STM32F3 Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

Cuauhtémoc Carbajal

01/11/2013

STM32F3 Communication Interfaces

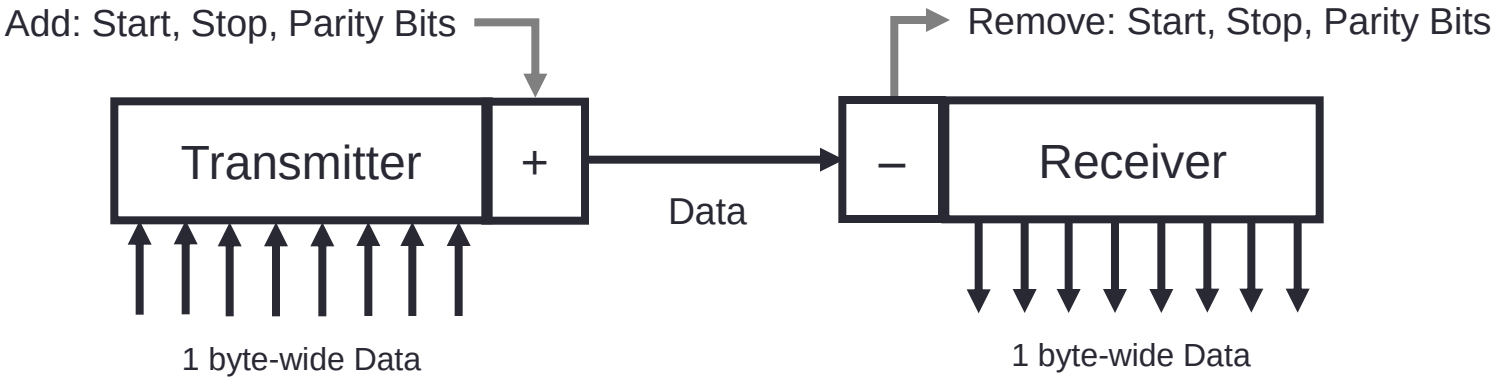
- CAN interface (2.0B Active)
- Two I2C Fast mode plus (1 Mbit/s) with 20 mA current sink, SMBus/PMBus, wakeup from STOP
- Up to five USART/UARTs (ISO 7816 interface, LIN, IrDA, modem control)
- Up to three SPIs, two with multiplexed I2S interface, 4 to 16 programmable bit frame
- USB 2.0 full speed interface
- Infrared Transmitter

Why Serial Communication?

- Parallel communication implies sending a whole byte (or more) of data over multiple parallel wires
  - Parallel data transfer requires many I/O pins. This requirement prevents the microcontroller from interfacing with as many devices as desired in the application.
  - Data synchronization for parallel transfer is difficult to achieve over a long distance.
- Serial communication implies sending data bit by bit over a single wire
  - Many I/O devices do not have high data rate to justify the use of parallel data transfer.
  - Consider cost.
- There are 2 types of serial communication:
  - Asynchronous
  - Synchronous

Asynchronous Serial Communication

- With **asynchronous** communication, the transmitter and receiver do not share a common clock



The Transmitter

- Shifts the parallel data onto the serial line using its own clock
- Also adds the start, stop and parity check bits

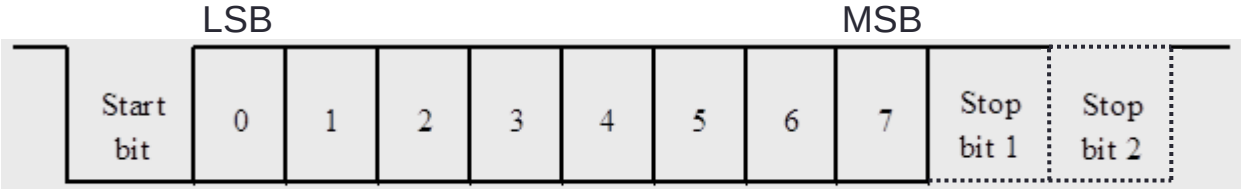
The Receiver

- Extracts the data using its own clock
- Converts the serial data back to the parallel form after stripping off the start, stop and parity bits

Data Format for Asynchronous Data Communication

- Data is transmitted character by character bit-serially.
- A character consists of
  - one start bit (0)
  - 7 to 8 data bits
  - an optional parity bit
  - one, or one and a half, or two stop bits (1)
  - least significant bit is transmitted first
  - most significant bit is transmitted last

A logic high is called a *mark*, and a logic low is called a *space*.

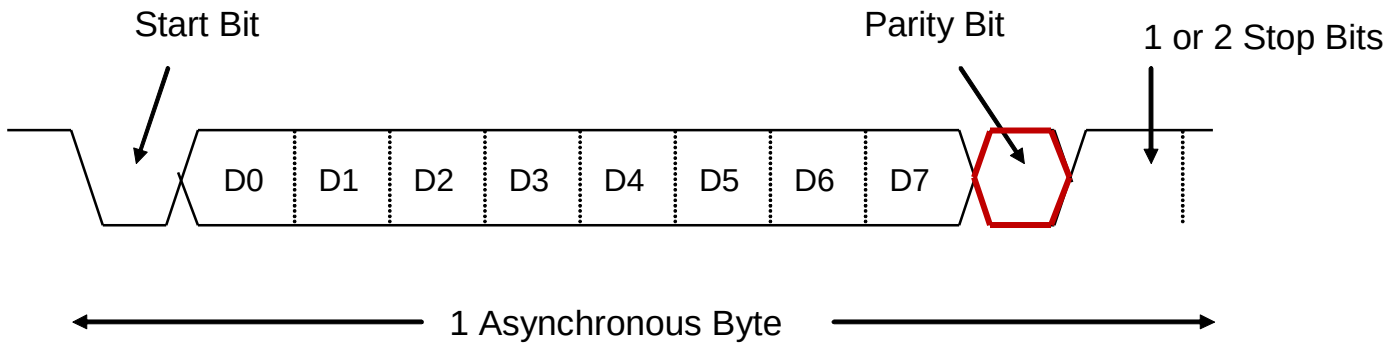


The format of a character

Asynchronous Serial Communication

- Start bit—indicates the beginning of the data word
- Stop bit—indicates the end of the data word
- Parity bit—added for error detection (optional)
- Data bits—the actual data to be transmitted
- Baud rate—the bit rate of the serial port
- Throughput—actual data transmitted per sec (total bits transmitted—overhead)
  - Example: 115200 baud = 115200 bits/sec
  - If using 8-bit data, 1 start, 1 stop, and no parity bits, the effective throughput is: 115200 * 8 / 10 = 92160 bits/sec

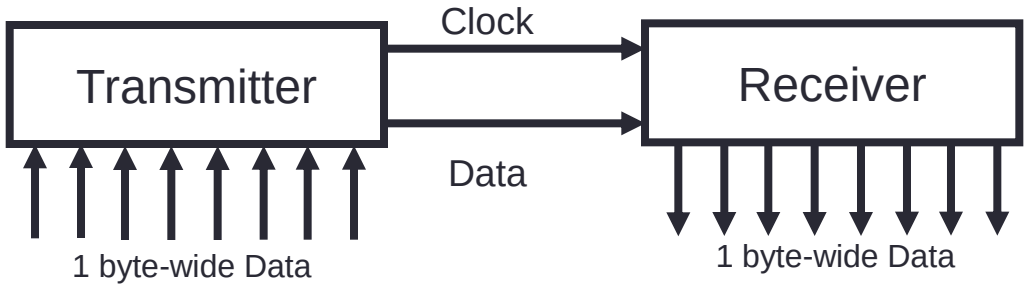
Asynchronous Serial Communication



- Parity options include even, odd, or no parity.
- Asynchronous transmission is easy to implement but less efficient as it requires an extra 2-3 control bits for every 8 data bits
- This method is usually used for low volume transmission

Synchronous Serial Communication

- In the **synchronous** mode, the transmitter and receiver share a common clock
- The transmitter typically provides the clock as a separate signal in addition to the serial data



The Transmitter

- Shifts the data onto the serial line using its own clock
- Provides the clock as a separate signal
- No start, stop, or parity bits added to data

The Receiver

- Extracts the data using the clock provided by the transmitter
- Converts the serial data back to the parallel form

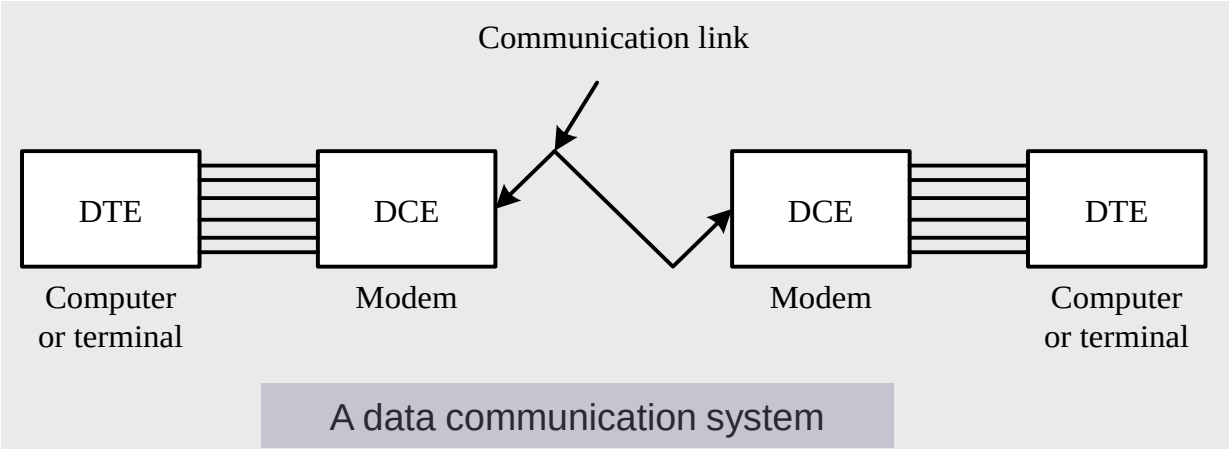
UART

What is UART?

- An interface designed to transfer data only in asynchronous mode that utilizes the EIA-232 standard.
- UARTs are cheap, easy to use, and until recently, very common.

Asynchronous Serial Data Communication

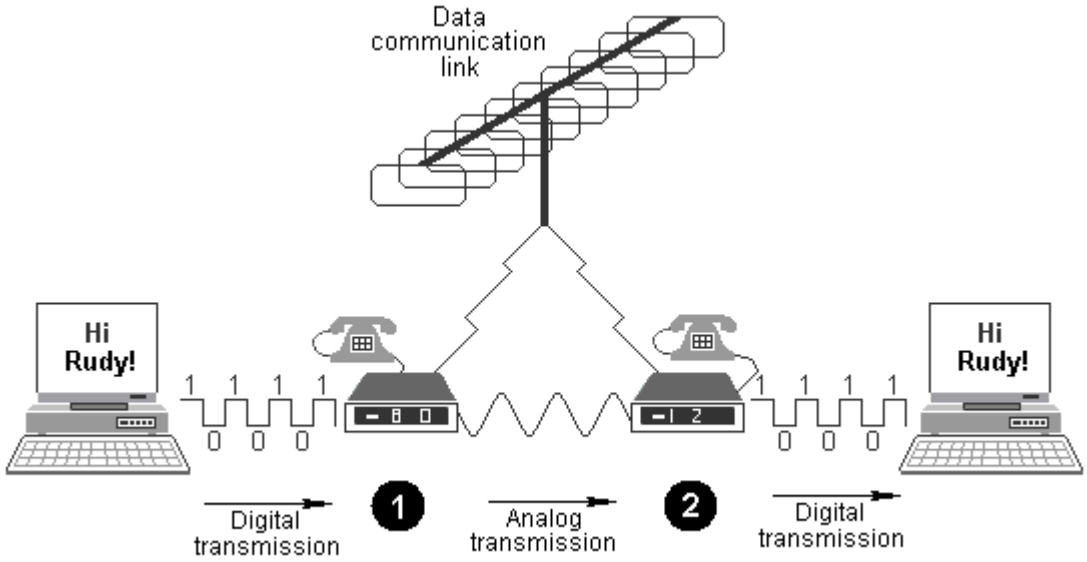
- It is often used for data communication between a DTE and a DCE with or without a modem.
- DTE stands for data terminal equipment and can be either a computer or a terminal.
- DCE stands for data communication equipment. A modem is a DCE.



