

STM8S+SDCC+SPL: использование модуля UART1 на примере функции printf()

(/2016/stm8_spl_printf/)

разделы: STM8 (/tags/stm8), дата: 20 июля 2016г.

UART обладает двумя важными преимуществами перед остальными интерфейсами: во-первых, через TTL-конвертер его напрямую можно подключить к компьютеру, а во-вторых, линии связи между двумя узлами могут достигать 50 метров, что позволяет без проблем построить, к примеру, локальную сеть типа SmartHome. В STM8 скорость передачи через UART может достигать 1Mbit/s при частоте системной шины в 16MHz.

Микроконтроллеры STM8S могут обладать несколькими UART модулями, правда характеристики этих модулей немного разнятся:

Table 52. UART configurations⁽¹⁾

Feature	UART1	UART2	UART3	UART4
Asynchronous mode	X	X	X	X
Multiprocessor communication	X	X	X	X
Synchronous communication	X	X	NA	X
Smartcard mode	X	X	NA	X
IrDA mode	X	X	NA	X
Single-wire Half-duplex mode	X	NA	NA	X
LIN master mode	X	X	X	X
LIN slave mode	NA	X	X	X

1. X = supported; NA = not applicable.

(/img/stm8/uart_conf.png)

Использование UART с помощью стандартной периферийной библиотеки(далее SPL), не сложнее, чем в AVR или даже Arduino. Библиотека, в папке Examples содержит готовые примеры использования UART, которые подробно прокомментированы. НО что бы не скатиться до потребительского отношения, бездумно используя чужой код, я предлагаю пошуршать немного мануалами, чтобы понять, чем же все-таки являются USART модули в STM8, и как ими пользоваться максимально эффективно.

Основные возможности UART1 в stm8s103f3:

Main features

- One Mbit/s full duplex SCI
- SPI emulation
- High precision baud rate generator
- SmartCard emulation
- IrDA SIR encoder decoder
- LIN master mode
- Single wire half duplex mode

Asynchronous communication (UART mode)

- Full duplex communication - NRZ standard format (mark/space)
- Programmable transmit and receive baud rates up to 1 Mbit/s (fCPU/16) and capable of following any standard baud rate regardless of the input frequency
- Separate enable bits for transmitter and receiver
- Two receiver wakeup modes:
 - Address bit (MSB)
 - Idle line (interrupt)
- Transmission error detection with interrupt generation
- Parity control

Synchronous communication

- Full duplex synchronous transfers
- SPI master operation
- 8-bit data communication
- Maximum speed: 1 Mbit/s at 16 MHz (fCPU/16)

LIN master mode

- Emission: Generates 13-bit synch. break frame
- Reception: Detects 11-bit break frame

(/img/stm8/uart_features.png)

LIN и CAN это локальные сети созданные для использования в автомобилестроении (<https://ru.wikipedia.org/wiki/LIN>):

Основные задачи, возлагаемые на LIN (<https://ru.wikipedia.org/wiki/LIN>) консорциумом европейских автомобильных производителей, – объединение автомобильных подсистем и узлов (таких как дверные замки, стеклоочистители, стеклоподъемники, управление магнитолой и климат-контролем, электролюк и так далее) в единую электронную систему. LIN-протокол утверждён Европейским Автомобильным Консорциумом как дешёвое дополнение к сверхнадёжному протоколу CAN (https://ru.wikipedia.org/wiki/Controller_Area_Network).

LIN и CAN дополняют друг друга и позволяют объединить все электронные автомобильные приборы в единую многофункциональную бортовую сеть. Причём область применения CAN – участки, где требуется сверхнадёжность и скорость; область же применения LIN – объединение дешёвых узлов, работающих с малыми скоростями передачи информации на коротких дистанциях и сохраняющих при этом универсальность, многофункциональность, а также простоту разработки и отладки.

Так же имеется поддержка IrDA (<http://www.ixbt.com/peripheral/irda.html>)

Сам порт IrDA основан на архитектуре коммуникационного COM-порта ПК, который использует универсальный асинхронный приемо-передатчик UART и работает со скоростью передачи данных 2400–115200 bps.

Связь в IrDA полудуплексная, т.к. передаваемый ИК-луч неизбежно засвечивает соседний PIN-диодный усилитель приемника. Воздушный промежуток между устройствами позволяет принять ИК-энергию только от одного источника в данный момент.

Кроме этого имеется еще Smartcard (<https://habrahabr.ru/post/257279/>)

Интерфейс является синхронным режимом USART-драйвера, это значит, что передачу каждого бита информации мы синхронизируем частотой на выводе CLK, но здесь есть одно важное отличие от прочих синхронных интерфейсов (вроде того же SPI): для тактирования одного бита информации нужен не один импульс на CLK, а 372 импульса (это магическое число прописано в 3 части ISO7816, и носит название ETU (Elementary Time Unit)), т.е., один бит данных тактируется каждым 372-м (в идеальном случае) фронтом. Сама частота должна лежать в пределах от 1 до 5 МГц.

Как видно, вариантов использования UART много, но сейчас нас будет интересовать классический UART протокол: 8N1.

Регистры модуля UART1 в SPL описываются следующей структурой:

```
typedef struct UART1_struct
{
    __IO uint8_t SR;      /*!< UART1 status register */
    __IO uint8_t DR;      /*!< UART1 data register */
    __IO uint8_t BRR1;    /*!< UART1 baud rate register */
    __IO uint8_t BRR2;    /*!< UART1 DIV mantissa[11:8] SCIDIV fraction */
    __IO uint8_t CR1;     /*!< UART1 control register 1 */
    __IO uint8_t CR2;     /*!< UART1 control register 2 */
    __IO uint8_t CR3;     /*!< UART1 control register 3 */
    __IO uint8_t CR4;     /*!< UART1 control register 4 */
    __IO uint8_t CR5;     /*!< UART1 control register 5 */
    __IO uint8_t GTR;     /*!< UART1 guard time register */
    __IO uint8_t PSCR;    /*!< UART1 prescaler register */
}
UART1_TypeDef;
```

DR - регистр данных, BRRx - определяют скорость интерфейса, SR - флаговые/статусные регистры, CRx - управляющие режимами работы UART регистры, PSCR - позволяет понизить тактовую частоту UART модуля. Если к примеру, скорость передачи 9600 bod, то незачем модуль гонять вхолостую, можно поставить делитель на 64. GTR - Guard Time Register используется в Smartcard, задает величину защищенного интервала в тактах системной шины тактирования.

