Исследование скорости передачи данных в беспроводных сетях Nanonet

Алексей МОЩЕВИКИН,
к. ф.-м. н.
alextmou@lab127.karelia.ru

В статье отражены результаты проведенного исследования режимов работы трансивера NA1TR8 (Nanonet) и особенностей его программирования для достижения максимальной пропускной способности при передаче данных между двумя узлами. Показано, что полезная скорость передачи при использовании небольших кадров (длиной до 128 байтов) не превышает 628 кбит/с и 1126 кбит/с для режимов с битовыми скоростями 1 и 2 Мбит/с соответственно.

Введение

Технология Nanonet разрабатывается компанией Nanotron Technologies (h ttp://ww w. nanotron. com) и позиционируется на рынке как беспроводная сеть «последней мили», которая может быть применена, в частности, для сбора информации с различного рода датчиков и устройств. Максимальная заявленная битовая скорость составляет 2 Мбит/с, однако в некоторых случаях для увеличения дальности и повышения надежности передачи данных имеет смысл использовать скорости 1 Мбит/с и 500 кбит/с.

Технология Nanonet поддерживает два режима доступа к среде передачи: TDMA (с разделением по времени) и CSMA/CA (с конкуренцией за право передачи и механизмом предотвращения коллизий). TDMA подразумевает необходимость программно-аппаратной организации узла-арбитра, который будет выдавать разрешение остальным узлам по очереди передавать в какие-то определенные моменты времени. В CSMA/CA момент старта передачи не определен заранее, поэтому в работу трансиверов вводятся специальные процедуры, максимально предотвращающие возможность почти одновременного начала трансляции своих кадров двумя узлами.

Определение максимальной скорости передачи кадров

В расширенном режиме CSMA/CA узел — инициатор передачи (1) — сначала генерирует кадр запроса на передачу Req2S, извещая все другие узлы в сети о своем намерении отослать какому-то определенному узлу (2) кадр с данными. Если в сети присутствует адресат (2), то он должен ответить на запрос Req2S согласием Clr2S принять следующий кадр (подтверждение запроса на передачу). Такая процедура необходима для того, чтобы, во-первых, обеим сторонам убедиться в возможности осуществления взаимосвязи и, во-вторых, заставить все остальные узлы в сети прекратить на некоторое время попытки завладеть средой передачи. После этого узел (1) отсылает кадр с данными Data и ждет от узла (2) специального пакета подтверждения приема данных Ack.

Для определения максимально возможной скорости передачи кадров в режиме с гарантированной доставкой имеет смысл воспользоваться усеченным режимом CSMA/CA, в котором отсутствует процедура «установления канала связи» (Req2S-Clr2S), но каждый переданный кадр с данными Data подтверждается кадром Ack (см. рис. 1).

Рис. 1. Кадр подтверждения после передачи данных

В случае отсутствия подтверждения в течение определенного промежутка времени узел-отправитель посылает кадр вновь до тех пор, пока транзакция не будет успешно завершена или не истечет счетчик попыток.

Документация к трансиверам NA1TR8[1] позволяет выделить ряд временных интервалов, из которых состоит цикл взаимодействия между двумя узлами сети (так как целью данного исследования является вычисление максимальной скорости передачи данных, будем считать, что условия радиосвязи позволяют отсылать и принимать абсолютно все кадры без ретрансляции).

Рис. 2. Последовательность действий при получении команды «начать передачу»

После получения трансивером команды начать передачу startTxCmd, он должен выдер-

жать фиксированную паузу (CIFS, Carrier Sense InterFrame Space) в 24 микросекунды для проверки среды на незанятость, а затем, следуя алгоритму предотвращения коллизий (CSMA/CA), подождать еще некоторое время, определяемое генератором случайных чисел (от 0 до 7 временных интервалов Backoff Time Slots по 24 мкс для первой попытки отослать кадр). Введение элемента случайности и последовательное увеличение диапазона генерации интервала ожидания для повторных попыток эффективно позволяет избегать коллизий. После окончания передачи информационного кадра приемная сторона должна через 8 микросекунд (Switching InterFrame Space) послать в эфир подтверждение Ack. Учитывая время прохождения сигнала между трансиверами, временной промежуток

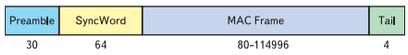

Рис. 3. Общий формат кадра

между кадрами Data и Ack может составлять до 24 мкс.

