

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

Using hardware and software to make new stuff

Search

Categories

- [3D Printing](#) (1)
- [Affinity Photo](#) (1)
- [Aide-memoir](#) (1)
- [Analog](#) (1)
- [Ansible](#) (3)
- [Dates to Remember](#) (3)
- [Docker](#) (1)
- [Electronics](#) (144)
- [ESP8266](#) (12)
- [FPGA](#) (4)
- [Garden](#) (1)
- [General](#) (8)
- [Home Built CPU](#) (6)
- [Informatica](#) (1)
- [Internet of Things](#) (8)
- [iOS](#) (2)
- [KiCad](#) (4)
- [MSP430](#) (2)
- [Netduino](#) (49)
- [NuttX](#) (9)
- [Photography](#) (3)
- [Pico](#) (10)
- [Raspberry Pi](#) (17)
- [Silverlight](#) (6)
- [Software Development](#) (88)
- [STM32](#) (5)
- [STM8](#) (54)
- [Tips](#) (5)

Archives

- [July 2024](#)
- [June 2024](#)
- [May 2024](#)
- [April 2024](#)
- [March 2024](#)
- [February 2024](#)
- [January 2024](#)
- [December 2023](#)
- [November 2023](#)
- [October 2023](#)

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

- [April 2020](#)
- [February 2020](#)
- [January 2020](#)
- [July 2019](#)
- [February 2018](#)
- [July 2017](#)
- [June 2017](#)
- [April 2017](#)
- [March 2017](#)
- [February 2017](#)
- [October 2016](#)
- [September 2016](#)
- [July 2016](#)
- [June 2016](#)
- [May 2016](#)
- [April 2016](#)
- [March 2016](#)
- [January 2016](#)
- [December 2015](#)
- [November 2015](#)
- [October 2015](#)
- [August 2015](#)
- [June 2015](#)
- [May 2015](#)
- [April 2015](#)
- [March 2015](#)
- [February 2015](#)
- [January 2015](#)
- [December 2014](#)
- [October 2014](#)
- [September 2014](#)
- [August 2014](#)
- [July 2014](#)
- [June 2014](#)
- [May 2014](#)
- [March 2014](#)
- [February 2014](#)
- [January 2014](#)
- [December 2013](#)
- [November 2013](#)
- [October 2013](#)
- [September 2013](#)
- [August 2013](#)
- [July 2013](#)
- [June 2013](#)
- [May 2013](#)
- [April 2013](#)
- [March 2013](#)
- [January 2013](#)
- [November 2012](#)
- [October 2012](#)
- [September 2012](#)
- [August 2012](#)
- [July 2012](#)
- [June 2012](#)
- [May 2012](#)

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

- [October 2011](#)
- [September 2011](#)
- [July 2011](#)
- [June 2011](#)
- [May 2011](#)
- [April 2011](#)
- [March 2011](#)
- [February 2011](#)
- [January 2011](#)
- [December 2010](#)
- [April 2010](#)

Using the UART on the STM8S

We have previously seen how to configure the STM8S using the STD Peripheral Library using both the high level API and the slightly lower level register access. In this post we will use the definitions in the *iosstm8s103f3.h* and we will also have a look at the registers which control the UART ending with a program which uses the UART to send data to a terminal emulator on a desktop computer (in my case a PC). We must remember that while this article looks at low speed communications with a PC, the UARTs on the STM8S have a variety of uses covering other protocols other than those discussed in this article.

The definition of the problem is simple, allow the STM8S to send debug information to a terminal emulator running at 115200,n,8,1 (for all of those who remember DOS MODE commands for serial communication).

It should be noted that UART1 is not available on the STM8S Discovery board and so UART2 is used instead.

The Registers

In order to set up the UART we will need to perform the following tasks:

1. set the parity and number of data bits
2. set the parity
3. set the number of stop bits
4. setup the baud rate
5. set the clock polarity etc.

It is important to remember that transmission and reception must both be disabled before we start to change these registers.

UART_CR1 – Data Bits and Parity

The number of data bits is selected using the M bit of CR1. This can be set to either 8 or 9 bits. We will be using 8 data bits and so will need to set UART_CR1_M to 0 (setting to 1 would enable 9 data bits).

To set the parity we would set PCEN and PS. In our case we are disabling parity so only need to worry about PCEN (Parity Control Enable). Setting this bit to 0 will disable the parity calculation.