Функцией UART1_DeInit(), в регистры записываются следующие значения по умолчанию:

```
#define UART1_SR_RESET_VALUE    ((uint8_t)0xC0)
#define UART1_BRR1_RESET_VALUE ((uint8_t)0x00)
#define UART1_BRR2_RESET_VALUE ((uint8_t)0x00)
#define UART1_CR1_RESET_VALUE  ((uint8_t)0x00)
#define UART1_CR2_RESET_VALUE  ((uint8_t)0x00)
#define UART1_CR3_RESET_VALUE  ((uint8_t)0x00)
#define UART1_CR4_RESET_VALUE  ((uint8_t)0x00)
#define UART1_CR5_RESET_VALUE  ((uint8_t)0x00)
#define UART1_GTR_RESET_VALUE  ((uint8_t)0x00)
#define UART1_PSCR_RESET_VALUE ((uint8_t)0x00)
```

Функционал модуля UART1 реализуется следующими функциями:

```
void UART1_DeInit(void);
void UART1_Init(uint32_t BaudRate, UART1_WordLength_TypeDef WordLength,
                UART1_StopBits_TypeDef StopBits, UART1_Parity_TypeDef Parity,
                UART1_SyncMode_TypeDef SyncMode, UART1_Mode_TypeDef Mode);
void UART1_Cmd(FunctionalState NewState);
void UART1_ITConfig(UART1_IT_TypeDef UART1_IT, FunctionalState NewState);
void UART1_HalfDuplexCmd(FunctionalState NewState);
void UART1_IrDAConfig(UART1_IrDAMode_TypeDef UART1_IrDAMode);
void UART1_IrDACmd(FunctionalState NewState);
void UART1_LINBreakDetectionConfig(UART1_LINBreakDetectionLength_TypeDef UART1_LINBreakDetectionLength);
void UART1_LINCmd(FunctionalState NewState);
void UART1_SmartCardCmd(FunctionalState NewState);
void UART1_SmartCardNACKCmd(FunctionalState NewState);
void UART1_WakeUpConfig(UART1_WakeUp_TypeDef UART1_WakeUp);
void UART1_ReceiverWakeUpCmd(FunctionalState NewState);
uint8_t UART1_ReceiveData8(void);
uint16_t UART1_ReceiveData9(void);
void UART1_SendData8(uint8_t Data);
void UART1_SendData9(uint16_t Data);
void UART1_SendBreak(void);
void UART1_SetAddress(uint8_t UART1_Address);
void UART1_SetGuardTime(uint8_t UART1_GuardTime);
void UART1_SetPrescaler(uint8_t UART1_Prescaler);
FlagStatus UART1_GetFlagStatus(UART1_Flag_TypeDef UART1_FLAG);
void UART1_ClearFlag(UART1_Flag_TypeDef UART1_FLAG);
ITStatus UART1_GetITStatus(UART1_IT_TypeDef UART1_IT);
void UART1_ClearITPendingBit(UART1_IT_TypeDef UART1_IT);
```

Пример программы для пересылки данных с микроконтроллера на компьютер через UART:

```

#include "stm8s.h"
#include "stm8s_gpio.h"
#include "stm8s_clk.h"
#include "stm8s_tim4.h"
#include "stm8s_uart1.h"
#include "stdio.h"

#define LED_PORT GPIOB
#define LED_GPIO_PIN_5

#define TIM4_PERIOD          124

#define PUTCHAR_PROTOTYPE void putchar (char c)

ErrorStatus status = FALSE;
volatile uint16_t count;
uint16_t i;

INTERRUPT_HANDLER(IRQ_Handler_TIM4, 23)
{
    if (count)
        count--;

    TIM4_ClearITPendingBit(TIM4_IT_UPDATE);
}

void delay_ms(uint16_t ms)
{
    TIM4_Cmd(DISABLE);          // stop
    TIM4_TimeBaseInit(TIM4_PRESCALER_128, TIM4_PERIOD);
    TIM4_ClearFlag(TIM4_FLAG_UPDATE);
    TIM4_ITConfig(TIM4_IT_UPDATE, ENABLE);

    count = ms;

    TIM4_Cmd(ENABLE);          // let's go

    while(count);
}

int main( void )
{
    // ----- GPIO CONFIG -----
    GPIO_DeInit(LED_PORT);
    GPIO_Init(LED_PORT, LED, GPIO_MODE_OUT_PP_LOW_FAST);

    // ----- CLK CONFIG -----
    CLK_DeInit();

    CLK_SYSClkConfig(CLK_PRESCALER_CPUDIV1);
    CLK_SYSClkConfig(CLK_PRESCALER_HSIDIV1); // set 16 MHz for CPU

    TIM4_DeInit();
    // uncomment if use HSE on Quartz
    //status = CLK_ClockSwitchConfig(CLK_SWITCHMODE_AUTO, CLK_SOURCE_HSE, DISABLE,
    //                               CLK_CURRENTCLOCKSTATE_DISABLE);

    // ----- UART1 -----
    UART1_DeInit();

    // 8N1
    UART1_Init((uint32_t)9600, UART1_WORDLENGTH_8D, UART1_STOPBITS_1, UART1_PARITY_NO,
               UART1_SYNCMODE_CLOCK_DISABLE, UART1_MODE_TXRX_ENABLE);

    enableInterrupts();
    i=0;
    for(;;)
    {
        GPIO_WriteReverse(LED_PORT, LED);
        delay_ms(1000);

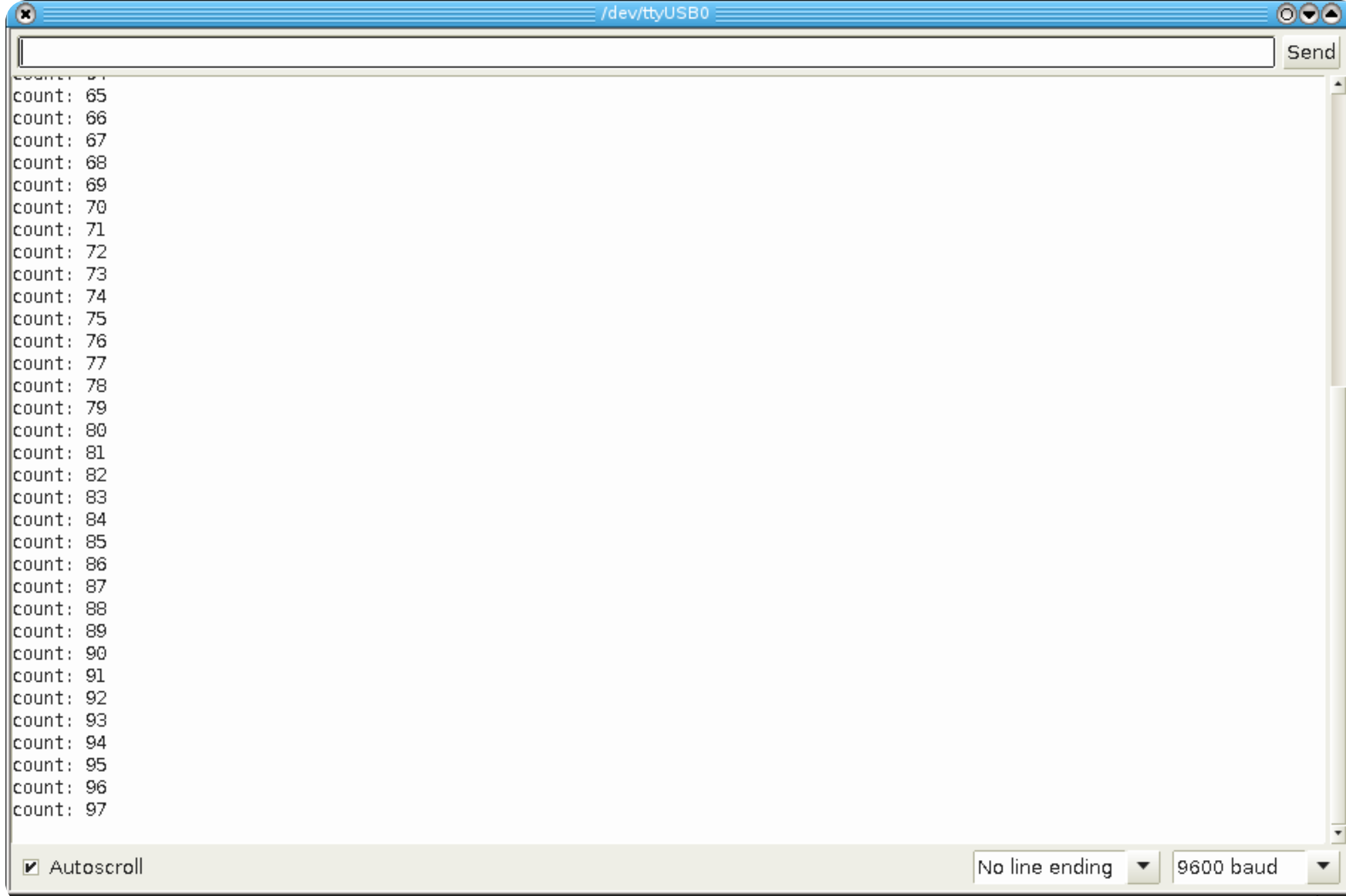
        printf("count: %u\n",++i);
    }
}

PUTCHAR_PROTOTYPE
{
    // Write a character to the UART1
    UART1_SendData8(c);
    // Loop until the end of transmission
    while (UART1_GetFlagStatus(UART1_FLAG_TXE) == RESET);
}