Общий формат кадра представлен ниже.

Преамбула (Preamble) и конечный ограничитель кадра (Tail) передаются за 30 мкс и 4 мкс соответственно. Синхронизирующая последовательность (SyncWord) состоит из 64 битов, длительность ее передачи зависит от битовой скорости, установленной в трансивере (1 или 2 Мбит/с).

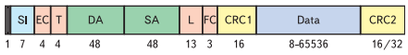

Рис. 4. Структура кадра с данными

В поле MAC Frame переменной длины может помещаться информационный кадр Data или кадр подтверждения Ack.

Кадр с данными состоит из следующих полей:

- SI (Scrambler Init, 8 битов, старший бит не определен) — биты инициализации скремблера на приемной стороне;
- EC (Encryption Control, 4 бита) — биты управления шифрованием;
- T (Type, 4 бита) — тип кадра Data (существуют 6 типов кадров: данные Data, подтверждение принятых данных Ack, широковещательный BrdCast, синхронизации времени TimeB, запрос на передачу Req2S, подтверждение запроса на передачу Clr2S);
- DA (Destination Address, 48 битов) — адрес получателя;
- SA (Source Address, 48 битов) — адрес отправителя;
- L (Length, 13 битов) — длина поля данных (в байтах);
- FC (Frame Control, 3 бита) — поле управления потоком кадров;
- CRC1 (Cyclic Redundancy Check, 16 битов) — контрольная сумма заголовка;

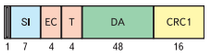

Рис. 5. Структура кадра подтверждения

- Data (1–8192 байтов) — поле данных;
- CRC2 (16 или 32 бита) — контрольная сумма поля данных.

Кадр подтверждения принятых данных намного короче и состоит из:

Т а б л и ц а 1. Время передачи для битовых скоростей 1 Мбит/с и 2 Мбит/с

Название поля (временного интервала)	Битовая длина, бит / длительность, мкс	Время передачи [мкс] при битовойскорости	
		1 Мбит/с	2 Мбит/с
CIFS	24 мкс	24	24
random	(0..7)*24 мкс, 84 в среднем	84	84
Preamble	30 мкс	30	30
SyncWord	64 бит	64	32
SI+EC+T+DA+SA+L+ +FC+CRC1+CRC2(32b)	176 бит	176	88
Data (128 / 8192 байтов)	128*8 б / 8192*8 байт	1024 / 65 536	512 / 32 768
Tail	4 мкс	4	4
SIFS	<24 мкс (8)	8	8
Preamble	30 мкс	30	30
SyncWord	64 бит	64	32
SI+T+DA+CRC1	80 бит	80	40
Tail	4 мкс	4	4
ИТОГО:	Длительность кадра	1592 мкс / 66 104 мкс	888 мкс / 33 144 мкс
	Макс. скорость передачи данных	628 кбит/с / 968 кбит/с	1126 кбит/с / 1931 кбит/с

- поля инициализации скремблера,
- поля управления шифрованием (все биты выставлены в 0),
- поля тип кадра Ack,
- адреса получателя,
- а также поля контрольной суммы.

Поскольку трансиверы NA1TR8 могут работать в разных режимах битовой скорости 1 Мбит/с и 2 Мбит/с, время, затрачиваемое на передачу одного кадра с полем данных в 128 байтов и 8192 байта (максимально возможная длина), указано в таблице 1[2].

Создание устройств, способных передавать информацию кадрами длиннее 128 байтов, затруднено вследствие необходимости проектировать аппаратное и программное обеспечение модулей приемо-передатчиков с согласованными скоростями обмена данными между чипом NA1TR8 и управляющим микроконтроллером. Размер одного банка (буфера) памяти трансивера NA1TR8 составляет 128 байтов. В случае возникновения потребности передать кадр бóльшего размера, необходимо разработать дополнительные процедуры управления буферами приема и передачи данных. При попеременной передаче в эфир 64 байтов данных из верхней или нижней половины буфера надо успевать записывать новую информацию в другую половину, не используемую в данный момент. На приемной стороне, соответственно, подобные действия должны быть выполнены при получении данных. Далее в статье этот режим передачи информации в длинных кадрах обсуждаться не будет.

