ECE3411

Lecture 5

USART: Universal Synchronous & Asynchronous Receiver & Transmitter

Sung Yeul Park
Department of Electrical & Computer Engineering
University of Connecticut
Email: sung_yeul.park@uconn.edu

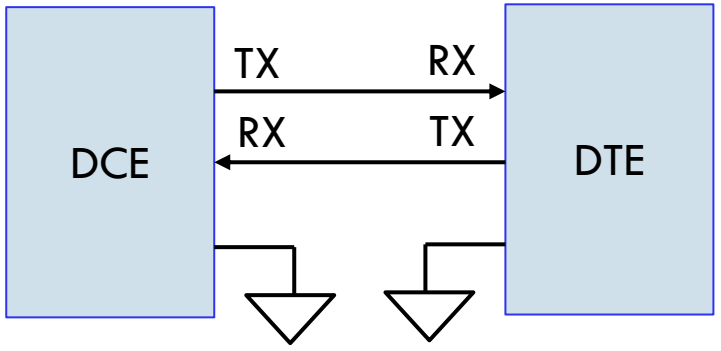
Adapted from Lecture 2a, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider
Adapted from John Chandy ECE3411 – Fall 2024

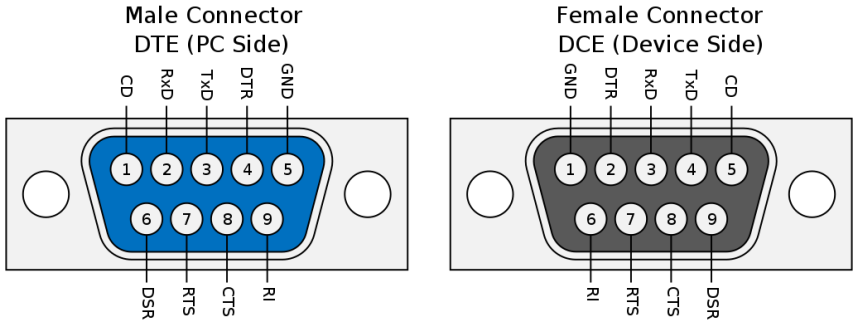
Learning Objectives

- Understand USART communication principles and protocols
- Configure AVR128DB48 USART peripherals for various applications
- Implement interrupt-driven serial communication
- Debug serial communication issues using proper tools
- Create practical applications using USART for embedded systems
- Apply flow control and error handling techniques

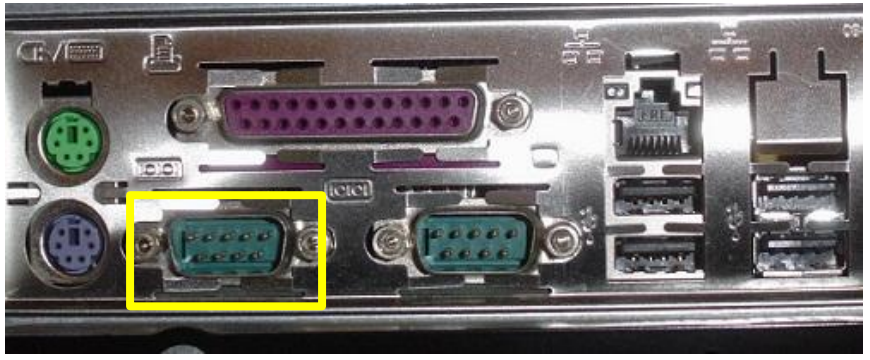
USART

- Universal Synchronous/Asynchronous Receiver Transceiver (USART) implements serial communications
- Asynchronous mode communicates over a 3-wire cable: TX, RX, Gnd
- Designed for modems a long time ago and protocol is slow