```

Так же как в AVR, для использования функции printf() нужно только задать функцию putchar(char c), после чего системная библиотека сделает все за вас. Кстати, прошивка вместе с printf(), начинает весить уже от 5 Кбайт.

Результат работы выглядит как-то так:



я предлагаю заглянуть в код функции UART1_SendData8(c):

```
void UART1_SendData8(uint8_t Data)
{
    /* Transmit Data */
    UART1->DR = Data;
}
```

а так же UART1_GetFlagStatus(UART1_FLAG_TXE):

```
FlagStatus UART1_GetFlagStatus(UART1_Flag_TypeDef UART1_FLAG)
{
    FlagStatus status = RESET;

    /* Check parameters */
    assert_param(IS_UART1_FLAG_OK(UART1_FLAG));

    /* Check the status of the specified UART1 flag*/
    if (UART1_FLAG == UART1_FLAG_LBDF)
    {
        if ((UART1->CR4 & (uint8_t)UART1_FLAG) != (uint8_t)0x00)
        {
            /* UART1_FLAG is set*/
            status = SET;
        }
        else
        {
            /* UART1_FLAG is reset*/
            status = RESET;
        }
    }
    else if (UART1_FLAG == UART1_FLAG_SBK)
    {
        if ((UART1->CR2 & (uint8_t)UART1_FLAG) != (uint8_t)0x00)
        {
            /* UART1_FLAG is set*/
            status = SET;
        }
        else
        {
            /* UART1_FLAG is reset*/
            status = RESET;
        }
    }
    else
    {
        if ((UART1->SR & (uint8_t)UART1_FLAG) != (uint8_t)0x00)
        {
            /* UART1_FLAG is set*/
            status = SET;
        }
        else
        {
            /* UART1_FLAG is reset*/
            status = RESET;
        }
    }
    /* Return the UART1_FLAG status*/
    return status;
}
```

