разъёмах и драйверах физического уровня, но и о существующих функционально законченных устройствах, которые отдельно продаются на рынке и имеют крайне низкую цену из-за огромных объёмов производства. Если говорить о Ethernet, то таковыми устройствами являются хабы, роутеры, персональные компьютеры и т.д. Кроме аппаратных решений в приложение можно будет интегрировать и программные продукты. Это могут быть платные, а могут быть и бесплатные решения. Важно в данной ситуации только то, что эти решения соответствуют определённому стандарту и это позволило создать их вне различных приложений. Выбрав определённый стандарт связи, вы легко интегрируете всю его периферию в своё приложение.

Обратная сторона медали. Для того чтобы освоить новый стандарт связи надо затратить примерно столько же усилий, как на то чтобы освоить новый контроллер. Производство этого не любит. Существуют проверенные, отработанные решения. Уже хорошо изучены на практике их сильные и слабые стороны. Хочется их тиражировать не только на уровне производства, но и на уровне разработчика. Безусловно, освоение нового стандарта связи потребует определённых усилий. Важно во время освоения нового стандарта связи, помнить о тех выгодах, которые он принесёт, о которых рассказано выше.

2. Что такое Ethernet?

Начнём с того, что Ethernet – это протокол последовательной передачи данных. Стандарт подразумевает передачу данных через коаксиальный кабель, витую пару и оптоволокно. Мы в дальнейшем будем говорить о передаче данных по витой паре. На существующий момент это самое массовое, а значит самое эффективное по сочетанию цена/возможности решение. Итак, у нас определён физический уровень. Это значит, что стандарт оговаривает тип среды, по которой происходит передача данных, тип разъёма, для подключения проводника уровни напряжений и вид модуляции, которым передаётся информация. Последние два параметра скрыты от пользователя. Т.е. не зная этих параметров можно интегрировать Ethernet в своё приложение. Поэтому мы их не будем рассматривать далее.

Кроме физического уровня стандарт Ethernet определяет уровень канальный. Т.е. определяет вид пакета данных. Как выглядит пакет данных (Рисунок 1).

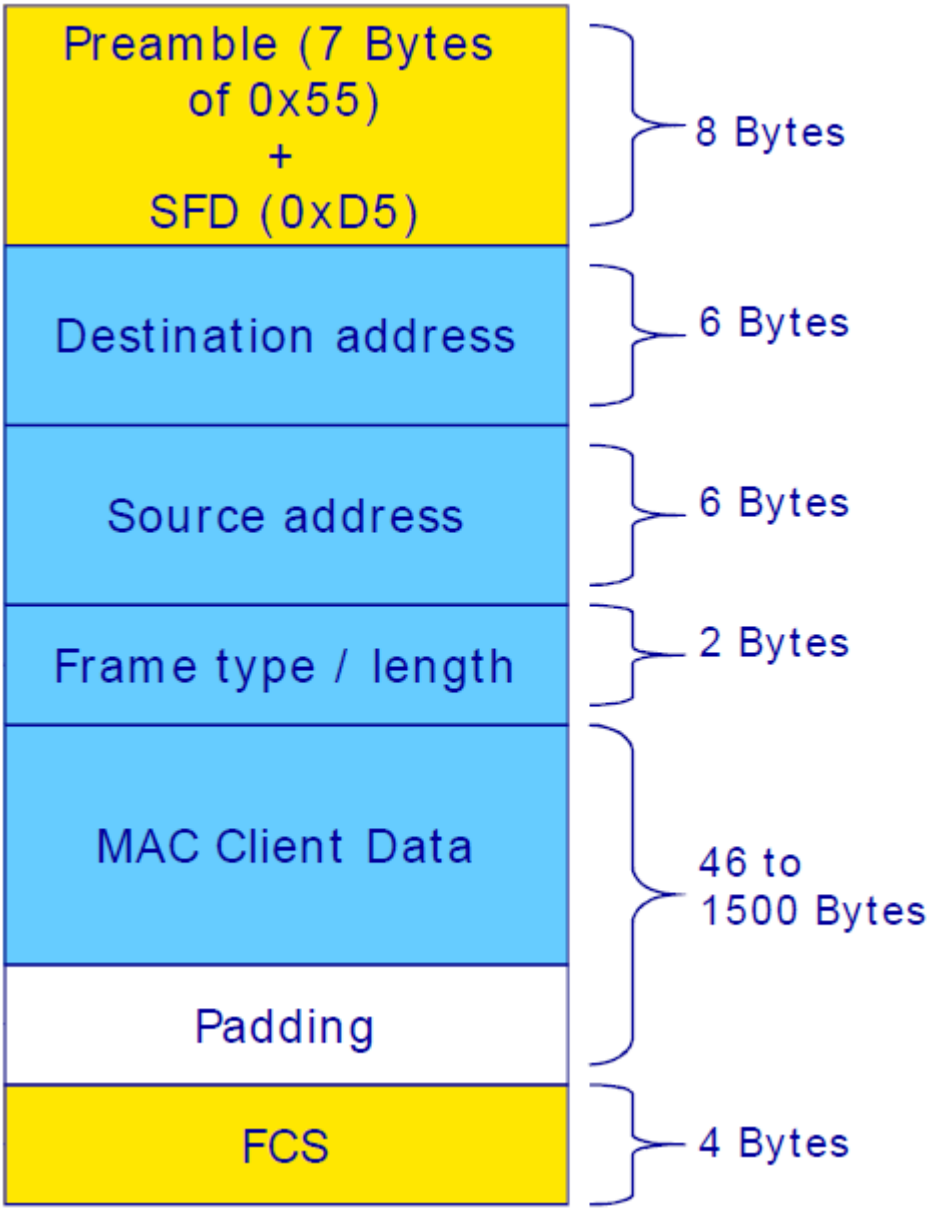


Рисунок 1. Формат пакета данных Ethernet

В начале трансляции пакета физический уровень генерирует преамбулу в виде набора чередующихся единиц и нулей. Данная операция требуется для синхронизации источника и приёмника сигнала. После окончания синхронизации генерируется команда начала пакета (SFD). Вся описанная процедура выполняется передачей восьми байт. Как правило, эту операцию выполняет физический уровень. Пользователь о ней может ничего не знать, при этом полноценно использовать Ethernet в своём устройстве. Зато пользователь совершенно точно должен будет указать адрес назначения пакета (Destination address) и

