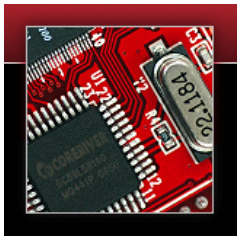


Подключение микроконтроллера к локальной сети: работаем с ENC28J60

Связь железа с компьютером.

Эта часть полностью посвящена работе с ENC28J60.

Кому-то она может показаться тупым копированием даташита. Но это не совсем так, тут есть и примеры кода и описание различных граблей.

Но даташит всё равно может пригодится. А так же errata.

Краткое содержание:

- Включение ENC28J60
- Архитектура ENC28J60
- Обмен данными по SPI
- Инициализация
- Отправка пакетов
- Приём пакетов
- Заключение

Примеры кода написаны под AVR. Впрочем из платформенно-зависимых вещей тут только работа со SPI.

ENC28J60 — Ethernet-адаптер (проще говоря, «сетевая карточка») на одном чипе, разработанный вражеской компанией Microchip. Микросхемка не требует для работы много обвязки из внешних компонентов, к МК подключается с помощью SPI. Полностью соответствует спецификации IEEE 802.3 и, кроме того, поддерживает много дополнительных прикольных фич (например, аппаратную фильтрацию пакетов).

А теперь, немного о грустном. Количество багов в ENC28J60 трудно описать печатными словами. Из-за них половина фич либо работает нестабильно, либо может нарушать работу других важных модулей. Хотя, главное, что принимать и отправлять пакеты девайс всё-таки умеет. :)

Подключаемся

Микросхемка выпускается в 28-ножечных DIP, SOIC и QFN корпусах. Попадаются и готовые модули со всей обвязкой и разъёмом для сетевого кабеля.

Вот стандартная схема включения ENC28J60 (распиновка для DIP корпуса):

Прямой эфир

Комментарии Публикации

Vga → Источник бесперебойного питания 34 → Силовая электроника

Make_Pic → Самый простой программный таймер 36 → Алгоритмы и программные решения

evsi → Понижающий преобразователь на UC3845 15 → Схемотехника

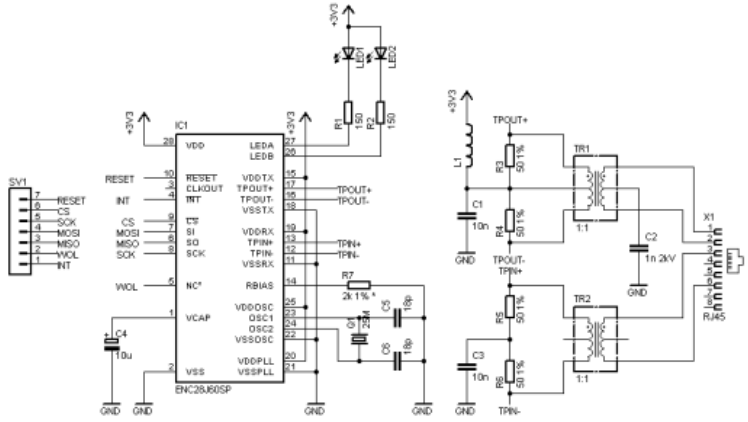
evsi → Farewell letter 192 → Блог им. e_mc2

anakost → Зарядное устройство для Li-ion на TP4056 91 → Деталька

Aljoska → Долгожданный паяльник A-BF GS110D 337 → Инструмент

Vga → 8 бит недорого для всех или как развлечь себя с помощью stm8 35 → Блог им. XOR

Vga → STM32f4 USB HS DMA HAL. Как это было 54 → STM32



Увеличить схему

Питание — 3.3 В. Но входы микросхемы совместимы с 5-вольтовыми TTL уровнями.

Потребляет микросхемка прилично — 250 мА. Нужно столько для питания драйверов передатчика. Есть режим «пониженного энергопотребления», когда вся «силовая» часть отключается.

VCAP — выход встроенного преобразователя на 2,5 В (именно такое напряжение используется при передачи данных по сетевому кабелю). К этому выводу нужно подключить конденсатор на 10 мкф. Даташит не рекомендует питать от этого вывода что-то ещё.

R7 (RBIAS) — резистор для какой-то балансировки. В даташите указан номинал 2 кОм с допуском 1%. Однако в ENC28J60 есть баг, из-за которого в ревизиях микросхемы 1 и 4 нужно использовать резистор на 2,7 кОм. А в ревизиях 5 и 7 — на 2,32 кОм. Иначе выходной сигнал не будет соответствовать спецификации IEEE. Западло, однако. Ревизию можно определить только чтением соответствующего регистра. Мне попалась ревизия 7 — судя по всему, последняя. Так что, стоит запаять резистор на 2,32 кОм, прочитать ревизию, и, если вдруг попадётся другая ревизия, резистор заменить. Где взять резистор на такой необычный номинал? Можно бегать по магазинам и искать, или просто взять горсточку обычных неточных резисторов на 2,2 кОм и методом тыка выбрать наиболее близкий к требуемому номиналу. Впрочем, когда я только начинал экспериментировать с ENC28J60, об этом баге не знал и ставил резистор на 2 кОм. Проблем не было. :)

TR1 и TR2 — не абы какие трансформаторы, а специальные Ethernet-фильтры (Ethernet magnetics). Представляют собой систему из нескольких катушек на ферритовых колечках. Обычно выпускаются в виде готовых сборок (оба фильтра в одном корпусе, совместимом с DIP-16). Нужны они, судя по всему, для развязки, защиты от статики, etc. (сетевой кабель может иметь длину до 100, а то и 300 м — представь какой статический потенциал может быть между девайсами на таком расстоянии). Где их взять? Вариант номер раз — взять мёртвую/ненужную (или хотя-бы дешёвую) сетевую карточку, и выпаять из неё. Кстати, в ней же найдутся точные резисторы на 50 Ом, розеточка для сетевого кабеля и, если повезёт, кварц на 25 МГц. Вариант номер два — использовать сетевой разъём с уже встроенными фильтрами («MagJack»). Деталька редкая и дорогая. Впрочем, достать её можно из трупика материнской платы — обычно там используются именно они. Правда, чтобы выпаять такую массивную деталь из толстенной платы с кучей слоёв меди, понадобится паяльная лампа или что-то в этом роде. К тому же часто деталька идёт в виде неразделимого блока с USB-A розеточками, которые будут вносить неприятную энтропию в конструкцию (если, конечно, в ней не будет USB-хоста, хе-хе).

Катушка L1 — ферритовое колечко диаметром 5мм с несколькими витками проволоки. Если колечка под рукой нет, можно поставить дроссель, например, на 100 мкГн.