но самая интересная функция, это конечно UART1_Init():

```

void UART1_Init(uint32_t BaudRate, UART1_WordLength_TypeDef WordLength,
                UART1_StopBits_TypeDef StopBits, UART1_Parity_TypeDef Parity,
                UART1_SyncMode_TypeDef SyncMode, UART1_Mode_TypeDef Mode)
{
    uint32_t BaudRate_Mantissa = 0, BaudRate_Mantissa100 = 0;

    /* Check the parameters */
    assert_param(IS_UART1_BAUDRATE_OK(BaudRate));
    assert_param(IS_UART1_WORDLENGTH_OK(WordLength));
    assert_param(IS_UART1_STOPBITS_OK(StopBits));
    assert_param(IS_UART1_PARITY_OK(Parity));
    assert_param(IS_UART1_MODE_OK((uint8_t)Mode));
    assert_param(IS_UART1_SYNCMODE_OK((uint8_t)SyncMode));

    /* Clear the word length bit */
    UART1->CR1 &= (uint8_t)(~UART1_CR1_M);

    /* Set the word length bit according to UART1_WordLength value */
    UART1->CR1 |= (uint8_t)WordLength;

    /* Clear the STOP bits */
    UART1->CR3 &= (uint8_t)(~UART1_CR3_STOP);
    /* Set the STOP bits number according to UART1_StopBits value */
    UART1->CR3 |= (uint8_t)StopBits;

    /* Clear the Parity Control bit */
    UART1->CR1 &= (uint8_t)(~(UART1_CR1_PCEN | UART1_CR1_PS ));
    /* Set the Parity Control bit to UART1_Parity value */
    UART1->CR1 |= (uint8_t)Parity;

    /* Clear the LSB mantissa of UART1DIV */
    UART1->BRR1 &= (uint8_t)(~UART1_BRR1_DIVM);
    /* Clear the MSB mantissa of UART1DIV */
    UART1->BRR2 &= (uint8_t)(~UART1_BRR2_DIVM);
    /* Clear the Fraction bits of UART1DIV */
    UART1->BRR2 &= (uint8_t)(~UART1_BRR2_DIVF);

    /* Set the UART1 BaudRates in BRR1 and BRR2 registers according to UART1_BaudRate value */
    BaudRate_Mantissa = ((uint32_t)CLK_GetClockFreq() / (BaudRate << 4));
    BaudRate_Mantissa100 = (((uint32_t)CLK_GetClockFreq() * 100) / (BaudRate << 4));
    /* Set the fraction of UART1DIV */
    UART1->BRR2 |= (uint8_t)((uint8_t)(((BaudRate_Mantissa100 - (BaudRate_Mantissa * 100)) << 4) / 100) & (uint8_t)0x0F);
    /* Set the MSB mantissa of UART1DIV */
    UART1->BRR2 |= (uint8_t)((BaudRate_Mantissa >> 4) & (uint8_t)0xF0);
    /* Set the LSB mantissa of UART1DIV */
    UART1->BRR1 |= (uint8_t)BaudRate_Mantissa;

    /* Disable the Transmitter and Receiver before setting the LBCL, CPOL and CPHA bits */
    UART1->CR2 &= (uint8_t)(~(UART1_CR2_TEN | UART1_CR2_REN));
    /* Clear the Clock Polarity, lock Phase, Last Bit Clock pulse */
    UART1->CR3 &= (uint8_t)(~(UART1_CR3_CPOL | UART1_CR3_CPHA | UART1_CR3_LBCL));
    /* Set the Clock Polarity, lock Phase, Last Bit Clock pulse */
    UART1->CR3 |= (uint8_t)((uint8_t)SyncMode & (uint8_t)(UART1_CR3_CPOL |
                                                         UART1_CR3_CPHA | UART1_CR3_LBCL));

    if ((uint8_t)(Mode & UART1_MODE_TX_ENABLE))
    {
        /* Set the Transmitter Enable bit */
        UART1->CR2 |= (uint8_t)UART1_CR2_TEN;
    }
    else
    {
        /* Clear the Transmitter Disable bit */
        UART1->CR2 &= (uint8_t)(~UART1_CR2_TEN);
    }
    if ((uint8_t)(Mode & UART1_MODE_RX_ENABLE))
    {
        /* Set the Receiver Enable bit */
        UART1->CR2 |= (uint8_t)UART1_CR2_REN;
    }
    else
    {
        /* Clear the Receiver Disable bit */
        UART1->CR2 &= (uint8_t)(~UART1_CR2_REN);
    }
    /* Set the Clock Enable bit, lock Polarity, lock Phase and Last Bit Clock
    pulse bits according to UART1_Mode value */
    if ((uint8_t)(SyncMode & UART1_SYNCMODE_CLOCK_DISABLE))
    {
        /* Clear the Clock Enable bit */
        UART1->CR3 &= (uint8_t)(~UART1_CR3_CKEN);
    }
    else
    {
        UART1->CR3 |= (uint8_t)((uint8_t)SyncMode & UART1_CR3_CKEN);
    }
}

```

И здесь кроется довольно массивный **"подводный камень"**. Дело в следующем. Как помнится (/2015/uart_part1/), в AVR была хитрая формула для преобразования скорости передачи в значение регистра UBRR. В STM8 имеется не менее хитрая формула для регистров BRRx:

22.3.4 High precision baud rate generator

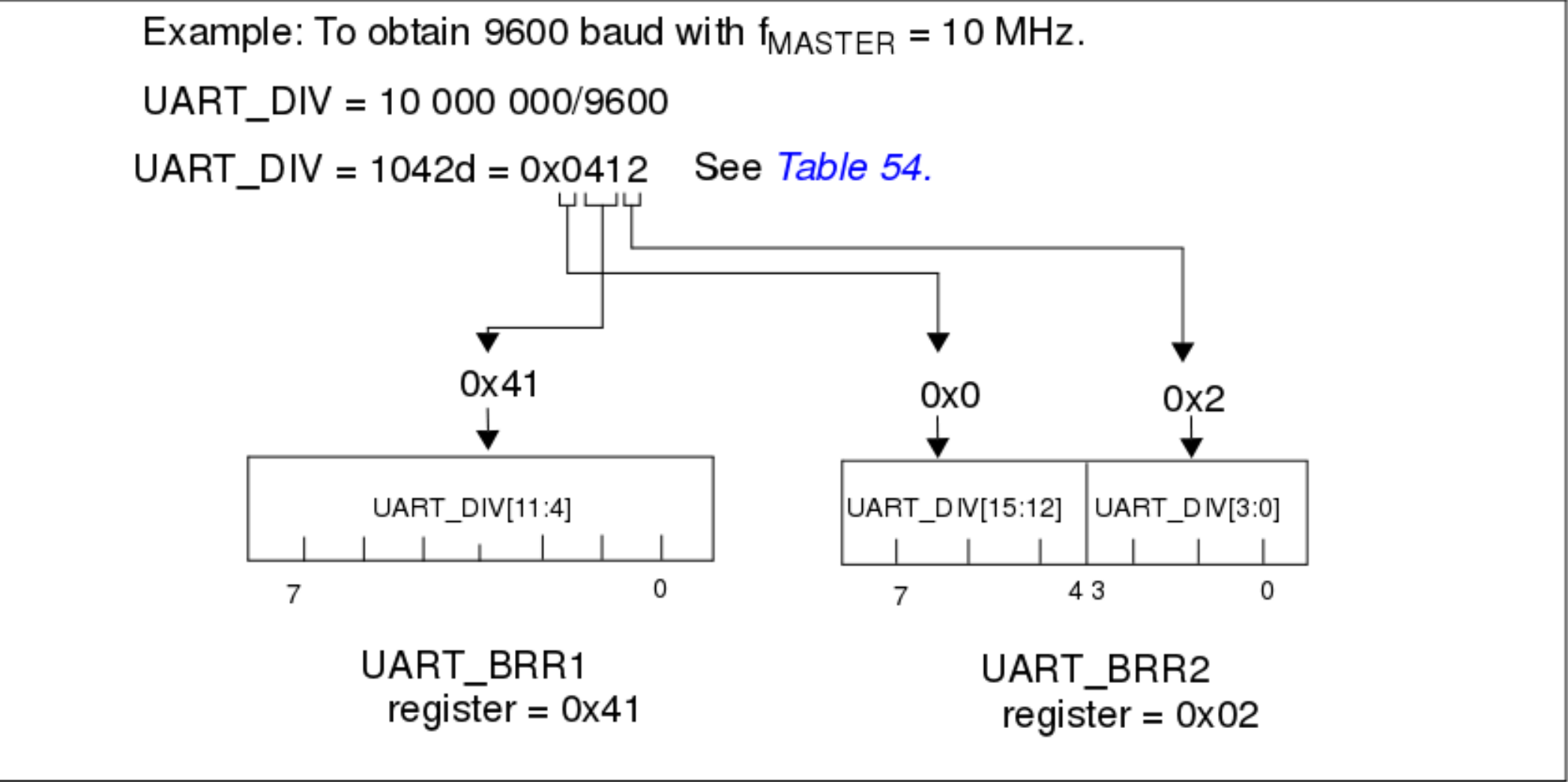
The receiver and transmitter (Rx and Tx) are both set to the same baud rate programmed by a 16-bit divider UART_DIV according to the following formula:

