Построение обмена по сети Ethernet на контроллерах STM32F107

Денис Ягов, инженер по применению, ТД «Промэлектроника», yagov@promelec.ru

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 решение по сочетанию цена/возможности. 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, с помощью которого 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, или вид пакета данных. 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 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. По старшим трём байтам MAC-адреса можно также 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 указывает тип пакета, длина которого формально может иметь любой размер. Минимальная длина пакета составляет 46 байт. При необходимости обмена пакетами меньшей длины следует дополнить их пустыми данными. Контроллеры STM32 выполняют эту операцию аппаратно.

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 требуется до работы с принятым пакетом проверить её соответствие принятым данным. И ту, и другую операцию контроллер STM 32 выполняет аппаратно.

Ethernet в STM32

В состав микроконтроллеров STM32F107xx, STM32F207xx и STM32F217xx входит периферия Ethernet MAC. Основные свойства периферии:

- полноценный MAC-уровень с подключением к внешнему физическому уровню;
- работа на скоростях 10 и 100 Мбит/с;
- 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 (см. рис. 1).

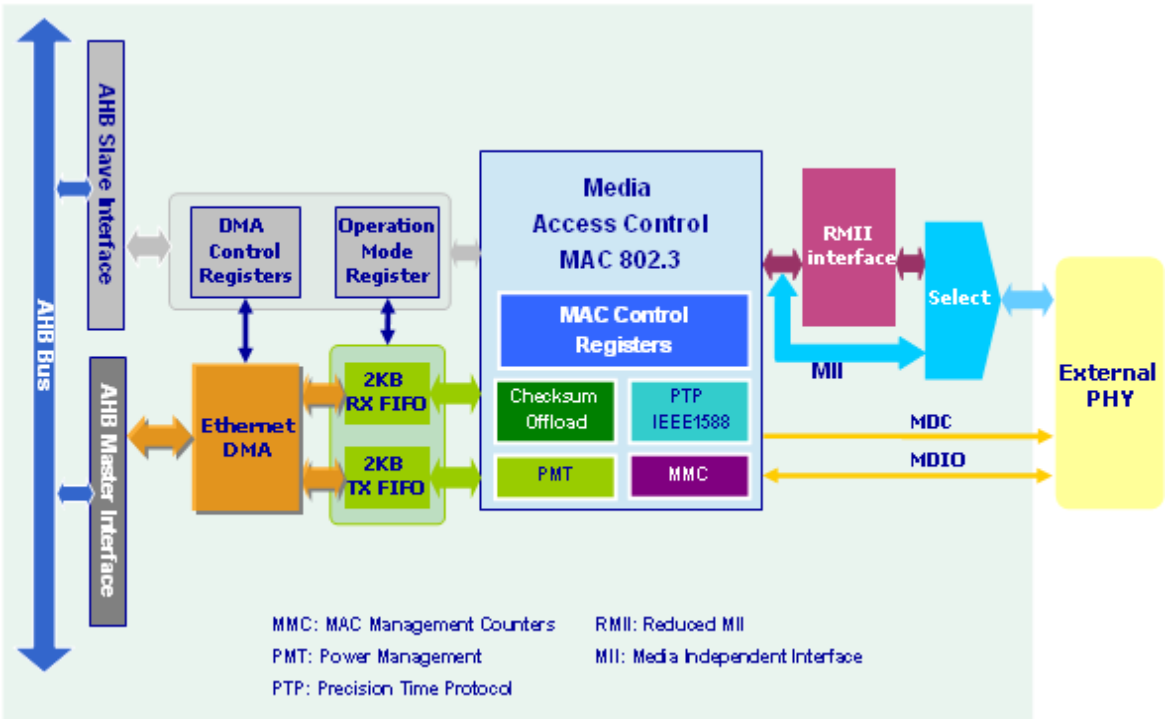

Рис. 1. Блок-схема Ethernet MAC в контроллере STM32F107

Всю блок-схему можно разделить на три части: интерфейс подключения Ethernet MAC к 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 также 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 обстоит ситуация с приёмом. Если из очереди приёма данные попали в обозначенную область ОЗУ, можно считать, что они прошли все настроенные фильтры и, в некоторых случаях, – проверку контрольной суммы. Таким образом, процедура приёма окончена.

Кроме непосредственных операций с потоком данных, управляемых контрольными регистрами, контроллер MAC 802.3 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 – 2 Кбайт. 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.

Кроме этого, контроллер поддерживает стандарт PTP IEEE 1588, который позволяет синхронизировать работу сети во времени, что необходимо, 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 об очередности трансляции сообщений, можно существенно уменьшить число коллизий на линии.

Чтобы производить синхронизацию, каждое из устройств сети должно иметь внутренние часы. Часы каждого из устройств показывают 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 1 мкс. 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 также 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 отреагировать, 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. Выход из энергосберегающего режима осуществляется, например, при получении пакета специального вида. Пакет, посредством которого контроллер выводится из энергосберегающего режима, должен пройти все фильтры, иначе он не произведёт каких-либо действий.

Контроллер MAC 802.3 следит за состоянием очереди приёма данных из Ethernet. Что произойдёт, если очередь приёмных сообщений 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 возобновить трансляцию.

STM32 может иметь до четырёх MAC-адресов. Нулевой MAC-адрес у устройства имеется по умолчанию. Остальные три могут быть разрешены либо запрещены. Все MAC-адреса можно использовать в качестве фильтров сообщений. Кроме фильтрации по точным адресам контроллер может применять относительную фильтрацию, называемую hash-фильтрацией. Основное свойство данного механизма в фильтрации сообщений, презназначенных для групп получателей.