ENC28J60 автоматически определяет полярность подключенных светодиодов. Причём полярность светодиода, подключенного к выводу LEDB влияет на дуплексный режим работы микросхемы. Если светодиод подключен как показано на схеме, катодом к микрухе — ENC28J60

yanvasilij → Beremiz - свободная среда программирования ПЛК. Часть 2 7 → PLC

Metallist64 → Музыкальный ночник 6 → Блог им. Metallist64

romanetz → Beremiz - свободная среда программирования ПЛК. Часть 1 57 → PLC

yauheni_lychkouski → I2S в FPGA на Verilog 18 → ПЛИС

Vitalik → Особенности ведения базы ЭРИ в САПР электроники Delta Design 55 → Софт для электронщика

birdie → Питание от Li-Ion с самоотключением 38 → Схемотехника

angel5a → Говнопост 38 → Блог им. angel5a

XOR → Первые впечатления от tiny817 11 → AVR

Vga → Обновленное IDE для STM8 14 → Блог им. Papandopala

shluzzzoid → Зарядное устройство на MAX713. Импульсный режим. 54 → Силовая электроника

Wan-Derer → Работа с Bluetooth модулем HC05 119 → Деталька

Vga → О припоях 170 → Осторожно, говно!

Весь эфир | RSS

1-Wire Altera arduino ARM Assembler Atmel AVR C++ DIY enc28j60 ethernet FPGA gcc I2C IAR KEIL LaunchPad LCD led linux LPCXpresso MSP430 nxp PCB PIC pinboard2 RS-485 RTOS STM32 STM8 STM8L TI UART USB алгоритм ассемблер АЦП библиотека блок питания вопрос деталька дисплей идея инструмент конкурс конкурс2 ЛУТ микроконтроллеры начинающим обзор Отладочная плата паяльник печатная плата плата ПЛИС поделки покупки программатор программирование светодиод софт схема схемотехника Технологии умный дом фоторезист халява хрень Часы юмор

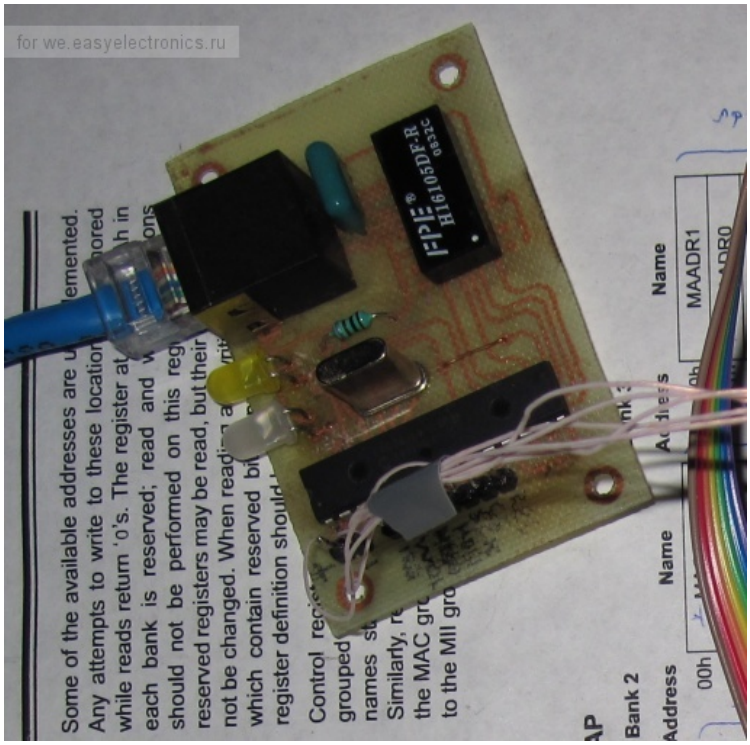
Блоги

Топ

STM8	**37.92**
AVR	**37.52**
Мусоровоз	**29.53**
STM32	**27.02**
Деталька	**23.24**
Связь железа с компьютером.	**22.58**
Схемотехника	**18.15**
Умный дом	**17.75**
MSP430	**17.13**

инициализируется в полнодуплексном режиме. Соответственно, если подключен анодом — то в полудуплексном. Если светодиод не подключен — состояние не определено. Впрочем, дуплексный режим можно изменить при инициализации.

Конденсатор C2 на 2 кВ служит для разрядки статики при подключении кабеля. Естественно, можно поставить конденсатор и на меньшее напряжение. Ни на что это не повлияет, разве что твой девайс не будет формально соответствовать стандарту.



~~Вход RESET уже подтянут к питанию внутри микрухи, так что его можно оставить болтаться — ENC28J60 поддерживает и «мягкий» сброс.~~

Как выяснилось, вход RESET у ENC28J60, несмотря на то, что сказано в даташите, не подтянут! Его обязательно нужно соединить с питанием, иначе микруха может сброситься в самый неудачный момент из-за любой наводки.

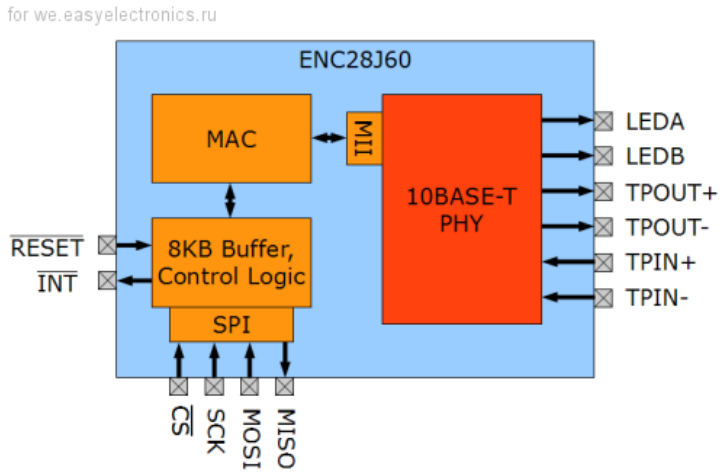
Выходы прерываний использовать не обязательно, и, как мне кажется, и не нужно. Забрать принятый пакет или загрузить пакет для отправки — слишком длительная процедура, чтобы выполнять её в обработчике прерывания. Лучше делать это из главного цикла.

С выхода CLKOUT можно снимать тактовый сигнал (с настраиваемым делителем). Правда, из-за бага, при входе в спящий режим сигнал пропадает (хотя главные часики продолжают тикать). Блин, инженеры из Microchip совсем забивают на тестирование?!

Таким образом, для связи с микроконтроллером можно использовать только 4 провода — стандартную шину SPI.

