LABORATORIO DI ARCHITETTURE E PROGRAMMAZIONE DEI SISTEMI ELETTRONICI INDUSTRIALI

Laboratory Lesson 7:

Universal Serial Asynchronous Receiver Transmitter (USART)

Prof. Luca Benini <luca.benini@unibo.it>

Filippo Casamassima <filippo.casamassima@unibo.it>
Domenico Balsamo <domenico.balsamo@unibo.it>

Course Organization

- Hands-on session LAB1 **Thursday 15.00 – 19.00**
- Prof Benini Friday **9.00 – 11.00 room 5.5**
- Lab is available **Friday 11.00 – 13.00**
- Check website for **announcements, course material:** http://www-micrel.deis.unibo.it/LABARCH
- Final Exam:
  - Homeworks (**to be checked weekly**)
  - Final project
  - Final discussion (homeworks + final project)

USART Characteristics

USART (Universal Synchronous-Asynchronous Receiver/Transmitter)

The USART is the most used serial communication interface (eg. PC RS232 interface, IC communication interface, BT and WiFi module interface)

FEATURES:

- Full duplex, asynchronous communications
- Fractional baud rate generator systems

– A common programmable transmit and receive baud rates up to 4.5 MBits/s

- Programmable data word length (8 or 9 bits)
- Configurable stop bits - support for 1 or 2 stop bits
- Transmitter clock output for synchronous transmission
- Single wire half duplex communication
- Configurable multibuffer communication using DMA (direct memory access)

– Buffering of received/transmitted bytes in reserved SRAM using centralized DMA

- Separate enable bits for Transmitter and Receiver

FLAGS:

– Receive buffer full
– Transmit buffer empty
– End of Transmission flags

- **Parity control:**

– Transmits parity bit
– Checks parity of received data byte

- **Four error detection flags:**

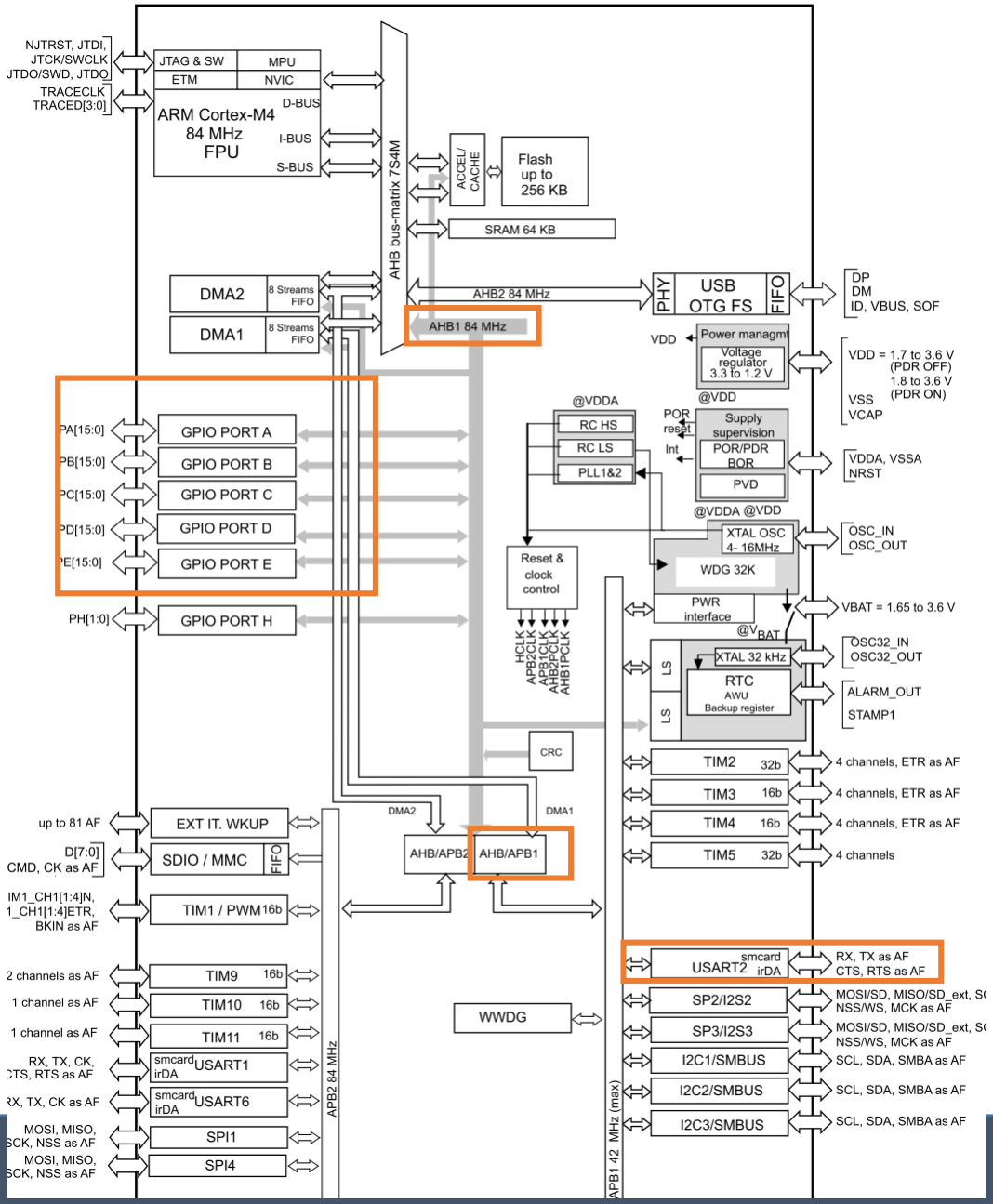
– Overrun error
– Noise error
– Frame error
– Parity error