$$\text{Tx/ Rx baud rate} = \frac{f_{\text{MASTER}}}{\text{UART_DIV}}$$

The UART_DIV baud rate divider is an unsigned integer, coded in the BRR1 and BRR2 registers as shown in [Figure 119](#).

Refer to [Table 54](#). for typical baud rate programming examples.

Figure 119. How to code UART_DIV in the BRR registers



Note: The Baud Counters will be updated with the new value of the Baud Registers after a write to BRR1. Hence the Baud Register value should not be changed during a transaction. The BRR2 should be programmed before BRR1.

Note: UART_DIV must be greater than or equal to 16d.

(/img/stm8/uart1.png)

своя табличка тоже наличествует:

Table 54. Baud rate programming and error calculation

Baud rate	$f_{\text{MASTER}} = 10 \text{ MHz}$					$f_{\text{MASTER}} = 16 \text{ MHz}$				
	Actual	% Error ⁽¹⁾	UART_DIV	BRR1	BRR2	Actual	% Error ⁽¹⁾	UART_DIV	BRR1	BRR2
2.4	2.399	−0.008%	0x1047	0x04	0x17	2.399	−0.005%	0x1A0B	0xA0	0x1B
9.6	9.596	−0.032%	0x0412	0x41	0x02	9.598	−0.020%	0x0693	0x68h	0x03
19.2	19.193	−0.032%	0x0209	0x20	0x09	19.208	−0.040%	0x0341	0x34	0x01
57.6	57.471	−0.224%	0x00AE	0x0A	0x0E	57.554	−0.080%	0x0116	0x11	0x06
115.2	114.942	−0.224%	0x0057	0x05	0x07	115.108	−0.080%	0x008B	0x08	0x0B
230.4	232.558	−0.937%	0x002B	0x02	0x0B	231.884	−0.644%	0x0045	0x04	0x05
460.8	454.545	−1.358%	0x0016	0x01	0x06	457.143	−0.794%	0x0023	0x02	0x03
921.6	NA	NA	NA	NA	NA	941.176	−2.124%	0x11	0x01	0x01

1. Error % = (Calculated - Desired) Baud Rate / Desired Baud Rate

Note: The lower the f_{MASTER} frequency, the lower will be the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with this data.

(/img/stm8/uart2.png)

Как не трудно заметить, функция прямо пропорциональна fMASTER. Значение fMASTER UART1_Init() берет из модуля CLK с помощью функции CLK_GetClock():

```
BaudRate_Mantissa    = ((uint32_t)CLK_GetClockFreq() / (BaudRate << 4));
BaudRate_Mantissa100 = (((uint32_t)CLK_GetClockFreq() * 100) / (BaudRate << 4));
```

А сама CLK_GetClock() выглядит так:

```
uint32_t CLK_GetClockFreq(void)
{
    uint32_t clockfrequency = 0;
    CLK_Source_TypeDef clocksource = CLK_SOURCE_HSI;
    uint8_t tmp = 0, presc = 0;

    // Get CLK source.
    clocksource = (CLK_Source_TypeDef)CLK->CMSR;

    if (clocksource == CLK_SOURCE_HSI)
    {
        tmp = (uint8_t)(CLK->CKDIVR & CLK_CKDIVR_HSIDIV);
        tmp = (uint8_t)(tmp >> 3);
        presc = HSIDivFactor[tmp];
        clockfrequency = HSI_VALUE / presc;
    }
    else if ( clocksource == CLK_SOURCE_LSI)
    {
        clockfrequency = LSI_VALUE;
    }
    else
    {
        clockfrequency = HSE_VALUE;
    }

    return((uint32_t)clockfrequency);
}
```

Так вот. Если чип будет использовать внешний резонатор HSE, значение частоты резонатора будет браться из именованной константы HSE_VALUE в stm8s.h:

```
#if !defined  HSE_Value
#if defined (STM8S208) || defined (STM8S207) || defined (STM8S007) || defined (STM8AF52Ax) || \
    defined (STM8AF62Ax) || defined (STM8AF622x)
#define HSE_VALUE ((uint32_t)24000000) /* Value of the External oscillator in Hz*/
#else
#define HSE_VALUE ((uint32_t)16000000) /* Value of the External oscillator in Hz*/
#endif /* STM8S208 || STM8S207 || STM8S007 || STM8AF62Ax || STM8AF52Ax || STM8AF622x */
#endif /* HSE_Value */
```

.т.е. если ваш кварц не на 16 МГц, и вы не вписали частоту кварца во время компиляции, или не поправили заголовочный файл stm8s.h, то **работать корректно ваш модуль UART не будет**. Такой вот тест на внимательность.

Разобравшись с BRRx, осталось пробежаться по управляющим регистрам UART1_CRx. Общая карта регистров для модуля UART1 в STM8S выглядит так:

22.7.13 UART register map and reset values

Table 61. UART1 register map

Address	Register name	7	6	5	4	3	2	1	0
0x00	UART1_SR Reset Value	TXE 1	TC 1	RXNE 0	IDLE 0	OR 0	NF 0	FE 0	PE 0
0x01	UART1_DR Reset Value	DR7 X	DR6 X	DR5 X	DR4 X	DR3 X	DR2 X	DR1 X	DR0 X
0x02	UART1_BRR1 Reset Value	UART_DIV[11:4] 00000000							
0x03	UART1_BRR2 Reset Value	UART_DIV[15:12] 0000				UART_DIV[3:0] 0000			
0x04	UART1_CR1 Reset Value	R8 0	T8 0	UARTD 0	M 0	WAKE 0	PCEN 0	PS 0	PIEN 0
0x05	UART1_CR2 Reset Value	T1EN 0	TC1EN 0	RIEN 0	ILIEN 0	TEN 0	REN 0	RWU 0	SBK 0
0x06	UART1_CR3 Reset Value	- 0	L1NEN 0	STOP 00		CKEN 0	CPOL 0	CPHA 0	LBCL 0
0x07	UART1_CR4 Reset Value	- 0	LBD1EN 0	LBDL 0	LBDF 0	ADD[3:0] 0000			
0x08	UART1_CR5 Reset Value	- 0	- 0	SCEN 0	NACK 0	HDSEL 0	IRLP 0	IREN 0	0
0x09	UART1_GTR Reset Value	GT7 0	GT6 0	GT5 0	GT4 0	GT3 0	GT2 0	GT1 0	GT0 0
0x0A	UART1_PSCR Reset Value	PSC7 0	PSC6 0	PSC5 0	PSC4 0	PSC3 0	PSC2 0	PSC1 0	PSC0 0

(/img/stm8/uart_registers_map.png)

UART1_Init() для конфигурации UART1 использует регистры: CR1, CR2, CR3.