Внешний физический интерфейс Ethernet подключается контроллеру по определённому стандарту – 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. На рисунке 2 показаны оба вида подключений.

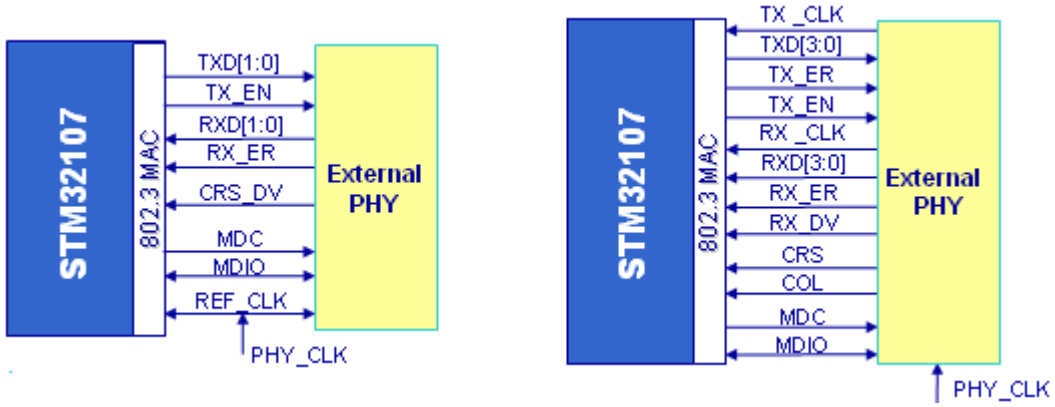

Рис. 2. Подключение по стандарту RMII (слева) и MII (справа)

Подключение по RMII требует всего восьми 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, можно предположить, что соединение по RMII более выгодно. Скорее всего, так и есть, однако имеется ряд нюансов. Интерфейс RMII работает на частоте 50 МГц, что вдвое выше, чем у MII. Соответственно, эмиссия помех такого решения выше. Более того, для генерации 50 МГц и выдаче этой частоты на микросхему физического уровня задействуется 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.

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 четыре слова длиной 32 бита каждое, размещённые в ОЗУ.

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 имеется 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 имеет возможность наблюдать результат действий.

Дескриптор также имеет поле, указывающее длину и адрес области памяти, которая подлежит трансляции/приёму. STM32 поддерживает два вида дескрипторов (не путать с разделением дескрипторов на приём и трансляцию). Один вид описывает одну область памяти и указывает адрес следующего дескриптора. Другой вид описывает сразу две области памяти, а следующий дескриптор располагается последовательно в памяти за данным. Отличие значений одного вида дескриптора от другого проявится в словах №№2 и 4 в теле дескриптора. В зависимости от типа в четвёртом поле дается указатель на адрес следующего дескриптора или на адрес следующей области памяти, подлежащей приёму/отправке данных. Во втором указаны длины областей памяти, предназначенных для получения/трансляции данных в зависимости от типа дескриптора. Значение каждого бита дескриптора подробно описано в точном описании RM0008 контроллеров STM32.

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 текущий 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 определенный вид дескриптора). В случае цепочной 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. Также контроллер прямого доступа к памяти закрепляет описываемый буфер за приложением.

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

Пример Ethernet-приложения для контроллера STM32

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

До задания параметров Ethernet следует 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 задаем 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 и внешнюю микросхему PHY-уровня. Далее должно быть выделено пространство буферов приёма, буферов передачи и для описывающих их дескрипторов.

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

// Ethernet DMA работает с дескрипторами – указатели массивов данных, которые следует транслировать (+ дополнительная служебная информация)
// Дескрипторы
ETH_DMADESCTypeDef DMARxDscrTab[ETH_RXBUFNB], DMATxDscrTab[ETH_TXBUFNB];
// массив буферов приёма и отправки данных
// на которые 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 цепочную 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 имеются 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, в ней находится принятая по Ethernet информация. 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 №№20 и 21 в принятом пакете. Мы их считываем и отдаём дескриптор и буфер в ведение DMA.

```
// Проверяем, кому принадлежит дескриптор – процессору или
if ((DMARxDscrTab->Status&ETH_DMARxDesc_OWN) == 0)
{
// Чтобы успешно принимать, следует 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 DMA Ethernet
DMARxDscrTab->Status = ETH_DMARxDesc_OWN;
// разрешаем приём
ETH_DMAReceptionCmd(ENABLE);
}
```

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

```
// а прошлая посылка отправлена?Если отправлена, то
if ((DMATxDscrTab->Status&ETH_DMARxDesc_OWN) == 0)
{
// Чтобы успешно отправлять, следует сначала отключить передачу (либо успевать обрабатывать все буферы, чтобы DMA не подвешивался).
ETH_DMATransmissionCmd(DISABLE);
//помещаем 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 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 помещаем данные. В нашем случае это два байта со смещением 20 и 21 от начала пакета и передаём дескриптор в распоряжение DMA.

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 разделе «Ethernet в STM32». 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.

```
// Настройка фильтра приёма
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 воспользуемся 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_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 гибче фильтровать приёмные сообщения.

Более подробную информацию, а также материалы тренинга по применению микроконтроллеров можно получить, обратившись в техническую поддержку компании «Промэлектроника» support@promelec.ru.