A data communication system

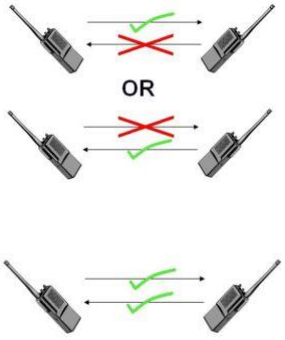
Modem

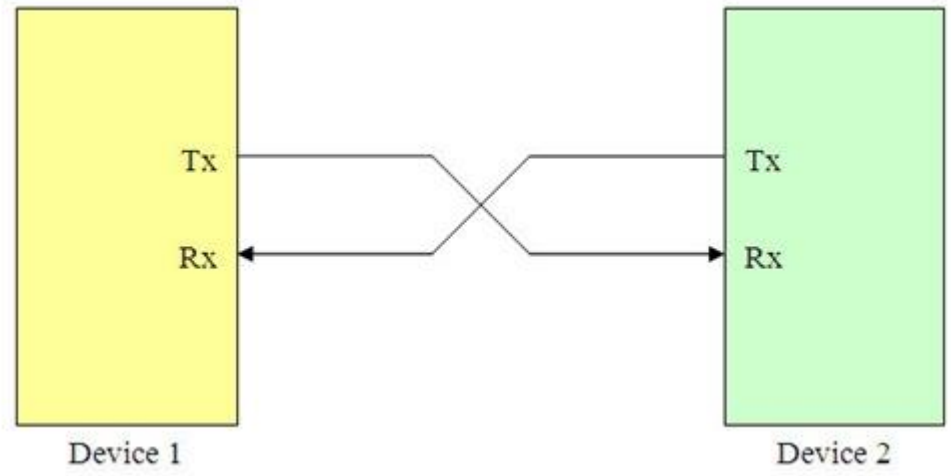
- A modem (modulator/demodulator) provides a way of encoding digital data as a set of audio signals that can be sent over a telephone line. Most modems communicate using RS232 and a set of hardware handshaking signals used to regulate data flow.



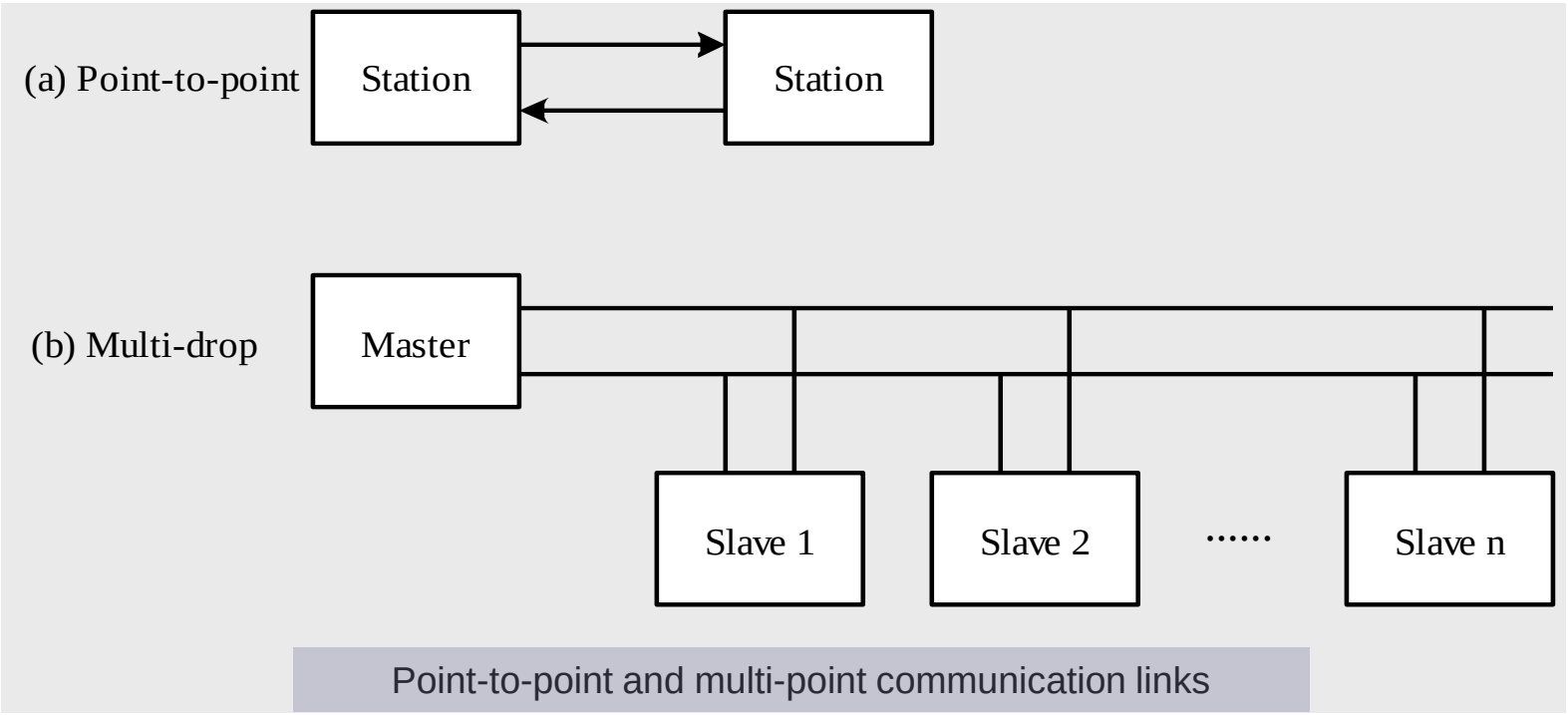
Kinds of data communication links

- Simplex link
- Half-duplex link
- Full-duplex link





Types of Communication Link Configuration



Point-to-point and multi-point communication links

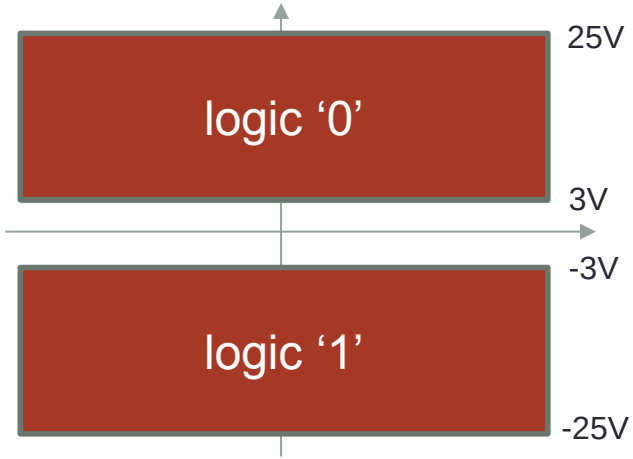
A multidrop system is a master and slave system. One master connects with a few slaves in the system. Each time, the master communicates with one of the slaves. When the master wants to transfer a block of data to a slave, it first sends out an address byte to identify the target slave. The 9th-bit of the data byte sent from the master is set to 1 to indicate the address byte while cleared to 0 to indicate the data byte. All the slave systems will compare the address byte with their own address. Only the target slave will respond to the master. The master then starts transmitting data bytes to the target slave. The non-addressed slave systems will ignore the incoming data until a new address byte is received.

The RS232 Standard

- Was the most widely used physical level interface for data communication
- Specifies 25 interchange circuits for DTE/DCE use
- Was established in 1960 by Electronics Industry Association (EIA)
- Was revised into RS232C in 1969
- Was revised into RS232D in 1987
- Was revised to RS232E in 1992 and renamed as EIA-232-E
- Four aspects: electrical, functional, procedural, and mechanical

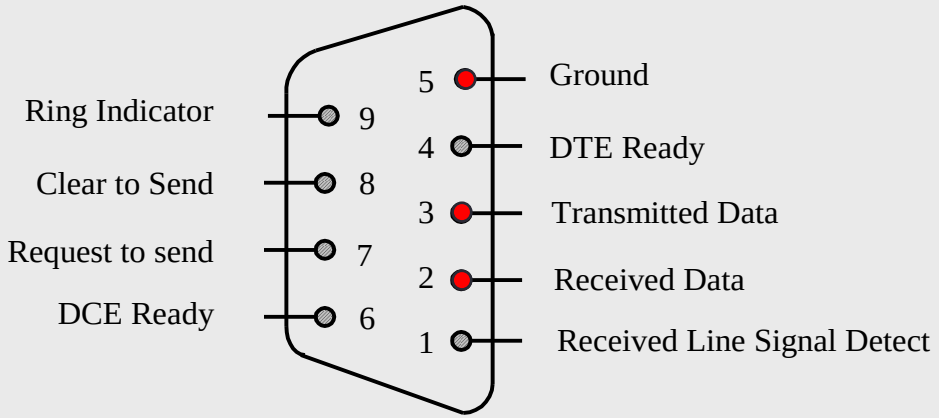
The EIA-232E Electrical Specifications (1 of 2)

- The interface is rated at a signal rate of < 20 kbps.
- The signal can transfer correctly within 15 meters.
- The maximum driver output voltage (with circuit open) is -25 V to +25 V.
- The minimum driver output voltage (loaded output) is -25 V to -5 V and +5 V to +25 V.
- The minimum driver output resistance when power is off is 300 Ω.
- The receiver input voltage range is -25 V to +25 V.
- The receiver output is high when input is open circuit.
- A voltage more negative than -3 V at the receiver input is interpreted as a logic 1.
- A voltage more positive than +3 V at the receiver input is interpreted as a logic 0.

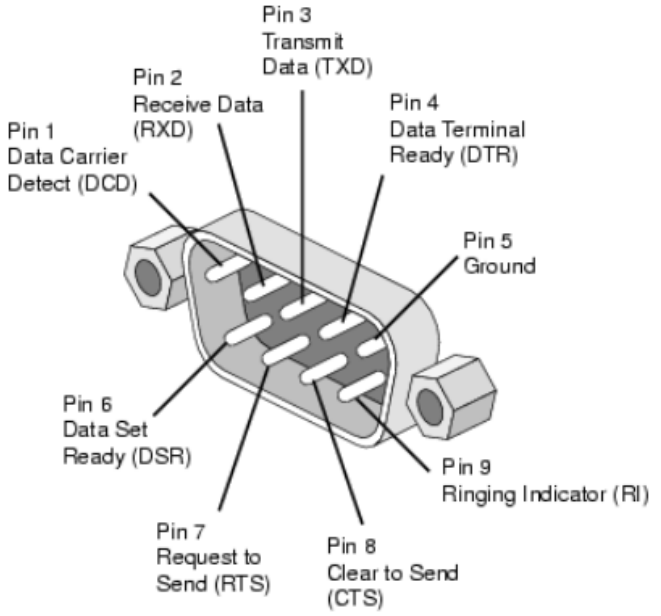


EIA-232-E Mechanical Specification (2 of 2)

- Only a small subset of the 25 pins are actually used in most data communications.
- Nine-pin is introduced to reduce the size and cost of the connector.