- **Ten interrupt sources with flags:**

– CTS changes
– Transmit data register empty
– Transmission complete
– Receive data register full
– Idle line received
– Overrun error
– Framing error
– Noise error
– Parity error

- **Multiprocessor communication - enter into mute mode if address match does not occur**

USART


NJTRST, JTDI, JTCK/SWCLK JTDO/SWD, JTDO
TRACECLK TRACED[3:0]
JTAG & SW
MPU
ETM
NVIC
ARM Cortex-M4 84 MHz FPU
D-BUS
I-BUS
S-BUS
AHB bus-matrix 7S4M
ACCEL/ CACHE
Flash up to 256 KB
SRAM 64 KB
DMA2
DMA1
8 Streams FIFO
AHB2 84 MHz
AHB1 84 MHz
PHY
USB OTG FS
FIFO
DP
DM
ID, VBUS, SOF
VDD
Power managmt
Voltage regulator 3.3 to 1.2 V
VDD = 1.7 to 3.6 V (PDR OFF)
1.8 to 3.6 V (PDR ON)
VSS
VCAP
@VDDA
@VDD
RC HS
RC LS
PLL1&2
POR reset
Int
Supply supervision
POR/PDR
BOR
PVD
VDDA, VSSA
NRST
@VDDA @VDD
XTAL OSC 4- 16MHz
OSC_IN
OSC_OUT
WDG 32K
Reset & clock control
PWR interface
VBAT = 1.65 to 3.6 V
@VBAT
XTAL 32 kHz
OSC32_IN
OSC32_OUT
LS
RTC
AWU
Backup register
ALARM_OUT
STAMP1
HCLK
APB2CLK
APB1CLK
AHB/8CLK
AHB1PCLK
CRC
PA[15:0]
GPIO PORT A
PB[15:0]
GPIO PORT B
PC[15:0]
GPIO PORT C
PD[15:0]
GPIO PORT D
PE[15:0]
GPIO PORT E
PH[1:0]
GPIO PORT H
TIM2 32b
TIM3 16b
TIM4 16b
TIM5 32b
4 channels, ETR as AF
4 channels, ETR as AF
4 channels, ETR as AF
4 channels
DMA2
DMA1
AHB/APB2
AHB/APB1
up to 81 AF
EXT IT. WKUP
D[7:0] CMD, CK as AF
SDIO / MMC
FIFO
IM1_CH1[1:4]N, 1_CH1[1:4]ETR, BKIN as AF
TIM1 / PWM 16b
2 channels as AF
TIM9 16b
1 channel as AF
TIM10 16b
1 channel as AF
TIM11 16b
RX, TX, CK, CTS, RTS as AF
smcard irDA USART1
X, TX, CK as AF
smcard irDA USART6
MOSI, MISO, SCK, NSS as AF
SPI1
MOSI, MISO, SCK, NSS as AF
SPI4
APB2 84 MHz
WWDG
USART2
smcard irDA
RX, TX as AF
CTS, RTS as AF
SP2/I2S2
MOSI/SD, MISO/SD_ext, S
NSS/WS, MCK as AF
SP3/I2S3
MOSI/SD, MISO/SD_ext, S
NSS/WS, MCK as AF
I2C1/SMBUS
SCL, SDA, SMBA as AF
I2C2/SMBUS
SCL, SDA, SMBA as AF
I2C3/SMBUS
SCL, SDA, SMBA as AF
APB1 42 MHz (max)

USART Tx/Rx

Transmitter

The transmitter can send data words of either 8 or 9 bits depending on the M bit status.
When the transmit enable bit (TE) is set, the data in the transmit shift register is output on the TX pin and the corresponding clock pulses are output on the CK pin.

