Роман Абраш
г. Новочеркасск
E-mail: **arv@radioliga.com**

Книга по работе с WinAVR и AVR Studio

Продолжение. Начало в №1-12/2010; №1/2011

avr/power.h – Поддержка PRR

Некоторые модели микроконтроллеров AVR имеют встроенный регистр PRR (Power Reduction Register – регистр снижения мощности) или несколько таких регистров PRRn. С их помощью имеется возможность отключить часть неиспользуемой программой периферии для снижения потребляемой микроконтроллером мощности.

Этот модуль определяет рад макросов, облегчающих использование PRR. Однако, нормальное их функционирование возможно только для тех микроконтроллеров, которые действительно содержат этот регистр (регистры) и соответствующую периферию; кроме того, иногда одна и та же периферия отличается в наименовании (например, USART и USART0).

Здесь перечислены все соответствующие макросы, однако о наличие поддержки ими выбранного программистом микроконтроллера следует справляться в документации. При попытке использовать неподдерживаемый микроконтроллер будут возникать ошибки компиляции.

power_adc_enable() – включение встроенного АЦП (не путать[35] с битом ADEN в соответствующем регистре)

power_adc_disable() – выключение АЦП

power_lcd_enable() – включение модуля ЖКИ-драйвера

power_lcd_disable() – выключение модуля ЖКИ-драйвера

power_psc0_enable() – включение модуля Power Stage Controller 0

power_psc0_disable() – выключение модуля Power Stage Controller 0

power_psc1_enable() – включение модуля Power Stage Controller 1

power_psc1_disable() – выключение модуля Power Stage Controller 1

power_psc2_enable() – включение модуля Power Stage Controller 2

power_psc2_disable() – выключение модуля Power Stage Controller 2

power_spi_enable() – включение модуля SPI

power_spi_disable() – выключение модуля SPI

power_timer0_enable() – включение таймера 0

power_timer0_disable() – выключение таймера 0

power_timer1_enable() – включение таймера 1

power_timer1_disable() – выключение таймера 1

power_timer2_enable() – включение таймера 2

power_timer2_disable() – выключение таймера 2

power_timer3_enable() – включение таймера 3

power_timer3_disable() – выключение таймера 3

power_timer4_enable() – включение таймера 4

power_timer4_disable() – выключение таймера 4

power_timer5_enable() – включение таймера 5

power_timer5_disable() – выключение таймера 5

power_twi_enable() – включение модуля TWI

power_twi_disable() – выключение модуля TWI

power_usart_enable() – включение модуля USART

power_usart_disable() – выключение модуля USART

power_usart0_enable() – включение модуля USART0

power_usart0_disable() – выключение модуля USART0

power_usart1_enable() – включение модуля USART1

power_usart1_disable() – выключение модуля USART1

power_usart2_enable() – включение модуля USART2

power_usart2_disable() – выключение модуля USART2

power_usart3_enable() – включение модуля USART3

power_usart3_disable() – выключение модуля USART3

power_usb_enable() – включение модуля USB

power_usb_disable() – выключение модуля USB

power_usi_enable() – включение модуля USI

power_usi_disable() – выключение модуля USI

power_vadc_enable() – включение модуля VADC

power_vadc_disable() – выключение модуля VADC

power_all_enable() – включить все модули

power_all_disable() – выключить все модули

Некоторые модели AVR имеют регистр CLKPR предварительного деления тактовой частоты, при помощи которого так же можно снижать потребляемую мощность. Для работы с этим регистром введен особый тип:

```
typedef enum{
    clock_div_1 = 0,
    clock_div_2 = 1,
    clock_div_4 = 2,
    clock_div_8 = 3,
    clock_div_16 = 4,
    clock_div_32 = 5,
    clock_div_64 = 6,
    clock_div_128 = 7,
    clock_div_256 = 8
} clock_div_t;
```

и 2 макроса:

clock_prescale_set(x) – установка значения делителя частоты (**x** – одно из значений **clock_div_t**)

clock_prescale_get() – получить значение текущего делителя (возвращаемый тип результата – **clock_div_t**)

avr/sfr_defs.h – Регистры специальных функций AVR

Данный модуль определяет символьные имена для всех регистров специальных функций, а так же вводит ряд макросов, упрощающих типичные действия над битовыми значениями этих регистров.