Ориентируясь на полученные цифры, можно утверждать, что полезная скорость передачи данных при использовании кадров небольшой длины (128 байтов) не может превышать 628 кбит/с и 1126 кбит/с для битовых скоростей 1 Мбит/с и 2 Мбит/с соответственно.

Таким образом, даже если трансиверы будут постоянно занимать среду передачи, то все равно около 40% пропускной способности канала будет уходить на накладные расходы.

Ограничение максимальной скорости передачи данных вследствие программно-аппаратных особенностей устройств

Автономное устройство, снабженное приемо-передатчиком Nanonet, обычно состоит из двух главных узлов, связанных между собой интер-

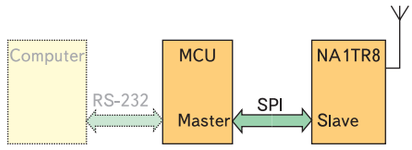

Рис. 6. Блок-схема устройства, содержащего трансивер Nanonet

фейсом SPI, — управляющего микроконтроллера (MCU) и трансивера (чип NA1TR8 или модуль Nanopan) (рис. 6).

В случае необходимости управлять устройством или сохранять и обрабатывать поступающую по радиоканалу информацию микроконтроллер подсоединяется к персональному компьютеру с помощью какого-нибудь проводного интерфейса (например, RS-232 или USB, на рисунке обозначено штриховой линией серым цветом). Существует также способ подключения трансивера к компьютеру напрямую (без использования MCU). Для этого, например, применяется преобразователь сигналов LPT порта в SPI интерфейс (такие переходники поставляются со стартовым отладочным комплектом).

В данной статье подробно разобран случай использования микроконтроллеров Atmel с ядром популярной серии AVR (например, ATmega32L). Из большого набора периферийных узлов в нем используется SPI интерфейс (тактовая частота до 4 МГц) для связи с трансивером и универ-

[1] NanoNET System Specifications PHY/MAC, ver.1.04 (разделы 3 «Описание форматов кадров», 7.2 «Алгоритм CSMA/CA», приложение A1 «Атрибуты и константы»).

[2] Существует еще, как уже было отмечено, режим битовой скорости 500 кбит/с, для достижения максимально возможных скоростей обмена данными его использовать нецелесообразно.

сальный приемо-передатчик USART для связи с компьютером через com-порт.

Микроконтроллер управляет работой чипа, записывая данные (инструкции) в регистры трансивера. Для того чтобы записать и одновременно считать n байтов из текущего банка памяти NA1TR8 по интерфейсу SPI[3], необходимо передать n+2 байта. В первом байте указывается вид операции (чтение/запись) и количество байтов для передачи, второй содержит адрес памяти, относительно которого будет выполняться операция, далее следуют собственно данные.

Например, для изменения значения одного из регистров NA1TR8 требуется передать 3 байта по интерфейсу SPI, затратив на эту передачу как минимум 6 микросекунд (SPI@4МГц, один бит передается за 0,25 мкс). Кроме этого, управляющим микроконтроллером будет дополнительно израсходовано время, связанное с подготовкой данных и организацией циклов ввода-вывода.

В поставляемом с отладочными наборами трансиверов Nanonet программном обеспечении передачу данных в NA1TR8 выполняет функция transSPI ().

```
void transSPI (CMDT command, MyByte8T offset,
MyByte8T *DataStr, MyByte8T anz)
{
// объявление переменных не приведено
if (command==WRITE_CMD)          // если получена
// команда «запись в трансивер»
{
SetSPISS (LOW);          // подключить чип NA1TR8
// к интерфейсу SPI
cmd=0x80 | anz;          // сформировать старший байт
// из количества anz
// байтов для чтения и кода операции (0x80)
SPIReadWrite (cmd);      // передать 1-ый байт
SPIReadWrite ((unsigned char) offset);// передать 2-ой
// байт (адрес памяти)
for (i=0;i<anz; i++)     // передать anz байтов данных
{
SPIReadWrite (DataStr [i]);
}
SetSPISS (HIGH);         // отключить чип NA1TR8
// от интерфейса SPI
}
else        // если получена команда «чтение в MCU»
{
SetSPISS (LOW);          // подключить чип NA1TR8
// к интерфейсу SPI
cmd=~0x80 & anz;         // сформировать старший байт
SPIReadWrite (cmd);      // передать 1-ый байт
SPIReadWrite ((unsigned char) offset);// передать 2-ой
// байт (адрес памяти)
for (i=0;i<anz; i++)     // получить в массив DataStr [i]
// anz байтов данных
{
ucReadback=SPIReadWrite ((unsigned char) 0xff);
DataStr [i] =ucReadback;
}
SetSPISS (HIGH);         // отключить чип NA1TR8
// от интерфейса SPI
}
}
```