Character transmission

During a USART transmission, data shifts out least significant bit first on the TX pin. In this mode, the USART_DR register consists of a buffer (TDR) between the internal bus and the transmit shift register
Every character is preceded by a start bit which is a logic level low for one bit period. The character is terminated by a configurable number of stop bits.
The following stop bits are supported by USART: 0.5, 1, 1.5 and 2 stop bits.
Note:
1 The TE bit should not be reset during transmission of data. Resetting the TE bit during the transmission will corrupt the data on the TX pin as the baud rate counters will get frozen. The current data being transmitted will be lost.
2 An idle frame will be sent after the TE bit is enabled.

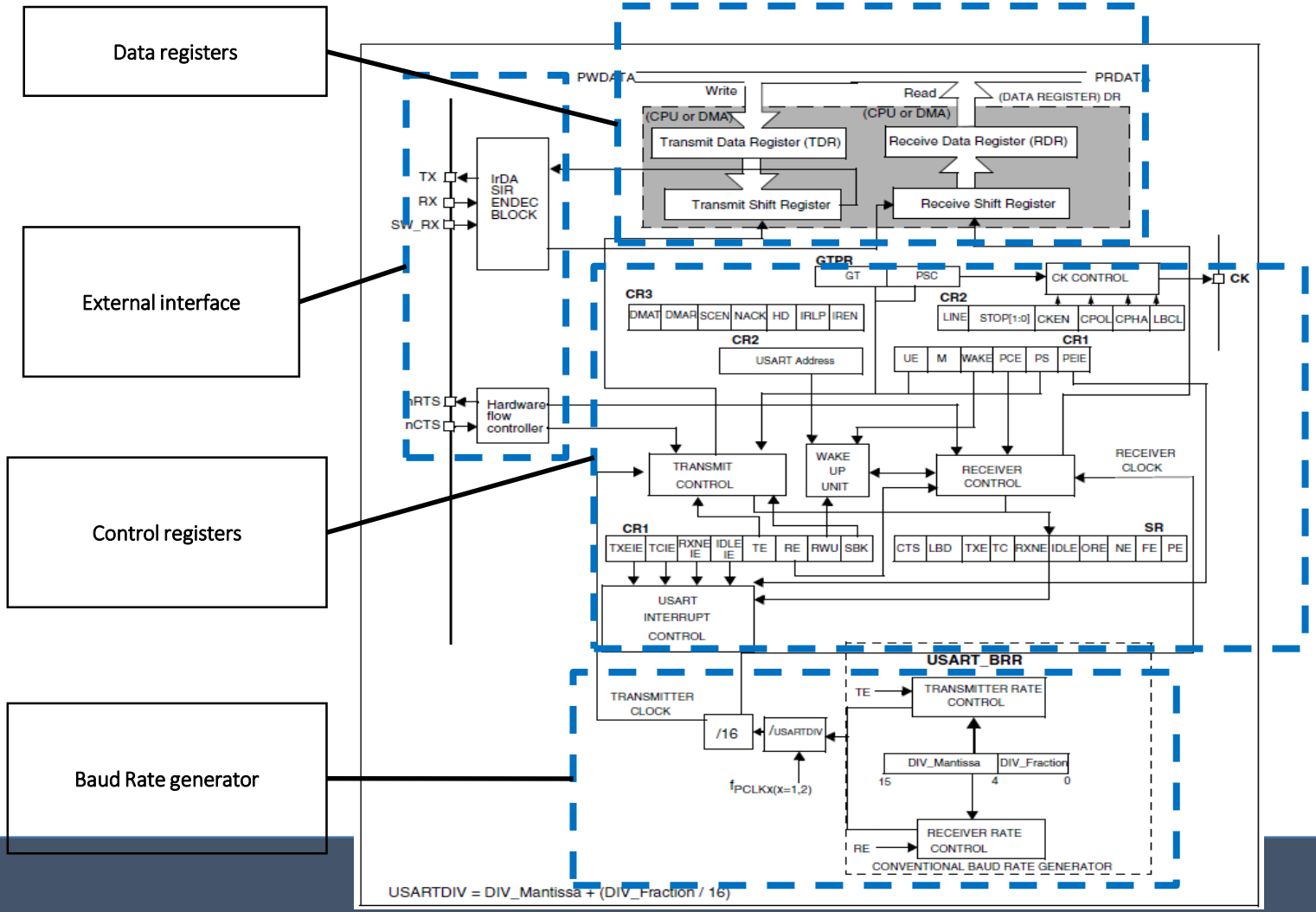
Receiver

The USART can receive data words of either 8 or 9 bits depending on the M bit in the USART_CR1 register.