Модуль подключается автоматически, если подключен модуль avr/io.h – Специфичные для AVR функции ввода-вывода, поэтому нет необходимости подключать его явно.

Макросы

Все макросы определены, как макросы-функции.

_BV()

_BV(bit) – макрос, преобразующий номер бита в битовую маску. Данная команда используется для работы с битами SFR, выполняя следующее действие: (1<<(bit)).

bit_is_set()

bit_is_set(sfr, bit) – макрос, возвращающий логическое значение ИСТИНА, если бит bit в указанном регистре sfr установлен. Используется для проверки состояния флагов.

bit_is_clear()

bit_is_clear(sfr, bit) – макрос, возвращающий логическое значение ИСТИНА, если бит bit в регистре sfr не установлен. Используется для проверки состояния флагов.

loop_until_bit_is_set()

loop_until_bit_is_set(sfr, bit) – макрос, выполняющий цикл ожидания (путем непрерывного опроса соответствующего sfr) до тех пор, пока бит bit в регистре sfr не будет установлен.

loop_until_bit_is_clear()

loop_until_bit_is_clear(sfr, bit) – макрос, выполняющий цикл ожидания (путем непрерывного опроса соответствующего sfr) до тех пор, пока бит bit в регистре sfr не будет сброшен.

avr/sleep.h – Управление энергосберегающими режимами и режимами «сна»

В этом модуле определен ряд макросов и функций, реализующих общий интерфейс управления режимами «сна».

[35] Данное примечание относится и к другим макросам – они обеспечивают именно отключение/включение питания соответствующей периферии при помощи регистров PRR.

Использование инструкции SLEEP позволяет снизить общее энергопотребление в некоторых применениях. Микроконтроллеры AVR реализуют несколько различных вариантов «сна», для которых в модуле определены следующие константы (о поддержке выбранным контроллером соответствующего режима следует справиться в документации к контроллеру, не все константы определены для любого контроллера):

SLEEP_MODE_IDLE – режим «бездействия» Idle

SLEEP_MODE_PWR_DOWN – режим «отключения» Power Down

SLEEP_MODE_PWR_SAVE – режим «экономии» Power Save

SLEEP_MODE_ADC – режим «снижения шума АЦП» ADC Noise Reduction

SLEEP_MODE_STANDBY – режим «ожидания»

SLEEP_MODE_EXT_STANDBY – расширенный режим «ожидания»

Практически управление режимом энергосбережения сводится к вызову макроса set_sleep_mode() для указания нужного режима и последующего включения этого режима макросом sleep_mode(). Следует помнить, что если нет необходимости остановить работу микроконтроллера вплоть до аппаратного сброса, необходимо разрешить прерывания до выполнения этих макросов.

В некоторых случаях может потребоваться большая гибкость в управлении режимами экономии, для чего предусмотрены отдельные функции sleep_enable(), sleep_disable() и sleep_cpu().

Функции и макросы модуля

set_sleep_mode()

set_sleep_mode(mode) – макрос, подготавливающий микроконтроллер к заданному режиму сна. Параметр mode – одна из определенных ранее констант режима. Этот макрос эквивалентен переходу в режим Idle, т.е. остановки тактового генератора и отключения периферии, если это требуется режимом, не происходит. Для полноценного включения режима требуется второй макрос sleep_mode()

sleep_mode()

sleep_mode() – макрос, переводящий микроконтроллер в режим сна. Его реализация такова, что программист вправе считать, что «возврат» из макроса произойдет уже после «пробуждения» контроллера.

sleep_enable()

void sleep_enable(void) – функция, устанавливающая бит разрешения режима сна (SE-бит)

sleep_disable()

void sleep_disable(void) – функция, сбрасывающая бит разрешения режима сна (SE-бит)

sleep_cpu()

void sleep_cpu(void) – функция, «усыпляющая» микроконтроллер, т.е. эквивалентная команде SLEEP

avr/version.h – Контроль версии avr-libc

Модуль определяет несколько констант, которые могут использоваться в программе для проверки совместимости исходного текста с используемой версией avr-libc.