UART_CR3 – Number of Stop Bits and Clock Settings

The number of stop bits can be set to 1, 1.5 or 2. This is controlled by setting UART_CR3_STOP to one of the following values:

Value Description

00 1 Stop bit

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

Next, we need to consider the clock settings.

UART_CR3_CPOL determines the idle state of the clock, 0 sets the clock low when idle, 1 sets it high when idle.

UART_CR3_CPHA determines if the data should be stable on the rising or falling edge of the signal. Setting this to 0 means the data is set on the rising edge of the clock signal. Setting this to 1 means the data is ready on the falling edge of the clock signal.

UART_CR3_LBCL determines if the clock pulse for the last data bit is set to the clock pin. A 0 means the last clock pulse is not generated while 1 means that the pulse is generated.

UART_BRR1 & UART_BRR2 – Baud Rate Registers

The baud rate of the UART is controlled by dividing f_{master} by the baud rate divisor. The result gives the clock speed of the serial port. In our case we have a 16 MHz clock speed for the microcontroller and we want the serial port to run at 115200 baud. So some simple rearranging of the formula and the UART divider is given by:

$$\begin{aligned}\text{UART Divider} &= f_{\text{master}} / \text{baud rate} \\ &= 16,000,000 / 115,200 \\ &= 138 \\ &= 0x008a\end{aligned}$$

Now we need to rearrange the number *0x008a* a little in order to get the right bits into BRR1 and BRR2 (Baud Rate Register 1 & 2). This was written as a 32 bit number to illustrate how this is put into the registers. To do this we split the number (represented by $d_3d_2d_1d_0$) into three parts:

- the first digit (d_3) – 0
- the next two digits (d_2d_1) – 08
- the last digit (d_0) – a

And set up the registers as follows:

$$\begin{aligned}\text{BRR1} &= d_2d_1 \\ &= 0x08 \\ \text{BRR2} &= d_3d_0 \\ &= 0x0a\end{aligned}$$

When setting these registers it is important to remember to set BRR2 *before* setting BRR1.

UART_CR2 & UART_CR3 – Enabling the UART

The first action we would need to take is to disable the UART and the last thing we should do is enable it. This is controlled by three bits in two different registers, namely registers UART_CR3 and UART_CR2. All three bits use 0 for disable and 1 for enable. The bits we been to set are:

UART_CR2_TEN Enable/disable transmission
UART_CR2_REN Enable/disable reception
UART_CR3_CKEN Enable/disable the clock

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

Software

Moving on to our software, we will need a standard STM8 project for the microcontroller you are using. I am using the STM8S103F3P3 and the default project I set up in a previous article.

Setting Up the UART

This code makes a fundamental assumption, namely that you have configured the chip and your circuit to run at 16 MHz. I did this by setting the chip to use the internal oscillator as its clock source and using a prescaler of 1 (see the previous article on [setting up the system clock](#) for more information). The code for this is in the *InitialiseSystemClock* method.

The next step is to configure the UART. This is performed in the *InitialiseUART* method.

```
1  //
2  // Setup the UART to run at 115200 baud, no parity, one stop bit, 8 data bit
3  //
4  // Important: This relies upon the system clock being set to run at 16 MHz.
5  //
6  void InitialiseUART()
7  {
8      //
9      // Clear the Idle Line Detected bit in the status register by a read
10     // to the UART1_SR register followed by a Read to the UART1_DR register.
11     //
12     unsigned char tmp = UART1_SR;
13     tmp = UART1_DR;
14     //
15     // Reset the UART registers to the reset values.
16     //
17     UART1_CR1 = 0;
18     UART1_CR2 = 0;
19     UART1_CR4 = 0;
20     UART1_CR3 = 0;
21     UART1_CR5 = 0;
22     UART1_GTR = 0;
23     UART1_PSCR = 0;
24     //
25     // Now setup the port to 115200,n,8,1.
26     //
27     UART1_CR1_M = 0;           // 8 Data bits.
28     UART1_CR1_PCEN = 0;       // Disable parity.
29     UART1_CR3_STOP = 0;       // 1 stop bit.
30     UART1_BRR2 = 0x0a;        // Set the baud rate registers to 115200 baud
31     UART1_BRR1 = 0x08;        // based upon a 16 MHz system clock.
32     //
33     // Disable the transmitter and receiver.
34     //
35     UART1_CR2_TEN = 0;        // Disable transmit.
36     UART1_CR2_REN = 0;        // Disable receive.
37     //
38     // Set the clock polarity, lock phase and last bit clock pulse.
39     //
40     UART1_CR3_CPOL = 1;
41     UART1_CR3_CPHA = 1;
```