адрес источника пакета (Source address). И тот и другой являются MAC адресами устройств. Для чего нужны MAC адреса? Главным образом, для фильтрации сообщений. Так же организация IEEE, которая продаёт диапазоны MAC адресов, заносит покупателя в базу данных. Таким образом, по старшим трём байтам MAC адреса можно определить производителя оборудования. За адресами получателя и источника пакета следует 2 байта, указывающие длину транслируемого пакета. Если число, указывающее длину последующих данных больше 1500, то данное поле будет указывать тип пакета, длина которого формально может иметь любой размер. Длина пакета имеет минимальный размер 46 байт. С чем это связано? Главным образом с возможностью 100% детекции коллизий. Что произойдёт, если сразу 2 устройства сети начнут транслировать данные? Проверяя физическое состояние линии во время трансляции оба устройства поймут, что передаваемый сигнал не соответствует тому, что физически присутствует на линии. Устройства обнаружили коллизию. По определённому алгоритму устройства сети заново отправят транслируемые пакеты, не мешая друг другу. Что может произойти, если пакет данных короткий, расстояние между отправителями информации достаточно большое. Транслируемый сигнал достигает другое устройство, транслирующее свой пакет, с задержкой. Если пакеты будут короткими, то устройства окончат трансляцию и уйдут с линии и имевшую место коллизию не обнаружат. В результате потеряются пакеты с информацией и устройства, транслировавшие их не будут об этом знать. Именно поэтому пакет имеет минимальную длину 46 байт. Если мы хотим отправить данные длиной меньше чем 46 байт, то мы обязаны дополнить отправляемый пакет пустыми данными. Некоторые контроллеры, такие как STM32, выполняют эту операцию на аппаратном уровне.

Достоверность принятой информации гарантируется контрольной суммой (поле FCS), которая выдаётся в конце пакета. Контрольная сумма вычисляется по алгоритму CRC32. При отправке пакета требуется вычислить и добавить её значение в конце. При приёме – до работы с принятым пакетом требуется проверить её соответствие принятым данным. И ту, и другую операцию контроллер STM 32 выполняет аппаратно.

3. Надстройки Ethernet

Для построения локальной сети стандарта Ethernet вполне достаточно. Существует ряд стандартов, с помощью которых можно объединять локальные сети в глобальные и далее производить следующие надстройки (Рисунок 2). Можно данную задачу решить самостоятельно и создать собственный алгоритм логической адресации и надстроек более высокого уровня. Но по причинам, описанным в первой части, объективно выгодно интегрировать существующие стандарты в своё приложение. Например, для адресации локальных сетей применить IP - стандарт (либо другой, подходящий).

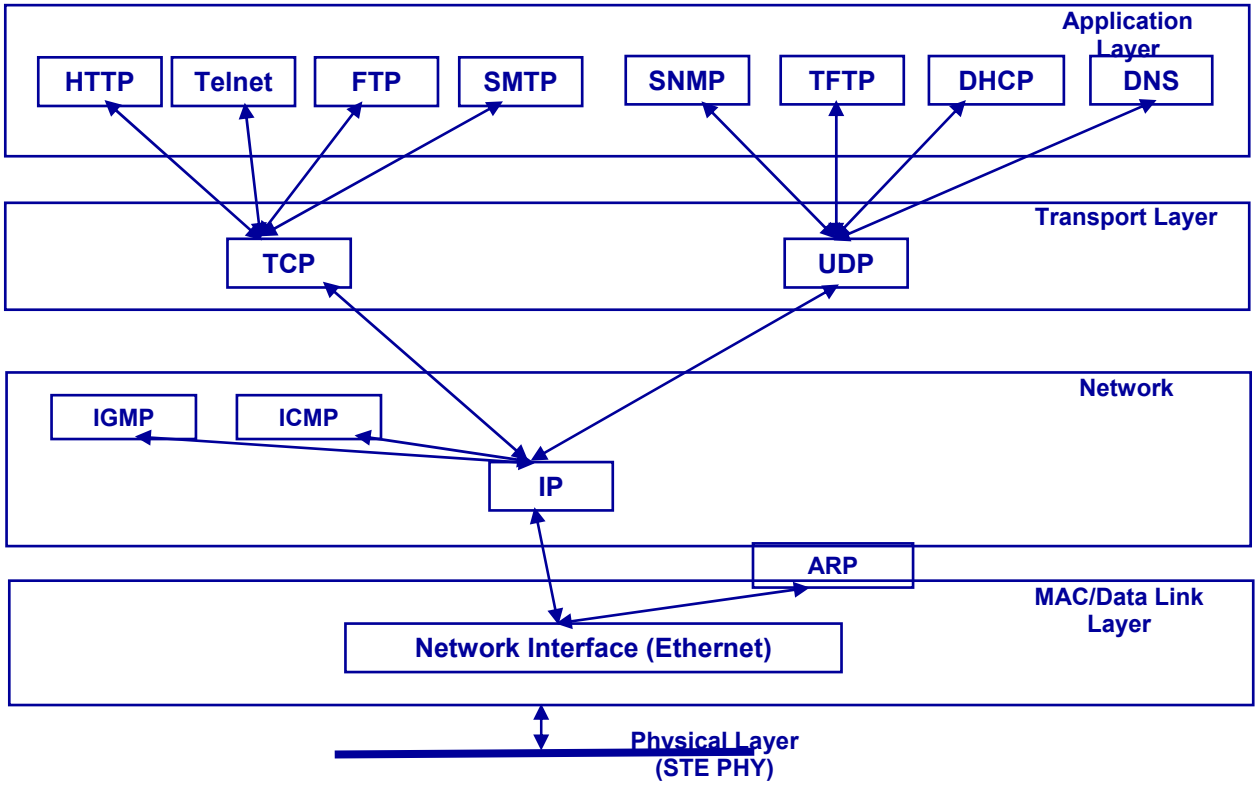


Рисунок 2. Стандартные надстройки Ethernet.