EIA-232E DB9 connector and signal assignment

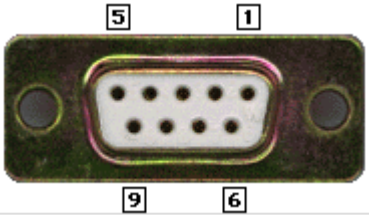


DB9 Female Connector DCE

DB9 Male Connector DTE

DB9 Male Connector DTE

DB9 Female Connector DCE



9 pin	25 pin	DTE	DTE Signal Name	DTE Direction	DCE	DCE Signal Name	DCE Direction	Description
1	8	DCD	Data Carrier Detect	IN	DCD	Data Carrier Detect	OUT	Modem connected to another
2	3	**RxD**	Receive Data	IN	**TxD**	Transmit Data	OUT	Receives bytes into the PC
3	2	**TxD**	Transmit Data	OUT	**RxD**	Receive Data	IN	Transmits bytes out of the PC
4	20	DTR	Data Terminal Ready	OUT	DTR	Data Terminal Ready	IN	I'm ready to communicate
5	7	**SG**	Signal Ground		**SG**	Signal Ground		
6	6	DSR	Data Set Ready	IN	DSR	Data Set Ready	OUT	I'm ready to communicate
7	4	**RTS**	Request To Send	OUT	**RTS**	Request To Send	IN	RTS/CTS flow control
8	5	**CTS**	Clear To Send	IN	**CTS**	Clear To Send	OUT	RTS/CTS flow control
9	22	RI	Ring Indicator	IN	RI	Ring Indicator	OUT	Telephone line ringing

How to Detect the Arrival of Start Bit

- Use a clock signal with frequency at least 16 times that of the data rate to sample the RxD signal.
- When the RxD pin is idle (high) for at least three sampling times and a falling edge follows, the SCI circuit checks the third, fifth, and seventh samples after the first sample. If the majority of them are low, then the start bit is considered detected.

How to Determine the Logic Value of a Data Bit

- Use a clock signal with frequency at least 16 times that of the data rate to sample the incoming data.
- Take the majority function of the eighth, ninth, and tenth samples. If the majority of them are 1s, then the logic value is determined to be 1.

Data Transmission Errors

- Framing error
  - The stop bit is not recognized on reception at the expected time, following either a desynchronization or excessive noise.
- Receiver overrun
  - One or more characters received, but not read by the CPU
- Parity error
  - Odd number of bits change value

ASCII Table

High Nibble

Low Nibble

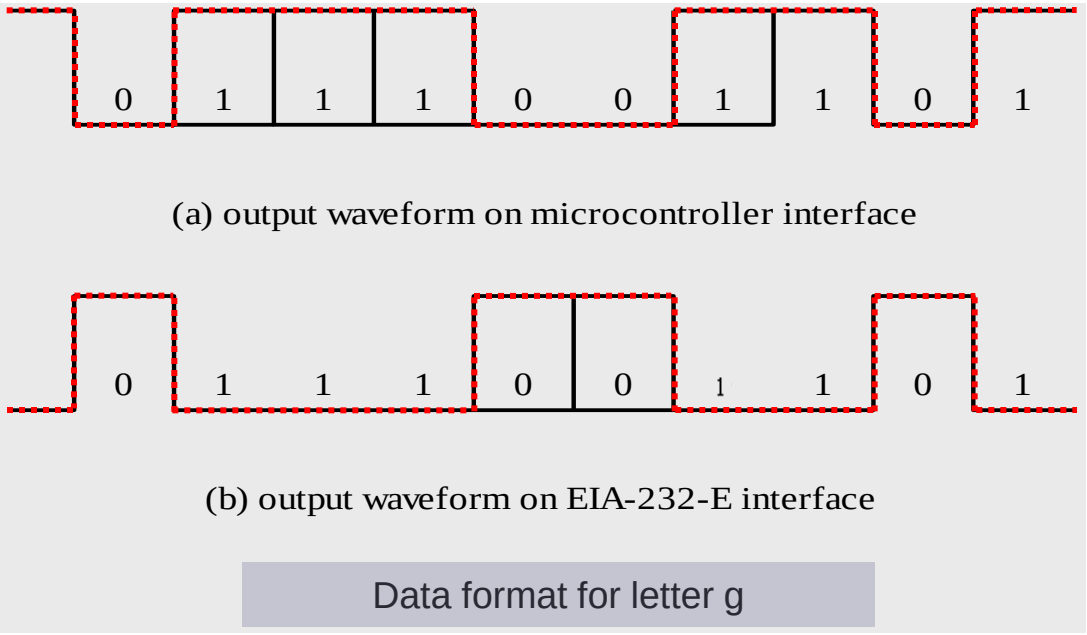
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Example:

- Sketch the output of the letter g when it is transmitted using the format of one start bit, 8 data bit, and 1 stop bit.

Solution:

- The ASCII code of letter g is $67 or %01100111. This code will be followed by a stop bit. The output from the DTE should be:



Data format for letter g

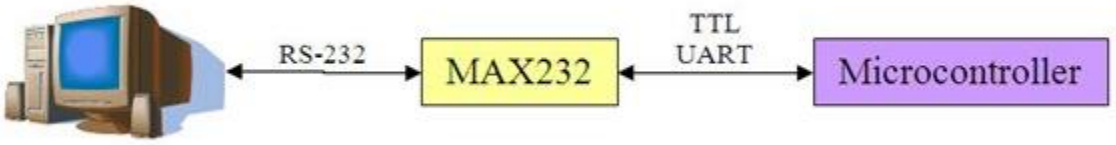
Null Modem Connection

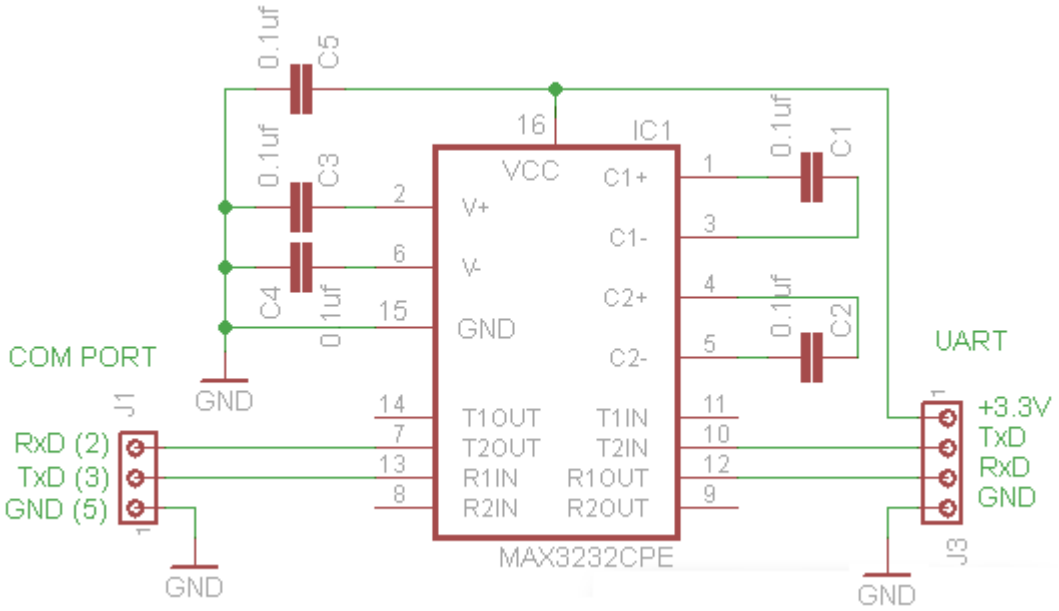
- When two devices that are both DTE or both DCE must be connected together without a modem or a similar media translator between them, a NULL modem must be used. The NULL modem electrically re-arranges the cabling so that the transmitter output is connected to the receiver input on the other device, and vice versa.

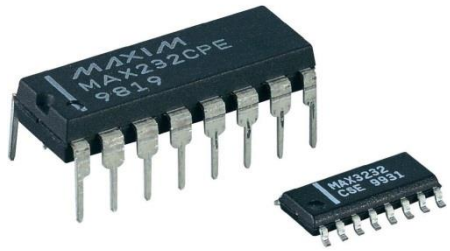
Signal Name	DTE 1		DTE 2		Signal Name
	DB25 pin	DB9 pin	DB9 pin	DB25 pin	
FG (frame ground)	1	-	-	1	FG
TD (transmit data)	2	3	2	3	RD
RD (receive data)	3	2	3	2	TD
RTS (request to send)	4	7	8	5	CTS
CTS (clear to send)	5	8	7	4	RTS
SG (signal ground)	7	5	5	7	SG
DSR (data set ready)	6	6	4	20	DTR
CD (carrier detect)	8	1	4	20	DTR
DTR (data terminal ready)	20	4	1	8	CD
DTR (data terminal ready)	20	4	6	6	DSR