В случае, когда четвертым параметром (количество байтов) передана «1», в теле функции три раза произойдет вызов SPIReadWrite ().

Следует обратить внимание, что функция transSPI () носит аппаратно независимый характер (не зависит от типа микроконтроллера и способа его подключения к трансиверу). В отличие от нее функции SetSPISS () и SPIReadWrite () выполняют уже конкретные действия с физическими сигналами на линиях, связывающих MCU и NA1TR8, следовательно, их код должен быть аппаратно зависим.

Например, в SPIReadWrite () происходит вызов функции spi (), код которой приведен ниже.

```
unsigned char spi (unsigned char data)
{
SPDR=data;
while ((SPSR & (1<<SPIF)) ==0);
return SPDR;
}
```

Переданный как параметр функции, байт data записывается в регистр данных (SPDR) микроконтроллера ATmega32L (значение data необходимо передать в трансивер). Далее в цикле ядро микроконтроллера постоянно читает старший бит (SPIF, флаг прерываний SPI) регистра SPSR (регистр состояния SPI). Факт изменения значения флага SPIF с 0 на 1 свидетельствует о произошедшем окончании цикла передачи байта по интерфейсу SPI. Поскольку в интерфейсе SPI передача информации идет сразу по двум линиям MISO и MOSI в обоих направлениях, то после передачи 8 битов в регистре SPDR оказывается байт, который был получен из трансивера. Именно его и возвращает функция spi ().

Приведенные выше листинги функций демонстрируют использование языка Си для программирования микроконтроллеров Atmel серии AVR (например, в среде разработчика CodeVisionAVR C Compiler, ht tp://w ww.hpinfotech.c om). Использование языка Ассемблер при создании многоуровневых приложений с передачей параметров в функции хотя и менее удобно, но заметно повышает быстродействие программ за счет разумной оптимизации кода.

Так, для записи в трансивер трех байтов по интерфейсу SPI необходимо выполнить следующие операции (в скобках указано число тактов микроконтроллера ATmega32L, необходимое для выполнения того или иного действия):

- подготовить байт команды, записать его во временный регистр (2 такта);
- подготовить байт адреса, записать его во временный регистр со смещением +1 относительно байта команды (~4 такта);
- подготовить байт данных, записать его во временный регистр со смещением +2 относительно байта команды (3 такта);
- в цикле брать последовательно каждый из трех байтов, записывать их в регистр SPDR и ожидать окончания передачи (выставления флага SPIF) (3* (~2 мкс, затраченные на передачу байта по SPI, +~5 тактов)).
- Итого, при тактовых частотах SPI — 4 МГц и микроконтроллера — 8 МГц общее время выполнения пересылки трех байтов должно составить немногим более 9 микросекунд.

Следующий пример демонстрирует неэффективность обычного подхода программирования на Си с множеством вложенных функций и компиляцией с опциями по умолчанию. В примере разобран дизассемблированный листинг упомянутой выше transSPI (), вернее, первой ее половины, связанной с записью данных в чип nanonet (в столбце справа указано количество тактов микроконтроллера, затрачиваемое на каждую операцию).