В любом случае, независимо от того каким образом мы захотим связывать локальные сети, нам придётся включить в свой пакет определённую служебную информацию, на основе которой будет производиться логическая адресация. Любая последующая надстройка будет занимать часть передаваемой информации. Выглядит это так:

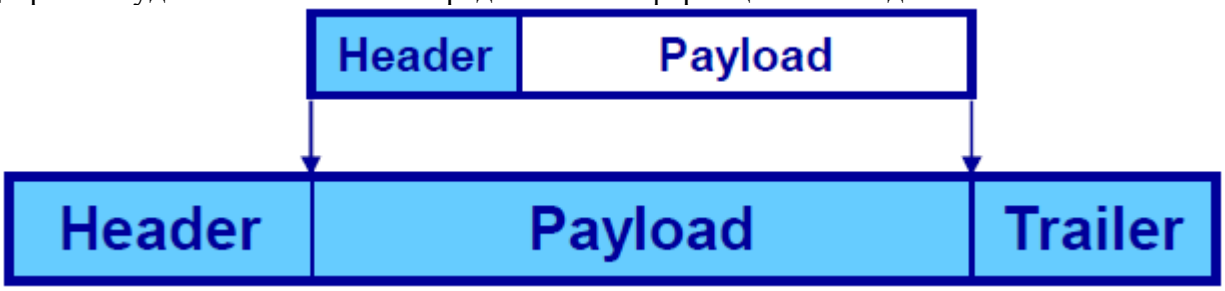


Рисунок 3. Добавление служебной информации в пакет данных

Итак, если мы добавляем к Ethernet IP-адресацию, то отправляемый/принимаемый пакет должен содержать информацию об IP адресах, а так же остальную служебную информацию данного протокола. Информацию о точной структуре пакета IP (или любого другого) протокола можно легко найти в интернете (например, тут: http://ru.wikipedia.org/wiki/IPv4), мы не будем уделять этому внимание. Аналогично этому механизму произойдёт добавление следующего транспортного уровня надстройки. Таким образом, структура передаваемого пакета будет выглядеть так:

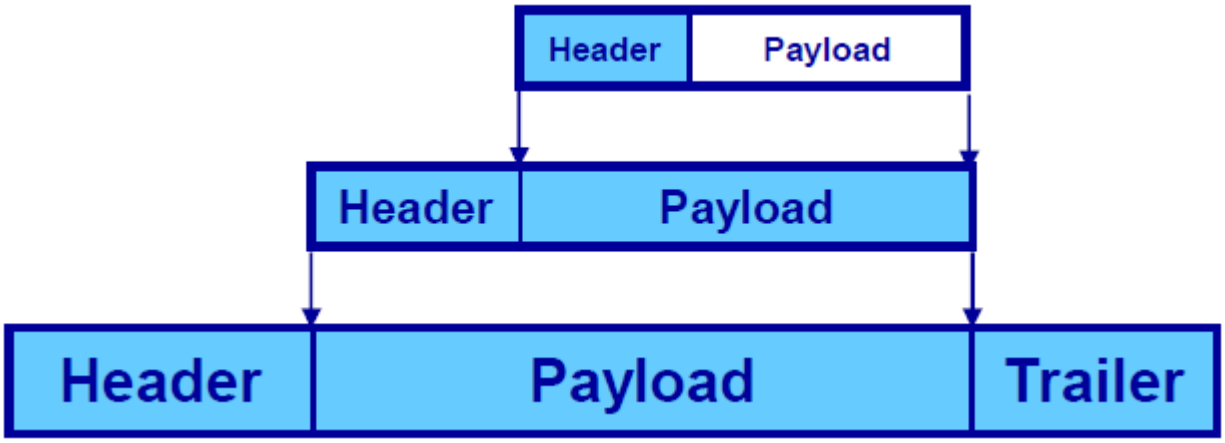


Рисунок 4. Структура передаваемого пакета, включающего несколько уровней надстройки.

Таким образом, разработчикам, избравшим Ethernet в качестве интерфейса соединяющего части системы в единую локальную сеть, открывается дорога по развитию своей системы и тесной интеграцией её с существующими решениями.

4. Ethernet в STM32

Микроконтроллеры STM32F107xx, STM32F207xx и STM32F217xx имеют в своём составе периферию Ethernet MAC. Основные свойства периферии:

- Полноценный MAC уровень с подключением к внешнему физическому
- Работа на скоростях 10 и 100 Мбит/сек
- Полу/полнодуплексные режимы работы
- Выделенный DMA контроллер с очередями приёма и передачи пакетов
- Поддержка привязки пакетов во времени
- Управление входом/выходом режимов низкого энергопотребления
- Интегрированный набор векторов прерываний

Рассмотрим блок-схему Ethernet MAC в контроллере STM32F107 (Рисунок 5).