Null Modem connection

MAX3232:
True +3.0V to +5.5V RS-232 Transceivers



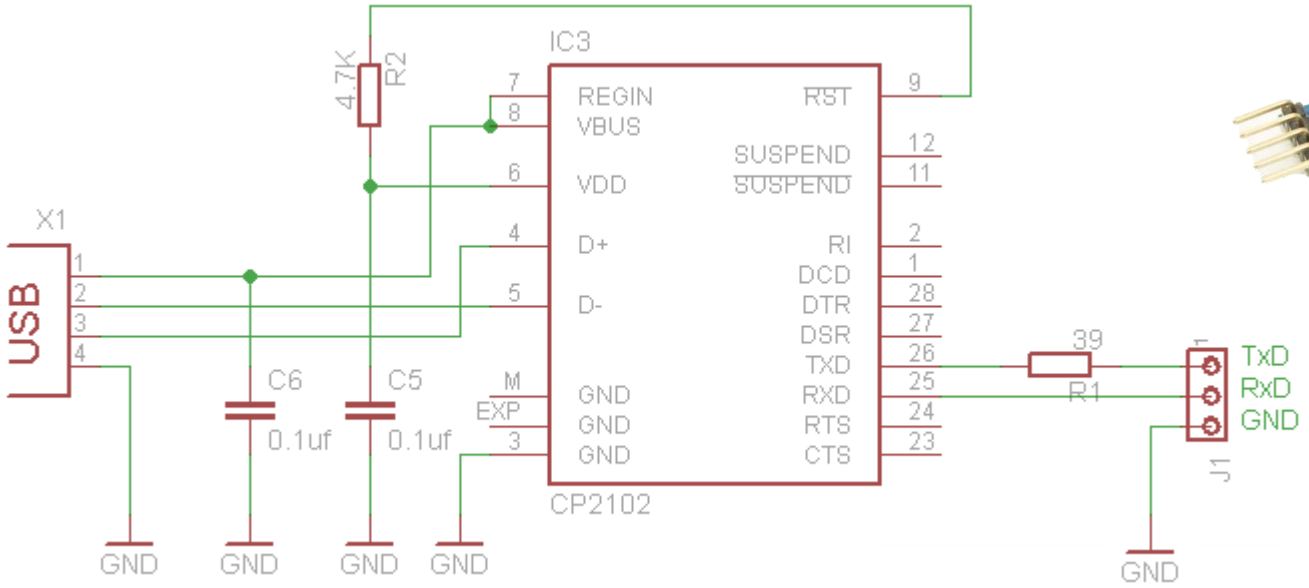


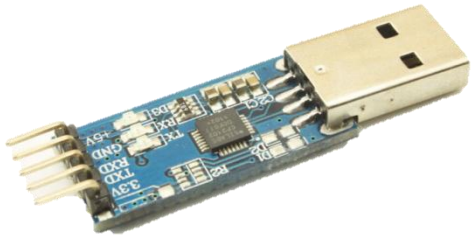


UART Future

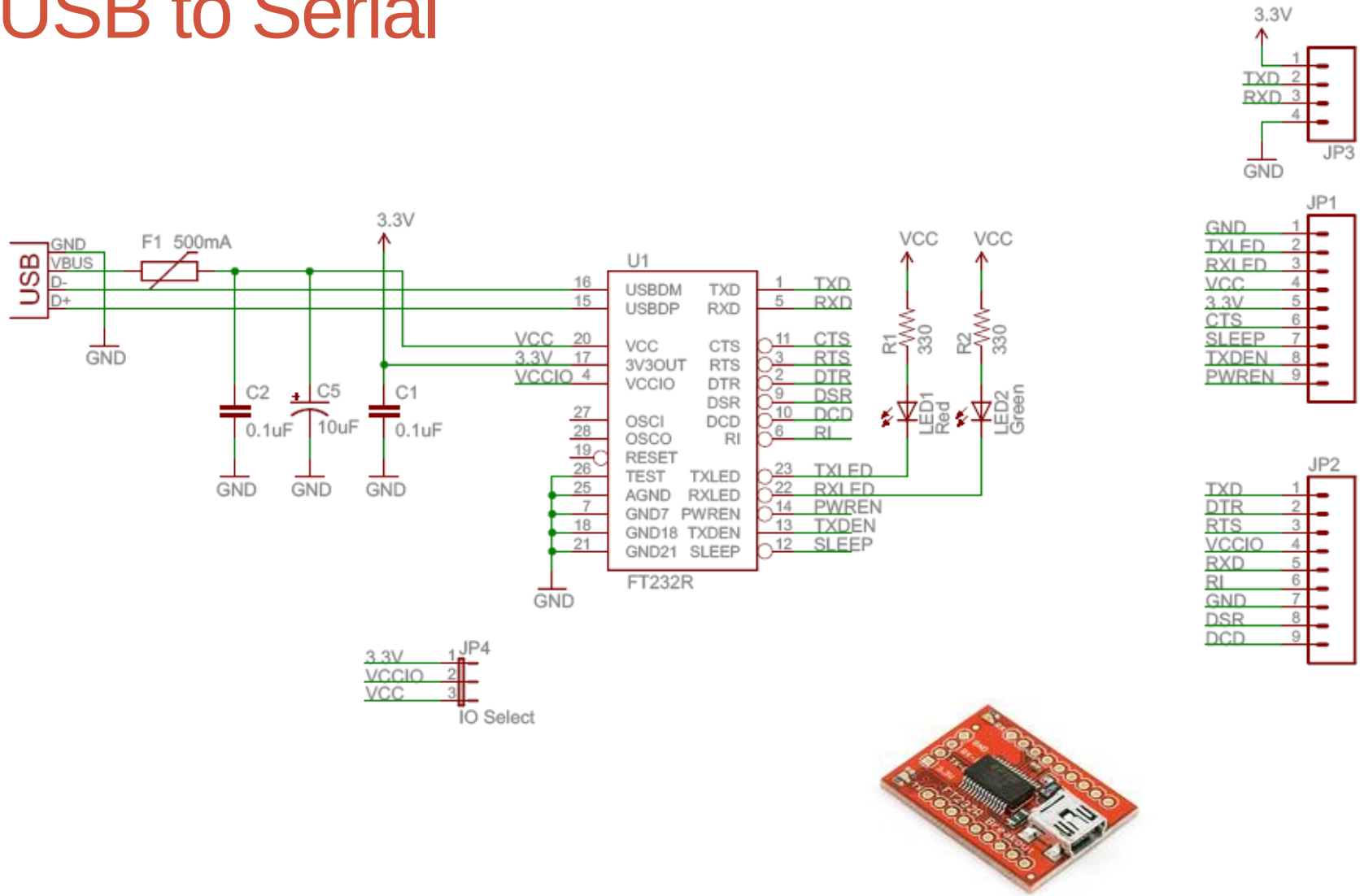
- USB is well on its way to replace the serial communication ports on PCs.
- So, are UARTs on their way to extinction?

CP2102:
Single-Chip USB to UART Bridge





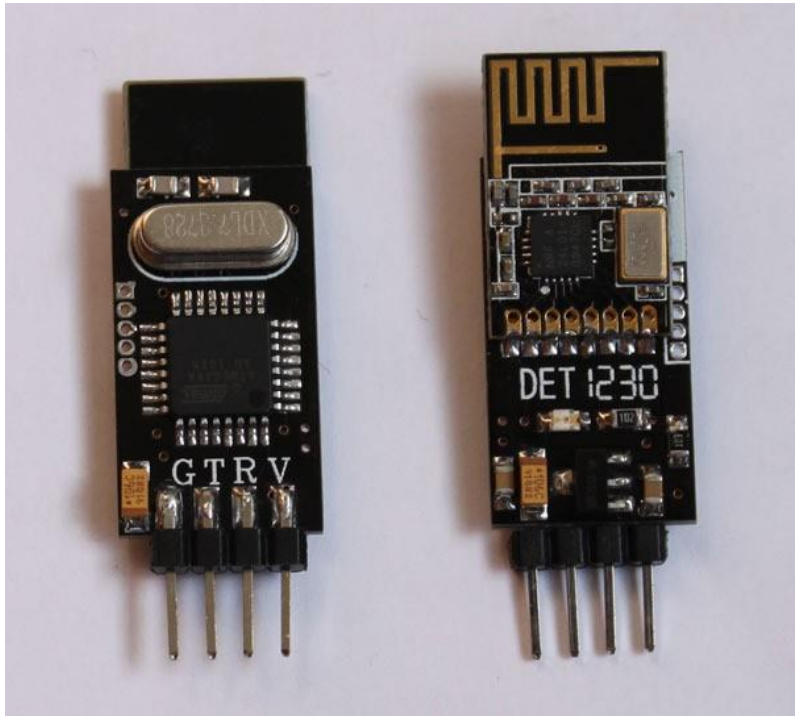
FT232RL: USB to Serial



https://www.sparkfun.com/products/718

UART interface NRF24L01 2.4G Wireless Module

- The nRF24L01 is a highly integrated, ultra low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM (Industrial, Scientific and Medical) band.



DRF1605H UART Serial Port to Zigbee Wireless Module Adapter w/ Antenna

- Easy to use: it can be used as an UART cable and ignore the ZigBee protocol.
- UART Interface (TX & RX) to Zigbee
- Built-in RS485 direction control
- High power module: the distance can reach 1600M

Bluetooth Master/Slave UART Board Wireless Transceiver Module

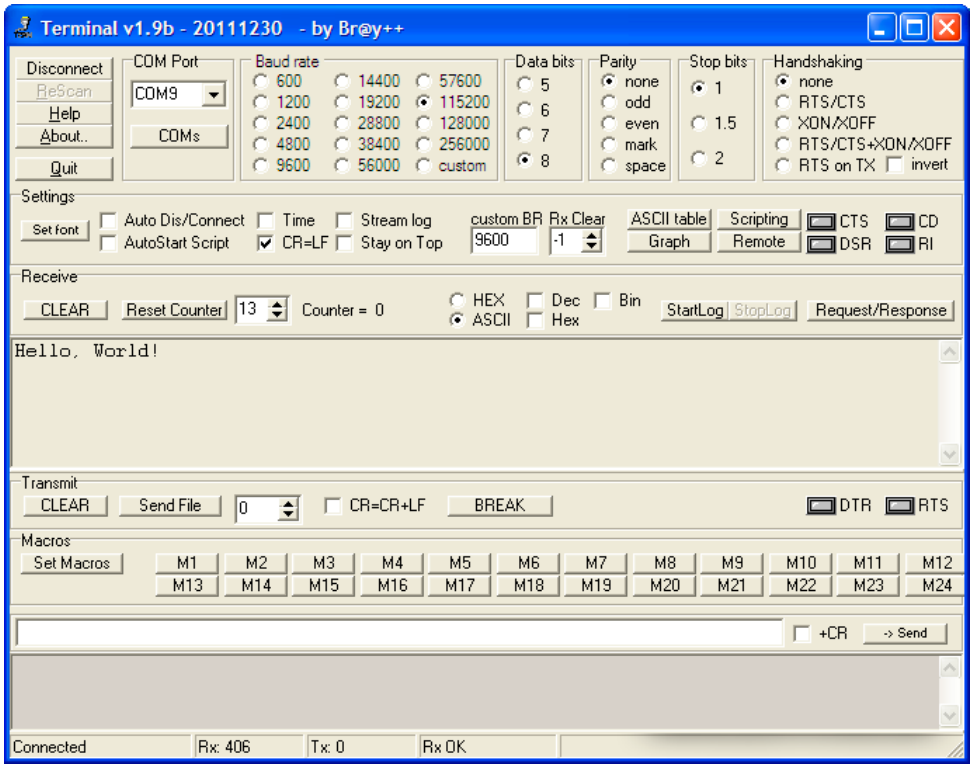
- For establishing communication, a Master Module (MM) and a Slave Module (SM) are required.
- MM communicate only with their SM.
- MM cannot communicate with another MM.
- SM is capable to communicate with a PC or mobile devices which features Bluetooth capability
- SM cannot communicate with another SM

Master module

Slave module

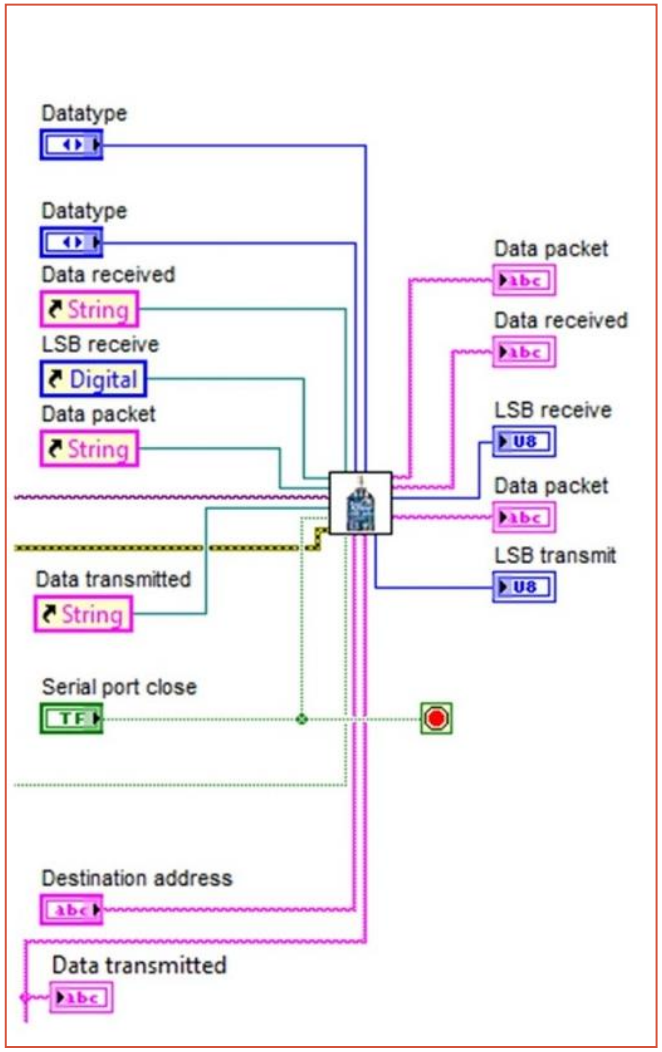
Com Port Development Tool

- Terminal is a simple serial port (COM) terminal emulation program. It can be used for communication with different devices such as modems, routers, embedded µC systems, GSM phones, GPS modules... It is very useful debugging tool for serial communication applications.