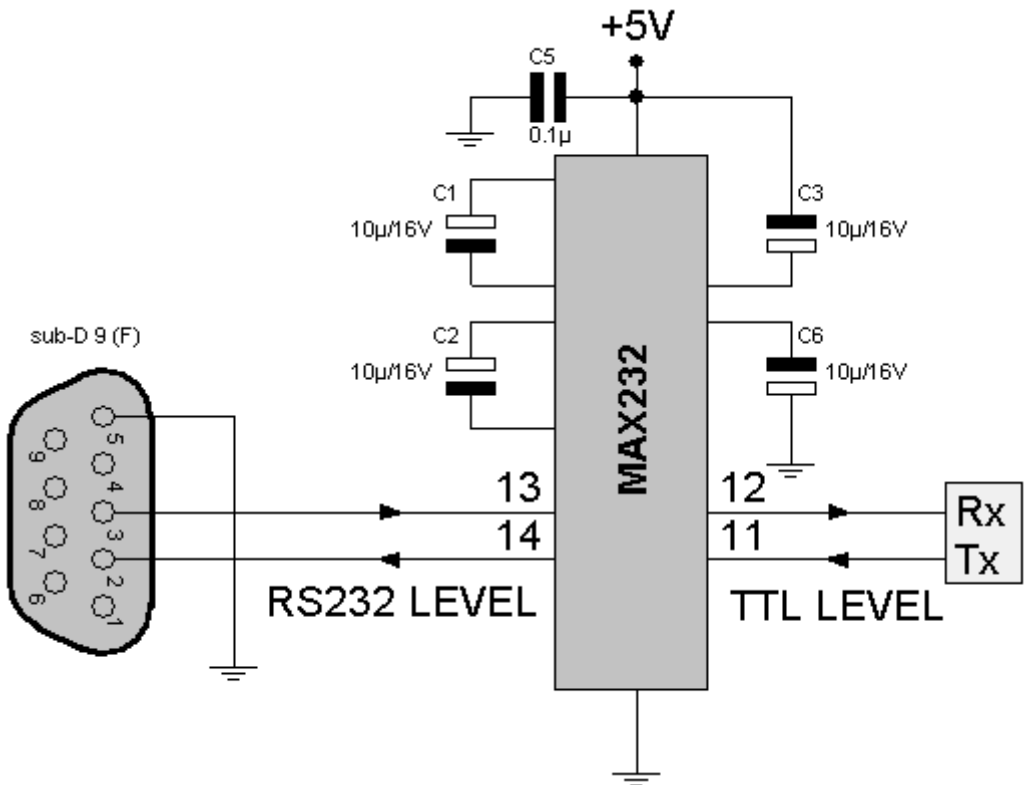


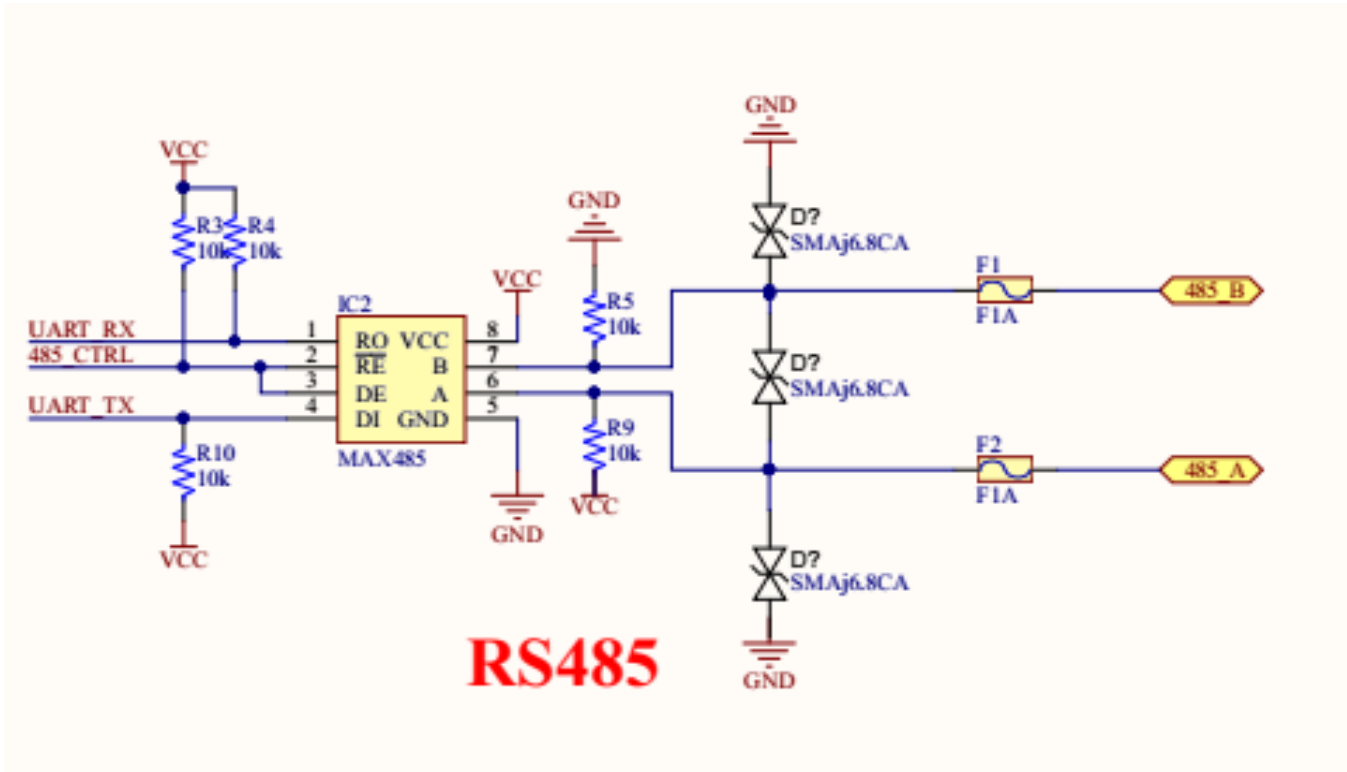


RS-232 Pinout for the DE-9 Connector



RS 232 / RS 485 Interface

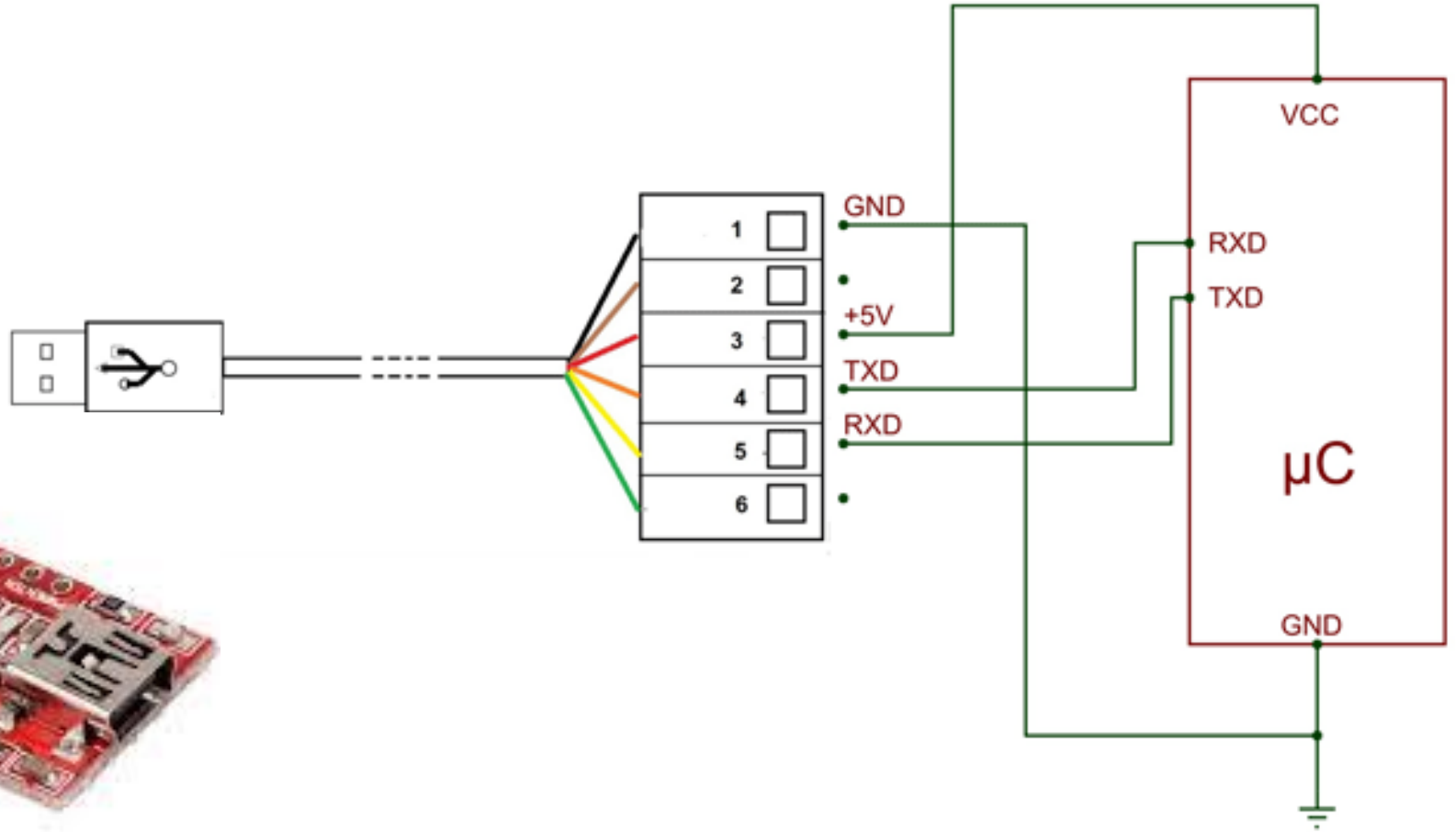




USART: RS-232 and RS-485

Feature	RS-232 Setup	RS-485 Setup
Interface IC	MAX232	MAX485 or SN75176
AVR Pins	USART TX/RX	USART TX/RX + GPIO for DE/RE control
Voltage Level	TTL (converted to ±12V)	TTL (converted to differential ±5V)
Mode	Point-to-point Full-duplex mode	Multi-drop (bus) Half-duplex mode