Архитектура ENC28J60

На этой картинке я нарисовал основные блоки ENC28J60:



- PHY — физический уровень. Приёмник, передатчик, драйверы, etc. В общем, всё, что необходимо для работы с определённой средой передачи данных (medium). В данном случае — с витой парой, по стандарту 10BASE-T. Доступ к PHY происходит исключительно через MII — Medium Independent Interface. MII задуман так, чтобы следующий (канальный) уровень мог абстрагироваться от типа среды передачи данных. PHY имеет свой набор 16-битных регистров (специфичных для среды передачи данных), доступ к которым осуществляется через MII. Не нужно пугаться аббревиатуры MII — это всего лишь набор регистров, через которые управляется PHY.
- MAC (Medium Access Controller) — канальный уровень. В него входит вся логика, необходимая для отправки и приёма пакетов в сети Ethernet. MAC занимается адресацией, рассчётом контрольной суммы, фильтрацией принимаемых пакетов, разрешением коллизий (в полудуплексном режиме), etc. Обменивается со следующим, сетевым уровнем готовыми пакетами, а с физическим — отправляемыми и принимаемыми «сырыми» байтами.
- Управляющая логика занимается всем остальным. В том числе, обслуживает буфер, из которого MAC берёт отправляемые данные и складывает принятые. Управляет режимами энергопотребления, etc.

Вся память в ENC28J60 делится на буфер для данных, управляющие регистры и регистры PHY.

Буфер для данных

В ENC28J60 есть буфер размером 8 КБ. Часть этого буфера обычно выделяется для приёма пакетов, остальное можно использовать как угодно. Например, для отправляемых данных.

Управляющие регистры

Управляющие регистры делятся на 4 банка (ну нравится Microchip'овским инженерам сегментированное адресное пространство). Каждый банк имеет размер в 32 регистра, причём последние 5 ячеек (0x1b..0x1f) всегда мапятся на одни и те же регистры, вне зависимости от того, какой банк выбран.

Страшно?

	Банк 0	Банк 1	Банк 2	Банк 3
0x00	ERDPTL	EHT0	MACON1	MAADR1
0x01	ERDPTH	EHT1	MACON2	MAADR0
0x02	EWRPTL	EHT2	MACON3	MAADR3
0x03	EWRPTH	EHT3	MACON4	MAADR2
0x04	ETXSTL	EHT4	MABBIPG	MAADR5
0x05	ETXSTH	EHT5	—	MAADR4
0x06	ETXNDL	EHT6	MAIPGL	EBSTSD
0x07	ETXNDH	EHT7	MAIPGH	EBSTCON
0x08	ERXSTL	EPMM0	MACLCON1	EBSTCSL
0x09	ERXSTH	EPMM1	MACLCON2	EBSTCSH
0x0A	ERXNDL	EPMM2	MAMXFLL	MISTAT
0x0B	ERXNDH	EPMM3	MAMXFLH	—
0x0C	ERXRDPTL	EPMM4	Reserved	—
0x0D	ERXRDPTH	EPMM5	MAPHSUP	—
0x0E	ERXWRPTL	EPMM6	Reserved	—
0x0F	ERXWRPTH	EPMM7	—	—
0x10	EDMASTL	EPMCSL	Reserved	—
0x11	EDMASTH	EPMCSH	MICON	—
0x12	EDMANDL	—	MICMD	EREVID
0x13	EDMANDH	—	—	—
0x14	EDMADSTL	EPMOL	MIREGADR	—
0x15	EDMADSTH	EPMOH	Reserved	ECOCON
0x16	EDMACSL	EWOLIE	MIWRL	Reserved
0x17	EDMACSH	EWOLIR	MIWRH	EFLOCON
0x18	—	ERXFCON	MIRDL	EPAUSL
0x19	—	EPKTCNT	MIRDH	EPAUSH
0x1A	Reserved	Reserved	Reserved	Reserved
0x1B	EIE	EIE	EIE	EIE
0x1C	EIR	EIR	EIR	EIR
0x1D	ESTAT	ESTAT	ESTAT	ESTAT
0x1E	ECON2	ECON2	ECON2	ECON2
0x1F	ECON1	ECON1	ECON1	ECON1

Пугаться количества не нужно. Сейчас всё структурируем, и станет просто.

Основная часть регистров имеет префикс E (Ethernet). Регистры MAC — с префиксом MA, регистры MII — с префиксом MI.

Регистры можно разделить на функциональные группы.

	Банк 0	Банк 1	Банк 2	Банк 3
0x00	ERDPTL	EHT0	MACON1	MAADR1
0x01	ERDPTH	EHT1	MACON2	MAADR0
0x02	EWRPTL	EHT2	MACON3	MAADR3
0x03	EWRPTH	EHT3	MACON4	MAADR2
0x04	ETXSTL	EHT4	MABBIPG	MAADR5
0x05	ETXSTH	EHT5	—	MAADR4
0x06	ETXNDL	EHT6	MAIPGL	EBSTSD
0x07	ETXNDH	EHT7	MAIPGH	EBSTCON
0x08	ERXSTL	EPMM0	MACLCON1	EBSTCSL
0x09	ERXSTH	EPMM1	MACLCON2	EBSTCSH
0x0A	ERXNDL	EPMM2	MAMXFLL	MISTAT
0x0B	ERXNDH	EPMM3	MAMXFLH	—
0x0C	ERXRDPTL	EPMM4	Reserved	—
0x0D	ERXRDPTH	EPMM5	MAPHSUP	—
0x0E	ERXWRPTL	EPMM6	Reserved	—
0x0F	ERXWRPTH	EPMM7	—	—
0x10	EDMASTL	EPMCSL	Reserved	—
0x11	EDMASTH	EPMCSH	MICON	—
0x12	EDMANDL	—	MICMD	EREVID
0x13	EDMANDH	—	—	—
0x14	EDMADSTL	EPMOL	MIREGADR	—
0x15	EDMADSTH	EPMOH	Reserved	ECOCON
0x16	EDMACSL	EWOLIE	MIWRL	Reserved
0x17	EDMACSH	EWOLIR	MIWRH	EFLOCON
0x18	—	ERXFCON	MIRDL	EPAUSL
0x19	—	EPKTCNT	MIRDH	EPAUSH
0x1A	Reserved	Reserved	Reserved	Reserved
0x1B	EIE	EIE	EIE	EIE
0x1C	EIR	EIR	EIR	EIR
0x1D	ESTAT	ESTAT	ESTAT	ESTAT
0x1E	ECON2	ECON2	ECON2	ECON2
0x1F	ECON1	ECON1	ECON1	ECON1

Основное
Указатели буфера
DMA
Фильтрация пакетов
MAC-адрес
Регистры MAC
Регистры MII
Управление потоком
Прерывания
Идентификатор ревизии (e.g. 0x06 = ревизия 7)
Управление ножкой CLKOUT
«Встроенный тест»

Основное

Назначение отдельных бит, как правило, понятно из их названий. :)
Здесь я опишу лишь то, что нам реально понадобится. Остальные биты описаны в даташите. Однако перед использованием той или иной фичи, нужно заглянуть в еррату и проверить нет ли с данной фичей проблем. Например, DMA (биты DMAST и CSUMEN) еррата использовать не рекомендует вообще. Так-то!