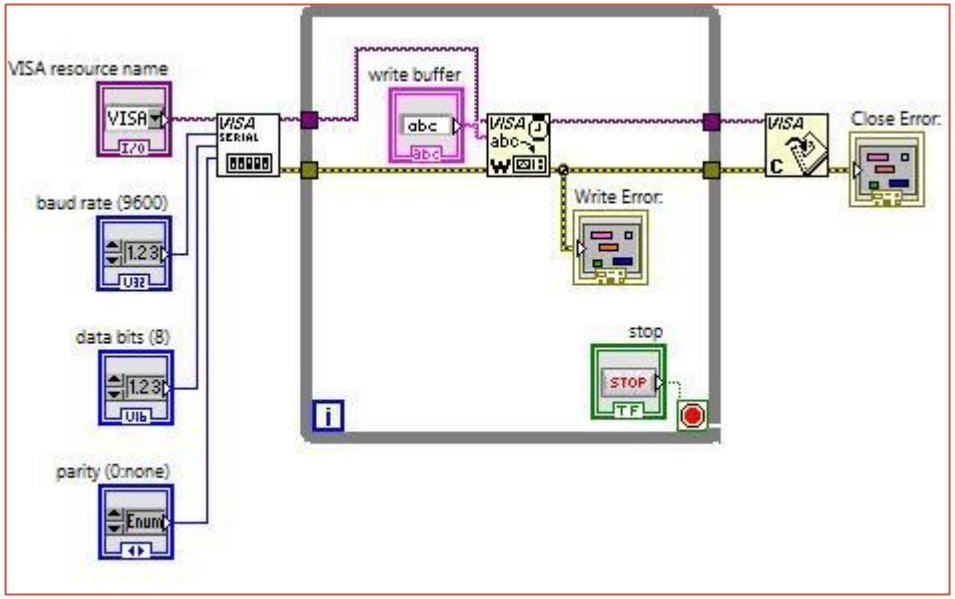


https://sites.google.com/site/terminalbpp/

Serial Communication Through LabVIEW



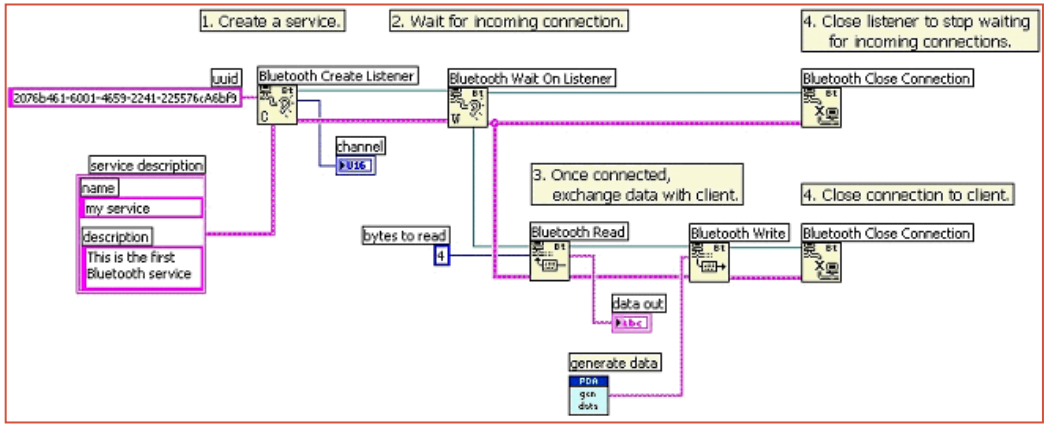
ZigBee



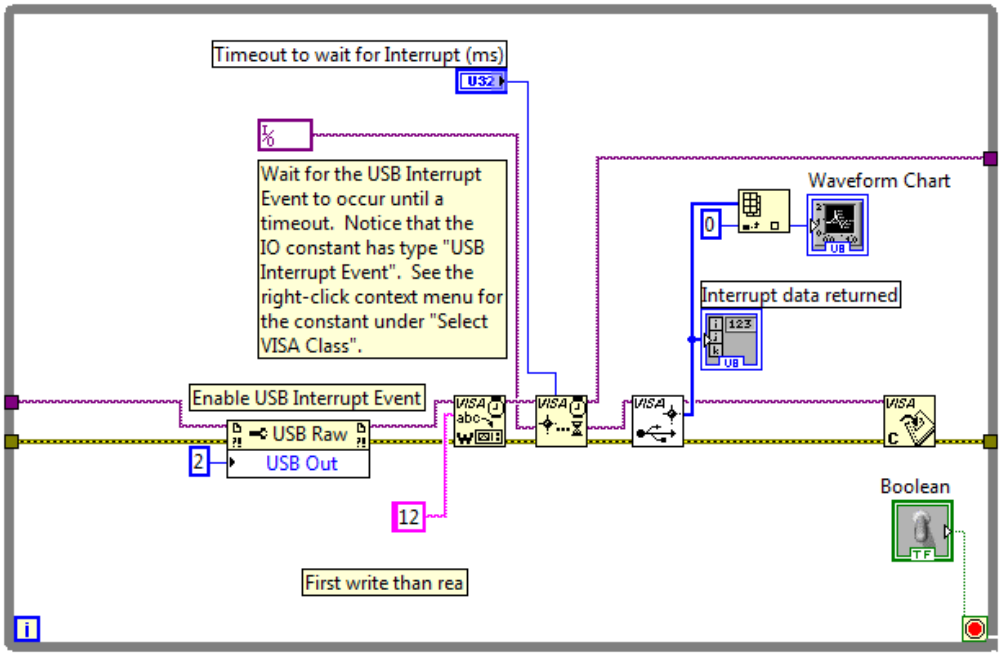
UART

http://www.ni.com/pdf/manuals/371253c.pdf

Serial Communication Through LabVIEW



Bluetooth



USB

USART

What is an USART?

- USART (Universal Synchronous/Asynchronous Receiver/Transmitter)
  - is a device that facilitates communication through a computer's serial port using the RS-232C protocol.
- Like a UART (Universal Asynchronous Receiver/Transmitter), a USART provides the computer with the interface necessary for communication with modems and other serial devices.
- However, unlike a UART, a USART offers the option of synchronous mode.
  - In program-to-program communication, the synchronous mode requires that each end of an exchange respond in turn without initiating a new communication. Asynchronous operation means that a process operates independently of other processes.

What is an USART?

- Practical differences between synchronous mode (which is possible only with a USART) and asynchronous mode (which is possible with either a UART or a USART) can be outlined as follows:
  - Synchronous mode requires both data and a clock. Asynchronous mode requires only data.
  - In synchronous mode, the data is transmitted at a fixed rate. In asynchronous mode, the data does not have to be transmitted at a fixed rate.
  - Synchronous data is normally transmitted in the form of blocks, while asynchronous data is normally transmitted one byte at a time.
  - Synchronous mode allows for a higher DTR (data transfer rate) than asynchronous mode does, if all other factors are held constant.

STM32F3 USART

USART Features

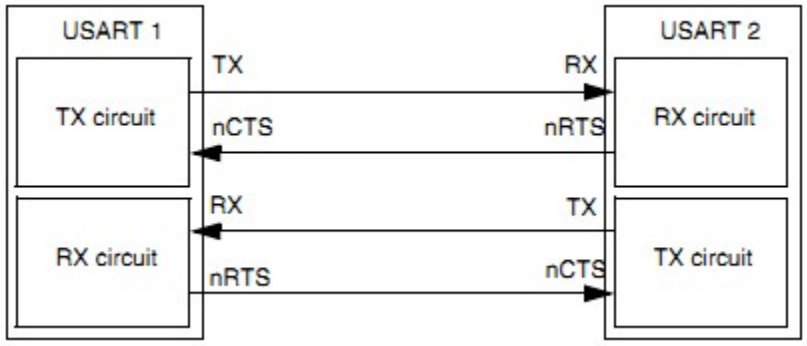
				UART	
USART modes/features(1)	USART1	USART2	USART3	USART4	USART5
Hardware flow control for modem	X	X	X		
Continuous communication using DMA	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X
Synchronous mode	X	X	X		
Smartcard mode	X	X	X		
Single-wire half-duplex communication	X	X	X	X	X
IrDA SIR ENDEC block	X	X	X	X	X
LIN mode	X	X	X	X	X
Dual clock domain and wakeup from Stop mode	X	X	X	X	X
Receiver timeout interrupt	X	X	X	X	X
Modbus communication	X	X	X	X	X
Auto baud rate detection	X	X	X		
Driver Enable	X	X	X		

X = supported

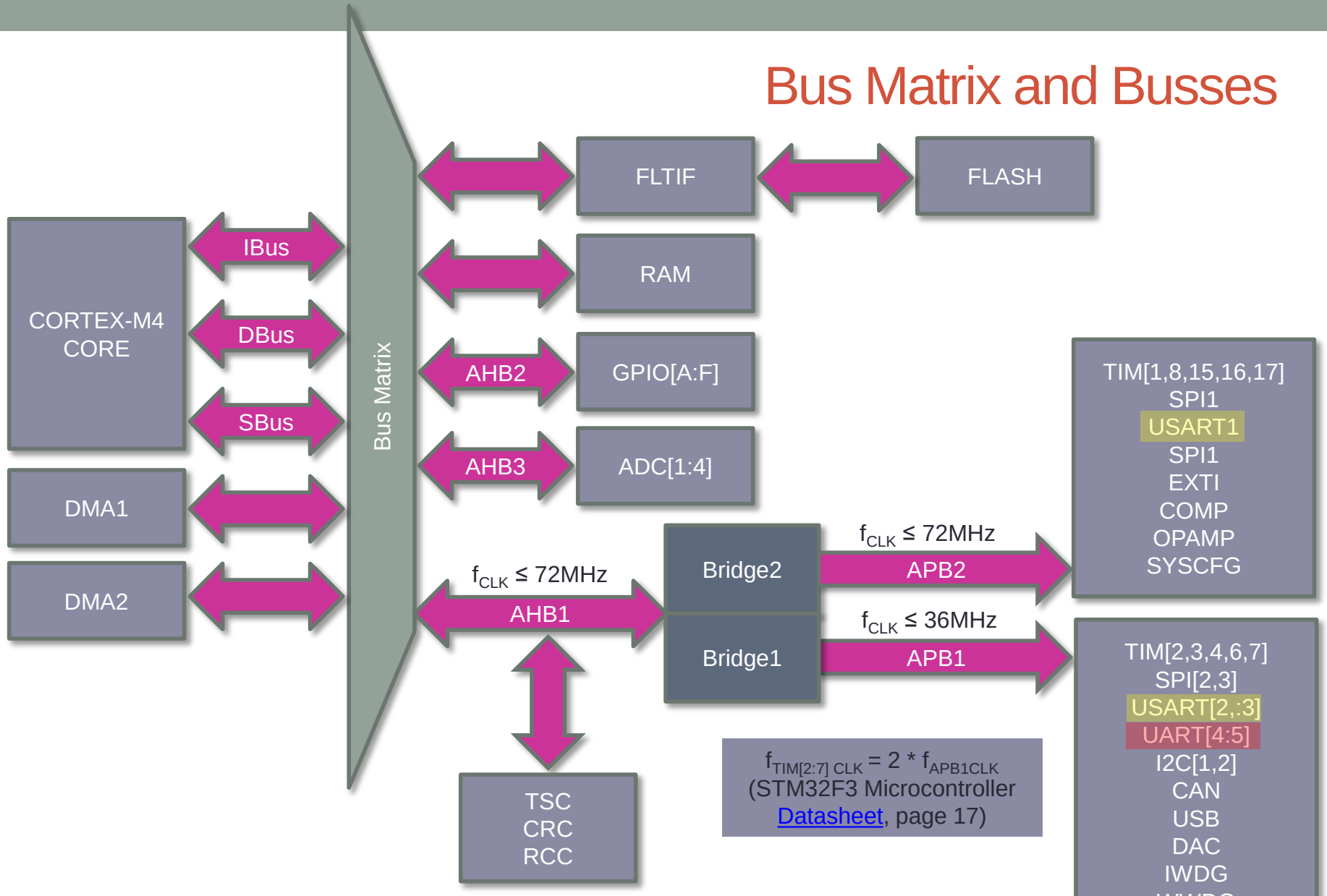
The USART interfaces are able to communicate at speeds of up to 9 Mbits/s.

RTS / CTS Hardware Flow Control

- RTS / CTS Flow Control is a flow control mechanism that is part of the RS232 standard.
- It makes use of two further pins on the RS232 connector, RTS (Request to Send) and CTS (Clear to Send). These two lines allow the receiver and the transmitter to inform each other of their state.
- A transmitter raises its RTS line, which causes an interrupt on the receiver
  - Hey can I send some data?
- If the receiver is in a position to receive the data it will assert its CTS line
  - Yes, you can start sending.
- The raising and lowering of these lines allows device drivers which implement hardware flow control code to maintain a reliable data connection between transmitter and receiver.