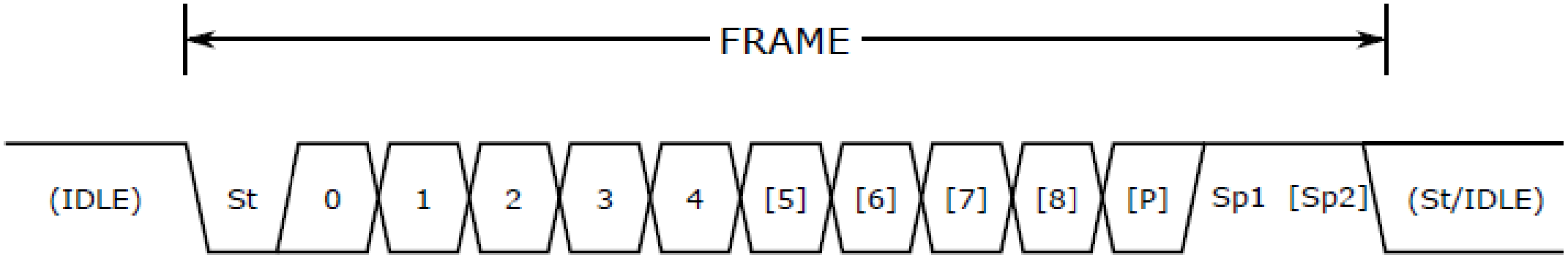
FTDI Interface



Frame Format

- To transmit a byte (i.e., one char) we need at least one start bit (receiving clock starts when falling edge is received), 8 data bits, and one stop bit: Total of 10 bits.

Figure 27-2. Frame Formats



St	Start bit, always low
(n)	Data bits (0 to 8)
P	Parity bit, may be odd or even
Sp	Stop bit, always high
IDLE	No transfer on the communication line (RxD or TxD). The Idle state is always high.