Отсюда хорошо видно, что унифицированная для разного аппаратного обеспечения функция SetSPISS () выполняется за 39 тактов микроконтроллера, тогда как в ней происходит всего лишь выставление сигнала логического нуля на одной

```
; void transSPI(CMDT command, MyByte8T offset, MyByte8T *DataStr, MyByte8T anz)
                                        ;{
                                        ; сохранение части параметров в ОЗУ
_transSPI:      CALL SaveParam          ; MCU                                   16[4]
                LDD R26,Y+8             ; чтение байта «команда»                2
                CPI R26,LOW(0x80)       ; if (command==WRITE_CMD)               1
                                        ; если команда «запись», переход на
                BRNE _0xA9              ; след. оператор                        1
                CALL SetSPISS           ; вызов SetSPISS(LOW);                  39
                ORI R30,0x80            ; cmd=0x80 | anz;                       1
                RCALL _SPIReadWrite     ; SPIReadWrite(cmd);                    59
                RCALL _SPIReadWrite     ; SPIReadWrite((unsigned char)offset);  59
                LDI R18,LOW(0)          ; for (i=0;i<anz; i++)                  1
                LDI R19,HIGH(0)         ; i — двухбайтовое целое                1
                                        ; если i=0, т. е. если все байты
_0xAB:          CALL Test_i             ; переданы                              15
                BRGE _0xAC              ; выход                                 1
                MOV R30,R18                                                     2
                MOV R31,R19
                                        ; расчет указателя на очередной байт
                CALL CalcAddress        ; данных                                14
                ST -Y,R30                                                       2
                RCALL _SPIReadWrite     ; SPIReadWrite(DataStr[i]);             59
                SUBI R18,LOW(-1)                                                1
                SBCI R19,HIGH(-1)                                               1
                                        ; переход на передачу следующего
                RJMP _0xAB              ; байта данных                          2
                ...                                                             ...
                RET                                                             4
;}
```

[3] NanoNET Serial Peripheral Interface Specifications, ver. 1.06.

[4] Описание микроконтроллера ATmega32L "8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash. ATmega32, ATmega32L", раздел "Instruction Set Summary"

из цифровых линий, а SPIReadWrite () за 59 тактов (более 7 микросекунд).

При проектировании программного обеспечения для микроконтроллеров AVR следует помнить, что вызов и возвращение из любой процедуры (инструкции CALL и RET) выполняются за 4 такта, следовательно, длятся суммарно одну микросекунду. Использование вложенных процедур улучшает читаемость созданного кода, но многократно усиливает эффект бесполезного расходования времени.

Для того чтобы зарегистрировать максимально возможную скорость передачи данных между двумя узлами, была написана программа, которая как можно чаще отсылает неизменные 128 байтов данных и автоматически получает на них подтверждение о приеме от удаленного трансивера. Структура такой программы представлена на рис. 7.

После установки режимов работы микроконтроллера ATmega32L, инициализации приемо-передатчика NA1TR8 и подготовки кадра программа входит в бесконечный цикл последовательных запусков функции DilSendMessage() — передачи кадра в эфир. Количество переданных данных может контролироваться с помощью прерываний от внутреннего таймера микроконтроллера, отмеряющего секундные интервалы.

```
void DilSendMessage (MyByte8T *buff, MyByte8T len)
// функция отсылки кадра
// первый параметр — указатель на строку байтов,
// второй — длина строки
{
// объявление переменных, проверка параметров
lengthCB [0] = 0x80;
lengthCB [1] = 0;
// установить длину кадра (0x80 — 128 байт)
setTxFrmType (frmTypeData);
// установка формата кадра (frmTypeData — кадр
// с данными)
ARQset = TRUE;
// установить режим повторной передачи кадра
// в случае неуспешной предыдущей попытки
setTxARQmax (3);
// установить максимальное количество попыток пе
// редачи кадра (3)
// перед передачей следующего кадра необходимо
// убедиться
// в успешной передаче предыдущего
if (WaitForTransmitterReady ())          // 890 тактов
{
packet_a++;                  // 15 тактов
// подсчет количества переданных кадров
}
else
{
// вывести сообщение «проблемы с трансмиттером»
}
setTxLengthCtrlB (lengthCB);             // 1272 такта
// установить длину кадра
cmd = 7;  // 1 такт
transSPI (WRITE_CMD, STxBufferCmd_O, &cmd, 1);
// 340 тактов
// выдать команду начать передачу:
// записать значение 7 в регистр STxBufferCmd_O
// действия по проверке различных ошибок
return;
}
```