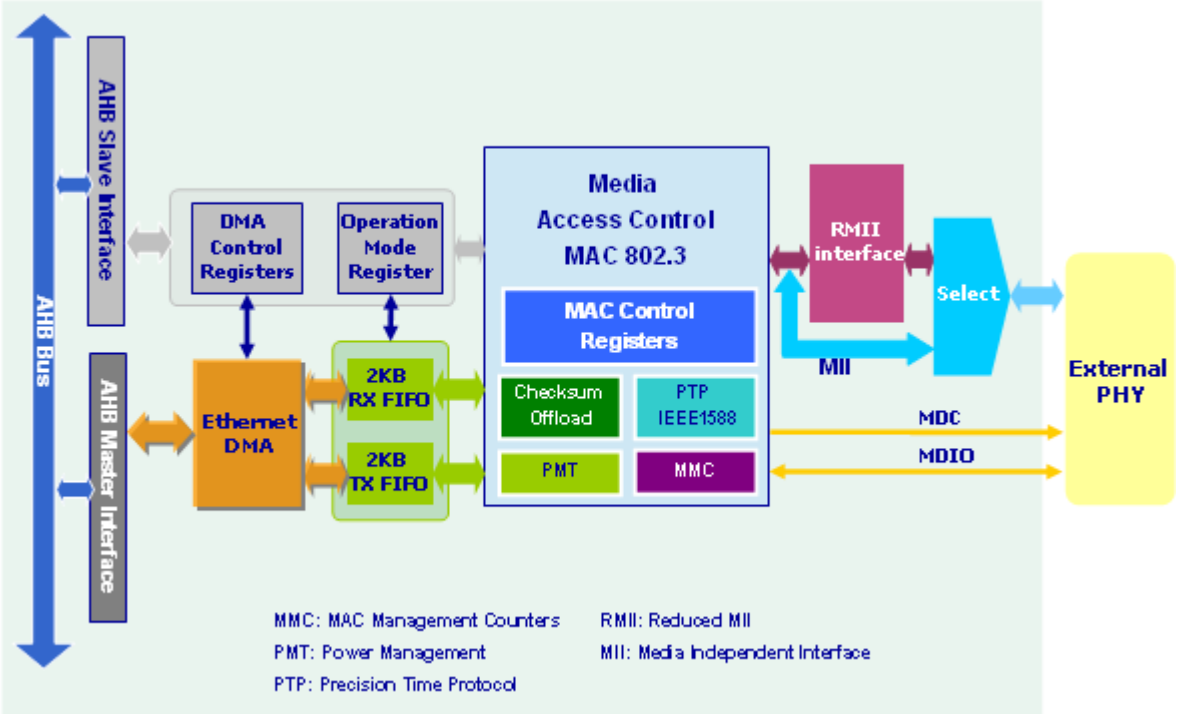


Рисунок 5. Блок-схема Ethernet MAC в контроллере STM32F107.

Всю блок-схему можно разделить на 3 части: интерфейс подключения Ethernet MAC внутренней шине AHB контроллера, непосредственно MAC 802.3 и интерфейс подключения к внешней микросхеме PHY уровня Ethernet.

Ethernet в STM32 устроен таким образом, что пользователь не должен создавать отправляемый пакет непосредственно внутри периферии. Формирование или приём пакета данных происходит в области памяти ОЗУ. Отправляются или принимаются данные в эти области посредством выделенного контроллера прямого доступа к памяти, который обслуживает только Ethernet и обладает специфичными свойствами, в отличие от остальных контроллеров DMA, представленных в STM32. Пользователь настраивает регистры DMA, а так же дескрипторы. Дескрипторы – это данные в ОЗУ, которые объясняют контроллеру прямого доступа к памяти, что делать с той или иной областью при работе Ethernet, т.е. это дополнительная служебная информация, используемая периферией Ethernet MAC, размещённая в ОЗУ. На основании контрольных регистров DMA и дескрипторов осуществляется копирование пакетов данных из ОЗУ в очередь FIFO при отправке и, наоборот, при приёме. Если при трансляции данные попали в очередь FIFO, то с некоторой задержкой произойдёт их отправка на физическом уровне. С этого момента пользователь может не заботиться о данной операции, она будет выполнена. Аналогично, приём. Если из очереди приёма данные попали в обозначенную область ОЗУ, то можно считать, что данные прошли все настроенные фильтры и в некоторых случаях проверку контрольной суммы, т.е. с полученными данными можно работать, процедура приёма окончена.

Контроллер MAC 802.3. Кроме непосредственных операций с потоком данных, управляемых контрольными регистрами выполняет ещё ряд функций. Данный модуль добавляет контрольную сумму CRC32 к отправляемым пакетам и проверяет её у принятых. Это происходит в случае, когда размер транслируемого пакета меньше размера очереди – 2кб. В случае несовпадения – принятый пакет удаляется из очереди. Таким образом, разработчик ПО может не задумываться о существовании проверки целостности принятого/отправляемого пакета данных, но пользоваться этим. Аппаратная поддержка контроллера выполнит это самостоятельно. Кроме этого, контроллер поддерживает стандарт PTP IEEE 1588. Этот стандарт позволяет синхронизировать работу сети во времени. Это может применяться, например, для одновременного измерения каких либо параметров в разных частях сети. Далее, пакеты с данными могут с различной задержкой доставляться в один из модулей для обработки и анализа. Синхронизация узлов сети позволяет рационально использовать её пропускную способность. Если, например, устройства договорятся кто в какой момент будет транслировать сообщение, то можно серьёзно снизить число коллизий на линии. Для того, чтобы производить синхронизацию, каждое из устройств сети должно иметь внутренние часы. Часы каждого из устройств будут показывать своё значение. Задача синхронизации заключается в том, чтобы разница показаний этих часов была минимальной. Стандарт PTP IEEE 1588, который поддерживается контроллерами STM32F107 позволяет синхронизовать внутренние часы отдельных узлов системы с точностью до 1мкс. Более поздний стандарт PTP IEEE 1588v2, поддерживаемый контроллерами STM32F207 теоретически позволяет синхронизировать узлы сети с точностью до наносекунд.

Интегрированный в STM32 контроллер MAC 802.3 имеет блок регистров статистики работы сети. Данный модуль аппаратно фиксирует количество успешно отправленных, принятых пакетов, а так же пакетов с ошибкой в контрольной сумме, количество пакетов отправленных с одной коллизией и количество пакетов отправленных с более чем одной коллизией. Приложение пользователя может получать данную статистику и как-то реагировать, например, уменьшить количество передаваемой информации.

По командам, получаемым из сети Ethernet, можно пробуждать контроллер. Эта функция возможна в STM32 благодаря блоку управления питания, интегрированного в MAC контроллер. Выйти из энергосберегающего режима можно получив пакет специального вида. Пакет, посредством которого контроллер выводится из

энергосберегающего режима, должен пройти все фильтры. В противном случае он не произведёт никаких действий.

Контроллер MAC 802.3 следит за состоянием очереди приёма данных из Ethernet. Что произойдёт, если очередь приёмных сообщений будет заполнена?

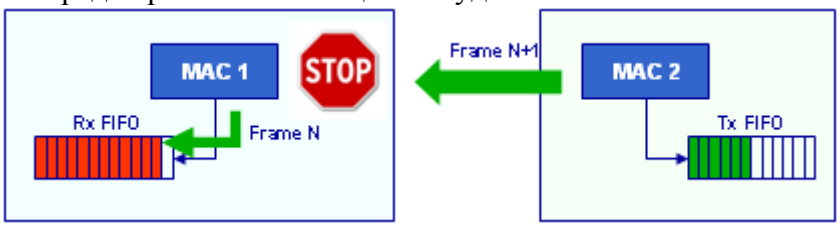


Рисунок 6. Передача сообщения, когда очередь приёмника заполнена.