USART

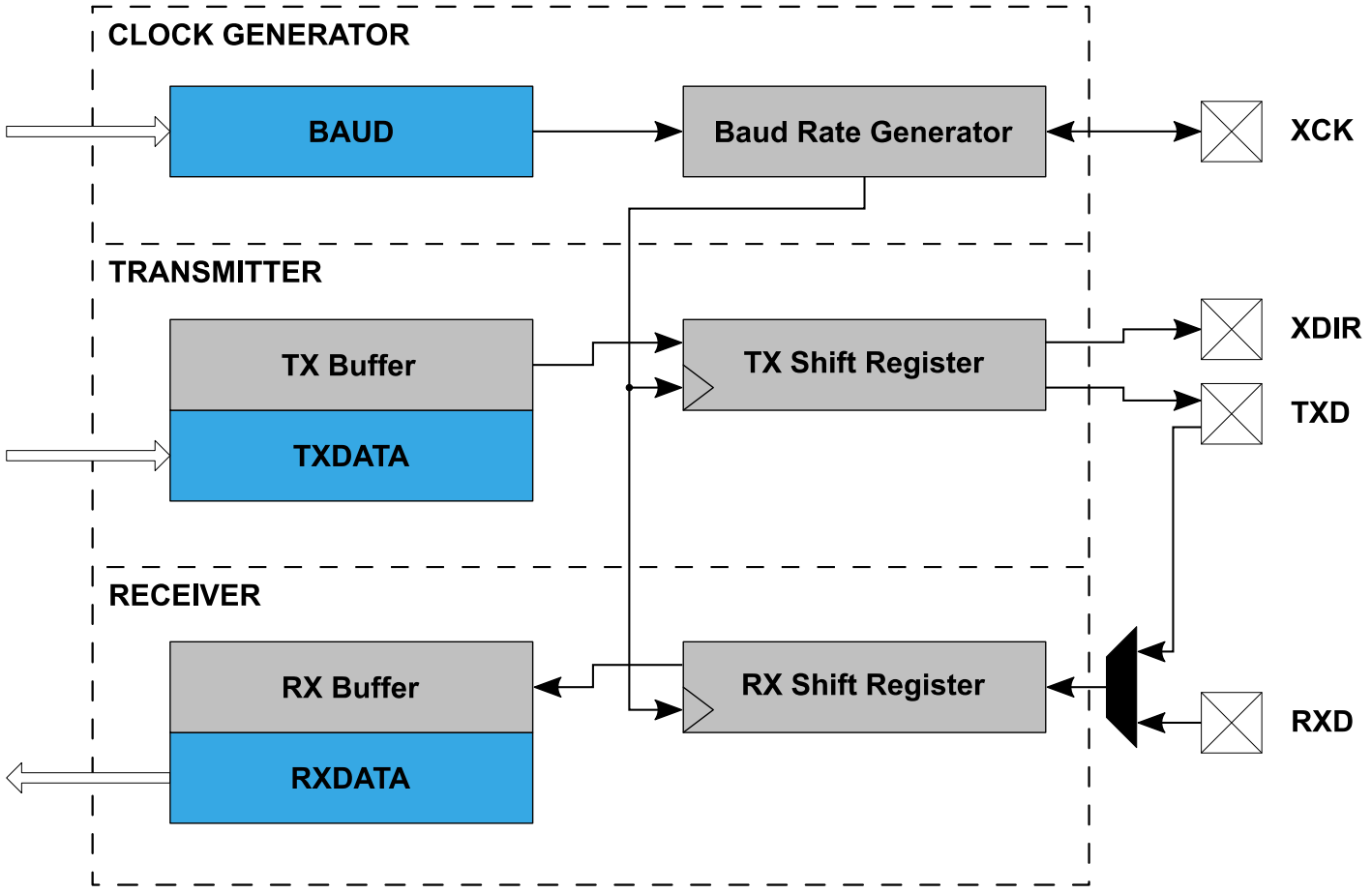
- Baud rate: bits per second transmission rate
- Each byte takes 10 bits (1 start bit, 8 data bits, 1 stop bit)
- 9600 baud = 960 bytes/s (approximately 1ms per byte)
- This is slow: Therefore, in SW start transmitting a character, then do something else!
- Baud rate can go up to 1Mbit per second
- The cable limits the maximum possible Baud rate

AVRDx USART

- USART = Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Clock generator, Transmitter, Receiver
- AVR128DB48 has five USARTs

Signal	Type	Description
XCK	Output/input	Clock for synchronous operation
XDIR	Output	Transmit enable for RS-485
TxD	Output/input	Transmitting line (and receiving line in One-Wire mode)
RxD	Input	Receiving line

Figure 25-1. USART Block Diagram



UBRR0H and UBRR0L

- Baud rate is translated relative to the system oscillator clock frequency f_{CLK_PER} to the BAUD register

Table 27-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Conditions	Baud Rate (Bits Per Seconds)	USART.BAUD Register Value Calculation
Asynchronous	$f_{BAUD} \leq \frac{f_{CLK\_PER}}{S}$ $USART.BAUD \geq 64$	$f_{BAUD} = \frac{64 \times f_{CLK\_PER}}{S \times BAUD}$	$BAUD = \frac{64 \times f_{CLK\_PER}}{S \times f_{BAUD}}$
Synchronous Host	$f_{BAUD} \leq \frac{f_{CLK\_PER}}{S}$ $USART.BAUD \geq 64$	$f_{BAUD} = \frac{f_{CLK\_PER}}{S \times BAUD[15:6]}$	$BAUD[15:6] = \frac{f_{CLK\_PER}}{S \times f_{BAUD}}$

- S = samples per bit – 16 (single speed) or 8 (double speed)

Curiosity Nana board Hardware

	TX	RX	Usage
USART2	PF4	PF5	External device communication
USART3	PB0	PB1	**Direct PC communication via USB**

Key Registers

Register	Address	Function
USARTn.BAUD	Base+0x00	Baud rate setting
USARTn.CTRLA	Base+0x01	Control A (interrupts)
USARTn.CTRLB	Base+0x02	Control B (enable, mode)
USARTn.CTRLC	Base+0x03	Control C (frame format)
USARTn.RXDATAL	Base+0x04	Receive data low
USARTn.RXDATAH	Base+0x05	Receive data high
USARTn.TXDATAL	Base+0x06	Transmit data low
USARTn.TXDATAH	Base+0x07	Transmit data high
USARTn.STATUS	Base+0x08	Status register

USARTn.BAUD

Bit	15	14	13	12	11	10	9	8
	BAUD[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	BAUD[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:8 – BAUD[15:8] USART Baud Rate High Byte
This bit field holds the MSB of the 16-bit Baud register.

