Полнодуплексный программный UART для ATtiny13

Привет всем пользователям Geektimes! Как-то раз по долгу службы мне потребовалось реализовать программный UART на популярном микроконтроллере ATtiny13. Загуглив, я нашел большое количество статей на эту тему, многие из них выложены здесь:

- UART в ATtiny13 или Как вывести данные из МК за 52р [1]
- Ещё один программный UART на ATtiny13 [2]

А есть и на других ресурсах:

- Программный UART для ATtiny13 [3]

Последняя реализация, в общем-то, удовлетворяет моим потребностям (полнодуплексная связь). Но, во-первых, код написан в CodeVision AVR, который я не использую по сугубо религиозным соображениям, во-вторых, слабо комментированные вставки на ассемблере тем более отбивают желание разбираться в коде. Я же поставил себе целью написать на чистом С понятную пользователям библиотеку полнодуплексного UART-а. А заодно написать про это статью, потому что задачка достаточно интересная ввиду очень ограниченного объема памяти и ресурсов контроллера (всего один 8-битный таймер). Для новичков в программировании микроконтроллеров это будет неплохой учебный материал, т.к. я сам в процессе написания библиотеки, практически с нуля освоил архитектуру AVR.

Кому интересно — добро пожаловать под кат, текста будет много, много исходников с комментариями в коде.

Итак, начнем. Заголовочный файл uart13.h я просто выложу в том виде, в котором он есть с комментариями в коде, там все просто.

Заголовочный файл uart13.h

```
/* Библиотека программной реализации UART для микроконтроллеров ATtiny */

#ifndef _UART13_H_
#define _UART13_H_ 1

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

/*
*       Ниже настраиваются порты и пины портов которые будут использоваться
*       как передатчик и приемник UART.
*/

#define TXPORT PORTB            // Имя порта для передачи
#define RXPORT PINB             // Имя порта на прием
#define TXDDR DDRB              // Регистр направления порта на передачу
#define RXDDR DDRB              // Регистр направления порта на прием
#define TXD 0                   // Номер бита порта для использования на передачу
#define RXD 1                   // Номер бита порта для использования на прием

/*
*       Ниже задаются константы, определяющие скорость передачи данных (бодрейт)
*       расчет BAUD_DIV осуществляется следующим образом:
*       BAUD_DIV = (CPU_CLOCK / DIV) / BAUD_RATE
*       где CPU_CLOCK - тактовая частота контроллера, BAUD_RATE - желаемая скорость UART,
*       а DIV - значение делителя частоты таймера, задающееся регистром TCCR0B.
*       Например, тактовая частота 9.6 МГц, делитель на 8, скорость порта 9600 бод:
*       BAUD_DIV = (9 600 000 / 8) / 9600 = 125 (0x7D).
*/

//#define T_DIV         0x01    // DIV = 1
#define T_DIV           0x02    // DIV = 8
//#define T_DIV         0x03    // DIV = 64
#define BAUD_DIV        0x7D    // Скорость = 9600 бод

/*
*       Ниже идут объявления глобальных переменных и функций для работы UART
*/

volatile uint16_t txbyte;
volatile uint8_t rxbyte;
volatile uint8_t txbitcount;
volatile uint8_t rxbitcount;
```

```
void uart_init();
void uart_send(uint8_t tb);
int16_t uart_recieve(uint8_t* rb);

#endif /* _UART13_H_ */
```

А вот описание кода реализации библиотеки я разобью на части, чтобы не превратить статью в один огромный спойлер с кодом.

Прерывание TIM0_COMPA

```
ISR(TIM0_COMPA_vect)
{
        TXPORT = (TXPORT & ~(1 << TXD)) | ((txbyte & 0x01) << TXD); // Выставляем в бит TXD младший б
        txbyte = (txbyte >> 0x01) + 0x8000;     // Двигаем txbyte вправо на 1 и пишем 1 в старший раз
        if(txbitcount > 0)                      // Если идет передача (счетик бит больше нуля),
        {
                txbitcount--;                   // то уменьшаем его на единицу.
        }
}
```

Здесь мы видим код обработчика прерывания таймера по сравнению с регистром OCR0A. Оно работает постоянно и случается каждый раз, когда таймер достигает значения, записанного в регистре OCR0A. Когда это происходит, значение таймера в регистре TCNT0 автоматически обнуляется (режим таймера CTC, задается в регистре TCCR0A). Это прерывание используется для отправки данных по UART. Переменная txbyte используется как сдвиговый регистр: каждый раз, когда происходит прерывание, младший разряд переменной txbyte выставляется на вывод TXD микросхемы, после чего происходит сдвиг содержимого переменной вправо, а в освободившийся старший разряд записывается единица.