Контроллер MAC 802.3 приёмного устройства на базе STM32 отправит множественную рассылку с адресом назначения 01-80-C2-00-00-01, в которую включит значение времени, по прошествии которого, можно будет возобновить трансляцию.

STM32 может иметь до четырёх MAC адресов. Нулевой MAC адрес у него есть всегда, по умолчанию. Остальные три, могут быть разрешены либо запрещены. Все MAC адреса можно использовать в качестве фильтров сообщений. Кроме фильтрации по точным адресам контроллер можно применять относительную фильтрацию, называемую hash-фильтрацией. Мы не будем подробно рассказывать, что это такое, однако отметим основное свойство данного фильтра, которое заключается в том, что можно фильтровать сообщения, предназначенные для групп получателей.

Внешний физический интерфейс Ethernet подключается контроллеру по определённому стандарту. Это может быть MII (Media Independent Interface) либо RMII (Reduced Media Independent Interface). Эти виды подключений отличаются друг от друга количеством линий и частотой работы. Контроллеры STM32 поддерживают оба интерфейса физического уровня. Ниже на рисунке 6 показаны оба вида подключений.

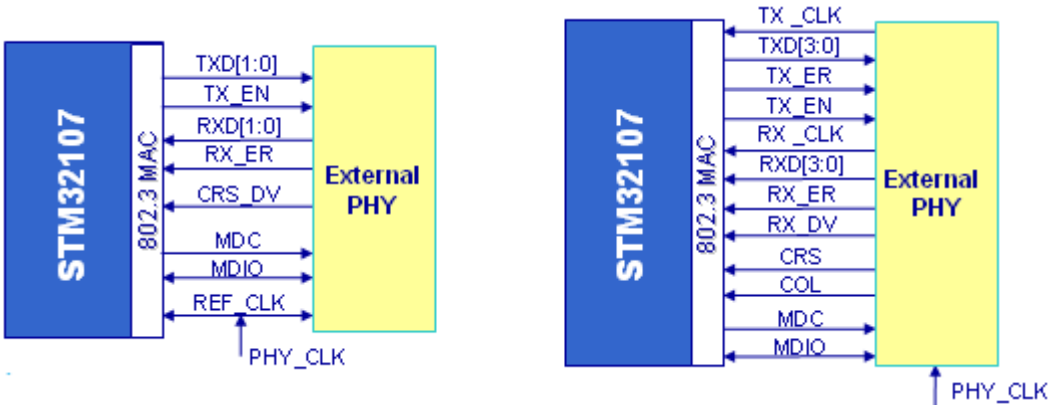


Рисунок 6. подключение по стандарту RMII (слева) и MII (справа)

Подключение по RMII требует всего 8 портов контроллера, в то время как для подключения физического уровня по стандарту MII требуется 16 портов контроллера. Если перевести количество свободных портов контроллера в стоимость, то можно предположить, что соединение по RMII более выгодно. Скорее всего так и есть, однако имеется ряд нюансов. Интерфейс RMII работает на частоте 50МГц, что вдвое выше, чем у MII, соответственно эмиссия помех такого решения будет выше. Более того для генерации частоты 50МГц и выдаче её на микросхему физического уровня будет задействован один из множителей частоты – рекомендуемое производителем решение по тактированию внешней микросхемы Ethernet PHY. И, наконец, контроллер STM32 может подключить до 32-х внешних микросхем физического уровня по интерфейсу MII и всего лишь одну – по интерфейсу RMII. В любом случае, у разработчика остаётся право выбора каким образом подключить внешнюю микросхему физического уровня Ethernet.

5. Дескрипторы

Прежде чем разбирать простейшее приложение на базе Ethernet в STM32 поговорим о дескрипторах. Дескрипторы объясняют контроллеру прямого доступа, связанного с Ethernet, что делать с той или иной областью памяти. Дескриптор – это 4 слова длиной 32-бит каждое, размещённые в ОЗУ. Вид дескриптора показан на рисунке 7.

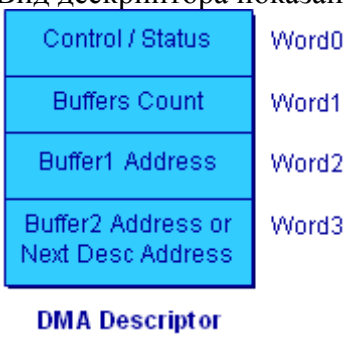


Рисунок 7. Дескриптор для DMA Ethernet

Каждый дескриптор содержит контрольное поле, описывающее ряд служебных параметров. Данное поле имеет различный вид для дескрипторов описывающих области памяти отправки и приёма сообщений. Тем не менее, есть ряд общих параметров:

- описываемая область памяти отдана в распоряжение DMA или приложению
- флаги ошибок приёма/передачи
- флаг некорректного описанного дескриптора
- признаки первого и последнего фрагмента разбитого на несколько частей сообщения.

По результатам отправки/приёма данных DMA видоизменяет контрольное поле дескриптора. Пользователь сможет увидеть результат действий.

Дескриптор так же имеет поле указывающее длину и адрес области памяти, подлежащей трансляции/приёму. Итак, есть контрольная информация, есть начало и длина копируемой области памяти ОЗУ, что ещё? STM32 поддерживает 2 вида дескрипторов (не путать с разделением дескрипторов на приём и трансляцию). Один вид дескрипторов описывает одну область памяти и указывает адрес следующего дескриптора. Другой вид дескрипторов описывает сразу две области памяти, а следующий дескриптор располагается последовательно в памяти за данным. Отличие значений одного вида от другого проявится в слове номер 2 и 4 в теле дескриптора. В зависимости от типа в четвёртом поле будет указатель на адрес следующего дескриптора или на адрес следующей области памяти, подлежащей приёму/отправке данных. Во втором будут указаны длины областей памяти, предназначенных для получения/трансляции данных, одна или две, опять же в зависимости от типа дескриптора. Значение каждого бита дескриптора подробно описано в точном описании RM0008 контроллеров STM32.