ECON1

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TXRST	**RXRST**	**DMAST**	**CSUMEN**	**TXRTS**	**RXEN**	**BSEL1**	**BSEL0**
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

- BSEL1:BSEL0 — выбор банка регистров.
- RXEN — разрешает приём данных.
- TXRTS — разрешает отправку пакета (автоматически сбрасывается после того, как отправка пакета будет завершена).

ECON2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
AUTOINC	**PKTDEC**	**PWRSV**	—	**VRPS**	—	—	—
R/W-1	W-0	R/W-0	U-0	R/W-0	U-0	U-0	U-0

- VRPS — разрешает перевод стабилизатора питания в экономичный режим при включении режима пониженного энергопотребления (бит PWRSV). Данный бит можно установить при инициализации и забыть про него.
- PWRSV — включает режим пониженного энергопотребления. Прежде чем устанавливать этот бит, следует запретить приём новых пакетов и убедиться что приём данных завершён. После выхода из режима пониженного энергопотребления, нужно подождать 1 мс, чтобы PHY вошёл в рабочий режим.
- PKTDEC — при установке этого бита значение счётчика пакетов уменьшается на 1.
- AUTOINC — включает автоматическое инкрементирование указателей чтения и записи буфера (для удобства последовательного чтения и записи данных). Этот бит установлен после сброса, и трогать его ни к чему.

ESTAT

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
INT	r	r	**LATECOL**	—	**RXBUSY**	**TXABRT**	**CLKRDY**
R-0	R/C-0	R-0	R/C-0	U-0	R-0	R/C-0	R/W-0

- TXABRT — флаг завершения передачи с ошибкой.
- RXBUSY — признак работы приёмника (установлен, если принимаются данные).

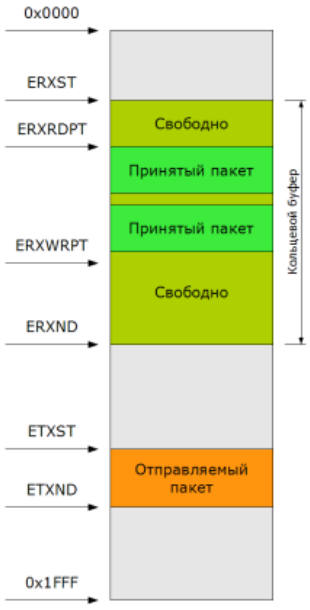
Регистр EPKTCNT — счётчик принятых пакетов. Автоматически инкрементируется при успешно принятом пакете. Уменьшается вручную, установкой бита PKTDEC в регистре ECON2. Вручную записывать в этот регистр ничего нельзя, т.к. все операции с ним должны выполняться атомарно.

Указатели буфера

Регистры, указывающие куда приёмник будет складывать данные, откуда данные будет брать передатчик, etc. Каждый указатель занимает два регистра. Например, младший байт ERDPT хранится в регистре ERDPTL, а старший — в ERDPTH.

ERDPT и EWRPT — указатели чтения и записи буфера. Указывают по какому адресу данные будут считываться из буфера или записываться в буфер микроконтроллером.

Если ECON2.AUTOINC установлен, данные будут считываться и записываться последовательно (соответствующий указатель будет инкрементироваться после каждого байта).



ETXST и ETXND — начало и конец отправляемого пакета. Например, если мы хотим отравить пакет размером 256 байт, лежащий в буфере по адресу 0x1800, устанавливаем ETXST в 0x1800 и ETXND в 0x18ff.

ERXST и ERXND — начло и конец кольцевого буфера, в который будут

приниматься пакеты. Из-за бага в ENC28J60, в ERXST можно записывать только 0. Например, если мы хотим выделить 4096 байт под приём пакетов, пишем в ERXST 0, а в ERXND 0x0fff. Когда приём пакетов разрешён, трогать эти регистры нельзя.

ERXRDPT и ERXWRPT — указатели кольцевого буфера. Доходя до конца буфера (ERXND), указатель затем перемещается на начало (ERXST).

ERXWRPT — указывает на место, куда приёмник положит следующий принятый пакет. Этот указатель доступен только для чтения. Он автоматически инициализируется вместе с ERXST и автоматически обновляется после приёма пакета.

ERXRDPT — указывает на то место, откуда микроконтроллер будет забирать принятые пакеты.

Если микроконтроллер долго не будет забирать пакеты из буфера, ERXWRPT может «догнать» ERXRDPT. В таком случае приёмнику некуда будет складывать данные и приходящие данные начнут выбрасываться. Чтобы освобождать место в буфере, микроконтроллер, после того, как заберёт пакет из буфера, должен перемещать ERXRDPT к следующему пакету.

DMA

В ENC28J60 есть модуль DMA, позволяющий выполнять операции над блоками. А именно, копирование блока внутри буфера и аппаратное вычисление контрольной суммы для IP и других протоколов. Второе даже могло бы пригодиться. Но из-за имеющегося бага, использование этой фичи может привести к потере входящих пакетов. Обидно.

Фильтрация пакетов

ENC28J60 отлично умеет фильтровать пакеты. Это важно, особенно, если сеть на хабах. :)

Правила фильтрации пакетов устанавливает регистр ERXFCON.

ERXFCON	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	UCEN	ANDOR	CRCEN	PMEN	MPEN	HTEN	MCEN	BCEN
	R/W-1	R/W-0	R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1

- UCEN включает фильтр Unicast-пакетов. Пакет проходит фильтр, если адрес получателя в нём равен нашему MAC-адресу.
- MCEN включает фильтр Multicast-пакетов. Пакет проходит фильтр, если является Multicast-пакетом.
- BCEN включает фильтр Broadcast-пакетов. Пакет проходит фильтр, если является широковещательным.
- MPEN включает фильтр волшебных пакетов. Пакет проходит фильтр, если является волшебным и направлен на наш MAC-адрес.
- PMEN включает фильтрацию по шаблону.
- HTEN включает фильтрацию по хэш-таблице.
- ANDOR — группировка фильтров. Если бит установлен — пакет принимается только при прохождении всех выбраных фильтров. Если сброшен — прохождения одного фильтра достаточно.
- CRCEN разрешает проверку контрольной суммы. Если установлен, принимаются пакеты только с корректной контрольной суммой.

Фильтрация по шаблону заключается в следующем. Из принятого пакета, по смещению, записанному в регистрах EPMO, берётся окно размером 64 байта. Из этого окна выбираются байты по маске, записанной в регистрах EPMM (например, если бит 0 в регистре EPMM0 установлен, выбирается байт 0 из окна, etc.). От выбранных байт рассчитывается контрольная сумма. Если она совпадает со значением в регистрах EPMCS, фильтр пройден.

При фильтрации по хэш-таблице, рассчитывается хэш от адреса получателя, указанного в заголовке пакета. Берётся соответствующий бит из регистров EHT. Например, если хэш равен 0x5, берётся бит 5 из регистра EHT0. Если бит установлен, фильтр пройден.