Character reception

During a USART reception, data shifts in least significant bit first through the RX pin. In this mode, the USART_DR register consists of a buffer (RDR) between the internal bus and the received shift register.

USART block diagram



USART registers

	Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Status register	0x00	**USART_SR**	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
		Reset value																							0	0	1	1	0	0	0	0	0	0
Data register	0x04	**USART_DR**	Reserved																						DR[8:0]									
		Reset value																							0	0	0	0	0	0	0	0	0	0
Baud Rate register	0x08	**USART_BRR**	Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
		Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Control register 1	0x0C	**USART_CR1**	Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
Control register 2	0x10	**USART_CR2**	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
		Reset value																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Control register 3	0x14	**USART_CR3**	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
		Reset value																						0	0	0	0	0	0	0	0	0	0	0
Prescaler register	0x18	**USART_GTPR**	Reserved																GT[7:0]								PSC[7:0]							
		Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

USART Interrupts

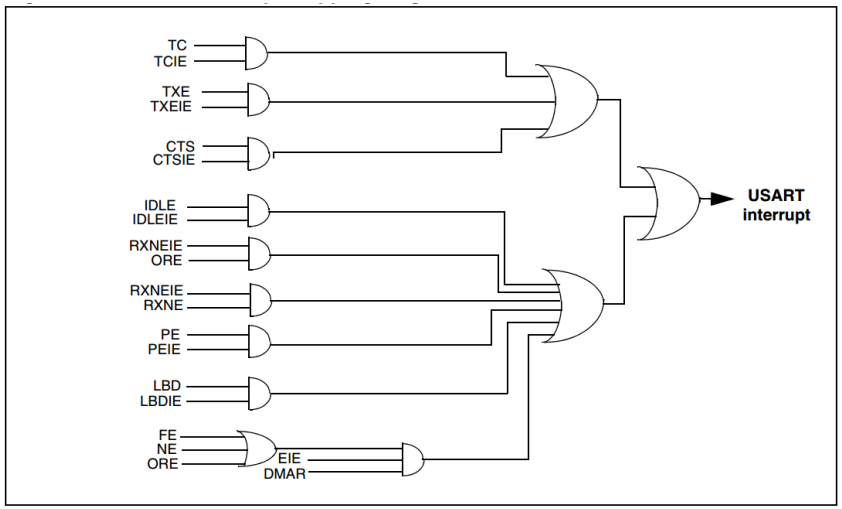
The USART interrupt events are connected to the same interrupt routine

During transmission: Transmission Complete, Clear to Send or Transmit Data Register empty interrupt.

While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission complete	TC	TCIE
Received data ready to be read	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
Break flag	LBD	LBDIE
Noise flag, Overrun error and Framing error in multibuffer communication	NE or ORE or FE	EIE(1)



USART Low Level procedure

TX

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAT) in USART_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the USART_BRR register.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_DR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_DR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Single byte communication

The TXE bit is always cleared by a write to the data register.
The TXE bit is set by hardware and it indicates:
● The data has been moved from TDR to the shift register and the data transmission has started.
● The TDR register is empty.
● The next data can be written in the USART_DR register without overwriting the previous data.
This flag generates an interrupt if the TXEIE bit is set.

RX

Procedure:

1. Enable the USART by writing the UE bit in USART_CR1 register to 1.
2. Program the M bit in USART_CR1 to define the word length.
3. Program the number of stop bits in USART_CR2.
4. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
5. Select the desired baud rate using the baud rate register USART_BRR
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit

Single byte communication

● The RXNE bit is set. It indicates that the content of the shift register is transferred to the RDR. In other words, data has been received and can be read (as well as its associated error flags).
● An interrupt is generated if the RXNEIE bit is set.
● The error flags can be set if a frame error, noise or an overrun error has been detected during reception.
● In multibuffer, RXNE is set after every byte received and is cleared by the DMA read to he Data Register.
● In single buffer mode, clearing the RXNE bit is performed by a software read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. The RXNE bit must be

USART (what)

- I want to use an USART. **What do I need to know?**
  - **Which bus USARTx are connected to?**
    ➞ Look at the architecture diagram (UM1669 Figure 5)
  - **Which Pin and Port are we going to use?**
    ➞ Look at the development board documentation (UM1669 Table 6)