Работа DMA показана на рисунке 8.

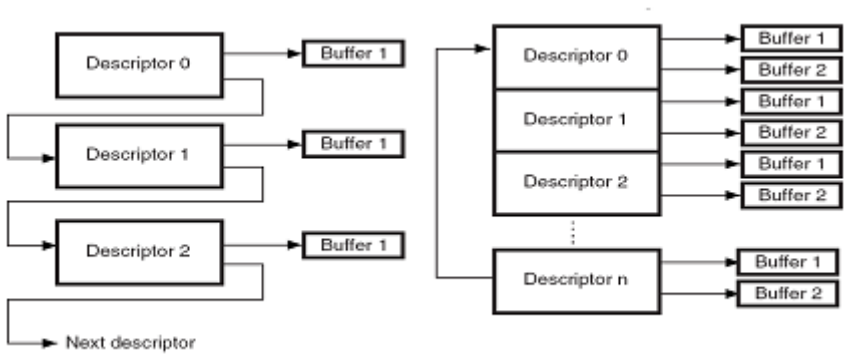


Рисунок 8. Работа DMA Ethernet с двумя видами дескрипторов и связанными с ними режимами: цепочечный (слева) и кольцевой (справа)

Итак, для трансляции данных по Ethernet мы указываем интегрированному в него контроллеру прямого доступа к памяти адрес первого дескриптора. Контроллер DMA находит его в памяти и определяет за кем в данный момент закреплена описываемая этим дескриптором область памяти: за ним или за приложением. Если за приложением, то DMA находит следующий дескриптор. Это происходит в зависимости от режима работы DMA (которому соответствует соответствующий вид дескриптора). В случае цепочечной обработки, адрес следующего дескриптора указан в одном из полей данного. В случае кольцевой обработки, следующий дескриптор расположен сразу за текущим. Если дескриптор закреплён за DMA, то производится работа с описанной им областью памяти: копирование в очередь трансляции по Ethernet или копирование очереди приёма. По результатам данной операции DMA сообщает результат своих действий в виде флагов в первом слове дескриптора. Так же контроллер прямого доступа к памяти закрепляет описываемый буфер за приложением.

В заключение отметим, знание работы DMA – ключ к применению Ethernet в контроллерах STM32.

6. Пример приложения с использованием Ethernet для контроллера STM32

Построим простое приложение, на базе которого будет происходить обмен данными по сети Ethernet. Мы не будем описывать настройки портов, тактирования периферии и прочие детали проекта, которые не имеют непосредственного отношения к приёму/передаче данных по сети Ethernet. Проект будет построен на базе прилагаемых стандартных библиотек для Ethernet.

До задания параметров Ethernet необходимо установить начальные значения. Выполнить это можно следующим образом:

```
// сброс настроек в начальное состояние Ethernet
  ETH_DeInit();

// сбрасываем Ethernet
  ETH_SoftwareReset();

// ждём подтверждения, что Ethernet сбросился
  while(ETH_GetSoftwareResetStatus()==SET);
```

Далее задать свойства сети:

```
// конфигурируем Ethernet
  ETH_StructInit(&ETH_InitStructure);

// Устанавливаем параметры ETH_InitStructure
  ETH_InitStructure.ETH_AutoNegotiation = ETH_AutoNegotiation_Enable  ;
  ETH_InitStructure.ETH_Speed = ETH_Speed_100M;
  ETH_InitStructure.ETH_LoopbackMode = ETH_LoopbackMode_Disable;
  ETH_InitStructure.ETH_Mode = ETH_Mode_FullDuplex;
  ETH_InitStructure.ETH_RetryTransmission = ETH_RetryTransmission_Disable;
  ETH_InitStructure.ETH_AutomaticPadCRCStrip = ETH_AutomaticPadCRCStrip_Disable;

//  Принимать всё подряд
     ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Enable;

  ETH_InitStructure.ETH_BroadcastFramesReception = ETH_BroadcastFramesReception_Enable;
  ETH_InitStructure.ETH_PromiscuousMode = ETH_PromiscuousMode_Disable;
  ETH_InitStructure.ETH_MulticastFramesFilter = ETH_MulticastFramesFilter_Perfect;
  ETH_InitStructure.ETH_UnicastFramesFilter = ETH_UnicastFramesFilter_Perfect;
// производим инициализацию параметров
  Value = ETH_Init(&ETH_InitStructure, PHY_ADDRESS);
```

Процедура ETH_Init настраивает не только внутренние регистры контроллера, но так же внешнюю микросхему PHY уровня. Далее должно быть выделено пространство буферов приёма, буферов передачи и для дескрипторов, описывающих их.

```
// Будем пользовать стандартные библиотеки, поэтому создаём соответствующие структуры
ETH_InitTypeDef ETH_InitStructure;

// Ethernet DMA работает с дескрипторами - указатели массивов данных которые надоъ
//транслировать (+ дополнительная служебная информация)
// Дескрипторы
ETH_DMADESCTypeDef  DMARxDscrTab[ETH_RXBUFNB], DMATxDscrTab[ETH_TXBUFNB];
// массив буферов приёма и отправки данных
// на котрые будут ссылаться дескрипторы
u8 Rx_Buff[ETH_RXBUFNB][ETH_MAX_PACKET_SIZE], Tx_Buff[ETH_TXBUFNB][ETH_MAX_PACKET_SIZE];
```

Описать дескрипторы можно вручную, заполнив все адреса и прочую служебную информацию. Но можно выполнить это гораздо быстрее, воспользовавшись библиотечной функцией. В данной ситуации создаём цепочечную структуру дескрипторов (дескриптор, который описывает одну область памяти и указывает адрес следующего дескриптора):

```
// Записываем часть параметров дескрипторов (указатели на буфер + на следующий дескриптор)
  ETH_DMATxDescChainInit(DMATxDscrTab, &Tx_Buff[0][0], ETH_TXBUFNB);
// Записываем длину буфера в дескриптор
  DMATxDscrTab->ControlBufferSize = 100;

// Устанавливаем дескрипторы для приёма
  ETH_DMARxDescChainInit(DMARxDscrTab, &Rx_Buff[0][0], ETH_RXBUFNB);
  DMARxDscrTab->ControlBufferSize = ETH_MAX_PACKET_SIZE | (1<<14);
```

Функции в качестве аргументов имеют начальные адреса дескрипторов, начальные адреса буферов данных и длину одного буфера. Аналогичные библиотечные функции есть для организации кольцевой структуры дескрипторов.