Ключевым моментом в каждой итерации внешнего цикла является ожидание разрешения начала передачи, реализованное в функции

```
WaitForTransmitterReady ().
MyBoolT WaitForTransmitterReady ()
{
// Объявление локальных переменных
if (TransmitterIsActive == TRUE)    // 12 тактов
// если трансмиттер включен
{
```

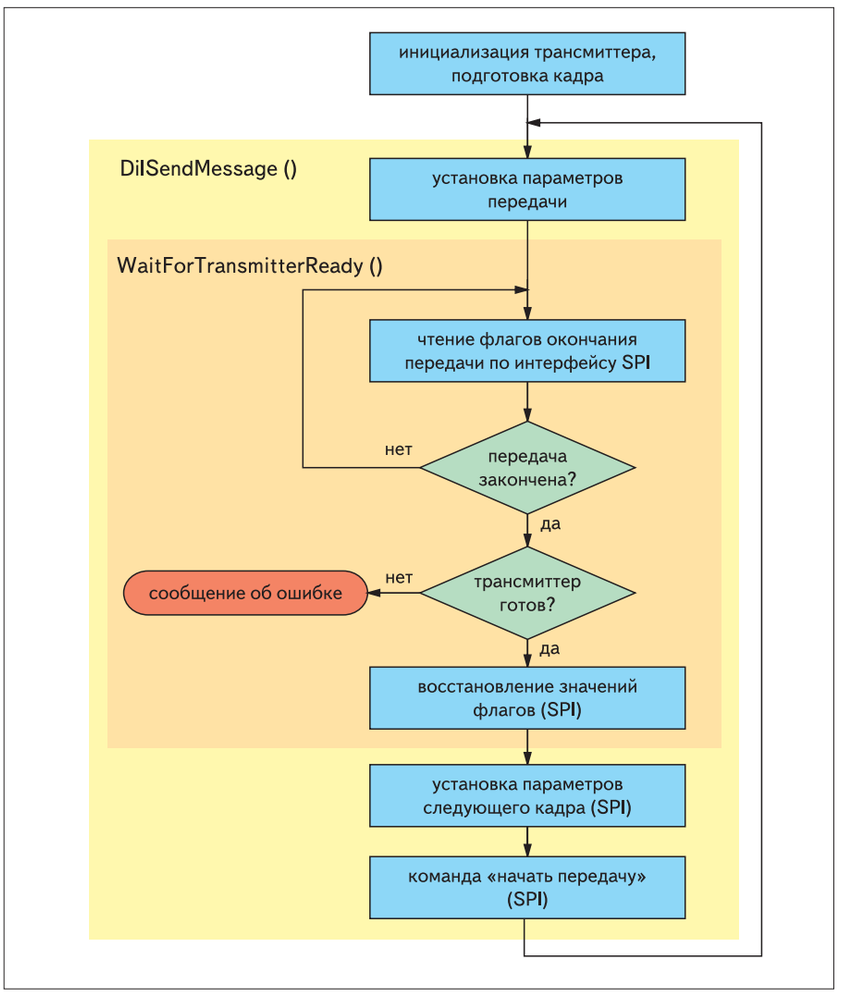

Рис. 7. Алгоритм программы для оценки максимально возможной скорости передачи данных между двумя узлами

```
// 9 тактов
for (i=0;i<TXWAIT;i++)                // 16 тактов
{
getTxIRQ (&irqFlag);     // 400 тактов
if ((irqFlag & TXENDIRQ) == TXENDIRQ)          //  4
// такта
{
if ((irqFlag & TXURUNIRQ) == TXURUNIRQ)       //  5
// тактов
// проверка корректности предыдущей передачи
// если некорректное завершение,
{
sTXstat. urunCount++;                  // 40 тактов
// заполнение массива статистических данных
}
resetTxIRQ (&irqFlag);  // 793 такта
TransmitterIsActive=FALSE;             // 3 такта
// трансмиттер выключить
break;      // 2 такта
}
// 22 такта
}
if (i >= TXWAIT)        // 17 тактов
// если трансмиттер слишком долго не освобождает-
ся // для передачи
{
sEXEPflags. TXexep = 1;                // 6 тактов
// выставить соответствующий флаг
return FALSE;              // 3 такта
// возврат «трансмиттер не готов»
}
}
return TRUE;               // 1 такт
// возврат «трансмиттер готов к передаче»        // 12
// тактов
}
```

Перед стартом передачи очередного информационного кадра необходимо убедиться в том, что предыдущий кадр был успешно отправлен. Сигналом этого события является выставленный флаг TXENDIRQ в регистре 0x11 (состояния прерываний трансмиттера).