PA2	TIM2_CH3, TIM5_CH3, TIM9_CH1, USART2_TX, ADC1_2	25														14	
PA3	TIM2_CH4, TIM5_CH4, TIM9_CH2, USART2_RX, ADC1_3	26														13	

  - **What do I need to do with this USART?** (input, output, ...)
    ➞ Configure for 9600 8 bit data, No Parity, 1 stop bit

USART (where)

- I want to use an USART. **Where can I gather these information?**

  ➞ The **datasheet** contains all the information we need

  ➞ Look at the **UM0368 User Manual**

USART code structure:

- The operation that are needed to use the USART are:
  - Initialization of the USART PIN as Alternate Function (not standard GPIO)
  - Initialization of USART
  - Set data in the Data Register / read data from Data Register
- Part 2:
  - Set an Interrupt for reception
  - Use the DMA for transmission

USART code - GPIO

```
void GPIO_init(void){
    GPIO_InitTypeDef GPIO_InitStructure;
    /* Enable GPIO clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    /* Connect PA2 to USARTx_Tx*/
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2);

    /* Connect PA3 to USARTx_Rx*/
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2);

    /* Configure USART Tx as alternate function */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART Rx as alternate function */
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}
```

Pins are used in Alternate Function, this means that are not used as GPIO, but as USART pin

USART - Initialization

```
void USART_Config(void)
{
    USART_InitTypeDef USART_InitStructure;

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* USARTx configured as follows:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
    */
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    /* USART initialization */
    USART_Init(USART2, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART2, ENABLE);

}
```

Structure definition and Clock enabled

USART - Initialization

```
void USART_Config(void)
{
    USART_InitTypeDef USART_InitStructure;

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* USARTx configured as follows:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
    */
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    /* USART initialization */
    USART_Init(USART2, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART2, ENABLE);

}
```

Set the baudrate (up to 2Mbaud, make sure that two usart devices connected among them have same baud)

USART - Initialization

```
void USART_Config(void)
{
    USART_InitTypeDef USART_InitStructure;

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* USARTx configured as follows:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
    */
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    /* USART initialization */
    USART_Init(USART2, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART2, ENABLE);

}
```

Configure USART for: no parity bit, 1 stop bit and 8 bit word length

USART - Initialization

```
void USART_Config(void)
{
    USART_InitTypeDef USART_InitStructure;

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* USARTx configured as follows:
        - BaudRate = 9600 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
    */
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    /* USART initialization */
    USART_Init(USART2, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART2, ENABLE);

}
```

Do not use hardware flow control (two additional lines for handshaking) and enable transmission and reception

USART - Initialization

```
void USART_Config(void)
{
    USART_InitTypeDef USART_InitStructure;

    /* Enable UART clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

    /* USARTx configured as follows:
        - BaudRate = 115200 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
    */
    USART_InitStructure.USART_BaudRate = 9600;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

    /* USART initialization */
    USART_Init(USART2, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART2, ENABLE);

}
```

Initialize peripheral and enable it

USART – printf enable

```
int __io_putchar(int ch)
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART */
  USART_SendData(USART2, (uint8_t) ch);

  /* Loop until the end of transmission */
  while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET)
  {}

  return ch;
}
```

If w want to use printf, the putchar function need to be redefined, every character is now sent to the USART