__AVR_LIBC_VERSION_STRING__ – строчное представление версии библиотеки, например, "1.6.2"

__AVR_LIBC_VERSION__ – версия библиотеки в виде длинного целого числа без знака, например, 10602UL

__AVR_LIBC_DATE_STRING__ – строчное представление даты релиза библиотеки (совпадает с релизом WinAVR), например, "20080403"

__AVR_LIBC_DATE__ - дата релиза библиотеки в виде длинного целого числа без знака, например, 20080403UL

__AVR_LIBC_MAJOR__ – версия библиотеки, например, 1

__AVR_LIBC_MINOR__ – «подверсия» библиотеки, например, 6

__AVR_LIBC_REVISION__ – номер релиза подверсии, например, 2

avr/wdt.h – Поддержка WDT AVR

Модуль реализует необходимый и достаточный минимум макросов, при помощи которых реализуется взаимодействие со сторожевым таймером (WDT) любого микроконтроллера AVR.

Для работы с WDT требуется выполнение определенной четко регламентированной процедуры, которую макросы реализуют автоматически. В частности, в нужных случаях на время происходит глобальное запрещение прерываний. Программист может не беспокоиться о дополнительных действиях, используя предлагаемые макросы.

Единственное, о чем следует помнить, это особенность, присущая новым моделям микроконтроллеров (например, ATMega88 и т.п.): после аппаратного сброса WDT продолжает работать, причем с минимальным предделителем (обычно, период WDT составляет 15 мс). Поэтому следует выключать его (если нужно) как можно раньше. При этом может потребоваться особый подход для того, чтобы определить причину аппаратного сброса. Рекомендуется придерживаться следующего подхода (см. *врезку*):

В приведенном примере для сохранения значения MCUSR использована переменная mcusr_mirror, которая не получает значение 0 по умолчанию (характерное для Си-программ) – секция «.noinit». А функция, которая осуществляет инициализацию этой переменной содержимым MCUSR и одновременно запрещает работу WDT, get_mcusr() размещается в секции автоматической инициализации пользователя «.init3», т.е. будет выполнена сразу после инициализации стека. Функция не имеет пролога и эпилога, поэтому ее код будет помещен в соответствующей секции как inline.

Дополнительно о секциях памяти и порядке их инициализации см. в главе Секции памяти.

Макросы и константы модуля

В модуле определен ряд констант для задания периода переполнения WDT. Так как некоторые модели микроконтроллеров могут иметь и другие значения программируемых периодов, реальное количество констант может быть другим (нетрудно выяснить, взглянув в содержимое файла wdh.h). Из наименования константы очевидно ее смысловое значение, поэтому детальное описание не приводится.

WDTO_15MS
WDTO_30MS
WDTO_60MS
WDTO_120MS
WDTO_250MS
WDTO_500MS
WDTO_1S
WDTO_2S
WDTO_4S
WDTO_8S

Так же в модуле определены три макроса типа функция.

wdt_enable()

wdt_enable(value) – макрос, включающий WDT с периодом переполнения, заданным значением value, которое должно быть одной из рассмотренных ранее констант WDTO_xxxS.

wdt_reset()

wdt_reset() – макрос, осуществляющий сброс счетчика WDT, эквивалент инструкции WDR.

```
#include <stdint.h>
#include <avr/wdt.h>
// объявление «неинициализируемой» переменной
uint8_t mcusr_mirror _attribute_ ((section (I.noinitI)));

// объявление функции «запоминания» значения MCUSR
void get_mcusr(void) \
__attribute__((naked)) \ /* без пролога и эпилога */
__attribute__((section(I.init3I))); /* в секции автоматической инициализации пользователя */

void get_mcusr(void){
mcusr_mirror = MCUSR;
MCUSR = 0;
wdt_disable();
}
```

wdt_disable()

wdt_disable() – макрос, выключающий WDT. Если WDT включен постоянно (т.е. так задано fuse-битами), этот макрос, естественно, не выполняет своей функции.

ВСПОМОГАТЕЛЬНЫЕ МОДУЛИ

К вспомогательным модулям отнесены те из входящих в AVR-LIBC, которые служат для упрощения некоторых действий, часто необходимых в системах на базе микроконтроллеров AVR. Это сервисные функции и функции, разработанные по инициативе сообщества программистов (GCC и WinAVR – это открытый проект, любой доброволец имеет возможность пополнить их функциональность, если она одобрена остальным сообществом).

util/atomic.h – Атомарные и неатомарные участки кода

Модуль реализует средства управления «атомарностью»[36] участков кода.