Пока данных для передачи нет, в переменной хранится число 0xFFFF и, тем самым, на выходе TXD непрерывно поддерживается высокий логический уровень. Когда мы хотим передать данные, мы должны записать в счетчик бит число бит для передачи: 1 стартовый, 8 бит данных и 1 стоповый, итого 10 (0x0A), и записать в txbyte данные для передачи вместе со стартовым битом. После этого они немедленно начнут передаваться. Формированием посылки занимается функция void uart_send(uint8_t tb).

Прерывание TIM0_COMPB

```
ISR(TIM0_COMPB_vect)
{
        if(RXPORT & (1 << RXD))                 // Проверяем в каком состоянии вход RXD
                rxbyte |= 0x80;                 // Если в 1, то пишем 1 в старший разряд rxbyte

        if(--rxbitcount == 0)                   // Уменьшаем на 1 счетчик бит и проверяем не стал ли
        {
                TIMSK0 &= ~(1 << OCIE0B);       // Если да, запрещаем прерывание TIM0_COMPB
                TIFR0 |= (1 << OCF0B);          // Очищаем флаг прерывания TIM0_COMPB
                GIFR |= (1 << INTF0);           // Очищаем флаг прерывания по INT0
                GIMSK |= (1 << INT0);           // Разрешаем прерывание INT0
        }
        else
        {
                rxbyte >>= 0x01;                // Иначе сдвигаем rxbyte вправо на 1
        }
}
```

Здесь мы видим обработчик прерывания таймера по сравнению с регистром OCR0B. Оно работает аналогично прерыванию TIM0_COMPA, но, в отличие от него, при выполнении этого прерывания не происходит обнуления таймера TCNT0. Это прерывание разрешается только тогда, когда мы принимаем данные, в остальное время оно запрещено. Когда оно случается, мы проверяем логическое состояние входа RXD и, если оно в единице, то пишем единицу в старший разряд переменной приема rxbyte, затем мы уменьшаем на единицу счетчик принятых бит и, если он стал нулем, заканчиваем прием. Иначе сдвигаем вправо переменную rxbyte, чтобы подготовить ее к приему следующего бита.

Прерывание INT0

```
ISR(INT0_vect)
{
        rxbitcount = 0x09;                      // 8 бит данных и 1 стартовый бит
        rxbyte = 0x00;                          // Обнуляем содержимое rxbyte
        if(TCNT0 < (BAUD_DIV / 2))              // Если таймер не досчитал до середины текущего перио
        {
                OCR0B = TCNT0 + (BAUD_DIV / 2); // То прерывание произойдет в текущем периоде спустя
        }
        else
        {
                OCR0B = TCNT0 - (BAUD_DIV / 2); // Иначе прерывание произойдет уже в следующем период
        }
        GIMSK &= ~(1 << INT0);                  // Запрещаем прерывание по INT0
        TIFR0 |= (1 << OCF0A) | (1 << OCF0B);   // Очищаем флаги прерываний TIM0_COMPA (B)
        TIMSK0 |= (1 << OCIE0B);                // Разрешаем прерывание по OCR0B
}
```

Прерывание INT0. Срабатывает по заднему фронту импульса на входе INT0, используется для отслеживания начала приема информации. Выставляет счетчик бит равным 9, обнуляет содержимое переменной rxbyte. Задает значение для регистра OCR0B, определяющего периодичность срабатывания прерывания TIM0_COMPB, оно должно приходиться по времени на середину принимаемого бита. После чего прерывание TIM0_COMPB разрешается, а прерывание INT0 запрещается.

Далее идут пользовательские функции для работы с UART.

