Using the USART

Content

1. Introduction

The STM32 Discovery is equipped with three USARTs. USART stands for Universal Synchronous Asynchronous Receiver Transmitter. The USARTs on the STM32 support a wide range of serial protocols, the usual asynchronous ones, plus things like IrDA, SPI etc.

In this article I am going to describe how the STM32 Discovery could be connected to the serial port (RS232) of a PC, and how the STM32 has to be programmed to communicate with the PC. We will cover both: sending and receiving.

Since the STM32 works on 3.3V levels, a level shifting component is needed to connect the USART of the STM32 to a PC serial port. Most PCs today come without an native RS232 port, thus an USB to serial converter is also needed. For the level shifting, and the USB to serial conversation, we use the FTDI based FT232RL in its **breadboard variant from sparkfun (http://www.sparkfun.com/products/718)** . For more details and a very basic usage example see **this (./?page_id=27)** article.

2. Prerequisites

- **Toolchain for building (./?page_id=131)** binaries for the STM32 is in place
- **Toolchain for flashing (./?page_id=148)** binaries to the STM32 is in place
- Basic understanding of the STM32 (GPIOs)
- **Basic understanding of the FT232RL (./?page_id=27)**
- Basic understanding of how to program the STM32 in C

3. Parts Needed

- **STM32 Discovery (http://www.st.com/internet/evalboard/product/250863.jsp)** board

- FT232RL (**breadboard version from sparkfun (http://www.sparkfun.com/products/718)**)
- A breadboard plus some breadboard wires

4. The Setup

In this example, we simply connect the RX of the STM32 USART1 to the TX of the FT232RL, and the TX of the USART1 to the RX of the FT232RL. The program we load onto the STM32 then waits until a character is received, takes that character, wraps square braces around it, and echoes back the received char in braces. E.g. if the character “A” is received, “[A]” is send back. Thus sending “Hello” will echo “[H][e][l][l][o]“. Every time a character is received, the build in green LED on the STM32 is flashed, every time the echo is send, the build in blue LED is flashed. On the PC side we use a serial terminal program (with local echo turned off) to send and receive characters to and from the STM32.

5. Wiring the Hardware

The wiring we do here is pretty simple. Since we do not use hardware flow control or any other fancy stuff, only the RX/TX pins of the STM32 and the FT232RL are needed. The table below shows the pin outs of the three USARTS available on the STM32 Discovery:

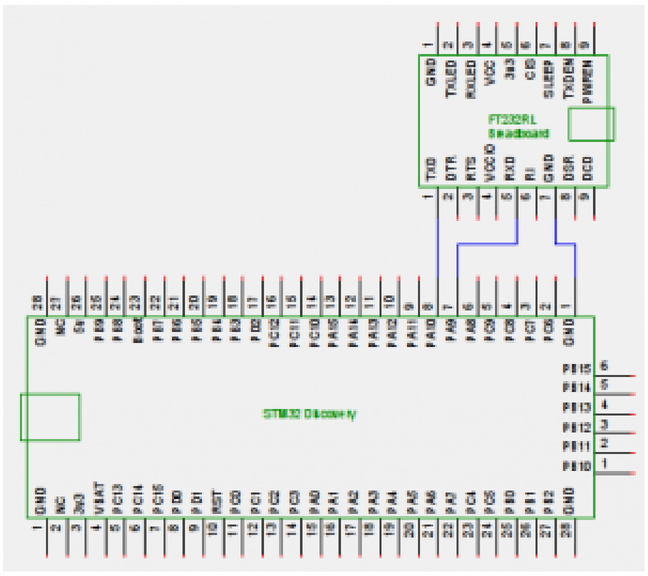
Pin	Function
PA8	USART1_CK
PA11	USART1_CTS
PA12	USART1_RTS
PA9	USART1_TX
PA10	USART1_RX
PA4	USART2_CK
PA0	USART2_CTS
PA1	USART2_RTS
PA2	USART2_TX
PA3	USART2_RX
PB12	USART3_CK
PB13	USART3_CTS
PB14	USART3_RTS
PB10	USART3_TX
PB11	USART3_RX

As one can see in the figure above, all the USARTs are fully equipped. The main difference is, that only USART1 is connected to the high-speed bus APB2. All the other USARTs are connected to APB1.