USART - main

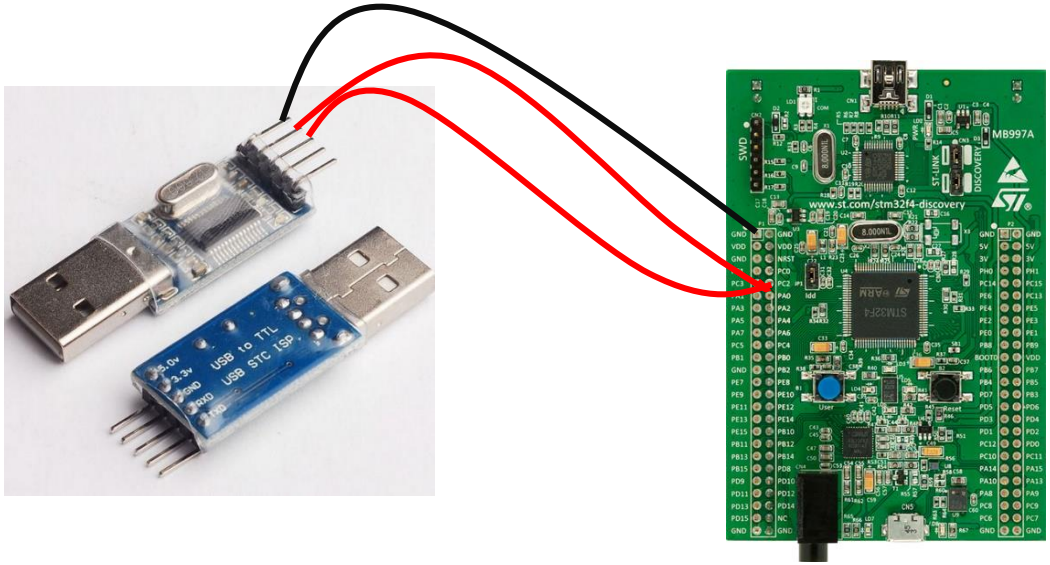
```
int main(void)
{

char ch;
    GPIO_init() ;
    /* USART configuration */
    USART_Config();

  /* Output a message on Hyperterminal using printf function */
  printf("\n\rUSART Printf Example: retarget the C library printf function to the USART\n\r");
    while (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET)
    {}
        ch = USART_ReceiveData(USART2);
  printf("\n\r USART2\n\r");

  while (1)
  {
  }
}
```

Wait until RX buffer is not empty (is full)

The Printf uses putchar for every character, after that the program wait for data on the serial port
And then print another string (USART2)
\r\n are for carriage return (a capo)

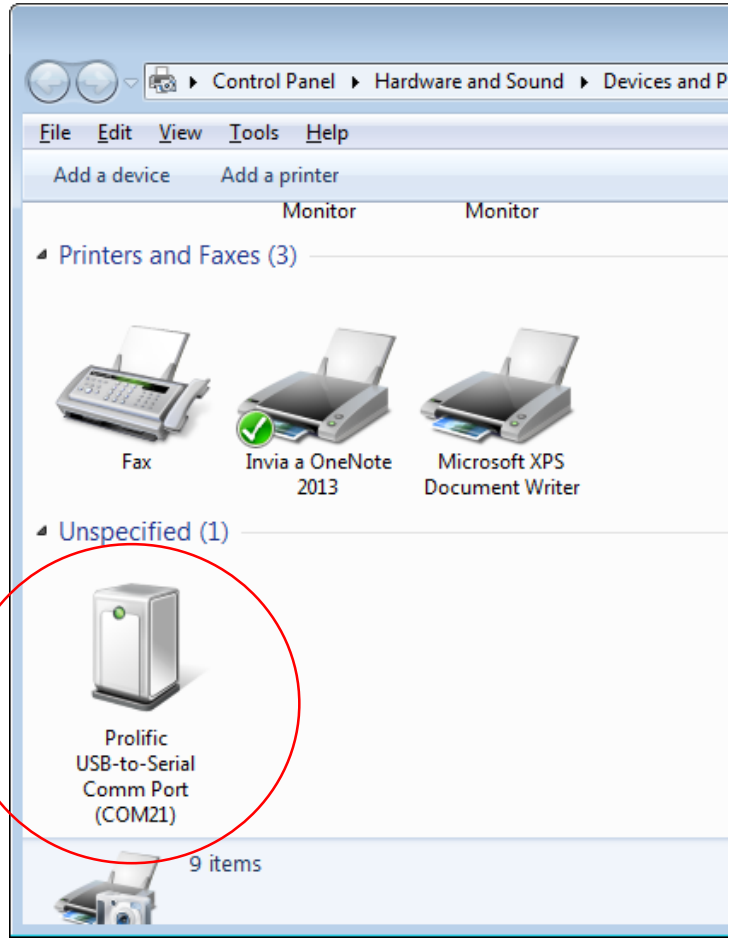
Board Connection

- Install USB drivers
- Connect Tx->PA3 and Rx->PA2 and GND->GND
- DO NOT CONNECT VCC
- Open a terminal (e.g. putty)