MAC-адрес

Те самые 6 байт, которые будут идентифицировать наш девайс в локальной сети. Нужны ENC28J60 для фильтрации входящих пакетов. Хранятся в

обратном порядке, т.е. для адреса 01:23:45:67:89:ab в MAADR0 пишем 0xab, в MAADR1 — 0x89, etc.

Регистры MAC

MACON1	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	—	—	—	**LOOPBK**	**TXPAUS**	**RXPAUS**	**PASSALL**	**MARXEN**
	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

- MARXEN — разрешить MAC принимать пакеты.
- TXPAUS, RXPAUS — включают аппаратное управление потоком.

MACON3	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	PADCFG2	**PADCFG1**	**PADCFG0**	**TXCRCEN**	**PHDRLEN**	**HFRMEN**	**FRMLNEN**	**FULDPX**
	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

- FULLDPX — включить полнодуплексный режим. Дуплексный режим PHY (PHCON1.PDPXMD) должен быть таким же. Значение после сброса зависит от полярности подключения светодиода к ножке LEDB.
- FRMLNEN — включить автоматическую проверку длины принимаемых и отправляемых фреймов.
- TXCRCEN — включить автоматическое добавление контрольной суммы к фрейму.
- PADCFG2:PADCFG0 — настройка паддинга фреймов:
  - 001 — выравнять пакет нулями до 60 байт, затем добавить контрольную сумму (4 байта). Бит TXCRCEN также должен быть установлен.
  - 000 — не выравнивать пакеты.

Регистры MAMXF — максимальная длина принимаемого и отправляемого пакета. Обычно 1518 байт или меньше. Ставим столько, сколько сможет утащить наш МК. Пакеты большего размера будут отбрасываться.

MABBIPG, MAIPGL и MAIPGH — задержка (gap) между отправляемыми пакетами. Стандартные значения:

- MABBIPG — 0x15 (в полнодуплексном режиме) или 0x12 (в полудуплексном).
- MAIPGL — 0x12.
- MAIPGH — 0x0c.

MACLCON1 и MACLCON2 — настройка задержки и ретрансмиссий при возникновении коллизии. Оставляем по умолчанию.

Регистры MII

Регистры MII служат для доступа к регистрам PHY. Во как!

MICON	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	RSTMII	—	—	—	—	—	—	—
	R/W-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0

MICMD	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	—	—	—	—	—	—	**MIISCAN**	**MIIRD**
	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0

MISTAT	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	—	—	—	—	r	**NVALID**	**SCAN**	**BUSY**
	U-0	U-0	U-0	U-0	R-0	R-0	R-0	R-0

Для чтения регистра PHY:

1. Выставляем его адрес в регистре MIREGADR.
2. Устанавливаем бит MICMD.MIIRD.
3. Ждём, пока MISTAT.BUSY очистится.
4. Вручную очищаем MICMD.MIIRD.
5. Забираем данные из регистров MIRD.

Для записи в регистр PHY:

1. Выставляем его адрес в регистре MIREGADR.
2. Записываем данные в регистры MIWR. Сначала MIWRL, затем MIWRH.
3. Ждём, пока MISTAT.BUSY очистится.

Управление ножкой CLKOUT

В ENC28J60 можно брать тактовый сигнал (с делителем) с ножки CLKOUT. Из-за бага, сигнал может пропадать при входе в режим пониженного

энергопотребления.

ECOCON

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	—	—	—	**COCON2**	**COCON1**	**COCON0**
U-0	U-0	U-0	U-0	U-0	R/W-1	R/W-0	R/W-0

Биты ECOCON2:0 устанавливают делитель:

- 000 — ножка CLKOUT подтянута к земле.
- 001 — делитель на 1 (25 МГц).
- 010 — делитель на 2 (12,5 МГц).
- 011 — делитель на 3 (8,333333 МГц).
- 100 — делитель на 4 (6,25 МГц).
- 101 — делитель на 8 (3,125 МГц).

Регистры PHY

Регистры PHY раположены в отдельном адресном пространстве. Получить к ним доступ можно через регистры MII. Размер адресного пространства — 32 регистра, всего заюзано 9 адресов.

0x00	PHCON1
0x01	PHSTAT1
0x02	PHID1
0x03	PHID2
0x10	PHCON2
0x11	PHSTAT2
0x12	PHIE
0x13	PHIR
0x14	PHLCON

Регистры PHY 16-битные. Используются для различных настроек PHY. Целый регистр выделен под настройки светодиодов. Эстетика!

PHCON1

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
PRST	**PLOOPBK**	-	-	**PPWRSV**	**PDPXMD**	-	-
0	0	-	-	1	?	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
r	-	-	-	-	-	-	-
0	-	-	-	-	-	-	-

- PDPXMD — дуплексный режим PHY. Должен соответствовать дуплексному режиму MAC (MACON3.FULLDPX). Начальное значение зависит от полярности светодиода, подключенного к ножке LEDB.

PHCON2

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	**FRCLNK**	**TXDIS**	r	r	**JABBER**	r	**HDLDIS**
-	0	0	0	0	0	0	0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
r	r	r	r	r	r	r	r
0	0	0	0	0	0	0	0

- HDLDIS — запрещает «заворот назад» (loopback) отправляемых данных в полудуплексном режиме.

PHSTAT1

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	-	**PFDPX**	**PHDPX**	-	-	-
-	-	-	1	1	-	-	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	-	-	**LLSTAT**	**JBSTAT**	-
-	-	-	-	-	0	0	-

- LLSTAT — «асинхронный» бит сосотяния линка. Читается как 1 если линк есть и не пропадал с момента предыдущего чтения этого бита.

PHSTAT2

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
-	-	**TXSTAT**	**RXSTAT**	**COLSTAT**	**LSTAT**	**DPXSTAT**	-
-	-	0	0	0	0	?	-

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-	-	-	**PLRITY**	-	-	-	-
-	-	-	?	-	-	-	-

- LSTAT — состояние линка. Бит установлен, если линк есть.

PHLCON

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
r	r	r	r	**LACFG3:0**			
0	0	1	1	0	1	0	0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LBCFG3:0				**LFRQ1:0**		**STRCH**	r
0	0	1	0	0	0	1	x

Управление светодиодиками.

- STRCH — разрешает «растягивание» событий. Если бит включен, события будут отмечаться вспышкой светодиода определённой длительности. Если

выключен — светодиод будет зажигаться только во время события (передача/приём данных, etc.).
- LFRQ — длительность вспышки светодиода:
  - 00 — 40 мс.
  - 01 — 73 мс.
  - 10 — 139 мс.
- LACFG и LBCFG — выбираем, что именно будут показывать светодиоды, подключенные к ножкам LEDA и LEDB:
  - 0001 — передача.
  - 0010 — приём.
  - 0100 — состояние линка.
  - 0101 — дуплексный режим.
  - 0111 — приём и передача.
  - 1000 — включен.
  - 1001 — выключен.
  - 1100 — приём и состояние линка.
  - 1101 — приём, передача и состояние линка.