Bits 7:0 – BAUD[7:0] USART Baud Rate Low Byte
This bit field holds the LSB of the 16-bit Baud register.

```
#define F_CPU 4000000UL  // 4MHz default clock

#define BAUD_RATE 9600

void USART3_Init(uint32_t baud) {
// Calculate baud rate register value
uint16_t baud_setting = (F_CPU * 64) / (16 * baud);

// Set baud rate
USART3.BAUD = baud_setting;
```

Control register: CTRLA

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	DREIE	RXSIE	LBME	ABEIE		RS485
Access	R/W	R/W	R/W	R/W	R/W	R/W		R/W
Reset	0	0	0	0	0	0		0

- `RXCIE`: Receive character complete interrupt enable
- `TXCIE`: Enables interrupt for both members in TX queue being empty
- `DREIE`: Enables interrupt if the first of the output pipeline is empty. Ready to transmit.
- `RXSIE`: Receive start frame interrupt enable
- `LBME`: Loop-back mode enable
- `ABEIE`: Auto-baud error interrupt enable
- `RS485`: Enable RS-485 mode

Control register: CTRLB

Bit	7	6	5	4	3	2	1	0
	RXEN	TXEN		SFDEN	ODME	RXMODE[1:0]		MPCM
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

- RXEN: RX enable
- TXEN: TX enable → Must still enable the corresponding pin as an output
- SFDEN: Enable start of frame detection
- ODME: Open drain mode enable
- RXMODE: Receiver mode
- MPCM: Multiple processor address mode (can connect more than 2 devices to the line)

Value	Name	Description
0x00	NORMAL	Normal-Speed mode
0x01	CLK2X	Double-Speed mode
0x02	GENAUTO	Generic Auto-Baud mode
0x03	LINAUTO	LIN Constrained Auto-Baud mode

```
// Enable transmitter and receiver
USART3.CTRLB = USART_TXEN_bm | USART_RXEN_bm;
```

Control register: CTRLC

Bit	7	6	5	4	3	2	1	0
	CMODE[1:0]		PMODE[1:0]		SBMODE	CHSIZE[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	1	1

- CMODE: Communication mode

Value	Name	Description
0x00	ASYNCHRONOUS	Asynchronous USART
0x01	SYNCHRONOUS	Synchronous USART
0x02	IRCOM	Infrared Communication
0x03	MSPI	Host SPI

- PMODE: Parity mode

Value	Name	Description
0x0	DISABLED	Disabled
0x1	-	Reserved
0x2	EVEN	Enabled, even parity
0x3	ODD	Enabled, odd parity

- SBMODE: Stop bit mode: 0 – 1 stop bit; 1 – 2 stop bits
- CHSIZE: Character size

Value	Name	Description
0x00	5BIT	5-bit
0x01	6BIT	6-bit
0x02	7BIT	7-bit
0x03	8BIT	8-bit
0x04	-	Reserved
0x05	-	Reserved
0x06	9BITL	9-bit (Low byte first)
0x07	9BITH	9-bit (High byte first)

```
// Set frame format: 8 data bits, no parity, 1 stop bit
USART3.CTRLC = USART_CHSIZE_8BIT_gc;
```

Control register: STATUS

Bit	7	6	5	4	3	2	1	0
	RXCIF	TXCIF	DREIF	RXSIF	ISFIF		BDF	WFB
Access	R	R/W	R	R/W	R/W		R/W	W
Reset	0	0	1	0	0		0	0

- RXCIF: Receive character complete flag
- TXCIF: TX queue empty
- DREIF: First of the output pipeline is empty.
  Ready to transmit.
- RXSIF: Receive start frame interrupt enable
- ISFIF: Inconsistent synchronization field
  (during automaud)
- BDF: Break detected flag
- WFB: Enable wait-for-break feature

```
void USART3_SendChar(char data) {
// Wait for transmit buffer to be empty
while (!(USART3.STATUS & USART_DREIF_bm));

// Send data
USART3.TXDATAL = data;
}
```

```
char USART3_ReceiveChar(void) {
// Wait for receive complete
while (!(USART3.STATUS & USART_RXCIF_bm));

// Return received data
return USART3.RXDATAL;
}
```

RXDATA and TXDATA

- RXDATA is a 16-bit register comprised of two 8-bit registers (RXDATAH and RXDATAL).
- Usually, only the RXDATAL is relevant. RXDATAH is used for 9-data bit setups
- RXDATAH also has some status bits related to the data reception
- Similarly, TXDATA has two 8-bit registers (TXDATAH and TXDATAL).

```
void USART3_SendChar(char data) {
// Wait for transmit buffer to be empty
while (!(USART3.STATUS & USART_DREIF_bm));

// Send data
USART3.TXDATAL = data;
}
```

```
char USART3_ReceiveChar(void) {
// Wait for receive complete
while (!(USART3.STATUS & USART_RXCIF_bm));

// Return received data
return USART3.RXDATAL;
}
```

RXDATAH

Bit	7	6	5	4	3	2	1	0
	RXCIF	BUFOVF				FERR	PERR	DATA[8]
Access	R	R				R	R	R
Reset	0	0				0	0	0

Bit 7 – RXCIF USART Receive Complete Interrupt Flag

This flag is set when there are unread data in the receive buffer and cleared when the receive buffer is empty.

Bit 6 – BUFOVF Buffer Overflow

This flag is set if a buffer overflow is detected. A buffer overflow occurs when the receive buffer is full, a new frame is waiting in the receive shift register, and a new Start bit is detected. This flag is cleared when the Receiver Data (USARTn.RXDATAL and USARTn.RXDATAH) registers are read.
This flag is not used in the Host SPI mode of operation.

Bit 2 – FERR Frame Error

This flag is set if the first Stop bit is '0' and cleared when it is correctly read as '1'.
This flag is not used in the Host SPI mode of operation.

Bit 1 – PERR Parity Error

This flag is set if parity checking is enabled and the received data has a parity error, or else, this flag cleared. For details on parity calculation, refer to 27.3.4.1. Parity.
This flag is not used in the Host SPI mode of operation.

Bit 0 – DATA[8] Receiver Data Register

When using a 9-bit frame size, this bit holds the ninth bit (MSb) of the received data.
When the Receiver Mode (RXMODE) bit field in the Control B (USARTn.CTRLB) register is configured to LIN Constrained Auto-Baud (LINAUTO) mode, this bit indicates if the received data are within the response space of a LIN frame. This bit is cleared if the received data are in the protected identifier field and is otherwise set.