In this tutorial, we are going to use USART1. Thus we wire up the following:

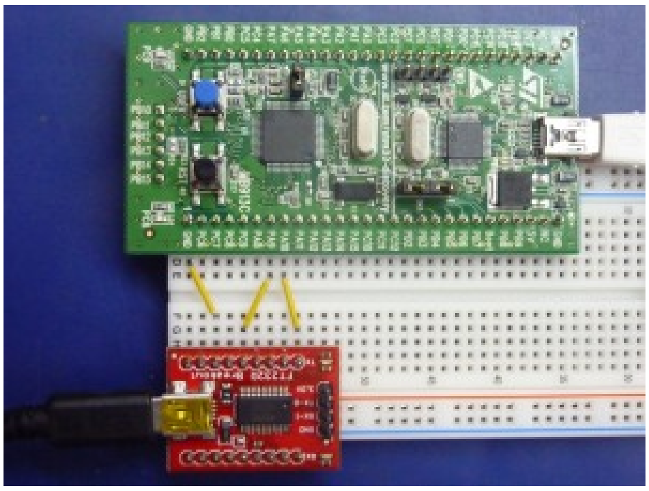
- PA9 from the STM32 (USART1_TX) gets connected to pin RXD of the FT232RL

- PA10 from the STM32 (USART1_RX) gets connected to pin TXD of the FT232RL
- GND of the STM32 gets connected to GND of the FT232RL



(http://gpio.kaltpost.de/wp-content/uploads/2011/07/stm32_usart.png)

Schamtic of STM32 USART connected to FT232RL



(http://gpio.kaltpost.de/wp-content/uploads/2011/07/stm32_usart_breadbord.jpg)

STM32 connected to FT232RL on breadbord

Thats it! Double check your wiring, connect the SMT32 and the FT232RL to your USB bus, and as always, be sensitive about funny smells comming from your components.

6. Writing the Software

For this tutorial we are going to program the STM32 on the "bare metal". The only things we use are some header files from the **STM32 library (http://www.st.com/internet/com/SOFTWARE_RESOURCES/SW_COMPONENT/FIRMWARE/stm32f10x_stdperiph_lib.zip)** to give us the needed register definitions.

To use an USART, the board has to be initialized properly first. In detail, we need to setup the following:

- Enable the system clock for USART1
- Setup the RX/TX pins for USART1
- Set the baudrate for USART1
- Enable USART1 and its RX- and TX-component

Since the presets for flow-control and stop-bit are already set to N and 1 by default, we don't need to configure them for this example.

To send/receive data, the DR (Data Register) register of the USART is used. If you read the register, you get the last byte received. If you write to the register, the byte written is transmitted. Now the tricky part is to know, if new data could be written to the register, or if new data could be read.

One (bad) solution for the sending thing is to use a delay after writing to the register, and if the delay is big enough, we can be sure that the data is transmitted. the bad solution for the receiving part is to poll the register until the byte read differs from the last byte read.

A better solution is to use the SR (Status Register) register of the USART1. The status register has a bit indicating if the data is transmitted as well as a bit indicating if new data was received. As soon one reads the DR register, the data received flag is automatically reset. This is exactly the solution we are going to use in out code.

An alternative would be to enable the USART1 interrupt for data received. By doing so, an interrupt handler is called each time new data was received.

In the next sub paragraphs I am going to detail how the setup, the sending and the receiving could be programmed in C. We will therefore look at the following operations:

- board_init: all the initialization is done her
- usart_rec: receive data blocking
- usart_snd: send data blocking

1. board_init

At least we need to enable the system clock for USART1 and the GPIOA block (since RX/TX are on that block). For using the build in LEDs, we also enable the GPIOC. This is done by setting the RCC register for the APB2 bus (both, GPIOA and USART1 are connected to this bus):

```
RCC->APB2ENR = 0
    // Turn on USART1
    | RCC_APB2ENR_USART1EN
    // Turn on IO Port A
    | RCC_APB2ENR_IOPAEN;
```

Now we are going to setup the GPIOs for PA9 (TX) and PA10 (RX). The TX pin must be setup as an output-pin with alternate function (USART1_TX). We configure this pin as output pin in push-pull mode at a frequency of 50 Mhz. Puzzling all the bits for the config register together, we end up in 0xB for this configuration. P10 (RX) must be setup as input. We use floating input for that since the connected FT232RL takes care of the pull-up/down stuff. To enable floating input for a pin, 0×4 must be written to the config register. In the code this would look like this:

```
// Put PA9  (TX) to alternate function output push-pull at 50 MHz
// Put PA10 (RX) to floating input
GPIOA->CRH = 0x000004B0;
```

Note: we use CHR since the pins 8-15 are in the H register, where as pins 0-7 are in the L part.