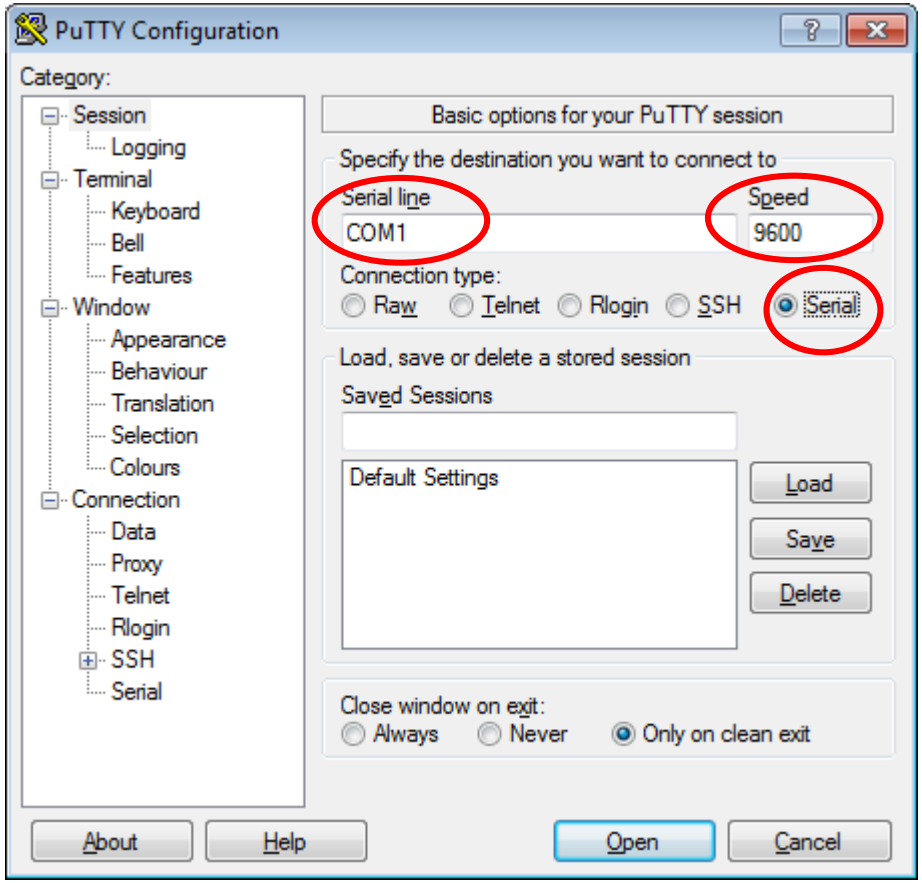
On the Computer

- **First INSTALL THE DRIVERS (PL2303_Prolific_Vista_332102.exe)**
- http://www-micrel.deis.unibo.it/LABARCH/2015/software/
- Connect the Serial adaptor to the PC
- Detect the serial port (devices and printers) in this
- case port is **COM21**



Open the terminal

- Open the terminal (we are using Putty, but you can use any other) (it is in you network drive F:)
- Choose “Serial”
- Set the correct COM port and Speed (Baudrate)
- Click “Open”
- You do not see what you write, only what you receive from serial port, to see what you write, connect TX and RX pin with Jumper



LABORATORIO DI ARCHITETTURE E PROGRAMMAZIONE DEI SISTEMI ELETTRONICI INDUSTRIALI

Laboratory Lesson 8:

USART – Interrupt and DMA

Prof. Luca Benini <luca.benini@unibo.it>

Filippo Casamassima <filippo.casamassima@unibo.it>
Domenico Balsamo <domenico.balsamo@unibo.it>

Why DMA and Interrupt

- Interrupt can stop main program execution when a new data is available
- DMA can transfer large quantity of data to USART leaving MCU free for other operations

Code - NVIC

```
void USART_DMA_NVIC_Configuration(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;

  /* Configure the Priority Group to 2 bits */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);

  /* Enable the USART2 RX DMA Interrupt */
  NVIC_InitStructure.NVIC_IRQChannel = DMA1_Stream6_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn; // we want to configure the USART1 interrupts
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; // this sets the priority group of the
USART1 interrupts
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // this sets the subpriority inside the group
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; // the USART2 interrupts are globally enabled
  NVIC_Init(&NVIC_InitStructure);
}
```

•

Enable Interrupt for DMA and USART Rx

Code - DMA

```
void DMA_Configuration(char* BufferAddr, u16 dataSize)
{
  DMA_InitTypeDef DMA_InitStructure;

  DMA_DeInit(DMA1_Stream6);

  DMA_InitStructure.DMA_Channel = DMA_Channel_4;
  DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral; // Transmit
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)BufferAddr;
  DMA_InitStructure.DMA_BufferSize = dataSize;
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&USART2->DR;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;

  DMA_Init(DMA1_Stream6, &DMA_InitStructure);

  /* Enable the USART Tx DMA request */
  USART_DMACmd(USART2, USART_DMAReq_Tx, ENABLE);
  /* Enable DMA Stream Transfer Complete interrupt */
  DMA_ITConfig(DMA1_Stream6, DMA_IT_TC, ENABLE);
  /* Enable the DMA RX Stream */
  DMA_Cmd(DMA1_Stream6, ENABLE);
}
```