Sample Code

```c
#define F_CPU 4000000UL  // 4MHz default clock

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>

#define BAUD_RATE 9600

void USART3_Init(uint32_t baud) {
// Calculate baud rate register value
uint16_t baud_setting = (F_CPU * 64) / (16 * baud);

// Set baud rate
USART3.BAUD = baud_setting;

// Set frame format: 8 data bits, no parity, 1 stop bit
USART3.CTRLC = USART_CHSIZE_8BIT_gc;

// For USART3 (already configured on Curiosity Nano)
PORTB.DIRSET = PIN0_bm;  // PA0 as output (TX)
PORTB.DIRCLR = PIN1_bm;  // PA1 as input (RX)

// Enable transmitter and receiver
USART3.CTRLB = USART_TXEN_bm | USART_RXEN_bm;
}

void USART3_SendChar(char data) {
// Wait for transmit buffer to be empty
while (!(USART3.STATUS & USART_DREIF_bm));

// Send data
USART3.TXDATAL = data;
}

void USART3_SendString(const char* str) {
while (*str) {
USART3_SendChar(*str);
str++;
}
}

char USART3_ReceiveChar(void) {
// Wait for receive complete
while (!(USART3.STATUS & USART_RXCIF_bm));

// Return received data
return USART3.RXDATAL;
}

int main(void) {
// Initialize USART0
USART3_Init(BAUD_RATE);

// Send welcome message
USART3_SendString("AVR128DB48 USART Demo\r\n");
USART3_SendString("Hello World!\r\n");
USART3_SendString("Type something: ");

while(1) {
// Echo received characters
char received = USART3_ReceiveChar();
USART3_SendChar(received);

// Send newline if Enter is pressed
if (received == '\r') {
USART3_SendString("\r\nYou typed: ");
}
}

return 0;
}
```

ASCII Table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Lab Practice #4: Changing LED Mode using UART

Change the previous lab to allow you to change the frequency and position of the LED based on input entered from the UART Terminal.

- The LED starts blinking at 2Hz
- Every 5 seconds, the program prints the following message on the terminal:
  `Do you want to change the frequency or position? (F/P)`
- If the user enters "`F`", the program should print out:
  `Frequency:`
  The user will then enter a frequency (from 1-10Hz) and the program should change the LED frequency accordingly
- If the user enters "`P`", the program should print out:
  `Position:`
  The user will then enter a frequency (from 0-7) and the program should change the LED position accordingly

stdio.h

- AVR library provides a `stdio.h` file that contains normal C stdout functions
- It depends on user-provided `uart_getchar` and `uart_putchar` functions – this code will be provided to you
- `FILE *stdout;`
- `FILE *stdin;`
- `printf(fmt_str, …)`
- `scanf("%s", buf);`
- `getchar()`
- `putchar(c)`

Using `stdio.h`

```
#include <stdio.h>
#include <string.h>
#include "uart.h"

int main()
{
      uart_init(3, 9600, NULL);
      while(1) {
            char buf[100];
            ...
            printf("Hello there!\n");
            scanf("%99s", buf);
            if (strcmp(buf, "Hello") == 0){
                  ...
            }
      }
}
```

uart.h

```
/*** uart.h ***/

/*
 * Perform UART startup initialization.
 */
FILE*  uart_init(uint8_t usart_num, uint32_t baud_rate, FILE* stream);

/*
 * Send one character to the UART.
 */
int    uart_putchar(char c, FILE *stream);

/*
 * Receive one character from the UART.  The actual reception is line-buffered,
 * and one character is returned from the buffer at each invocation.
 */
int    uart_getchar(FILE *stream);
```

uart.h

- `uart_init` takes three arguments:
  - `usart_num` is the number of the USART – should be within the range for the MCU
  - `baud_rate` is the desired baud rate
  - `stream` is an optional field that allows the user to create its own I/O stream. If `stream` is `NULL`, `uart_init` will initialize the `stdout`, `stdin`, and `stderr` streams.

```
int main()
{
    …
    uart_init(3, 9600, NULL);
    …
}
```

uart.c

- `uart.c` implements the `uart.h` functions but calls functions in the `uart-avrdx.c` file to do the MCU specific code

```
void* usart_init(uint8_t, uint32_t);
void usart_transmit_data(void*, char);
void usart_wait_until_transmit_ready(void*);
int usart_receive_data(void*);
```

Transmission

```
int uart_putchar(char c, FILE *stream)
{
    /* Alarm (Beep, Bell) */
    if (c == '\a') {
      fputs("*ring*\n", stderr);
      return 0;
    }

   /* Newline is translated into a CR */
   if (c == '\n')
      uart_putchar('\r', stream);

   void* usart = fdev_get_udata(stream);
   usart_wait_until_transmit_ready(usart);
   usart_transmit_data(usart, c);

   return 0;
}
```

Receiving

- `int uart_getchar(FILE *stream)` in `uart.c` is a simple line-editor that allows users to delete and re-edit the characters entered, until either `CR` or `NL` is entered
- printable characters entered will be echoed using `uart_putchar()`
  - So, you can see the character received by the MCU and you can verify whether the transmission was without error if you recognize the character as the transmitted one (as pressed by the keyboard)
- The core part in `uart_getchar` is

```
int uart_getchar(FILE *stream)
{
    ...
    void* usart = fdev_get_udata(stream);
    c = usart_receive_data(usart);
    ...
    uart_putchar(c, stream);
    ...
}
```