Данный модуль использует расширения СИ, вводимые стандартом ISO C99, и поэтому может использоваться только с опцией компилятора **–std=c99** или **–std=gnu99**.

Определения модуля

ATOMIC_BLOCK()

ATOMIC_BLOCK(type) – макрос, служащий для определения блока кода, ограниченного фигурными скобками, как исполняющегося атомарно. В качестве параметра type должна использоваться константа ATOMIC_FORCEON или ATOMIC_RESTORESTATE.

Пример использования макроса:

```
ATOMIC_BLOCK(ATOMIC_FORCEON){
ctr_copy = ctr;
}
```

NONATOMIC_BLOCK()

NONATOMIC_BLOCK(type) – макрос, служащий для определения блока кода, ограниченного фигурными скобками, как исполняющегося «не атомарно» (см. ATOMIC_BLOCK()). Это может потребоваться, если внутри большого атомарного блока есть участки, не требующие атомарности.

ATOMIC_RESTORESTATE

ATOMIC_RESTORESTATE – константа для использования в качестве параметра макроса ATOMIC_BLOCK(). Указывает, что в начале блока состояние SREG должно быть запомнено, а в конце блока – восстановлено в прежнем виде. Это гарантирует, что состояние флага глобального разрешения прерываний после завершения атомарного блока будет таким же, как и перед его началом (внутри атомарного блока прерывания, разумеется, запрещены).

ATOMIC_FORCEON

ATOMIC_FORCEON – константа для использования в качестве параметра макроса ATOMIC_BLOCK(). Указывает, что в конце атомарного блока прерывания просто должны быть принудительно разрешены. Использование этого параметра позволяет несколько сократить объем кода и повысить скорость исполнения атомарного блока, однако программист должен быть уверен, что глобальное разрешение прерываний после атомарного блока – это действительно то, что он хочет.

NONATOMIC_RESTORESTATE

NONATOMIC_RESTORESTATE – константа для использования в качестве параметра макроса NONATOMIC_BLOCK(). Указывает на то, что перед началом не-атомного блока должно быть сохранено значение SREG, а после его завершения – восстановлено. Это гарантирует, что блок действительно будет выполнен не атомарно, при этом состояние флага глобального разрешения прерываний после блока будет таким же, как и перед ним (внутри не-атомарного блока прерывания, естественно, разрешены).

NONATOMIC_FORCEOFF

NONATOMIC_FORCEOFF – константа для использования в качестве параметра макроса NONATOMIC_BLOCK(). Указывает на то, что по завершению не-атомарного блока прерывания просто должны быть принудительно запрещены. Использование этого параметра позволяет несколько сократить объем кода и повысить скорость исполнения не-атомарного блока, однако программист должен быть уверен, что глобальное запрещение прерываний после не-атомарного блока – это действительно то, что он хочет.

Важной особенностью ATOMIC-блоков кода является то, что внутри могут быть использованы любые операторы Си, в том числе **break** и **return**, при этом состояние аппарата прерываний будет корректно востановлено при исполнении этих операторов.

util/crc16.h – Вычисления CRC

Модуль содержит определения нескольких inline-функций для быстрого вычисления циклических контрольных сумм (CRC) по широкоизвестным полиномам.

Алгоритм использования этих функций для контроля целостности блока данных (основное назначение CRC) в общем случае следующий:

- вводится переменная, например, C_R_C (ей должно быть присвоено строго определенное начальное значение)

- последовательно для каждого из значений проверяемого блока вызывается соответствующая функция, которая модифицирует значение переменной C_R_C

- после того, как все данные «обсчитаны» сравнение полученного значения C_R_C с заранее известным (правильным) значением CRC или нулем, если правильное значение CRC входило в состав обрабатываемого блока данных.

Функции модуля

_crc16_update()

uint16_t _crc16_update (uint16_t crc, uint8_t data)

Параметры:

uint16_t **crc** – текущее значение CRC

uint8_t **data** – байт данных

Возвращаемое значение: новое значение CRC

Описание: функция подсчета 16-разрядной контрольной суммы по полиному, типичному для дисковых накопителей $x^{16} + x^{15} + x^2 + 1$ (0xA001). Для верного подсчета контрольной суммы начальное значение **crc** должно быть 0xFFFF.