22.7.5 Control register 1 (UART_CR1)

Address offset: 0x04

Reset value: 0x00

7	6	5	4	3	2	1	0
R8	T8	UARTD	M	WAKE	PCEN	PS	PIEN
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **R8**: *Receive Data bit 8.*

This bit is used to store the 9th bit of the received word when M=1

Bit 6 **T8**: *Transmit data bit 8.*

This bit is used to store the 9th bit of the transmitted word when M=1

Bit 5 **UARTD**: *UART Disable (for low power consumption).*

When this bit is set the UART prescaler and outputs are stopped at the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: UART enabled

1: UART prescaler and outputs disabled

Bit 4 **M**: *word length.*

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit (n depending on STOP[1:0] bits in the UART_CR3 register)

1: 1 Start bit, 9 Data bits, 1 Stop bit

Note: The M bit must not be modified during a data transfer (both transmission and reception) In LIN slave mode, the M bit and the STOP[1:0] bits in the UART_CR3 register should be kept at 0.

Bit 3 **WAKE**: Wakeup method.

This bit determines the UART wakeup method, it is set or cleared by software.

0: Idle Line

1: Address Mark

Bit 2 **PCEN**: Parity control enable.

– UART Mode

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCEN is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

– LIN slave mode

This bit enables the LIN identifier parity check while the UART is in LIN slave mode.

0: Identifier parity check disabled

1: Identifier parity check enabled

Bit 1 **PS**: Parity selection.

This bit selects the odd or even parity when the parity generation/detection is enabled (PCEN bit set) in UART mode. It is set and cleared by software. The parity will be selected after the current byte.

0: Even parity

1: Odd parity

Bit 0 **PIEN**: Parity interrupt enable.

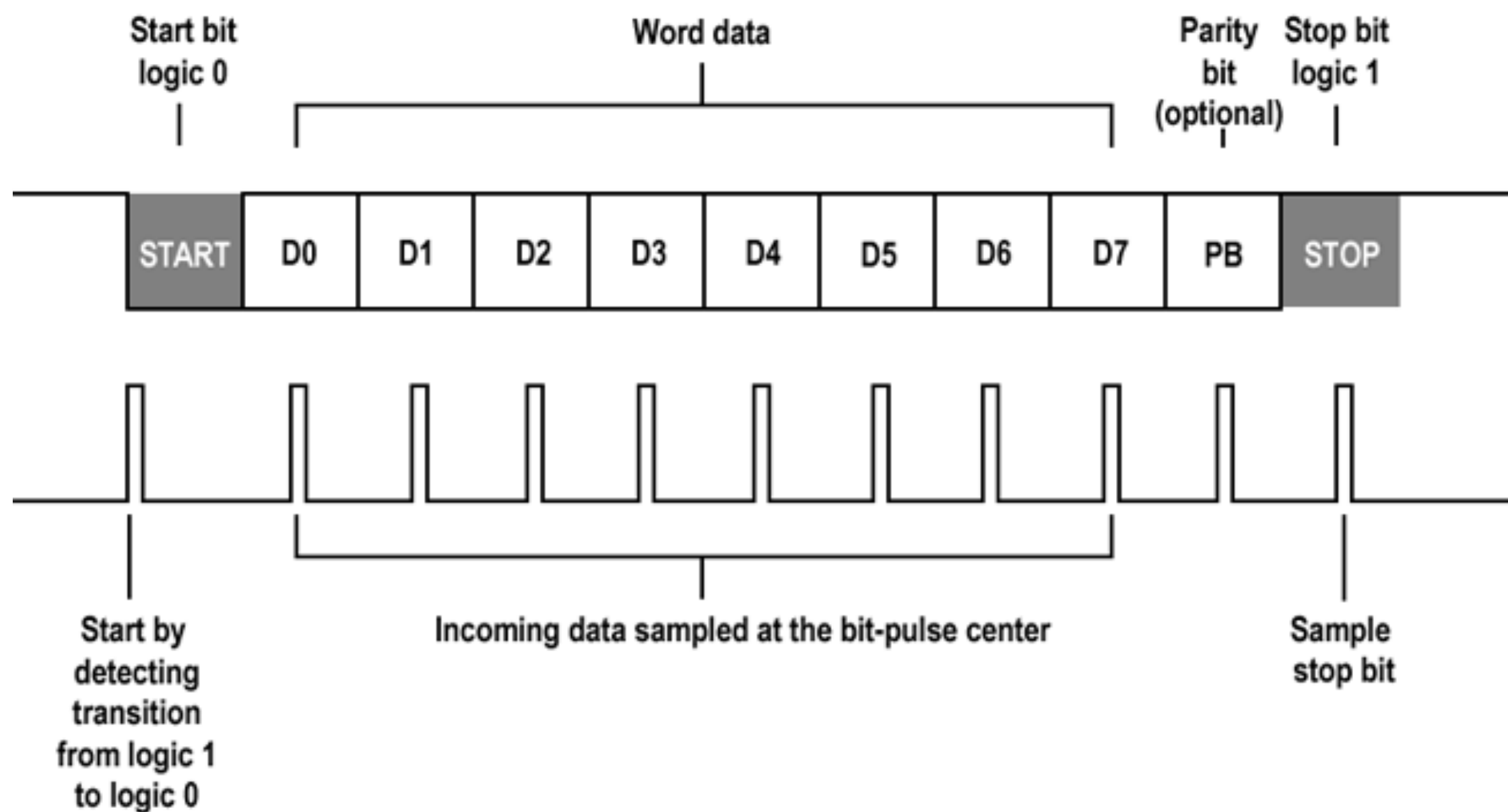
This bit is set and cleared by software.