Итак, дескрипторы описаны, свойства сети заданы, запускаем Ethernet:

```
// Разрешаем приём
  DMARxDscrTab->Status = ETH_DMARxDesc_OWN;

// Запускаем Ethernet
  ETH_Start();
```

Как принимать сообщение. Первое, что мы должны сделать – проверить до манипуляций принадлежность дескриптора. Если область памяти отдана приложению, то в ней находится принятая информация по Ethernet. В нашем приложении всего одна область памяти, отданная под приём. Поэтому требуется отключение и включение приёма по каналам DMA для корректной работы. В приложении нас интересуют байты номер 20 и 21 в принятом пакете. Мы их считываем и отдаём дескриптор и буфер в ведение DMA.

```
// Получение посылки по Ethernet
{
  // Проверяем кому принадлежит дескриптор - процессору или
 if ((DMARxDscrTab->Status&ETH_DMARxDesc_OWN) == 0)
 {
 // Чтобы успешно принимать надо сначала отключить приём (либо успевать обрабатывать
 //все буферы, чтобы DMA - не подвешивался).
  ETH_DMAReceptionCmd(DISABLE);
  // считываем яркость. Она находится в 20 и 21-м байте посылки (место может быть и иным :) )
  Bright_Rx = Rx_Buff[0][20]+(Rx_Buff[0][21]<<8);
  // отдаём дескриптор в руки DMA Ethernet
  DMARxDscrTab->Status = ETH_DMARxDesc_OWN;
  // разрешаем приём
  ETH_DMAReceptionCmd(ENABLE);
 }
```

Отправка данных аналогична приёму, с той лишь разницей, что при предыдущей отправке могли возникнуть набор ошибок, которые надо аннулировать.

```
 // а прошлая посылка отправлена? Если отправлена то
if ((DMATxDscrTab->Status&ETH_DMARxDesc_OWN) == 0)
{
// Чтобы успешно отправлять надо сначала откючить передачу (либо успевать
  //обрабатывать все буферы, чтобы DMA - не подвешивался).
 ETH_DMATransmissionCmd(DISABLE);
 //кладём данные по яркости в 20 и 21-й байт посылки
 Tx_Buff[0][20] = Bright_Tx&0xFF;
 Tx_Buff[0][21] = Bright_Tx>>8;
 // отдаём дескриптор в руки DMA Ethernet
 DMATxDscrTab->Status = ETH_DMARxDesc_OWN | ETH_DMATxDesc_TCH |
   ETH_DMATxDesc_TTSE | ETH_DMATxDesc_LS | ETH_DMATxDesc_FS;
 // разрешаем передачу
 ETH_DMATransmissionCmd(ENABLE);
```

В отправляемый пакет кладём данные, в нашем случае это два байта со смещением 20 и 21 от начала пакета и отдаём дескриптор в распоряжение DMA.

Итак, мы рассмотрели необходимые

7. А где же MAC адреса?

Действительно, в предыдущем примере мы настроили приёмную часть Ethernet таким образом, что контроллер получал все сообщения. Именно поэтому мы не формировали пакетов. Как это делать показано ниже:

```
  // адрес назначения МАССОВАЯ РАССЫЛКА
Tx_Buff[0][0]=0xff;
Tx_Buff[0][1]=0xff;
Tx_Buff[0][2]=0xff;
Tx_Buff[0][3]=0xff;
Tx_Buff[0][4]=0xff;
Tx_Buff[0][5]=0xff;
  // адрес источника
Tx_Buff[0][6]=0x17;
Tx_Buff[0][7]=0x11;
Tx_Buff[0][8]=0x12;
Tx_Buff[0][9]=0x13;
Tx_Buff[0][10]=0x14;
Tx_Buff[0][11]=0x15;
  // длина пакета
Tx_Buff[0][6]=0x0;
Tx_Buff[0][7]=0x55;
```

Мы самостоятельно заполняем преамбулу пакета, аналогично структуре, описанной в главе 2. При этом мы не заполняем синхронизирующий пакет с командой начала данных – первые восемь байт, а так же контрольную сумму CRC32, которая будет добавлена автоматически.

Если мы собираемся принимать пакеты определённой структуры, то мы можем ввести фильтрацию. Далее показан вариант настройки Ethernet периферии с организацией фильтрации по адресу источника.

```
//  Принимать всё подряд
//    ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Enable;

//  Принимать не всё подряд
//---------------------------------------
  ETH_InitStructure.ETH_ReceiveAll = ETH_ReceiveAll_Disable;
  ETH_InitStructure.ETH_SourceAddrFilter = ETH_SourceAddrFilter_Normal_Enable;
  ETH_InitStructure.ETH_DestinationAddrFilter = ETH_DestinationAddrFilter_Normal;
//---------------------------------------
```

Далее, необходимо настроить фильтр. В качестве фильтра будем использовать MACAddress1.

```
//настраиваем MAC адрес устройства
Mac_addr0[0]=0x0;
Mac_addr0[1]=0x55;
Mac_addr0[2]=0x12;
Mac_addr0[3]=0x13;
Mac_addr0[4]=0x14;
Mac_addr0[5]=0x17;

//ETH_MACAddressConfig(ETH_MAC_Address0, Mac_addr0);

ETH_MACAddressConfig(ETH_MAC_Address1, Mac_addr0);
ETH_MACAddressFilterConfig(ETH_MAC_Address1,ETH_MAC_AddressFilter_SA);
ETH_MACAddressPerfectFilterCmd(ETH_MAC_Address1, ENABLE);
```

Обратите внимание, что первые два байта фильтра Mac_addr0[0] и Mac_addr0[1] – обозначают длину принимаемого пакета. Кроме того, каждые 4 бита фильтра можно индивидуально подключать/отключать, что позволяет более гибко фильтровать приёмные сообщения.