В цикле по переменной i периодически проверяется состояние регистра 0x11 чипа NA1TR8. Если предыдущая передача была завершена, функция WaitForTransmitterReady () вернет значение TRUE. Длительность цикла, как уже подчеркивалось ранее, определяется тактовой частотой микроконтроллера и качеством написанного и скомпилированного кода.

Для случая (MCU@8МГц/SPI@4МГц) и не завершившейся на момент чтения регистра 0x11 передачей (такой цикл назовем «неудачным») одна итерация выполняется за 442 такта (16+400+4+22) или 55 мкс.

Если результатом выполнения функции getTxIRQ () будет выставленный бит TXENDIRQ (передача закончена), то необходимо, во-первых, его сбросить, а, во-вторых, прекратить выполнять тело цикла и перейти к процедурам подготовки передачи нового пакета. Эти действия плюс возврат из функции WaitForTransmit terReady () занимают 890 тактов (111 мкс).

Как уже было сказано выше, каждые 55 микросекунд происходит очередное тестирование состояния трансмиттера. Количество таких опро-

сов можно заранее определить, так как оно зависит от длины предыдущего кадра. Время простоя системы после выставленного флага TXENDIRQ до момента его проверки внутри функции getTxIRQ () варьируется в пределах от 1 до 442 такта, что составляет в среднем 228мкс. Погрешность расчетов в этом случае не превысит 28 мкс, что составляет малую часть от общей длительности цикла передачи. Далее, следуя вычислениям, на завершение функции getTxIRQ () (после обнаруженного сигнала TXENDIRQ) и возврат из функции WaitForTransmitterReady() уходит примерно 50+890 тактов или 117 мкс.

Таким образом, после момента окончания предыдущей передачи и возвратом из WaitForTran smitterReady () в среднем проходит до 145 мкс. В общем, скрупулезный подсчет количества тактов микроконтроллера между моментом чтения выставленного бита TXENDIRQ в регистре 0x11 трансмиттера NA1TR8 и моментом получения этим чипом команды начать передачу следующего кадра приводит к числу 2518 (890+15+1272+1+340) тактов. При указанных условиях этот временной промежуток будет длиться около 315 микросекунд.

Хотелось бы еще раз обратить внимание на постоянное использование неоптимизированной функции transSPI (), заставляющей микроконтроллер работать «вхолостую». В таблице 2 приведены значения числа тактов, затрачиваемых на выполнение этой функции для разного количества переданных байтов данных, и двух разных тактовых частотах SPI интерфейса — 4 и 16 МГц. К сожалению, ATmega32L не поддерживает обмен данными по SPI на частотах выше 4 МГц; для увеличения скорости необходимо использовать либо внешние микросхемы, реализующие интерфейс SPI, либо микроконтроллеры других производителей (например, фирмы Freescale)

При необходимости записать или считать сразу 6 байтов из приемо-передатчика (например, аппаратный MAC-адрес источника или отправителя кадра), микроконтроллер будет отрабатывать эту процедуру примерно в течение 100 микросекунд (!).

Для приведенной выше программы посылки кадров в бесконечном цикле промежуток между ними будет составлять примерно 315 мкс, что, в сумме с временем, затрачиваемым на передачу, составляет 1907 мкс для режима 1 Мбит/с и 1203 мкс для битовой скорости 2 Мбит/с. При этом максимальные полезные скорости передачи (для 128-байтных пакетов) получаются 524 кбит/с и 831 кбит/с соответственно.

Проведенные эксперименты по регистрации максимальных скоростей передачи дают следующие результаты (табл. 3).