0: Parity interrupt disabled

1: Parity interrupt is generated whenever PE=1 in the UART_SR register

(/img/stm8/uart1_cr1.png)

В CR1 у нас устанавливается длина фрейма равной восьми(установкой бита 4 **M**), а также сбрасывается аппаратный контроль четности(биты 2 **PCEN** и бит 1 **PS**) . Для ясности напомним формат UART кадра:



Остальные биты:

- Биты 7 и 6 отвечают за передачу кадром 9 бит. Если не ошибаюсь, такое используется в Multi-Processor communication когда локальную сеть строят на одном UART модуле. Тогда несколько бит используются для адресации устройства. Подробнее об этом можно почитать здесь: "Время говорить с камнями или USART Multi-processor Communication Mode" (<http://we.easyelectronics.ru/AVR/vremya-govorit-s-kamnyami-ili-usart-multi-processor-communication-mode.html>), сам я с этой штукой ни разу не сталкивался.
- Бит 5 отключает тактирование от UART, в целях энергосбережения.
- Бит 3 WAKE - определяет способ пробуждения из энергосберегающего режима. 0 - по освобождению линии, 1 - по принятию адреса.
- Бит 0 PIEN - разрешает/запрещает прерывание по четности. Прерывание генерируется когда устанавливается флаг PE в регистре UART1_SR.

Кстати, про прерывания. Прерывания UART могут вызывать следующие события:

22.6 UART interrupts

Table 60. UART interrupt requests

Interrupt event	Event flag	Enable control bit	Exit from Wait	Exit from Halt
Transmit data register empty	TXE	TIEN	Yes	No
Transmission complete	TC	TCIEN	Yes	No
Received data ready to be read	RXNE	RIEN	Yes	No
Overrun error detected / LIN header error	OR/LHE		Yes	No
Idle line detected	IDLE	ILIEN	Yes	No
Parity error	PE	PIEN	Yes	No
Break flag	LBDF	LBDIEN	Yes	No
Header Flag	LHDF	LHDIEN	Yes	No

- Note:**
- 1 The UART interrupt events are connected to two interrupt vectors (see [Figure 139](#)).
 - a) Transmission Complete or Transmit Data Register empty interrupt.
 - b) Idle Line detection, Overrun error, Receive Data register full, Parity error interrupt, and Noise Flag.
 - 2 These events generate an interrupt if the corresponding Enable Control Bit is set and the interrupt mask in the CC register is reset (RIM instruction).

Полезно будет эту табличку запомнить на будущее. А пока рассмотрим регистр UART_CR2:

22.7.6 Control register 2 (UART_CR2)

Address offset: 0x05

Reset value: 0x00

7	6	5	4	3	2	1	0
TIEN	TCIEN	RIEN	ILIEN	TEN	REN	RWU	SBK
rw	rw	rw	rw	rw	rw	rw	rw

Bit 7 **TIEN**: Transmitter interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever TXE=1 in the UART_SR register

Bit 6 **TCIEN**: Transmission complete interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever TC=1 in the UART_SR register

Bit 5 **RIEN**: Receiver interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever OR=1 or RXNE=1 in the UART_SR register

Bit 4 **ILIEN**: IDLE Line interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An UART interrupt is generated whenever IDLE=1 in the UART_SR register

Bit 3 **TEN**: Transmitter enable ⁽¹⁾ ⁽²⁾

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Bit 2 **REN**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

Bit 1 **RWU**: Receiver wakeup

– UART mode

This bit determines if the UART is in mute mode or not. It is set and cleared by software and can be cleared by hardware when a wakeup sequence is recognized.⁽³⁾ ⁽⁴⁾

– LIN slave mode (UART2, UART3 and UART4 only, if bits LINE and LSLV are set)

While LIN is used in slave mode, setting the RWU bit allows the detection of Headers only and prevents the reception of any other characters. Refer to [Mute mode and errors on page 350](#). In LIN slave mode, when RXNE is set, the software can not set or clear the RWU bit.

0: Receiver in active mode

1: Receiver in mute mode

Bit 0 **SBK**: Send break

This bit set is used to send break characters. It can be set and cleared by software. It should be set by software, and will be reset by hardware during the stop bit of break.

0: No break character is transmitted

1: Break character will be transmitted

1. During transmission, a "0" pulse on the TEN bit ("0" followed by "1") sends a preamble (idle line) after the current word.
2. When TEN is set there is a 1 bit-time delay before the transmission starts.
3. Before selecting Mute mode (by setting the RWU bit) the UART must first receive a data byte, otherwise it cannot function in Mute mode with wakeup by Idle line detection.
4. In Address Mark Detection wakeup configuration (WAKE bit=1) the RWU bit cannot be modified by software while the RXNE bit is set.

(/img/stm8/uart1_cr2.png)

Здесь нас интересуют биты 3 **TEN** и 2 **REN**, разрешение передачи и приема соответственно. В принципе, т.к. микроконтроллер работает только на передачу, можно было выставить только **TEN**, тогда пин Rx можно было бы использовать в других задачах.

Биты с седьмого по четвертый устанавливают источник вызова прерывания.

Бит **RWU**, определяет находится ли UART в "спящем" режиме или в активном. Как я понял из описания, UART можно вывести из режима сна, передачей специальной последовательности.

SBK - этот бит используется для отправки разрывающего стоп-бита. Устанавливается программно, сбрасывается аппаратно.

Регистр UARTx_CR3:

22.7.7

Control register 3 (UART_CR3)

Address offset: 0x06

Reset value: 0x00

7	6	5	4	3	2	1	0
Reserved	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL
r	rw	rw	rw	rw	rw	rw	rw

Bit 7 Reserved, must be kept cleared.

Bit 6 **LINEN**: LIN mode enable
This bit is set and cleared by software.
0: LIN mode disabled
1: LIN mode enabled

Bits 5:4 **STOP**: STOP bits.
These bits are used for programming the stop bits.
00: 1 Stop bit
01: Reserved
10: 2 Stop bits
11: 1.5 Stop bits
Note: For LIN slave mode, both bits should be kept cleared.

Bit 3 **CLKEN**: Clock enable
This bit allows the user to enable the SCLK pin.
0: SLK pin disabled
1: SLK pin enabled
Note: This bit is not available for UART3.

Bit 2 **CPOL**: Clock polarity ⁽¹⁾
This bit allows the user to select the polarity of the clock output on the SCLK pin. It works in conjunction with the CPHA bit to produce the desired clock/data relationship
0: SCK to 0 when idle
1: SCK to 1 when idle.
Note: This bit is not available for UART3.