DMA1 channel 4 Stream 6 is connected to Usart2 Tx

The memory buffer and data size are passed as parameters

USART data register is used as destination

DMA automatically set new character in USART2 data register when old character has been sent

Code Usart Interrupt

```
void USART2_IRQHandler(void)
{
    static int rx_index = 0;

    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        char rx = USART_ReceiveData(USART2);
        rx_buffer[rx_index++] = rx;
        if(rx == '\n' || rx == '\r' || rx_index ==BuffSize ){
            dataReady = 1;
            rx_index = 0;
        }
    }
}
```

Every time a new character is received, it is copied into a buffer

Code Usart Interrupt

```
void USART2_IRQHandler(void)
{
    static int rx_index = 0;

    if (USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
    {
        char rx = USART_ReceiveData(USART2);
        rx_buffer[rx_index++] = rx;
        if(rx == '\n' || rx == '\r' || rx_index ==BuffSize ){
            dataReady = 1;
            rx_index = 0;
        }
    }
}
```

When a carriage return, or the maximum buffer size has been reached a flag “dataReady” is set

Code - main

```
int main (void){

...

while (1)
    {

    if (dataReady == 1){
       value = atoi(rx_buffer); // Extract integer from RX buffer
       dataSize = sprintf(buffer,"You have entered the value: %d
\r\n",value); //Create a string
       DMA_Configuration(buffer, dataSize);// Send the string using DMA
       memset (rx_buffer,'\0',BuffSize); // Reset the RX Buffer
       dataReady = 0; //Reset the Data Ready flag

        }

}
```

If data is ready, extract integer from input string

Code - main

```
int main (void){

...

while (1)
    {

    if (dataReady == 1){
       value = atoi(rx_buffer); // Extract integer from RX buffer
       dataSize = sprintf(buffer,"You have entered the value: %d
\r\n",value); //Create a string
       DMA_Configuration(buffer, dataSize);// Send the string using DMA
       memset (rx_buffer,'\0',BuffSize); // Reset the RX Buffer
       dataReady = 0; //Reset the Data Ready flag

        }

}
```

We cannot use printf with DMA, so we create a string using sprint and save it to an array called “buffer”

Code - main

```
int main (void){

...

while (1)
    {

    if (dataReady == 1){
       value = atoi(rx_buffer); // Extract integer from RX buffer
       dataSize = sprintf(buffer,"You have entered the value: %d
\r\n",value); //Create a string
       DMA_Configuration(buffer, dataSize);// Send the string using DMA
       memset (rx_buffer,'\0',BuffSize); // Reset the RX Buffer
       dataReady = 0; //Reset the Data Ready flag

        }

}
```

We transmit the data using DMA

Code - main

```
int main (void){

...

while (1)
    {

    if (dataReady == 1){
       value = atoi(rx_buffer); // Extract integer from RX buffer
       dataSize = sprintf(buffer,"You have entered the value: %d
\r\n",value); //Create a string
       DMA_Configuration(buffer, dataSize);// Send the string using DMA
       memset (rx_buffer,'\0',BuffSize); // Reset the RX Buffer
       dataReady = 0; //Reset the Data Ready flag

        }

}
```

Input buffer is resetted together with dataReady flag

USART Esercizi

- 7.1 Per la USART in polling (senza DMA) nel while(1) del main inserisci un timeout (per non restare bloccati sulla ricezione del carattere) e stampa quanti millisecondi sono trascorsi dall'ultima volta che è stato ricevuto un carattere.
  - Suggerimento: puoi usare il systick per contare i millisecondi e inserire una condizione in:

```
while (USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET) {}
```

  - Per uscire dal while trascorso il timeout

- 8.1 Nell'esempio con il DMA, modifica il codice in maniera che dopo avr inserito due numeri, vengano inviate due stringhe, una con I due numeri e una con la loro somma ad es:

```
>Inserisci due numeri:
```

qui l'untente inserisce I numeri da terminale

```
>Hai inserito I numeri 21 e 45
>La loro somma è 21 + 45 = 66
```

Fai attenzione, aspetta che il DMA finisca di inviare una stringa prima di inviarne un'altra

USART Domande

- 7.a Quale flag devo controllare e quale registro devo scrivere per inviare dei dati sulla seriale?
- 7.b Descrivi il funzionamento dei Flag: TXE, TC e RXNE (guarda RM0368 – 19.6)
- 8.a Quale DMA, quale Canale e quale Stream dovrei settare se volgio usare il DMA con la USART1?