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

```

47     UART1_CR2_REN = 1;
48     UART1_CR3_CKEN = 1;
49 }

```

We will need to provide a method of sending a simple string to the serial port. The algorithm is simple:

1. Set a pointer to the start of the string
2. If the character pointed to be the pointer is not a null (i.e. 0) character then
 1. Transfer the character pointed to be the pointer into the UART data register
 2. Wait until the data register has been sent (Transmission Empty is true)
 3. Move the pointer on one byte

```

1  //
2  //  Send a message to the debug port (UART1).
3  //
4  void UARTPrintf(char *message)
5  {
6      char *ch = message;
7      while (*ch)
8      {
9          UART1_DR = (unsigned char) *ch;    // Put the next character into t
10         while (UART1_SR_TXE == 0);        // Wait for transmission to comp
11         ch++;                               // Grab the next character.
12     }
13 }

```

And finally we need a main program to control the application.

```

1  //
2  //  Main program loop.
3  //
4  void main()
5  {
6      __disable_interrupts();
7      InitialiseSystemClock()
8      InitialiseUART()
9      __enable_interrupts();
10     while (1)
11     {
12         UARTPrintf("Hello from my microcontroller...\n\r");
13         for (long counter = 0; counter < 250000; counter++);
14     }
15 }

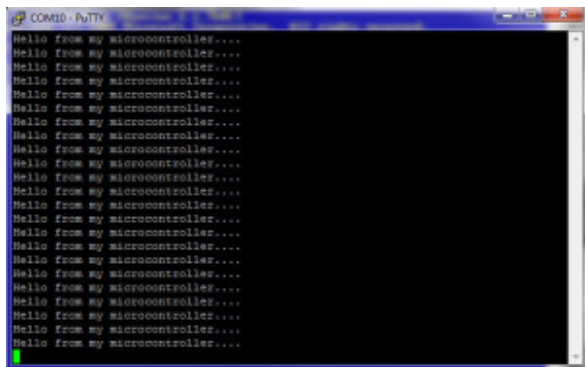
```

The full application code can be [downloaded from here](#). Simply unzip the files and open the project with IAR. The application can be downloaded to the chip by pressing Ctrl-D. Once downloaded to the microcontroller press F5 to run the application.

This application has been tested on my reference platform, the Variable Labs Protomodule and the STM8S Discovery board.

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)

terminal emulator (I used PuTTY) and connect to the COM port using the correct protocol. You should see the following output:



Output from the STM8S shown on a PuTTY terminal

Conclusion

Adding these methods to your project should allow you to generate debug output from an application running on the STM8S or communicate with devices which are controlled using a serial communication protocol.

Source Code Compatibility

System	Compatible?
STM8S103F3 (Breadboard)	✓
Variable Lab Protomodule	✓
STM8S Discovery	✓

Tags: [Electronics](#), [Software Development](#), [STM8](#), [The Way of the Register](#)

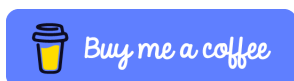
Monday, August 27th, 2012 at 10:04 am • [Electronics](#), [Software Development](#), [STM8](#) • [RSS 2.0](#) feed Both comments and pings are currently closed.

Comments are closed.

Pages

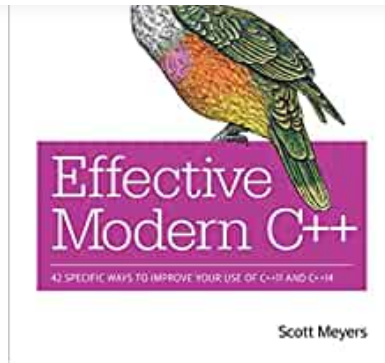
- [About](#)
- [Making a Netduino GO! Module](#)
- [The Way of the Register](#)
- [Making an IR Remote Control](#)
- [Weather Station Project](#)
- [NuttX and Raspberry Pi PicoW](#)

Support This Site



Currently Reading

This website uses cookies to improve your experience and to gather page view statistics. This site does not collect user information. [Accept & Close](#) [Read More](#)



© 2010 - 2024 Mark Stevens