В общем, большая часть регистров используется исключительно для конфигурации и после завершения инициализации не трогается.

SPI

Обмен по SPI ведётся в режиме 0 (CPOL=0, CPHA=0). ENC28J60 поддерживает скорость передачи данных по SPI до 10 мбит/с.

```
// Указываем как у нас подключено
#define ENC28J60_SPI_DDR    DDRB
#define ENC28J60_SPI_PORT    PORTB
#define ENC28J60_SPI_CS        (1<<PB4)
#define ENC28J60_SPI_MOSI    (1<<PB5)
#define ENC28J60_SPI_MISO    (1<<PB6)
#define ENC28J60_SPI_SCK    (1<<PB7)

#define enc28j60_select() ENC28J60_SPI_PORT &= ~ENC28J60_SPI_CS
#define enc28j60_release() ENC28J60_SPI_PORT |= ENC28J60_SPI_CS

// Инициализация ENC28J60
void enc28j60_init()
{
    // Настроим ножки
    ENC28J60_SPI_DDR |= ENC28J60_SPI_CS|ENC28J60_SPI_MOSI|ENC28J60_SPI_S
    ENC28J60_SPI_DDR &= ~ENC28J60_SPI_MISO;
    enc28j60_release();

    // Максимальная скорость SPI (CLK/2)
    SPCR = (1<<SPE)|(1<<MSTR);
    SPSR |= (1<<SPI2X);

    // Остальная инициализация
    // ...
}

// Передача данных через SPI
uint8_t enc28j60_rxtx(uint8_t data)
{
    SPDR = data;
    while(!(SPSR & (1<<SPIF)))
        ;
    return SPDR;
}
```

Обмен данными с ENC28J60 выполняется транзакциями. Транзакция начинается с отправки микроконтроллером команды. Затем идут

опциональные данные (приём или передача). Завершается транзакция «поднятием» ножки CS.

Чтение:

При чтении данных уровень на линии MOSI не имеет значения.

Запись:

При записи линия MISO находится в Z-состоянии (т.е. не подключена ни к чему).

```
#define enc28j60_rx() enc28j60_rxtx(0xff)
#define enc28j60_tx(data) enc28j60_rxtx(data)
```

Команда состоит из опкода и аргумента. При чтении или записи регистра, аргумент содержит адрес регистра.

	Опкод			Аргумент				
	7	6	5	4	3	2	1	0
Чтение регистра	0	0	0	a	a	a	a	a
Чтение буфера	0	0	1	1	1	0	1	0
Запись регистра	0	1	0	a	a	a	a	a
Запись в буфер	0	1	1	1	1	0	1	0
Установка бит по маске	1	0	0	a	a	a	a	a
Снятие бит по маске	1	0	1	a	a	a	a	a
Мягкий сброс	1	1	1	1	1	1	1	1

Операции с регистрами

Для чтения регистра контроллер отправляет ENC28J60 команду чтения регистра и забирает значение. При чтении регистров MAC или MII, контроллер должен пропустить 1 «ложный» байт, затем прочитать значение.

```
// Операция чтения
uint8_t enc28j60_read_op(uint8_t cmd, uint8_t adr)
{
    uint8_t data;

    // Низкий уровень на CS
    enc28j60_select();

    // Отправляем команду
    enc28j60_tx(cmd | (adr & 0x1f));
```

```
    // При необходимости, пропускаем "ложный" байт
    if(adr & 0x80)
        enc28j60_rx();

    // Читаем данные
    data = enc28j60_rx();

    // Высокий уровень на ножке CS
    enc28j60_release();
    return data;
}

// Операция записи
void enc28j60_write_op(uint8_t cmd, uint8_t adr, uint8_t data)
{
    enc28j60_select();

    // Отправляем команду
    enc28j60_tx(cmd | (adr & 0x1f));

    // Отправляем значение
    enc28j60_tx(data);

    enc28j60_release();
}
```

Перед тем, как осуществлять доступ к определённому регистру, нужно выбрать банк. Чтобы не отправлять команду переключения банка при каждой операции с регистром, можно закэшировать текущий банк и переключать банк только при необходимости.

Также, нам понадобится заголовочный файл с определениями регистров. Для удобства, в определение регистра можно включить также адрес банка и признак регистра MII/MAC.

```
// С этого адреса начинаются глобальные для всех банков регистры
#define ENC28J60_COMMON_CR    0x1B

// Банк 0
#define ERDPTL                0x00
#define ERDPTH                0x01
#define ERDPT                ERDPTL
//...


// Банк 1
#define EHT0                (0x00 | 0x20)
#define EHT1                (0x01 | 0x20)
//...

// Банк 2, регистры MAC/MII
#define MACON1                (0x00 | 0x40 | 0x80)
//...

// Банк 3
#define EREVID                (0x12 | 0x60)
//...
```

Кстати, быстро превратить таблицы из даташита в заголовочный файл поможет любой офисный пакет с редактором электронных таблиц. :)

```
#define ENC28J60_SPI_RCR    0x00
#define ENC28J60_SPI_WCR    0x40
#define ENC28J60_SPI_BFS    0x80
#define ENC28J60_SPI_BFC    0xA0

uint8_t enc28j60_current_bank = 0;
```

```
// Выбор банка регистров
void enc28j60_set_bank(uint8_t adr)
{
    uint8_t bank;

    // Регистр относится к определённому банку?
    if( (adr & ENC28J60_ADDR_MASK) < ENC28J60_COMMON_CR )
    {
        // Получаем номер банка
        bank = (adr >> 5) & 0x03; //BSEL1|BSEL0=0x03

        // Если выбран "не тот" банк
        if(bank != enc28j60_current_bank)
        {
            // Выбираем банк
            enc28j60_write_op(ENC28J60_SPI_BFC, ECON1, 0x03);
            enc28j60_write_op(ENC28J60_SPI_BFS, ECON1, bank);
            enc28j60_current_bank = bank;
        }
    }
}

// Чтение регистра
uint8_t enc28j60_rcr(uint8_t adr)
{
    enc28j60_set_bank(adr);
    return enc28j60_read_op(ENC28J60_SPI_RCR, adr);
}

// Чтение пары регистров (L и H)
uint16_t enc28j60_rcr16(uint8_t adr)
{
    enc28j60_set_bank(adr);
    return enc28j60_read_op(ENC28J60_SPI_RCR, adr) |
        (enc28j60_read_op(ENC28J60_SPI_RCR, adr+1) << 8);
}

// Запись регистра
void enc28j60_wcr(uint8_t adr, uint8_t arg)
{
    enc28j60_set_bank(adr);
    enc28j60_write_op(ENC28J60_SPI_WCR, adr, arg);
}

// Запись пары регистров (L и H)
void enc28j60_wcr16(uint8_t adr, uint16_t arg)
{
    enc28j60_set_bank(adr);
    enc28j60_write_op(ENC28J60_SPI_WCR, adr, arg);
    enc28j60_write_op(ENC28J60_SPI_WCR, adr+1, arg>>8);
}

// Очистка битов в регистре (reg[adr] &= ~mask)
void enc28j60_bfc(uint8_t adr, uint8_t mask)
{
    enc28j60_set_bank(adr);
    enc28j60_write_op(ENC28J60_SPI_BFC, adr, mask);
}

// Установка битов в регистре (reg[adr] |= mask)
void enc28j60_bfs(uint8_t adr, uint8_t mask)
{
    enc28j60_set_bank(adr);
    enc28j60_write_op(ENC28J60_SPI_BFS, adr, mask);
}
```