Bus Matrix and Busses

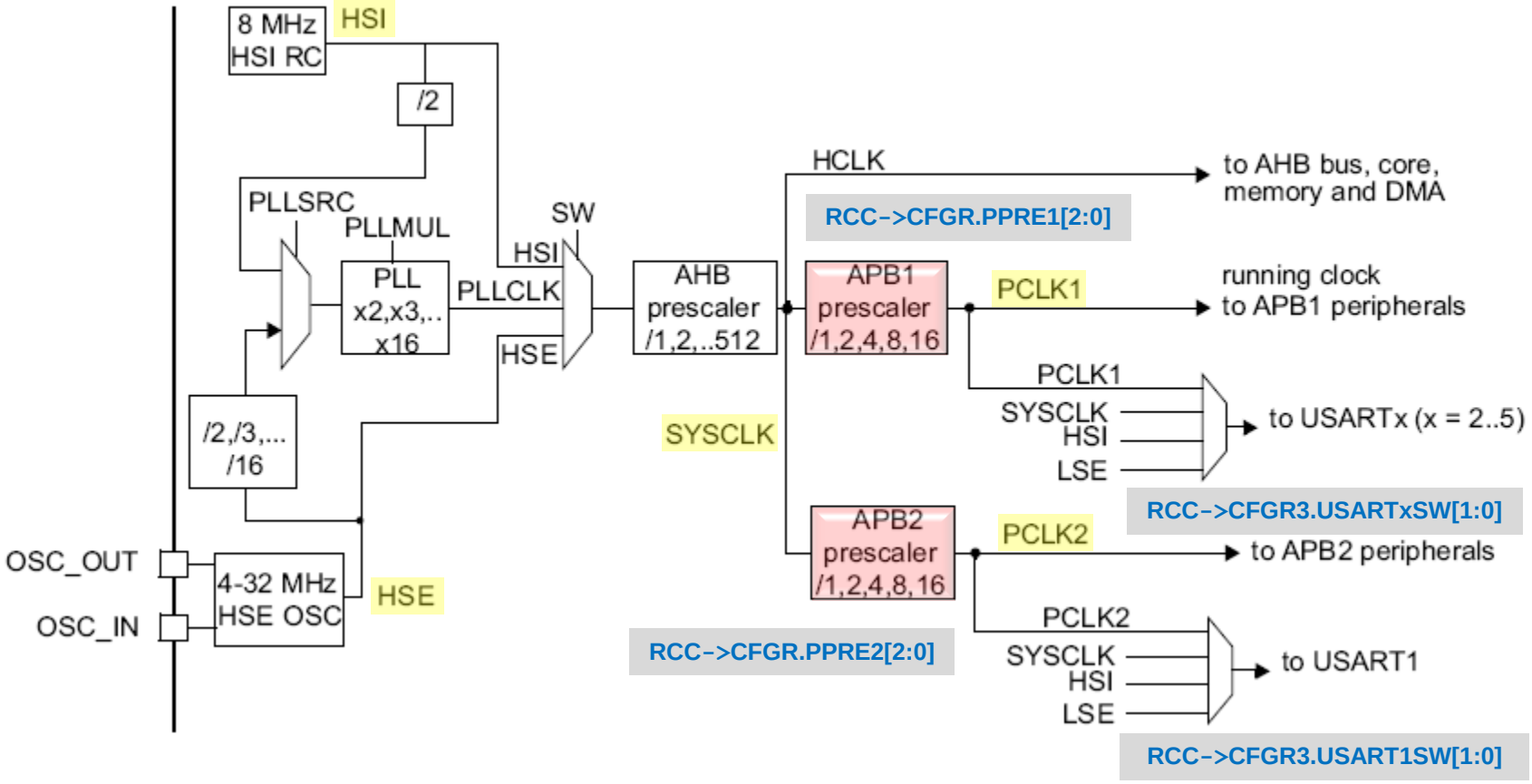


$f_{TIM[2:7]\ CLK} = 2 * f_{APB1CLK}$
(STM32F3 Microcontroller Datasheet, page 17)

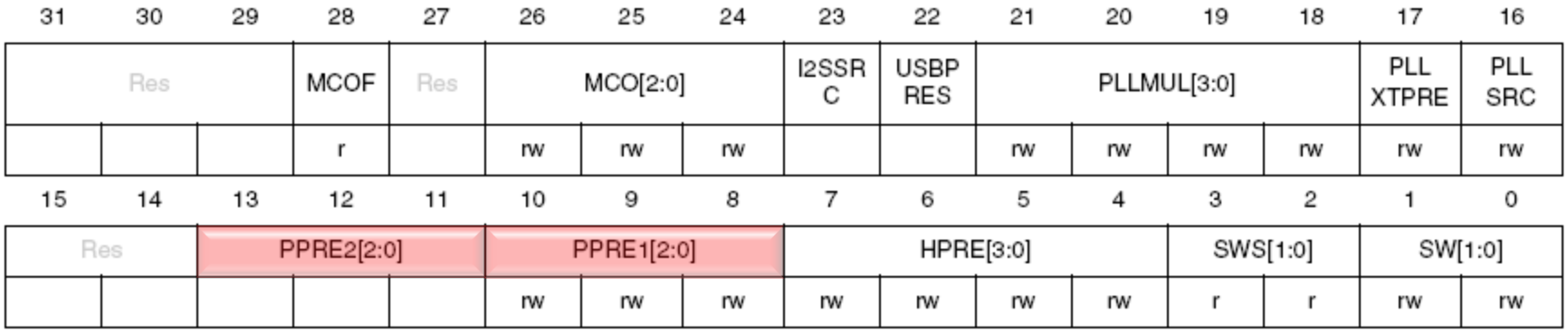
AHB[1;3]: Advanced High-performance Bus
APB: Advanced Peripheral Bus
RCC: Reset and Clock Control

STM32F3 Microcontroller Reference Manual, pages 41-44

Clock Tree (detail)



Clock configuration register (RCC_CFGR)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res			MCOF	Res	MCO[2:0]			I2SSRC	USBPRES	PLLMUL[3:0]				PLLXTPRE	PLLSRC
			r		rw	rw	rw			rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res		PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
					rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 13:11 **PPRE2:** APB high-speed prescaler (APB2)
Set and cleared by software to control the division factor of the APB clock (PCLK).
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Bits 10:8 **PPRE1:**APB Low-speed prescaler (APB1)
Set and cleared by software to control the division factor of the APB clock (PCLK).
0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Clock configuration register 3 (RCC_CFGR3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	UART5SW[1:0]		UART4SW[1:0]		USART3SW[1:0		USART2SW[1:0]	
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	Res	Res	Res	Res	Res	TIM8SW	TIM1SW	Res	Res	I2C2SW	I2C1SW	Res	Res	USART1SW[1:0]	
						rw	rw			rw	rw			rw	rw

USARTxSW[1:0]: USARTx clock source selection
This bit is set and cleared by software to select the USARTx clock source.

00: PCLK selected as USARTx clock source (default)
01: System clock (SYSCLK) selected as USARTx clock
10: LSE clock selected as USARTx clock
11: HSI clock selected as USARTx clock

APB1 Peripheral Clock Enable Register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res		DAC EN	PWR EN	Res	Res	Res	Res	Res	I2C2 EN	I2C1 EN	USART5 EN	USART 4EN	USART 3EN	USART 2EN	Res
	rw	rw	rw						rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3E N	SPI2E N	Res	Res	WWD GEN	Res					TIM7E N	TIM6EN	Res	TIM4EN	TIM3EN	TIM2E N
	rw			rw			rw			rw	rw		rw	rw	rw

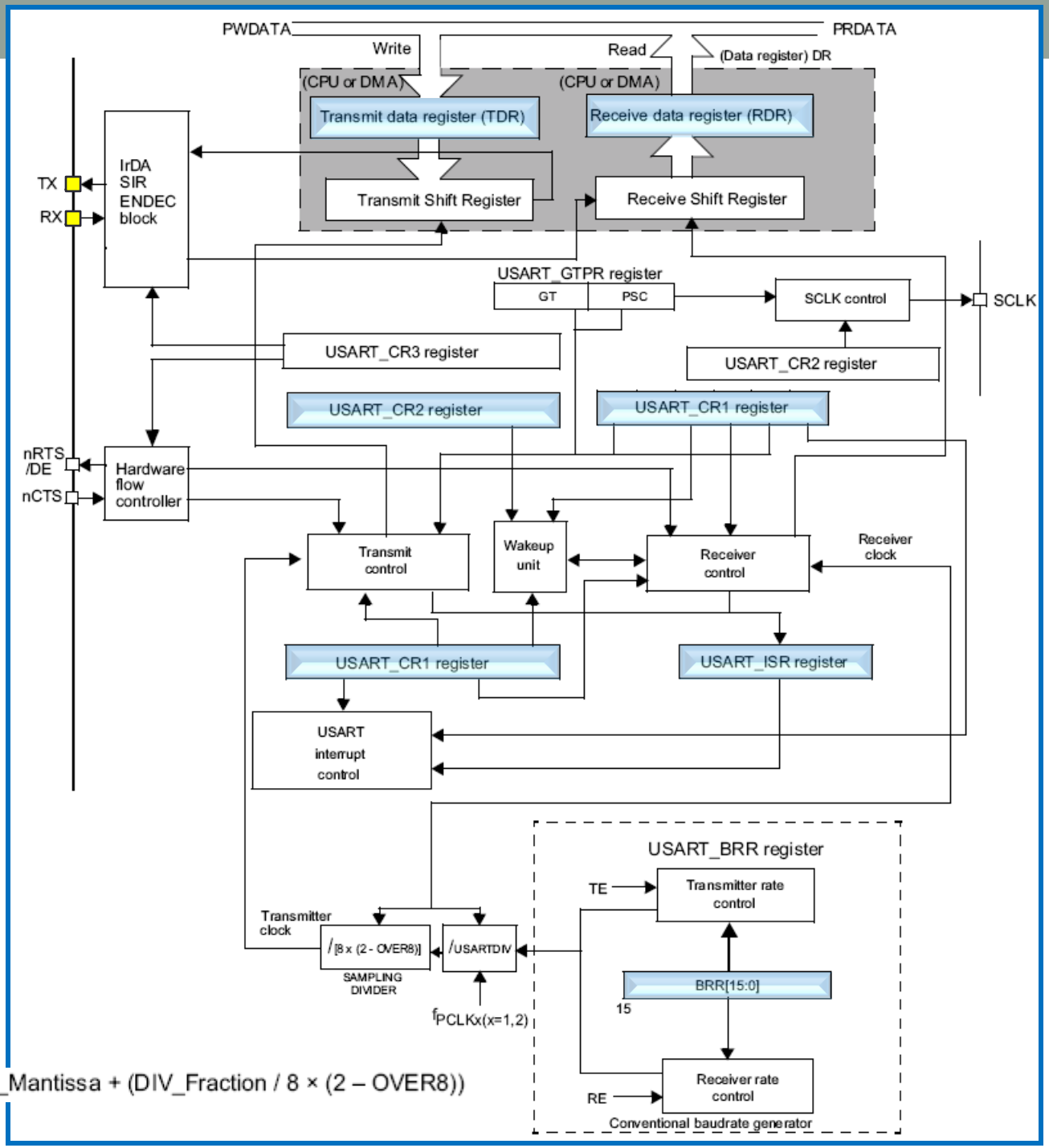
APB2 Peripheral Clock Enable Register (RCC_APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	TIM17 EN	TIM16 EN	TIM15 EN
									rw				rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res	USART1EN	TIM8 EN	SPI1 EN	TIM1 EN	Res	Res	Res	Res	Res	Res	Res	Res	Res	Res	SYS CFG EN
	rw	rw	rw	rw											rw

USART Tx & Rx Pins

	PA	AF	PB	AF	PC	AF	PD	AF	PE	AF
USART1_TX	PA9	7	PB6	7	PC4	7			PE0	7
USART1_RX	PA10	7	PB7	7	PC5	7			PE1	7
USART2_TX	PA2, PA14	7	PB3	7			PD5	7		
USART2_RX	PA3, PA15	7	PB4	7			PD6	7		
USART3_TX			PB10	7	PC10	7	PD8	7		
USART3_RX			PB11	7	PC11	7	PD9	7		
UART4_TX					PC10	5				
UART4_RX					PC11	5				
UART5_TX					PC12	5				
UART5_RX							PD2	5		



USARTDIV = DIV_Mantissa + (DIV_Fraction / 8 × (2 – OVER8))