Next we need to setup the USART. First we are going to define the baud rate to use. This is done by using the USARTs BRR register. Since the STM32 has a fractional generator any baud rate must be derived from the systems bus clock. The value to store in the BRR register simply could by calculated by this simple formula:

USART_BRR = [BUS_FREQUENCY_in_Hz] / [BAUD_RATE]

To setup the USART for 38400 bauds, we do the following in our code:

```
    // Configure BRR by deviding the bus clock with the baud rate
    USART1->BRR = 8000000/38400;
```

The last thing we have to do is to enable the UART, the TX unit of the URAT and the RX unit of the USART. This is done by setting the corresponding flags on th config register 1 of the UART1:

```
USART1->CR1 = USART_CR1_UE | USART_CR1_TE | USART_CR1_RE;
```

Thats it. Your USART is now ready to communicate with 38400 bauds at 8 data bits, no flow control and one stop-bit.

2. usart_rec

To read in received data, all we have to do is to wait until RXNE (RX not empty) bit is set in the status register SR of the USART1. As soon as we realized, this bit is set, we read out the data from the DR (data register) of the USART. Since the DR register is 32 bit, but only 9 Bits are used, we apply the mask 0x1FF to the incoming data to make sure we receive no additional garbage. As soon as we read the DR register, the USART resets the RXNE flag automatically. In our program it would look like this:

```
   /* Wait until the data is ready to be received. */
   while ((USART1->SR & USART_SR_RXNE) == 0);

   // read RX data, combine with DR mask (we only accept a max of 9 Bits)
   return USART1->DR & 0x1FF;
```

3. board_snd

To send data, we simply assign it to the DR register. Then we wait until the TXE (TX empty) flag in the SR register is set to be sure the data is transmitted. In the code it looks like this:

```
   USART1->DR = data;

   // wait for TX
   while ((USART1->SR & USART_SR_TXE) == 0);
```

7. Downloads

- Example source code, including schematic: **stm32_usart_loopback_example.tar.gz (http://gpio.kaltpost.de/wp-content/uploads/2011/07/stm32_usart_loopback_example.tar.gz)**

1 Comment to “Using the USART”

You can follow all the replies to this entry through the **comments (http://gpio.kaltpost.de/?feed=rss2&page_id=167)** feed.

1. [() S.S.](#)
   1. Januar 2013 at 14:46 | **Permalink (http://gpio.kaltpost.de/?page_id=167#comment-182)**

   Nice tutorial! I like the bare-bone programming method!

   Keep it up!

Log in to reply. (http://gpio.kaltpost.de/wp-login.php?redirect_to=http%3A%2F%2Fgpio.kaltpost.de%2F%3Fpage_id%3D167)

Leave a Reply

You must be **logged in (http://gpio.kaltpost.de/wp-login.php?redirect_to=http%3A%2F%2Fgpio.kaltpost.de%2F%3Fpage_id%3D167)** to post a comment.

Search

Search... Search

Stuff

- **HOWTOs (http://gpio.kaltpost.de/?page_id=82)**
- **libemb (http://gpio.kaltpost.de/?page_id=708)**
- **YWasp (http://gpio.kaltpost.de/?page_id=748)**
- **github (https://github.com/wendlers)**
- **uSherpa (http://gpio.kaltpost.de/?page_id=1267)**
- **Pynetsense (http://gpio.kaltpost.de/?page_id=1641)**
- **pyscmpd (http://gpio.kaltpost.de/?page_id=1588)**