_crc_xmodem_update()

uint16_t _crc_xmodem_update (uint16_t crc, uint8_t data)

Параметры:

uint16_t **crc** – текущее значение CRC

uint8_t **data** – байт данных

Возвращаемое значение: новое значение CRC

Описание: функция подсчета 16-разрядной контрольной суммы по полиному, требуемому протоколом Xmodem $x^{16} + x^{12} + x^5 + 1$ (0x1021). Для верного подсчета контрольной суммы начальное значение **crc** должно быть 0.

_crc_ccitt_update()

uint16_t _crc_ccitt_update (uint16_t crc, uint8_t data)

Параметры:

uint16_t **crc** – текущее значение CRC

uint8_t **data** – байт данных

Возвращаемое значение: новое значение CRC

Описание: функция подсчета 16-разрядной контрольной суммы по полиному, требуемому стандартом CCITT (используется в протоколах PPP и IrDA) $x^{16} + x^{12} + x^5 + 1$ (0x8408). Для верного подсчета контрольной суммы начальное значение **crc** должно быть 0xFFFF.

Примечание: хотя полином выглядит точно так же, как и для Xmodem, существует принципиальная разница в алгоритме вычисления CRC, связанная с порядком поступления битов на обработку (старший-младший).

_crc_ibutton_update()

uint8_t _crc_ibutton_update (uint8_t crc, uint8_t data)

Параметры:

uint8_t **crc** – текущее значение CRC

uint8_t **data** – байт данных

Возвращаемое значение: новое значение CRC

Описание: функция подсчета 8-разрядной контрольной суммы по полиному, используемому в приборах iButton[37] $x^8 + x^5 + x^4 + 1$ (0x8C). Для верного подсчета контрольной суммы начальное значение **crc** должно быть 0.

[36] Полноценное определение термина «атомарный» не приводится, в данном контексте можно считать, что это означает исполнение одного или более операторов Си как одной команды, т.е. без возможности прерывания посередине участка. Не смотря на кажущуюся странность этого термина в применении к одному оператору, это так: оператор **long tmp = 0;** фактически будет оттранслирован в 5 ассемблерных инструкций, исполняемых последовательно; между ними вполне возможно возникновение прерывания, что будет соответствовать прерыванию «в середине» оператора.

[37] Зарегистрированная торговая марка Maxim-Dallas.

util/delay.h – Реализация задержек программными циклами

Модуль определяет функции, являющиеся всего лишь более удобной «оберткой» базовым функциям задержек программными циклами (см. util/delay_basic.h – Базовые задержки программными циклами). Для нормального функционирования функций необходимо, чтобы было определено значение константы **F_CPU**, равное тактовой частоте микроконтроллера в герцах, до подключения модуля. Возможно определить эту константу внутри make-файла или, что то же самое, непосредственно в параметрах компилятора.

Функции задержек реализованы следующим образом. По значению задержки, полученному как параметр функции (в виде константы) на этапе компиляции вычисляется количество пустых программных циклов, которые при заданной тактовой частоте нужно выполнить для получения этой задержки. В итоге компилятор строит код, который и выполняет эту задачу.

Чтобы это было возможно, необходимо, чтобы вычисления над константами с плавающей точкой были проведены на этапе компиляции, т.е. необходимо, чтобы была включена оптимизация при компиляции. Если проект скомпилирован при отключенной оптимизации – задержки, обеспечиваемые функциями, не гарантируются.

Таким образом, для использования модуля **delay.h** необходимо выполнить 2 условия:

1. Объявить F_CPU, равное тактовой частоте контроллера в герцах до подключения модуля

2. Включить оптимизацию (любого уровня) при компиляции.

Примечание: функции реализуют задержки путем программных циклов, которые могут быть могут быть прерваны запросом прерывания. Разумеется, в этом случае длительность задержки окажется больше заданного значения, и чем чаще происходят прерывания и чем сложнее (т.е. длительнее) обработчики прерываний, тем эта погрешность выше.

Функции модуля

_delay_us()

void _delay_us(double us)

Описание: функция осуществляет задержку в **us** микросекунд. Максимально возможное значение задержки **768 / F_CPU** микросекунд (где F_CPU в ***мегагерцах***). Программист не обязан следить за тем, чтобы параметр **us** находился в допустимых пределах – если его значение окажется больше, чем допустимо, вызов функции автоматически будет перенаправлен в функцию _delay_ms() (никаких уведомлений об этом не осуществляется).