Чтение и запись буфера

Для чтения буфера отправляем команду чтения, затем читаем столько байт,

сколько нам нужно. Завершается операция поднятием линии CS.

```
#define ENC28J60_SPI_RBM    0x3A

// Чтение данных из буфера (по адресу в регистрах ERDPT)
void enc28j60_read_buffer(uint8_t *buf, uint16_t len)
{
    enc28j60_select();
    enc28j60_tx(ENC28J60_SPI_RBM);
    while(len--)
        *(buf++) = enc28j60_rx();
    enc28j60_release();
}
```

Запись происходит аналогично. Команда, передача данных, поднятие CS.

```
#define ENC28J60_SPI_WBM    0x7A

// Запись данных в буфер (по адресу в регистрах EWRPT)
void enc28j60_write_buffer(uint8_t *buf, uint16_t len)
{
    enc28j60_select();
    enc28j60_tx(ENC28J60_SPI_WBM);
    while(len--)
        enc28j60_tx(*(buf++));
    enc28j60_release();
}
```

Мягкий сброс

Сброс выполняется отправкой команды 0xff. После сброса ждём 1 мс, чтобы ENC28J60 мог выполнить внутреннюю инициализацию.

```
#define ENC28J60_SPI_SC        0xFF

void enc28j60_soft_reset()
{
    // Отправляем команду
    enc28j60_select();
    enc28j60_tx(ENC28J60_SPI_SC);
    enc28j60_release();

    // Ждём, пока ENC28J60 инициализируется
    _delay_ms(1);

    // Не забываем про банк
    enc28j60_current_bank = 0;
}
```

Инициализация

Типичная последовательность инициализации ENC28J60 выглядит примерно так:

- Настраиваем размер FIFO для приёма данных (ERXST, ERXND), инициализируем указатель для чтения данных из FIFO (ERXRDPT).
- Настраиваем фильтрацию входящих пакетов. По умолчанию, ENC28J60 пропускает пакеты, приходящие на наш MAC-адрес и широковещательные пакеты. В принципе, можно так и оставить.
- Настраиваем MAC:
  - Очищаем MACON2.MARST чтобы снять сброс MAC.
  - Устанавливаем MACON1.MARXEN чтобы разрешить приём данных MAC.

- Устанавливаем MACON1.RXPAUS и MACON1.TXPAUS для включения аппаратного управления потоком.
- Настраиваем биты PADCFG, TXCRCEN в MACON3. Для большинства приложений подойдёт выравнивание пакета до 60 байт и автоматическое добавление контрольной суммы.
- Устанавливаем максимальный размер фрейма в регистрах MAMXF.
- Устанавливаем размер промежутка между фреймами в регистрах MABBIPG, MAIPGL и MAIPGH.
- Устанавливаем MAC-адрес в регистрах MAADR.

- Настраиваем PHY:

  - Включаем бит PHCON2.HDLDIS, если не хотим получать свои пакеты обратно в полудуплексном режиме.
  - Выбираем как на различные события будут реагировать светодиоды LEDA и LEDB в регистре PHLCON.

- Настраиваем дуплексный режим, если хотим переопределить значение, определяемое полярностью светодиода LEDB. Для включения полного дуплекса устанавливаем биты PHCON1.PDPXMD и MACON3.FULDPX.
- Разрешаем приём пакетов

```
#define ENC28J60_SPI_DDR    DDRB
#define ENC28J60_SPI_PORT    PORTB
#define ENC28J60_SPI_CS        (1<<PB4)
#define ENC28J60_SPI_MOSI    (1<<PB5)
#define ENC28J60_SPI_MISO    (1<<PB6)
#define ENC28J60_SPI_SCK    (1<<PB7)

#define ENC28J60_BUFSIZE    0x2000
#define ENC28J60_RXSIZE        0x1A00
#define ENC28J60_MAXFRAME    1500

#define ENC28J60_RXSTART    0
#define ENC28J60_RXEND        (ENC28J60_RXSIZE-1)
#define ENC28J60_TXSTART    ENC28J60_RXSIZE
#define ENC28J60_BUFEND        (ENC28J60_BUFSIZE-1)

uint16_t enc28j60_rxrdpt = 0;

void enc28j60_init(uint8_t *macadr)
{
    // Настраиваем SPI
    ENC28J60_SPI_DDR |= ENC28J60_SPI_CS|ENC28J60_SPI_MOSI|ENC28J60_SPI_S
    ENC28J60_SPI_DDR &= ~ENC28J60_SPI_MISO;
    enc28j60_release();

    SPCR = (1<<SPE)|(1<<MSTR);
    SPSR |= (1<<SPI2X);

    // Выполняем сброс
    enc28j60_soft_reset();

    // Настраиваем размер буфера для приёма пакетов
    enc28j60_wcr16(ERXST, ENC28J60_RXSTART);
    enc28j60_wcr16(ERXND, ENC28J60_RXEND);

    // Указатель для чтения принятых пакетов
    enc28j60_wcr16(ERXRDPT, ENC28J60_RXSTART);
    enc28j60_rxrdpt = ENC28J60_RXSTART;

    // Настраиваем MAC
    enc28j60_wcr(MACON2, 0); // очищаем сброс
    enc28j60_wcr(MACON1, MACON1_TXPAUS|MACON1_RXPAUS| // включаем управл
        MACON1_MARXEN); // разрешаем приём данных
    enc28j60_wcr(MACON3, MACON3_PADCFG0| // разрешаем падднинг
        MACON3_TXCRCEN| // разрешаем рассчёт контрольной суммы
        MACON3_FRMLNEN| //разрешаем контроль длины фреймов
        MACON3_FULDPX);// включаем полный дуплекс
    enc28j60_wcr16(MAMXFL, ENC28J60_MAXFRAME); // устанавливаем максимал
```

```
    enc28j60_wcr(MABBIPG, 0x15); // устанавливаем промежуток между фрейм
    enc28j60_wcr(MAIPGL, 0x12);
    enc28j60_wcr(MAIPGH, 0x0c);
    enc28j60_wcr(MAADR5, macadr[0]); // устанавливаем MAC-адрес
    enc28j60_wcr(MAADR4, macadr[1]);
    enc28j60_wcr(MAADR3, macadr[2]);
    enc28j60_wcr(MAADR2, macadr[3]);
    enc28j60_wcr(MAADR1, macadr[4]);
    enc28j60_wcr(MAADR0, macadr[5]);

    // Настраиваем PHY
    enc28j60_write_phy(PHCON1, PHCON1_PDPXMD); // включаем полный дуплек
    enc28j60_write_phy(PHCON2, PHCON2_HDLDIS); // отключаем loopback
    enc28j60_write_phy(PHLCON, PHLCON_LACFG2| // настраиваем светодиодик
        PHLCON_LBCFG2|PHLCON_LBCFG1|PHLCON_LBCFG0|
        PHLCON_LFRQ0|PHLCON_STRCH);

    // разрешаем приём пакетов
    enc28j60_bfs(ECON1, ECON1_RXEN);
}
```