Blog Categories

- **3D-Printing (http://gpio.kaltpost.de/?cat=224)**
- **Beaglebone (http://gpio.kaltpost.de/?cat=127)**
- **C/C++ (http://gpio.kaltpost.de/?cat=80)**
- **Carambola (http://gpio.kaltpost.de/?cat=153)**
- **Cubieboard (http://gpio.kaltpost.de/?cat=210)**
- **Electronics (http://gpio.kaltpost.de/?cat=3)**
- **libemb (http://gpio.kaltpost.de/?cat=86)**
- **Linux (http://gpio.kaltpost.de/?cat=23)**
- **Microcontrollers (http://gpio.kaltpost.de/?cat=7)**
- **Misc (http://gpio.kaltpost.de/?cat=1)**
- **MSP430 (http://gpio.kaltpost.de/?cat=55)**
- **Neo FreeRunner (http://gpio.kaltpost.de/?cat=101)**
- **Off Topic (http://gpio.kaltpost.de/?cat=16)**
- **PIC (http://gpio.kaltpost.de/?cat=15)**
- **Programming (http://gpio.kaltpost.de/?cat=79)**
- **Propeller (http://gpio.kaltpost.de/?cat=172)**
- **pyscmpd (http://gpio.kaltpost.de/?cat=180)**
- **Python (http://gpio.kaltpost.de/?cat=157)**
- **Raspberry Pi (http://gpio.kaltpost.de/?cat=179)**
- **ROCKETuC (http://gpio.kaltpost.de/?cat=164)**
- **STM32 (http://gpio.kaltpost.de/?cat=8)**

- uSherpa (http://gpio.kaltpost.de/?cat=182)
- YWasp (http://gpio.kaltpost.de/?cat=88)

Last Comments

- Fabian bei MSP430G2553 SPI to nRF24l01+ (http://gpio.kaltpost.de/?page_id=983#comment-296)
- krishnak bei MSP430G2553 Hardware UART (http://gpio.kaltpost.de/?page_id=972#comment-293)
- Stefan bei MSP430G2553 Hardware UART (http://gpio.kaltpost.de/?page_id=972#comment-292)
- Robo bei MSP430G2553 Hardware UART (http://gpio.kaltpost.de/?page_id=972#comment-288)
- Stefan bei MSP430G2553 Hardware UART (http://gpio.kaltpost.de/?page_id=972#comment-285)

Archive

- Mai 2014 (http://gpio.kaltpost.de/?m=201405)
- April 2014 (http://gpio.kaltpost.de/?m=201404)
- März 2014 (http://gpio.kaltpost.de/?m=201403)
- Februar 2014 (http://gpio.kaltpost.de/?m=201402)
- Januar 2014 (http://gpio.kaltpost.de/?m=201401)
- Dezember 2013 (http://gpio.kaltpost.de/?m=201312)
- November 2013 (http://gpio.kaltpost.de/?m=201311)
- Juni 2013 (http://gpio.kaltpost.de/?m=201306)
- Mai 2013 (http://gpio.kaltpost.de/?m=201305)
- Februar 2013 (http://gpio.kaltpost.de/?m=201302)
- Januar 2013 (http://gpio.kaltpost.de/?m=201301)
- November 2012 (http://gpio.kaltpost.de/?m=201211)
- Oktober 2012 (http://gpio.kaltpost.de/?m=201210)
- September 2012 (http://gpio.kaltpost.de/?m=201209)
- August 2012 (http://gpio.kaltpost.de/?m=201208)
- Juli 2012 (http://gpio.kaltpost.de/?m=201207)
- Juni 2012 (http://gpio.kaltpost.de/?m=201206)
- Mai 2012 (http://gpio.kaltpost.de/?m=201205)
- April 2012 (http://gpio.kaltpost.de/?m=201204)
- März 2012 (http://gpio.kaltpost.de/?m=201203)
- Februar 2012 (http://gpio.kaltpost.de/?m=201202)
- Januar 2012 (http://gpio.kaltpost.de/?m=201201)
- Dezember 2011 (http://gpio.kaltpost.de/?m=201112)
- November 2011 (http://gpio.kaltpost.de/?m=201111)
- Oktober 2011 (http://gpio.kaltpost.de/?m=201110)
- September 2011 (http://gpio.kaltpost.de/?m=201109)
- August 2011 (http://gpio.kaltpost.de/?m=201108)
- Juli 2011 (http://gpio.kaltpost.de/?m=201107)