Функция uart_send

```
void uart_send(uint8_t tb)
{
        while(txbitcount);              // Ждем пока закончится передача предыдущего байта
        txbyte = (tb + 0xFF00) << 0x01; // Пишем в младшие разряды txbyte данные для передачи и сдвиг
        txbitcount = 0x0A;              // Задаем счетчик байт равным 10
}
```

Функция передачи байта по UART. Принимает в качестве аргумента байт для передачи, возвращаемого значения нет. Если в момент вызова функции идет передача байта, то она ждет пока передача закончится, после чего записывает в младшие 8 бит переменной txbyte байт для передачи, а старшие 8 бит остаются 0xFF, затем сдвигает переменную влево, создавая таким образом стартовый бит в младшем разряде. Задает счетчик бит равным 10.

Функция uart_recieve

```
int16_t uart_recieve(uint8_t* rb)
{
        if(rxbitcount < 0x09)           // Если счетчик бит на прием меньше 9
        {
                while(rxbitcount);      // Ждем пока завершится текущий прием
                *rb = rxbyte;           // Пишем по адресу указателя принятый байт
                rxbitcount = 0x09;      // Восстанавливаем значение счетчика бит
                return (*rb);           // Возвращаемся
        }
        else
        {
                return (-1);            // Иначе возвращаем -1 (принимать нечего)
        }
}
```

Функция приема байта по UART. Принимает в аргумент указатель на 8-битную переменную, где будет содержаться принятый байт. Возвращает принятый байт, если байт принят, а если принимать нечего, возвращает (-1). Если в момент вызова функции идет прием, функция будет ждать его завершения. Если функцию вызвать дважды, то первый раз она возвратит принятый байт, а во второй раз (-1).

Функция uart_init

```
void uart_init()
{
        txbyte = 0xFFFF;                // Значение буфера на передачу - все единицы
        rxbyte = 0x00;                  // Значение буфера на прием - все нули
        txbitcount = 0x00;              // Значение счетчика преедаваемых бит - ноль (ничего пока не
        rxbitcount = 0x09;              // Значение счетчика бит на прием - 9 (ожидаем возможного при

        TXDDR |= (1 << TXD);            // Задаем направление порта на передачу как выход
        RXDDR &= ~(1 << RXD);           // Задаем направление порта на прием как вход
        TXPORT |= (1 << TXD);           // Пишем единицу в выход TXD
        RXPORT |= (1 << RXD);           // Подтягиваем к единице вход RXD
        OCR0A = BAUD_DIV;               // Задаем значение регистра OCR0A в соответствии с бодрейтом
        TIMSK0 |= (1 << OCIE0A);        // Разрешаем прерывание TIM0_COMPA
        TCCR0A |= (1 << WGM01);         // Режим таймера CTC (очистка TCNT0 по достижению OCR0A)
        TCCR0B |= T_DIV;                // Задаем скорость счета таймера в соответствии с делителем
        MCUCR |= (1 << ISC01);          // Задаем прерывание INT0 по заднему фронту импульса
        GIMSK |= (1 << INT0);           // Разрешаем прерывание INT0
        sei();                          // Разрешаем прерывания глобально
}
```

Функция инициализации UART. Аргументов нет, возвращаемого значения нет. Инициализирует глобальные переменные и регистры микроконтроллера. Из комментариев в коде должно быть все понятно.

Итак, пришло время написать какой нибудь простой main() с использованием нашей библиотеки и посмотреть что получилось в плане объема кода и проверить работоспособность.

main.c

```
#include "uart13.h"

int main(void)
{
        uint8_t b = 0;
        uart_init();
        while (1)
        {
```