Bit 1 **CPHA**: Clock phase ⁽¹⁾
This bit allows the user to select the phase of the clock output on the SCLK pin. It works in conjunction with the CPOL bit to produce the desired clock/data relationship
0: The first clock transition is the first data capture edge
1: The second clock transition is the first data capture edge
Note: This bit is not available for UART3.

Bit 0 **LBCL**: Last bit clock pulse. ⁽¹⁾⁽²⁾
This bit allows the user to select whether the clock pulse associated with the last data bit transmitted (MSB) has to be output on the SCLK pin.
0: The clock pulse of the last data bit is not output to the SCLK pin.
1: The clock pulse of the last data bit is output to the SCLK pin.
Note: This bit is not available for UART3.

1. These 3 bits (CPOL, CPHA, LBCL) should not be written while the transmitter is enabled.
2. The last bit is the 8th or 9th data bit transmitted depending on the 8 or 9 bit format selected by the M bit in the UART_CR1 register.

(/img/stm8/uart1_cr3.png)

Здесь нас интересуют **STOP** биты 5:6, которые определяют количество стоп-битов: одни, два или полтора;

LINEN включает LIN режим;

Остальные биты настраивают вывод синхронизирующего тактового сигнала SCLK pin для работы USART в синхронном режиме.

- **CLKEN** включает выдачу тактового сигнала для SCLK pin.
- **CPOL** задает полярность SCLK pin;
- **CPHA** задет фазу SCLK pin. Передача бита данных будет синхронизирована по первому фронту или по второму;
- **LBCL** задет вывод последнего бита данных на SCLK pin. Т.к. USART передает фрейм младшим битом вперед, то последним битом

будет соответственно старший 8-й или 9-й бит данных, в зависимости от формата фрейма.

Использование USART1 в чипе STM8L051F3 (добавлено 31 августа 2016г.)

В L-серии UART модули лишены LIN и CAN интерфейсов, в остальном же соответствуют USART модулям в STM8S.

Струтура писывающая модуль USART в SPL идентична структуре в S-серии, за тем исключением, что она общая для всех модулей:

```
typedef struct USART_struct
{
    __IO uint8_t SR;      /*!< USART status register */
    __IO uint8_t DR;      /*!< USART data register */
    __IO uint8_t BRR1;    /*!< USART baud rate register */
    __IO uint8_t BRR2;    /*!< USART DIV mantissa[11:8] SCIDIV fraction */
    __IO uint8_t CR1;     /*!< USART control register 1 */
    __IO uint8_t CR2;     /*!< USART control register 2 */
    __IO uint8_t CR3;     /*!< USART control register 3 */
    __IO uint8_t CR4;     /*!< USART control register 4 */
    __IO uint8_t CR5;     /*!< USART control register 5 */
    __IO uint8_t GTR;     /*!< USART guard time register */
    __IO uint8_t PSCR;    /*!< USART prescaler register */
}
USART_TypeDef;
```

Поэтому во всех функциях SPL имеется дополнительный параметр: USART_TypeDef* USARTx - номер модуля USART:

```
/* Function used to set the USART configuration to the default reset state */
void USART_DeInit(USART_TypeDef* USARTx);

/* Initialization and Configuration functions */
void USART_Init(USART_TypeDef* USARTx, uint32_t BaudRate, USART_WordLength_TypeDef
               USART_WordLength, USART_StopBits_TypeDef USART_StopBits,
               USART_Parity_TypeDef USART_Parity, USART_Mode_TypeDef USART_Mode);
void USART_ClockInit(USART_TypeDef* USARTx, USART_Clock_TypeDef USART_Clock,
                    USART_CPOL_TypeDef USART_CPOL, USART_CPHA_TypeDef USART_CPHA,
                    USART_LastBit_TypeDef USART_LastBit);
void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);
void USART_SendBreak(USART_TypeDef* USARTx);

/* Data transfers functions */
void USART_SendData8(USART_TypeDef* USARTx, uint8_t Data);
void USART_SendData9(USART_TypeDef* USARTx, uint16_t Data);
uint8_t USART_ReceiveData8(USART_TypeDef* USARTx);
uint16_t USART_ReceiveData9(USART_TypeDef* USARTx);

/* Multi-Processor Communication functions */
void USART_WakeUpConfig(USART_TypeDef* USARTx, USART_WakeUp_TypeDef USART_WakeUp);
void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);

/* Half-duplex mode function */
void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState);


/* Smartcard mode functions */
void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState);
void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);

/* IrDA mode functions */
void USART_IrDAConfig(USART_TypeDef* USARTx, USART_IrDAMode_TypeDef USART_IrDAMode);
void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);

/* DMA transfers management functions */
void USART_DMACmd(USART_TypeDef* USARTx, USART_DMAREq_TypeDef USART_DMAREq,
                 FunctionalState NewState);

/* Interrupts and flags management functions */
void USART_ITConfig(USART_TypeDef* USARTx, USART_IT_TypeDef USART_IT,
                   FunctionalState NewState);
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, USART_FLAG_TypeDef USART_FLAG);
void USART_ClearFlag(USART_TypeDef* USARTx, USART_FLAG_TypeDef USART_FLAG);
ITStatus USART_GetITStatus(USART_TypeDef* USARTx, USART_IT_TypeDef USART_IT);
void USART_ClearITPendingBit(USART_TypeDef* USARTx); //, USART_IT_TypeDef USART_IT);
```





Пример использования printf() для STM8L051F3 выглядит следующим образом:



Начать обсуждение...

войти с помощью

или через DISQUS



Имя

Прокомментируйте первым.

ТАКЖЕ НА COUNT-ZERO

ATmega8: ультразвуковой сенсор HC-SR04 на прерывании по захвату(capture interrupt)

2 комментариев • 2 года назад

flanker — >Чем закончилась работа по "перекрещиванию" двух УЗ-сонаров?Ничем. В смысле, вопрос отложен в долгий ящик. На мой взгляд, размещать на поворотной ...

3-х уровневый конвейер STM8: перевод глав 3, 4, 5 руководства по программированию ...

1 комментарий • 10 месяцев назад

Alexander Kovalyov — Для многих не профессионалов микроконтроллеры - хобби для домашних поделок. Я в их числе.Спасибо хозяину сайта за все статьи, не только по ...

ATmega8: работа на Си с USART/UART

4 комментариев • 2 года назад

flanker — 9600 для начала, и тщательное тестирование при крайних рабочих температурах. ну и там уже по результатам тестирования выставите нужную скорость.

Связь двух микроконтроллеров на примере подключения 4-х разрядного семисегментного ...

3 комментариев • год назад

Андрей Лебедев — спасибо... так я и думал... у меня в качестве передающего байт - Arduino Nano 328P с кварцем, а в качестве приёмника ATtiny13A... поиграл со ...