UART Setup: Using `uart.h` Library

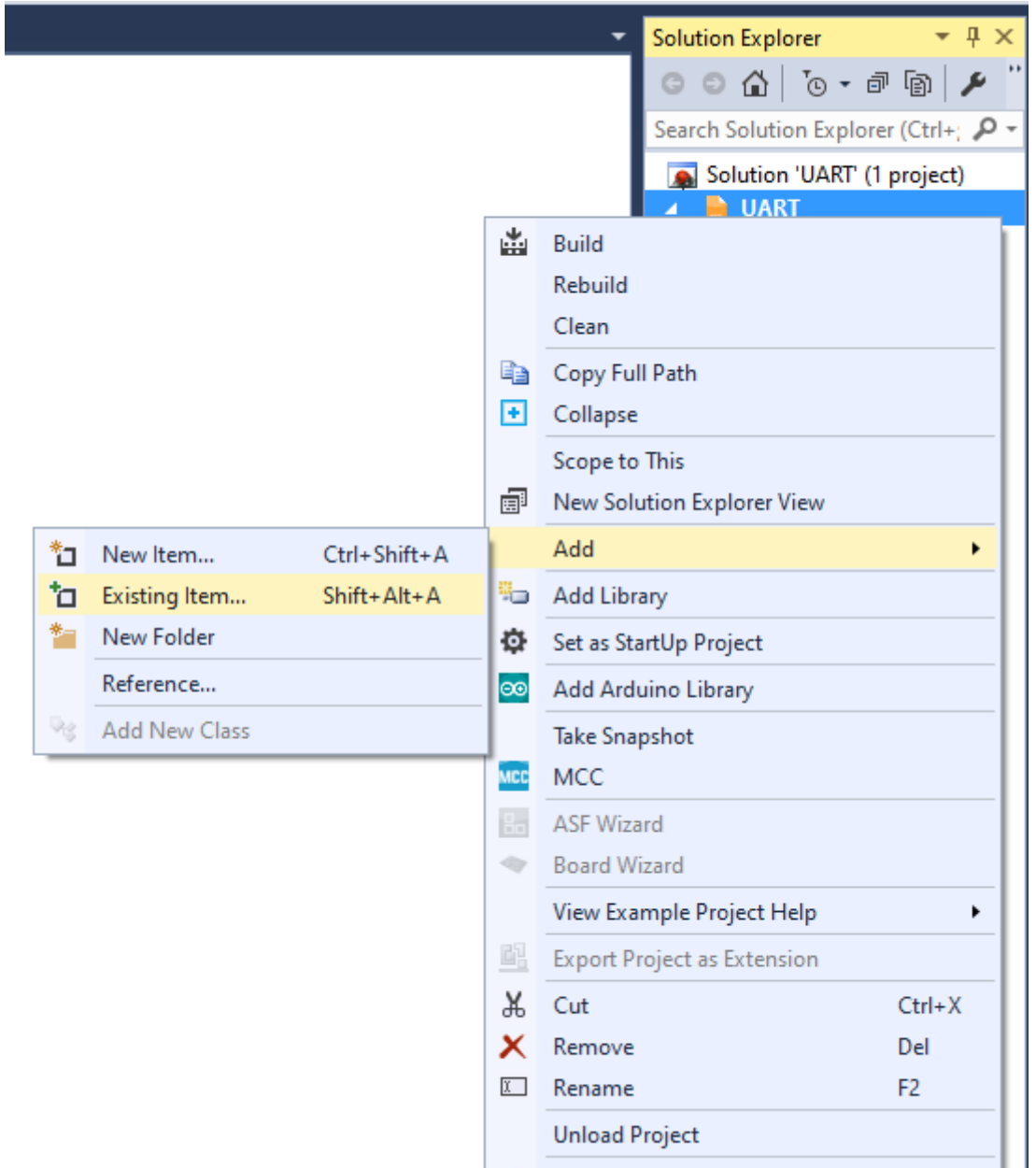
- In order to make it a little bit easier for you, the library files "`uart.c`" and "`uart-avrdx.c`" provided and it defines useful basic UART functions.
  - `uart.h`, `uart.c`, and `uart-avrdx.c` can be downloaded from HuskyCT
- The corresponding prototypes of the functions are declared in `uart.h`
- In order to use the functions provided by `uart.c`, you need to:
  1. Add `uart.c`, `uart-avrdx.c`, and `uart.h` files to your Microchip Studio project source files
  2. Include `uart.h` as a header file in your code, i.e.

```
#include "uart.h"
```

Adding Header and C Files to a Project

- In order to add the uart files to the project:
  1. Copy the files `uart.c`, `uart-avrdx.c`, and `uart.h` into the project directory.
  2. In the 'Solution Explorer' window, right click on the project's name → Add → Existing Item …
  3. Select `uart.c`, `uart-avrdx.c`, and `uart.h` and click "Add".
  4. Don't forget to declare/include the header file in your code by adding `#include "uart.h"`
- See the next few slides for illustration

Adding Header and C Files to a Project



uart-avrdx.c

- The `FILE` stream struct has a `udata` field that allows implementation specific data to be added to the stream.
  - `usart_init` will return a pointer to implementation specific data - in this case a pointer to the `USART_t` struct which is just the block of USART registers.
  - This pointer is assigned to `udata`
  - Subsequent calls to the `usart_xxx` functions will pass that pointer.

uart-avrdx.c

```
void* usart_init(uint8_t usartnum, uint32_t baud_rate)
{
        USART_t* usart;

        if (usartnum == 0) {
                usart = &USART0;
                PORTA.DIRSET = PIN0_bm; // enable USART0 TX pin
        }
        else if (usartnum == 1) {
                usart = &USART1;
                PORTC.DIRSET = PIN0_bm; // enable USART1 TX pin
        }
        else if (usartnum == 2) {
                usart = &USART2;
                PORTF.DIRSET = PIN0_bm; // enable USART2 TX pin
        }
        else if (usartnum == 3) {
                usart = &USART3;
                PORTB.DIRSET = PIN0_bm; // enable USART3 TX pin
        }

        // set BAUD and CTRLB registers
        return usart;
}
```

uart-avrdx.c

```
void usart_transmit_data(void* ptr, char c)
{
	USART_t* usart = (USART_t*)ptr;
	...
}

void usart_wait_until_transmit_ready(void *ptr)
{
	USART_t* usart = (USART_t*)ptr;
	...
}

int usart_receive_data(void* ptr)
{
	USART_t* usart = (USART_t*)ptr;
	...
}
```