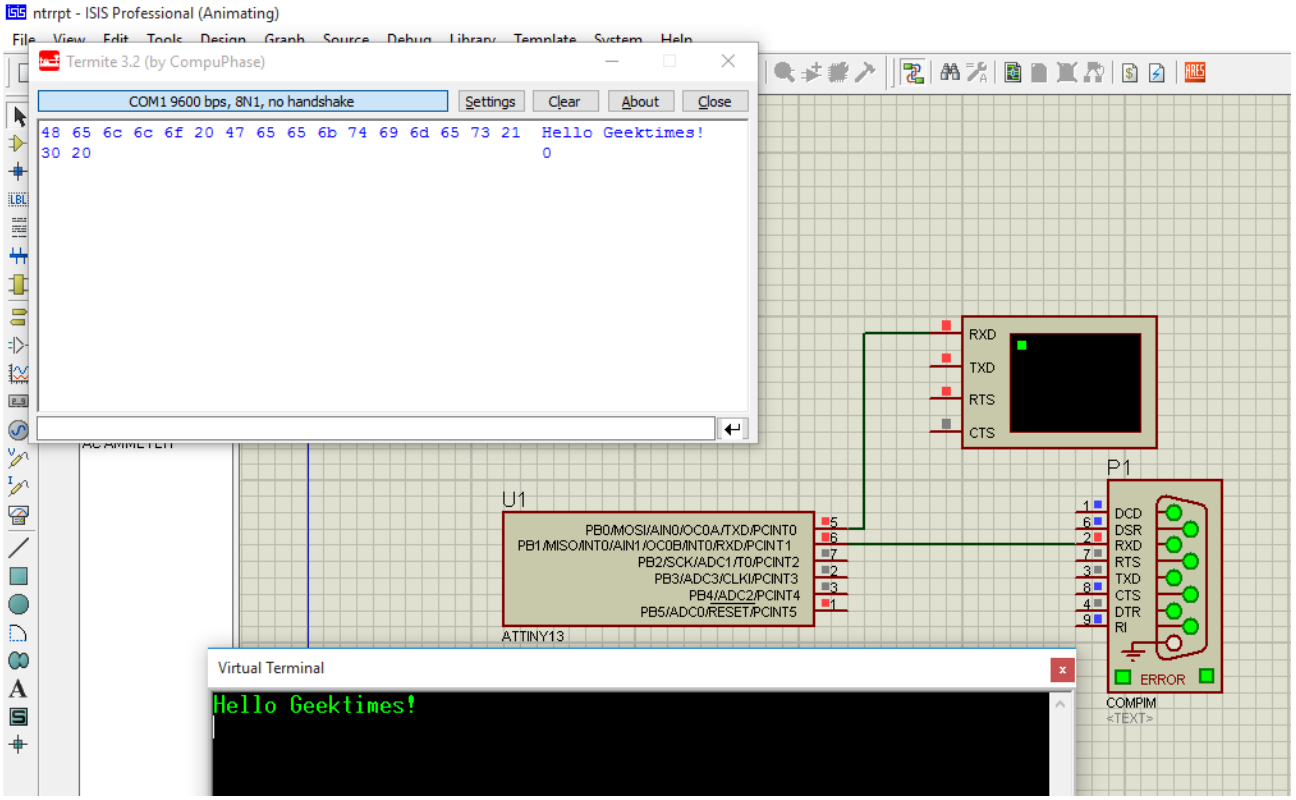
```
                if(uart_recieve(&b) >= 0)       // Если ничего не приняли, ничего и не передаем
                        uart_send(b);           // А если приняли, передаем принятое
        }
        return (0);
}
```

Компилируем:

```
Program Memory Usage: 482 bytes 47,1 % Full
Data Memory Usage: 5 bytes 7,8 % Full
```

Неплохо, у нас в запасе еще больше половины памяти микроконтроллера!
Проверяем в Proteus:

Симуляция ОК!



Итог: реализация получилась вполне годная к использованию, данные передаются и принимаются независимо, библиотека delay.h вообще не использована, а в запасе осталось больше половины памяти микроконтроллера.

Прилагаю исходники библиотеки, они компилируются в avr-gcc: Исходники на GitHub [4]

Автор: 8<

Источник [5]

Ссылки в тексте:

[1] UART в ATtiny13 или Как вывести данные из МК за 52р: **http://geektimes.ru/post/254792/**
[2] Ещё один программный UART на ATtiny13: **http://geektimes.ru/post/255010/**
[3] Программный UART для ATtiny13: **http://www.getchip.net/posts/046-programmnyjj-uart-dlya-attiny13/**
[4] Исходники на GitHub: **https://github.com/wiseholder/attiny13-uart**
[5] Источник: **http://geektimes.ru/sandbox/2450/**