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
0x00	**USART_CR1**	Res.	Res.	Res.	Res.	EOBIE	RTOIE	DEAT4	DEAT3	DEAT2	DEAT1	DEAT0	DEDT4	DEDT3	DEDT2	DEDT1	DEDT0	OVER8	CMIE	MME	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**USART_CR2**	ADD[7:4]				ADD[3:0]				RTOEN	ABRMOD1	ABRMOD0	ABREN	MSBFIRST	DATAINV	TXINV	RXINV	SWAP	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Res	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
0x08	**USART_CR3**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WUFIE	WUS [1:0]		SCAR CNT2:0]			Res.	DEP	DEM	DDRE	OVRDIS	ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value									0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**USART_BRR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BRR[15:4]												BRR[3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**USART_GTPR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	**USART_RTOR**	BLEN[7:0]								RTO[23:0]																							
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**USART_RQR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TXFRQ	RXFRQ	MMRQ	SBKRQ	ABRRQ
	Reset value																												0	0	0	0	0
0x1C	**USART_ISR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY	ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
	Reset value										0	0	0	0	0	0	0	0	0		0	0	0	0	0	1	1	0	0	0	0	0	0
0x20	**USART_ICR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	WUCF	Res.	Res.	CMCF	Res.	Res.	Res.	Res.	EOBCF	RTOCF	Res.	CTSCF	LBDCF	Res.	TCCF	Res.	IDLECF	ORECF	NCF	FECF	PECF
	Reset value												0			0					0	0		0	0		0		0	0	0	0	0
0x24	**USART_RDR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RDR[8:0]								
	Reset value																								X	X	X	X	X	X	X	X	X
0x28	**USART_TDR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDR[8:0]								
	Reset value																								X	X	X	X	X	X	X	X	X

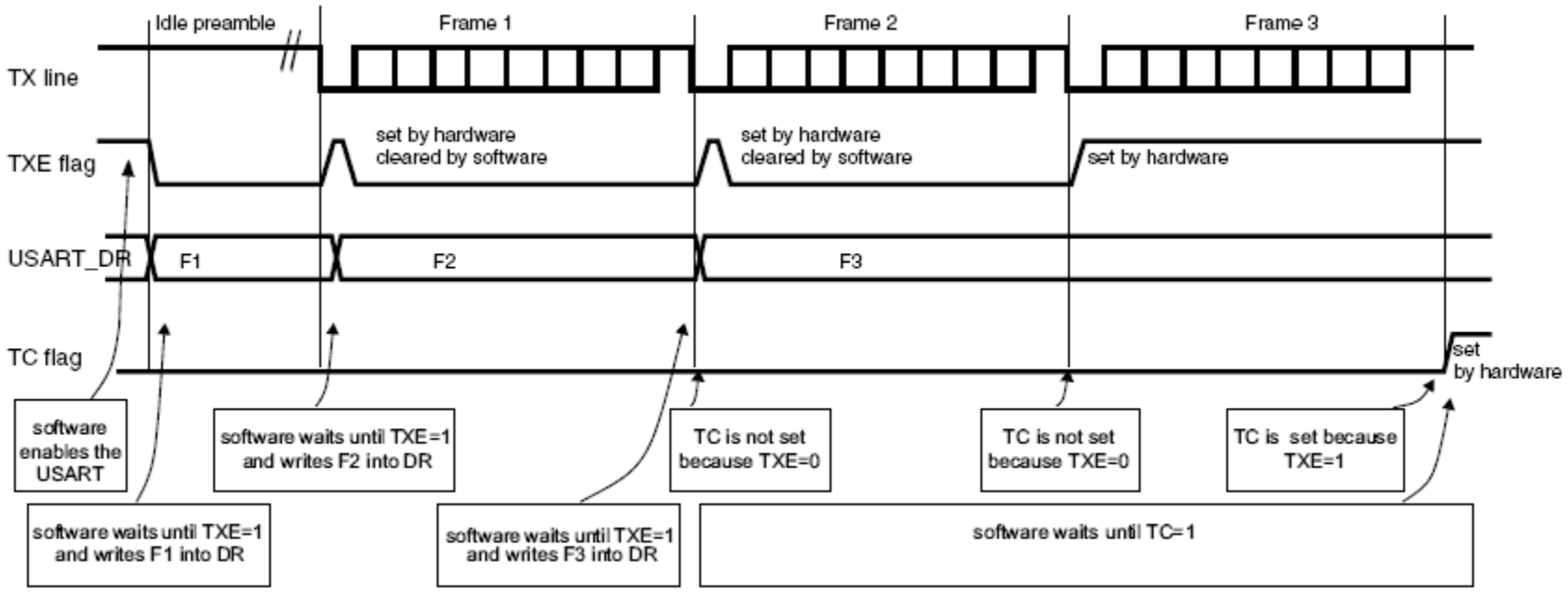
UART most important bits

Register	Bits	ID	Description	Operation
USARTx_CR1	0	UE	USART enable	
	2	RE	Receiver enable	
	3	TE	Transmitter enable	
	5	RXNEIE	RXNE interrupt enable	
	6	TCIE	Transmission complete interrupt enable	
	7	TXEIE	interrupt enable	
	9	PS	PS: Parity selection	0:even, 1:odd
	10	PCE	Parity control enable	
	12	M	Word length	0:8, 1:9 data bits
	15	OVER8	Oversampling mode	0:16, 1:8
USARTx_CR2	13:12	STOP[1:0]	STOP bits	0:1, 1: r, 2:2; 3:1.5
USARTx_BRR	15:4	BRR[15:4]	USARTDIV[15:4]	
	3:0	BRR[3:0]		**if** (OVER8==0) **{**BRR[3:0]=USARTDIV[3:0]**}** **else {**BRR[3:0]=USARTDIV[3:0] shifted 1 bit to the right; BRR[3] must be kept cleared;**}**
USARTx_ISR	5	RXNE	Read data register not empty	1:Received data is ready to be read
	6	TC	Transmission complete	
	7	TXE	Transmit data register empty	
USARTx_RDR	8:0	RDR[8:0]	Receive data value	
USARTx_TDR	8:0	TDR[8:0]	Transmit data value	

Character Transmission Configuration

1. Program USARTx_CR1.M to define the word length.
2. Select the desired baud rate using USARTx_BRR.
3. Program the number of stop bits in USARTx_CR2.STOP[1:0].
4. Enable the USART by setting USARTx_CR1.UE.
5. Select DMA enable (DMAT) in USARTx_CR3 if Multi buffer Communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the USARTx_CR1.TE to send an idle frame as first transmission.
7. Write the data to send in the USARTx_TDR register (this clears the USARTx_ISR.TXE). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USARTx_TDR register, wait until USARTx_ISR.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.

TC/TXE behavior when transmitting



Character Reception Configuration

1. Program the USARTx_CR1.M to define the word length.
2. Select the desired baud rate using USART_BRR.
3. Program the number of stop bits in USARTx_CR2.STOP[1:0].
4. Enable the USART by setting USARTx_CR1.UE.
5. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set USART_CR1.RE. This enables the receiver which begins searching for a start bit.

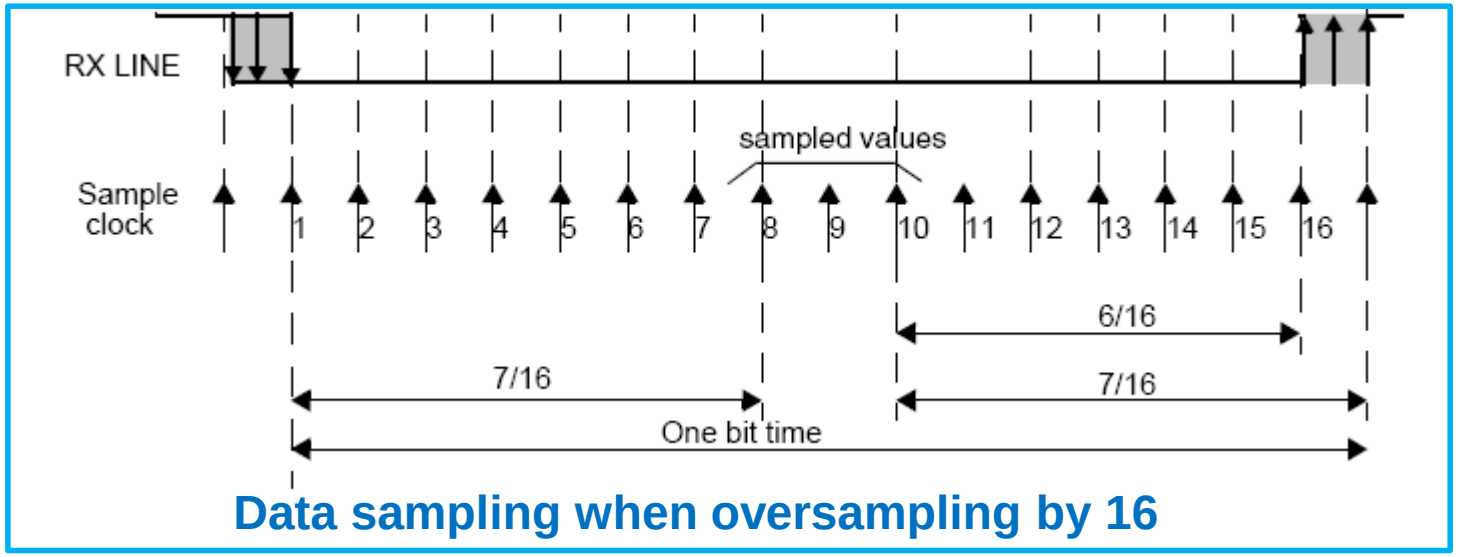
When a character is received

1. USARTx_ISR.RXNE is set. It indicates that the content of the shift register is transferred to USARTx_RDR. In other words, data has been received and can be read (as well as its associated error flags).
2. An interrupt is generated if USARTx_CR1.RXNEIE is set.
3. The error flags can be set if a frame error, noise or an overrun error has been detected during reception. PE flag can also be set with RXNE.
4. In multibuffer, USARTx_ISR.RXNE is set after every byte received and is cleared by the DMA read of the USARTx_RDR.
5. In single buffer mode, clearing RXNE is performed by a software read to USARTx_RDR. RXNE flag can also be cleared by writing 1 to USARTx_RQR.RXFRQ. RXNE must be cleared before the end of the reception of the next character to avoid an overrun error.

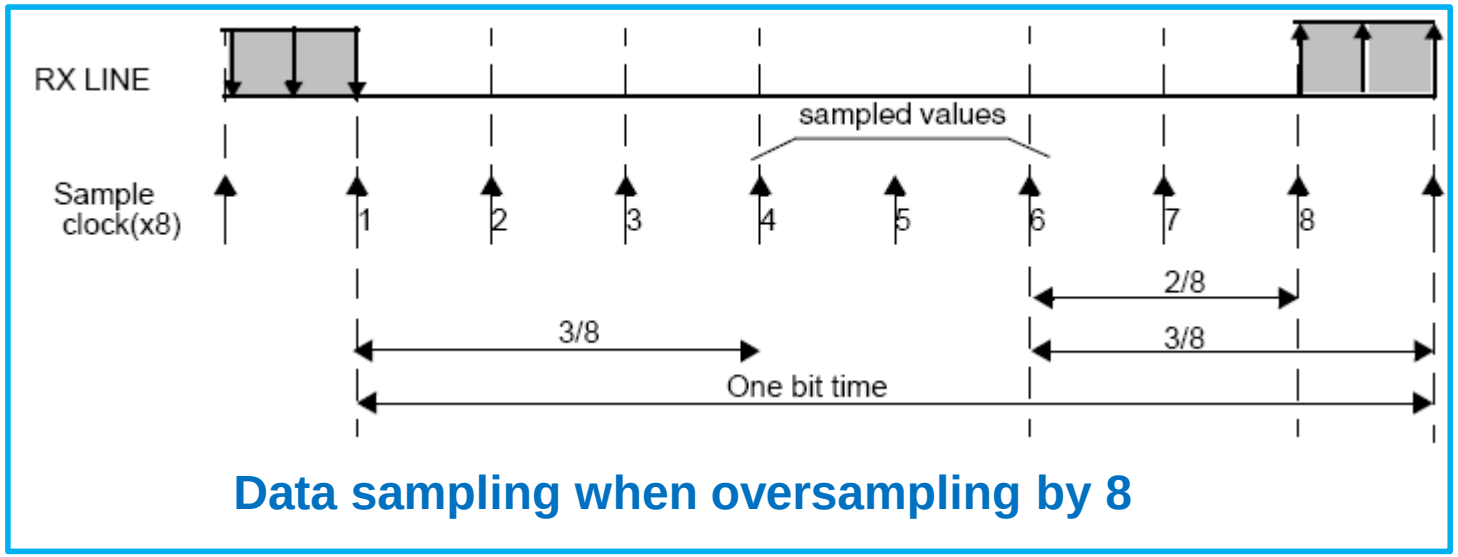
Oversampling

- The receiver implements different user-configurable oversampling techniques (except in synchronous mode) for data recovery by discriminating between valid incoming data and noise.
  - This allows a trade of between the maximum communication speed and noise/clock inaccuracy immunity.
- The oversampling method can be selected by programming the OVER8 bit in the USART_CR1 register and can be either 16 or 8 times the baud rate clock.