Более подробную информацию, а так же материалы тренинга можно получить обратившись в техническую поддержку компании Промэлектроника support@promelec.ru

8. Решения STMicroelectronics для Ethernet

Компания STMicroelectronics не ограничивается выпуском одних лишь микроконтроллеров с периферией Ethernet, а предлагает законченное решение для интеграции разрабатываемого приложения по данному стандарту. Что кроме контроллера может быть необходимо?

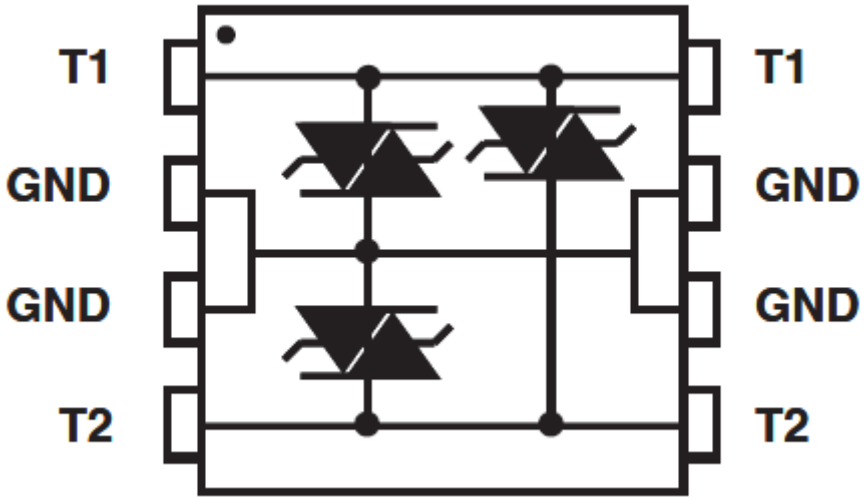
Во-первых, микросхема передачи сигнала Ethernet на физическом уровне. Далее приведены некоторые рекомендуемые фирмой STMicroelectronics решения PHY уровня.

Ethernet PHY Manufacturers	Part#	I/F supported	50MHz Internal PLL	package	Size
ST	ST802RT1A	MII/RMII/SMII	No	TQFP48	7x7
ST	STE101p	MII/RMII/SMII	No	TQFP64	10x10
National	DP83840TW	MII/RMII	Yes	LQFP48	7x7
National	DP83848CW	MII/RMII	No	LQFP48	7x7
Realtech	RTL8201N	MII/RMII	Yes	QFN or QFN32 (RMII only)	9x9 or 5x5
Micrel	KSZ8721CL	MII/RMII	No	LQFP48	7x7
SMSC	LAN8700	MII/RMII	No	QFN36	6x6
SMSC	LAN83C185	MII	No	TQFP64	10x10
Marvell	88E3015	MII/RGMII	No	QFN56	8x8

Мы рекомендуем применять ST802RT1A как оптимальные по сочетанию цена – качество.

Во-вторых, защита от поперечной помехи, способной вывести из строя микросхему передачи PHY уровня. Мы рекомендуем закладывать такого рода устройства на плату, но в массовом производстве её устанавливать исключительно по результатам испытаний на ЭМС. Некоторые приложения (например, бытовая электроника) могут работать и без

такой защиты. Специально для Ethernet компания STMicroelectronics выпустила устройство защиты ETP01.

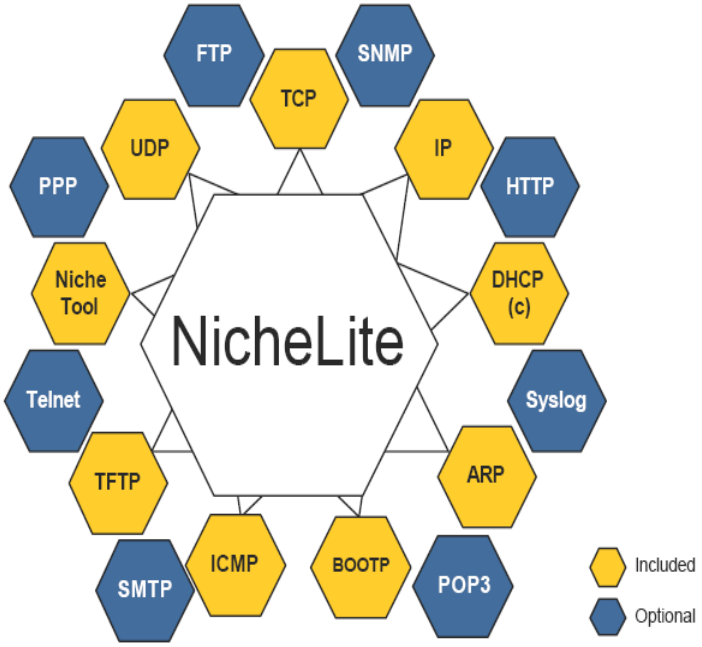


В-третьих, фирма STMicroelectronics выпускает микросхемы питания через Ethernet. На данный момент доступны две микросхемы – приёмников питания по линии Ethernet. Это PM8800 и PM8803. Так же в разработке находится источник питания POE.

В-четвёртых, имеется ряд отладочных комплектов, на базе которых, можно быстро провести отладку приложения с Ethernet. Вот некоторые из них:

- STM3210C-EVAL
- STM3210C-EVAL/IAR
- STM3210C-EVAL/Keil
-STEVAL-PCC010V1
-SK_MSTM32F107

В-пятых, компании партнёры STMicroelectronics создали ряд платных и бесплатных программных решений для Ethernet. Например, компания NichLite создала бесплатный TCP IP стек для контроллеров STM32.



Компания STMicroelectronics рекомендует так же обратить внимание на платное программное обеспечение фирм Micrium, Keil, IAR, Segger, CMX, Quadros и других.

Таким образом, разрабатывать устройства с интегрированной связью по сети Ethernet становится весьма просто. Работа компании STMicroelectronics в этой сфере яркий тому пример.