Tags

3D Printing (http://gpio.kaltpost.de/?tag=3d-printing-2) 8devices (http://gpio.kaltpost.de/?tag=8devices) AppNote (http://gpio.kaltpost.de/?tag=appnote) ARM (http://gpio.kaltpost.de/?tag=arm) AVR (http://gpio.kaltpost.de/?tag=avr) Bluetooth (http://gpio.kaltpost.de/?tag=bluetooth) Carambola (http://gpio.kaltpost.de/?

tag=carambola) Cut-n-Paste (http://gpio.kaltpost.de/?tag=cut-n-paste) DC-Motors (http://gpio.kaltpost.de/?tag=dc-motors) etractor (http://gpio.kaltpost.de/?tag=etractor) GPIO (http://gpio.kaltpost.de/?tag=gpio) Hacks (http://gpio.kaltpost.de/?tag=hacks) Howto (http://gpio.kaltpost.de/?tag=howto) Java (http://gpio.kaltpost.de/?tag=java) JTAG (http://gpio.kaltpost.de/?tag=jtag) Launchpad (http://gpio.kaltpost.de/?tag=launchpad) LED (http://gpio.kaltpost.de/?tag=led) Lego (http://gpio.kaltpost.de/?tag=lego) libemb (http://gpio.kaltpost.de/?tag=libemb) Links (http://gpio.kaltpost.de/?tag=links) MSP430 (http://gpio.kaltpost.de/?tag=msp430) MSP430G2231 (http://gpio.kaltpost.de/?tag=msp430g2231) MSP430G2553 (http://gpio.kaltpost.de/?tag=msp430g2553) Nordic (http://gpio.kaltpost.de/?tag=nordic) nRF24L01 (http://gpio.kaltpost.de/?tag=nrf24l01) OpenOCD (http://gpio.kaltpost.de/?tag=openocd) Parallax (http://gpio.kaltpost.de/?tag=parallax) Power Supply (http://gpio.kaltpost.de/?tag=power-supply) Propeller (http://gpio.kaltpost.de/?tag=propeller) PWM (http://gpio.kaltpost.de/?tag=pwm) python (http://gpio.kaltpost.de/?tag=python-2) R2D2 (http://gpio.kaltpost.de/?tag=r2d2) Range Finder (http://gpio.kaltpost.de/?tag=range-finder) Raspberry Pi (http://gpio.kaltpost.de/?tag=raspberry-pi) Robot (http://gpio.kaltpost.de/?tag=robot) Servo (http://gpio.kaltpost.de/?tag=servo) SPIN (http://gpio.kaltpost.de/?tag=spin) STM32 (http://gpio.kaltpost.de/?tag=stm32) TB6612FNG (http://gpio.kaltpost.de/?tag=tb6612fng) TI (http://gpio.kaltpost.de/?tag=ti) Tin Can (http://gpio.kaltpost.de/?tag=tin-can) Tutorial (http://gpio.kaltpost.de/?tag=tutorial) Tutorials (http://gpio.kaltpost.de/?tag=tutorials) UART (http://gpio.kaltpost.de/?tag=uart) Ubuntu (http://gpio.kaltpost.de/?tag=ubuntu)

Blogroll

- 43oh:MSP430 News (http://www.43oh.com)
- Dangerous Prototypes (http://dangerousprototypes.com/)
- Dave's EEV Blog (http://www.eevblog.com/)
- Electronics lab (http://www.electronics-lab.com/blog/)
- element14 TV (http://www.element14.com/community/community/element14tv)
- Embedded Lab (http://embedded-lab.com/)
- hackaday (http://hackaday.com)
- hackaweek (http://hackaweek.com)
- Highspeed Movies (http://hs-movies.de/)
- Let's Make Robots! (http://letsmakerobots.com/)
- PartSim Circuit Simulation (http://www.partsim.com/)
- Pyroelectro (http://www.pyroelectro.com/)

Meta

- Anmelden (http://gpio.kaltpost.de/wp-login.php)
- Beitrags-Feed (RSS) (http://gpio.kaltpost.de/?feed=rss2)

- **Kommentare als RSS (http://gpio.kaltpost.de/?feed=comments-rss2)**
- **WordPress.org (https://wordpress.org/)**