Отправка пакетов

Для отправки пакета, записываем указатели на его начало и конец в регистры ETXST и ETXND. Перед пакетом должен находиться управляющий байт, в котором можно переопределить некоторые настройки MAC для отправки этого пакета. После того, как пакет будет отправлен, после его конца будет записан блок, содержащий статус передачи.

Если хотим отправить пакет с настройками по-умолчанию, младший бит (POVERRIDE) управляющего байта должен быть сброшен.

```
void enc28j60_send_packet(uint8_t *data, uint16_t len)
{
    // Ждём готовности передатчика
    while(enc28j60_rcr(ECON1) & ECON1_TXRTS)
        ;

    // Записываем пакет в буфер
    enc28j60_wcr16(EWRPT, ENC28J60_TXSTART);
    enc28j60_write_buffer((uint8_t*)"\x00", 1);
    enc28j60_write_buffer(data, len);

    // Устанавливаем указатели ETXST и ETXND
    enc28j60_wcr16(ETXST, ENC28J60_TXSTART);
    enc28j60_wcr16(ETXND, ENC28J60_TXSTART + len);

    // Разрешаем отправку
    enc28j60_bfs(ECON1, ECON1_TXRTS);
}
```

Правда, в ENC28J60 есть баг, из-за которого бит TXRTS может не сбрасываться при серьёной ошибке передачи пакета. Соответственно, готовности передатчика мы не дождёмся. Еррата рекомендует проверять бит EIR.ERIF, и, если он установлен, выполнять сброс передатчика. Для этого изменим код вот так:

```
//...
while(enc28j60_rcr(ECON1) & ECON1_TXRTS)
{
    // При ошибке, сбрасываем передатчик
    if(enc28j60_rcr(EIR) & EIR_TXERIF)
    {
        enc28j60_bfs(ECON1, ECON1_TXRST);
        enc28j60_bfc(ECON1, ECON1_TXRST);
    }
}
//...
```

Приём пакетов

ENC28J60 записывает пакеты в кольцевой буфер в виде связанного списка:

Адрес первого непрочитанного пакета храниться в регистрах ERXRDPT. Забрав пакет, микроконтроллер записывает в ERXRDPT адрес следующего пакета. После этого место, которое занимал пакет считается свободным и ENC28J60 может использовать его для приёма новых пакетов.

Статус приёма — длина пакета (2 байта) и различные флаги (тоже 2 байта). Из флагов нас интересует только бит 7 — приём успешно завершён.

Все принятые пакеты ENC28J60 записывает в буфер с выравниванием на 2 байта. Таким образом, адрес пакета всегда чётный.

Для того, чтобы забрать принятый пакет, микроконтроллер делает следующее:

- Смотрит сколько принято пакетов (в регистре EPKTCNT).
- Читает пакет из буфера (по адресу ERXRDPT).
- Записывает в ERXRDPT адрес следующего пакета.
- Уменьшает значение счётчика пакетов установкой бита ECON2.PKTDEC.

Ну и последний на сегодня баг ENC28J60 — при записи чётного значения в регистр ERXRDPT, ENC28J60 может повредить данные в буфере (кстати, адрес пакета всегда чётный из-за выравнивания). Еррата рекомендует записывать в ENC28J60 всегда нечётное значение. Стоп, а как же мы узнаем

откуда брать новый пакет? Придётся хранить это значение в памяти микроконтроллера. Но ERXRDPT мы всё равно должны записывать, чтобы ENC28J60 знал сколько памяти доступно для приёма пакетов. Только записывать будем не адрес следующего пакета, а на адрес на 1 байт выше.

```
// "Правильное" значение ERXRDPT
uint16_t enc28j60_rxrdpt = 0;

uint16_t enc28j60_recv_packet(uint8_t *buf, uint16_t buflen)
{
    uint16_t len = 0, rxlen, status, temp;

    // Есть ли принятые пакеты?
    if(enc28j60_rcr(EPKTCNT))
    {
        // Считываем заголовок
        enc28j60_wcr16(ERDPT, enc28j60_rxrdpt);

        enc28j60_read_buffer((void*)&enc28j60_rxrdpt, sizeof(enc28j60_rx
        enc28j60_read_buffer((void*)&rxlen, sizeof(rxlen));
        enc28j60_read_buffer((void*)&status, sizeof(status));

        // Пакет принят успешно?
        if(status & 0x80)
        {
            // Выбрасываем контрольную сумму
            len = rxlen - 4;

            // Читаем пакет в буфер (если буфера не хватает, пакет обрез
            if(len > buflen) len = buflen;
            enc28j60_read_buffer(buf, len);
        }

        // Устанавливаем ERXRDPT на адрес следующего пакета - 1
        temp = (enc28j60_rxrdpt - 1) & ENC28J60_BUFEND;
        enc28j60_wcr16(ERXRDPT, temp);

        // Уменьшаем счётчик пакетов
        enc28j60_bfs(ECON2, ECON2_PKTDEC);
    }

    return len;
}
```

Заключение

Готовую библиотеку можно взять здесь.

Уфф, ну вот вроде и всё про ENC28J60 :)

В следующей части напишем простенькое приложение работающее с компом по UDP.

update: Микрочип время от времени обновляет даташит. Последнюю версию можно найти здесь. Статья написана на основе документа ревизии A.

Все статьи цикла

- Подключение микроконтроллера к локальной сети: Теория
- **Подключение микроконтроллера к локальной сети: работаем с ENC28J60**
- Подключение микроконтроллера к локальной сети: UDP-сервер
- Подключение микроконтроллера к локальной сети: UDP-клиент
- Подключение микроконтроллера к локальной сети: Широковещательные сообщения и DHCP
- Подключение микроконтроллера к локальной сети: TCP-клиент
- Подключение микроконтроллера к локальной сети: HTTP и CGI