Oversampling



Data sampling when oversampling by 16



Data sampling when oversampling by 8

Oversampling

- Depending on the application:
  - Select oversampling by 8 (OVER8=1) to achieve higher speed (up to $f_{CK}/8$).
    - In this case the maximum receiver tolerance to clock deviation is reduced.
  - Select oversampling by 16 (OVER8=0) to increase the tolerance of the receiver to clock deviations.
    - In this case, the maximum speed is limited to maximum $f_{CK}/16$, where f_{CK} is the clock source frequency.

Baud rate generation

- The baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the USART_BRR register
- The baud rate for standard USART, in case of oversampling by 16 (OVER8=0), is calculated as follows:

$$BRR[15:0] = USARTDIV = \frac{f_{CK}}{baud\ rate}$$

- in case of oversampling by 8 (OVER8=1), it is calculated as follows:

$$USARTDIV = \frac{2\ f_{CK}}{baud\ rate}$$

$$BRR[3:0] = USARDIV[3:0] \gg 1$$

$$BRR[15:4] = USARDIV[15:4]$$

How to derive USARTDIV from USART_BRR register values

- Example 1: To obtain 9600 baud with f_{CK} = 8 MHz.
  - In case of oversampling by 16:
    - USARTDIV = 8 000 000/9600
    - BRR[31:0] = USARTDIV = 833d = 0x0341

```
BRR = USARTDIV & 0xFFFF;
```

  - In case of oversampling by 8:
    - USARTDIV = 2 * 8 000 000/9600
    - USARTDIV = 1666,66 (1667d = 0x683)
    - BRR[3:0] = 0x3 >> 1 = 0x1
    - BRR = 0x681

```
BRR = (USARTDIV & 0xFFF0) | ( (USARTDIV & 0xF) >> 1);
```

How to derive USARTDIV from USART_BRR register values

- Example 2: To obtain 921.6 Kbaud with f_{CK} = 48 MHz.
  - In case of oversampling by 16:
    - USARTDIV = 48 000 000/921 600
    - BRR = USARTDIV = 52d = 0x34
  - In case of oversampling by 8:
    - USARTDIV = 2 * 48 000 000/921 600
    - USARTDIV = 104 (104d = 0x68)
    - BRR[3:0] = USARTDIV[3:0] >> 1 = 0x8 >> 1 = 0x4
    - BRR = 0x64

Code Example (1)

At last…

```
void GPIO_Config(void) {
 // PC4 configuration (TX)
  RCC->AHBENR     |=   1 << 19;      // enable GPIOC clock
  GPIOC->MODER    |=   2 << (4*2);   // GPIO_Mode_AF
  GPIOC->OTYPER   |=   1 << (4*1);   // GPIO_OType_OD
  GPIOC->OSPEEDR  |=   3 << (4*2);   // GPIO_Speed_50MHz
  GPIOC->PUPDR    &= ~(3 << (4*2)); // GPIO_PuPd_NOPULL
  GPIOC->AFR[0]   |=   7 << (4*4);   // AF7
 // PC5 configuration (RX)
  GPIOC->MODER    |=   2 << (5*2);   // GPIO_Mode_AF
  GPIOC->AFR[0]   |=   7 << (5*4);   // AF7
}
```

Code Example (2)

```
void USART1_Config(void){
 RCC->APB2ENR |= RCC_APB2ENR_USART1EN;  // Enable USART1 clock
 USART1->BRR   = 72000000/115200;
 USART1->CR1   &=  ~USART_CR1_OVER8;  // Oversampling mode = 16
 USART1->CR1   &=  ~USART_CR1_M;      // Word length = 8 bits
 USART1->CR1   &=  ~USART_CR1_PCE;    // No parity
 USART1->CR1   |=   USART_CR1_TE;     // Transmitter enable
 USART1->CR1   |=   USART_CR1_RE;     // Receiver enable
 USART1->CR1   |=   USART_CR1_UE;     // USART enable
 USART1->CR2   &= ~(USART_CR2_STOP_1 | USART_CR2_STOP_0); // one stop bit
}
```

Code Example (3)

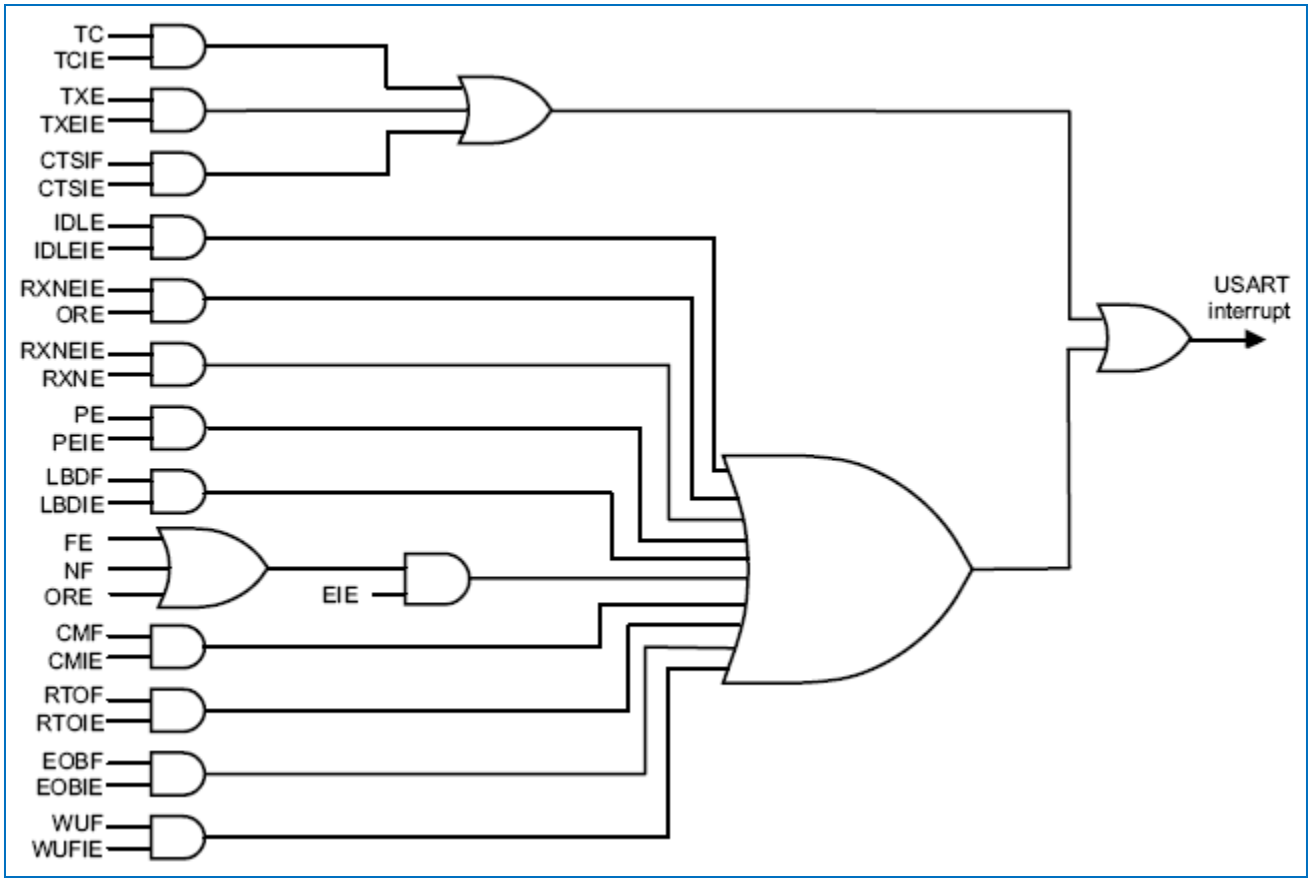
```
uint8_t SendChar (uint8_t ch)
{
  while (!(USART1->ISR & USART_ISR_TXE));
  USART1->TDR = (ch & 0xFF);
  return (ch);
}

uint8_t GetChar (void)
{  while (!(USART1->ISR & USART_ISR_RXNE));
  return ((uint8_t)(USART1->RDR & 0xFF));
}
```

Vector Table (detail)

IRQn

Position	Priority	Type of priority	Acronym	Description	Address
37	44	settable	USART1_EXTI25	USART1 global interrupt & EXTI Line 25	0x0000 00D4
38	45	settable	USART2_EXTI26	USART2 global interrupt & EXTI Line 26	0x0000 00D8
39	46	settable	USART3_EXTI28	USART3 global interrupt & EXTI Line 28	0x0000 00DC
52	59	settable	UART4_EXTI34	UART4 global and EXTI Line 34 interrupts	0x0000 0110
53	60	settable	UART5_EXTI35	UART5 global and EXTI Line 35 interrupts	0x0000 0114

USART interrupt mapping diagram



Summary of DMA2 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5
ADC	ADC2	ADC4	ADC2(1)	ADC4(1)	ADC3
SPI3	SPI3_RX	SPI3_TX			
UART4			UART4_RX		UART4_TX
TIM6 / DAC			TIM6_UP DAC_CH1		
TIM7 / DAC				TIM7_UP DAC_CH2	
TIM8	TIM8_CH3 TIM8_UP	TIM8_CH4 TIM8_TRIG TIM8_COM	TIM8_CH1		TIM8_CH2

References

- http://neuron.feld.cvut.cz/micro/stm32/tut-02-usart-en.html
- http://www.micromouseonline.com/2009/12/31/stm32-usart-basics/#axzz2iBnyv6V9
- http://pandafruits.com/stm32_primer/stm32_primer_uart.php
- http://easystm32.ru/interfaces/15-uart-in-stm32-part-1
- http://easystm32.ru/interfaces/16-uart-in-stm32-part-2
- http://cooldianzi.blog.163.com/blog/static/671115382013169575456 7/
- http://gpio.kaltpost.de/?page_id=167 *
- https://svn.kapsi.fi/jpa/paatti/io/io_gsm.c
- http://wiki.seabright.co.nz/wiki/HelloSTM32.html
- http://www.mikrocontroller.net/attachment/81212/usart.c
- http://nute.googlecode.com/svn/trunk/Armlet/Armlet2/Armlet2North/src/peripheral.cpp
- http://www.mikrocontroller.net/topic/308799
- http://www.codeproject.com/Articles/149950/STM32-Discovery-The-Basics-Creating-a-Project
- http://hobbymc.blogspot.mx/2011/01/stm32-discovery-basics-echo-serial-port.html
- http://www.keil.com/download/docs/359.asp

- LabVIEW
  - http://myweb.wit.edu/johnsont/Classes/LabView_Labs/Elec163LabVIEW8-RS232v1.htm
- CoX Peripheral Library
  - http://www.coocox.org/cox/manual/STM32F1xx/group___co_x___peripheral___lib.html