_delay_ms()

void _delay_ms(double ms)

Описание: функция осуществляет задержку в **ms** миллисекунд. Максимально возможное значение задержки **262.14 / F_CPU** миллисекунд (где F_CPU в ***мегагерцах***). Программист не обязан следить за тем, чтобы параметр **ms** находился в допустимых пределах – если его значение окажется больше, чем допустимо, произойдет автоматическое снижение точности выдержки интервала (никаких уведомлений об этом не происходит), таким образом, функция сможет реализовывать задержки вплоть до 65535 секунд.

util/delay_basic.h – Базовые задержки программными циклами

Модуль определяет две базовых inline-функции задержек программным циклом известной длительности, которые предназначены для получения очень коротких и точных задержек, и активно используются функциями _delay_us() и _delay_ms().

Функции не выполняют запрещения прерываний на время своей работы. Поэтому следует учитывать, что это может повлиять на точность формируемых задержек.

Для организации длительных задержек более эффективно использование аппаратных таймеров.

Функции модуля

_delay_loop_1()

void _delay_loop_1 (uint8_t count)

Описание: функция обеспечивает исполнение **count** раз (от 1 до 256) наикратчайшего цикла со счетчиком (т.е. одна итерация цикла длится 3 машинных такта). Дополнительно тратится время на инициализацию задействованного счетного регистра.

Примечание: для организации 256 повторений цикла следует передать в функцию параметр ноль.

_delay_loop_2()

void _delay_loop_2 (uint16_t count)

Описание: функция обеспечивает исполнение **count** раз (от 1 до 65536) наикратчайшего цикла со счетчиком (т.е. одна итерация цикла длится 4 машинных такта). Дополнительно тратится время на инициализацию задействованного счетного регистра.

Примечание: для организации 65536 повторений цикла следует передать в функцию параметр ноль.

util/parity.h – Генерация битов четности

Модуль определяет единственный макрос, реализующий оптимизированную ассемблерную функцию вычисления бита четности для байта данных (может использоваться при обмене по UART).

parity_even_bit()

parity_even_bit(val) – макрос, возвращающий 1, если число единиц а байте **val** нечетное.

util/setbaud.h – Упрощение вычисления скоростей UART

Модуль облегчает расчет значений регистров управления аппаратного UART. Реализуется это несколько необычным способом.

Перед подключением модуля следует определить две константы: **F_CPU** (тактовая частота микроконтроллера в герцах) и **BAUD** (желаемая скорость работы UART в бодах). Так же можно дополнительно определить значение **BAUD_TOL**, которое задает допустимую погрешность скорости в процентах (по умолчанию 2%). После определения этих констант следует подключить модуль setbaud.h, т.е. например, так:

```
#include <avr/io.h>

#define F_CPU 4000000 // указали тактовую
частоту контроллера

static void uart_9600(void){
#define BAUD 9600
#include <util/setbaud.h>
UBRRH = UBRRH_VALUE;
UBRRL = UBRRL_VALUE;
#if USE_2X
UCSRA |= (1 << U2X);
#else
UCSRA &= ~(1 << U2X);
#endif
}

static void uart_38400(void){
#undef BAUD // отменим определение, чтобы
избежать предупреждения компилятора
#define BAUD 38400
#include <util/setbaud.h>
UBRRH = UBRRH_VALUE;
UBRRL = UBRRL_VALUE;
#if USE_2X
UCSRA |= (1 << U2X);
#else
UCSRA &= ~(1 << U2X);
#endif
}
```

В приведенном примере показано определение двух функций, которые настраивают UART на скорости 9600 и 38400 бод соответственно. Делается это путем переопределения вышеупомянутых констант и многократного подключения модуля setbaud.h.

В самом модуле на основании заданных констант производится расчет значений других констант, которые затем используются для инициализации UBRR и UCSRA.

Таким образом, модуль использует ряд «входных» определений, формируя рад «выходных»:

UBRR_VALUE – значение для записи в UBRR

UBRRL_VALUE – значение для записи в UBRRL

UBRRH_VALUE – значение для записи в UBRRH

USE_2X – логический флаг, определяющий необходимость установки бита U2X в регистре UCSRA.

Продолжение в №3/2011