Tips

- Virtual COM on the Curiosity Nano is connected to USART3 which uses PORTB0 and PORTB1 – make sure nothing is connected to those pins
- You can use `scanf` for user input

```
int data;
scanf("%d", &data);
```

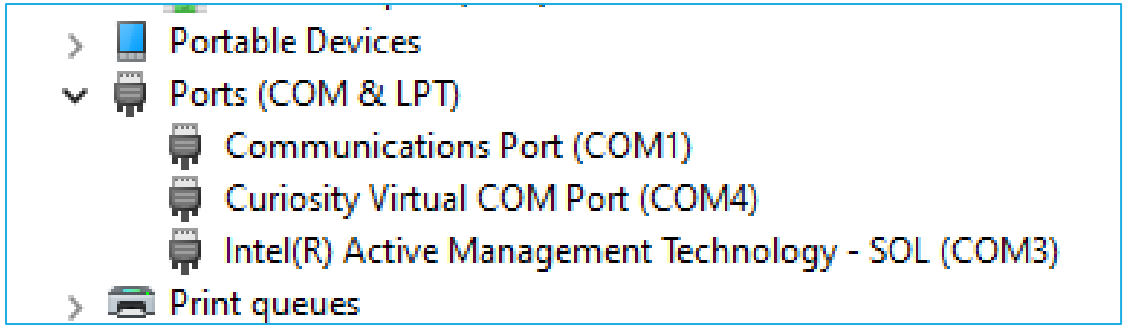
```
char str[10];
scanf("%9s", str);
```

- Make sure you check that the frequency and position are valid numbers
- Use a counter variable that keeps track of how many seconds have elapsed
  - When this counter reaches 5 seconds, you can print out the question on the serial port
  - You will increment this counter every time you go through the loop
  - You will need to increment this counter by the value of the LED period
- Another option is to count number of times through the loop
  - For example, at 1 HZ, you go through the loop 5 times; at 2 Hz, 10 times, etc.

UART Setup: COM Port Identification

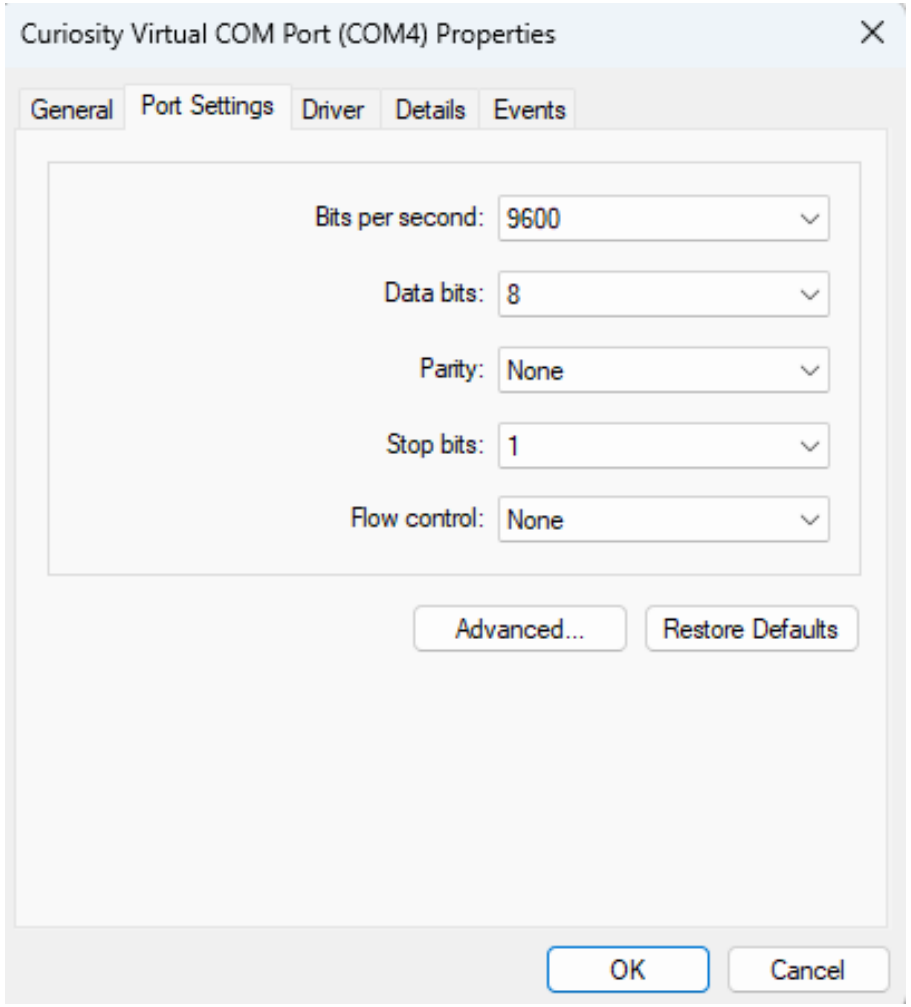
In order to setup UART communication between the Curiosity Nano and your PC, we first need to identify and setup the COM port used by Curiosity Nano board

- Connect the Curiosity Nano board to your computer via USB cable
- Go to: Control Panel → Device Manager
- Expand the Ports (COM & LPT) section as shown in the figure below.
- Note down the Port number shown for Curiosity Virtual COM Port, i.e. COM4 in the figure below.



UART Setup: COM Port Identification

- Double Click to open the Properties window of mEDBG Virtual COM Port.
- Make sure the Port Settings are the same as shown here.
- If necessary, the COM Port number can be changed under Advanced tab. However, generally the default COM Port number works just fine.



UART Setup: Putty

We will use PuTTY to send/receive data to the Curiosity Nano over UART

1. Download and Install PuTTY from Microsoft Store or from www.putty.org
2. In the PuTTY configuration window, select Serial Connection Type and type in the Curiosity Virtual COM Port number, e.g. COM4 (refer to the previous slide), as your Serial line.
3. Click Open to open up a terminal window