Первые 4 строки таблицы соответствуют условиям, при которых сторона-отправитель выжидает случайное время перед началом передачи (в среднем 96 мкс для первой попытки), вторые 4 — ситуации, когда кадр передается сразу же после окончания межкадрового промежутка. Данные в последнем столбце не учитывают затраты времени на подготовку внешним микроконтроллером следующего кадра и отражают предельно достижимые скорости передачи. Следует отметить, что режим работы CSMA без использования алгоритма предотвращения коллизий не рекомендовано использовать в сетях с несколькими активными узлами.

Таблица 2. Скоростные параметры обмена данными при использовании микроконтроллера ATmega32L

Количество передаваемых по SPI байтов данных (не включая байты команды и адреса)	Число тактов микроконтроллера ATmega32L (при тактовой частоте MCU 8 МГц, 8 тактов = 1 мкс)	
	SPI@4МГц	SPI@16МГц
0	276	252
1	373	337
2	470	422
3	567	507
4	664	592
5	761	677
6	858	762

Таблица 3. Результаты экспериментов по регистрации максимальных скоростей передачи данных

Режим работы трансивера	Поле данных кадра содержит		Битовая скорость		Количество кадров в секунду / скорость передачи данных	
	1 байт	128 байтов	1 Мбит/с	2 Мбит/с	продемонстр.	максимальн.
SPI@4МГц / MCU@8МГц, усеченный режим CSMA/CA с предотвращением коллизий	+		+		1141 /8,9 кбит/с	1656 /12,9 кбит/с
	+			+	1420 / 11 кбит/с	2500 / 19,5 кбит/с
		+	+		530 / 530 кбит/с	617 / 617 кбит/с
		+		+	840 /840 кбит/с	1101 / 1101 кбит/с
SPI@4МГц / MCU@8МГц, усеченный режим CSMA без алгоритма предотвращения коллизий	+		+		1300 / 10,2 кбит/с	1969 / 15,4 кбит/с
	+			+	1511 / 11,8 кбит/с	3289 / 25,7 кбит/с
		+	+		560 / 560 кбит/с	656 / 656 кбит/с
		+		+	918 / 918 кбит/с	1232 / 1232 кбит/с

Зафиксированные скорости передачи данных 530 кбит/с и 840 кбит/с близки к рассчитанным выше (524 кбит/с и 831 кбит/с) и находятся в пределах погрешностей.

Достижение максимально возможных скоростей передачи данных

Анализируя все вышеизложенное, можно выделить ряд рекомендаций по разработке и созданию программно-аппаратных модулей, конструируемых на основе трансиверов стандарта Nanonet. Для достижения высоких скоростей передачи данных необходимо выполнять следующие действия:

1. Оптимизировать быстродействие аппаратно-зависимых низкоуровневых процедур (например, с помощью создания части программного кода на языке Ассемблер и тщательного выбора способов передачи параметров в такие процедуры).
2. Использовать максимально высокие тактовые частоты для взаимодействия по интерфейсу SPI (в некоторых случаях применяя специализированные контроллеры SPI).
3. Связывать управляющий микроконтроллер и чип приемо-передатчика с компьютером заведомо быстрым интерфейсом (например, USB).
4. Продумывать схемотехнические решения и структуру создаваемых устройств с точки зрения возможности предварительной обработки данных (например, сохранения в энергонезависимой памяти, в режиме отложенной передачи).
5. Использовать кадры большей, чем 128 байтов, длины. Для этого необходимо либо перевести трансивер из режима полного дуплекса в полудуплексный режим (увеличив тем самым размер буферной памяти данных каждого кадра), либо воспользоваться возможностью попеременного последовательного доступа к ее верхней и нижней половинам.[5] БТ

Данное исследование проведено в рамках проекта «Научно-образовательный центр по фундаментальным проблемам приложений физики низкотемпературной плазмы» (RUX0-013-PZ-06), который поддерживается Министерством образования и науки РФ, Американским фондом гражданских исследований и развития (CRDF) и Правительством Республики Карелия, а также частично финансировалось Техническим научно-исследовательским центром Финляндии (VTT) в рамках договорных работ.

[5] Nanonet TRX (NA1TR8) Transceiver Register Description, ver. 1.06. раздел 5 "Двухпортовая память".