

graphic source: [http://exploreembedded.com/wiki/File:0\\_UART\\_main.gif](http://exploreembedded.com/wiki/File:0_UART_main.gif)

# Lecture 8.5

UART & Serial Port Programming (for debugging, communications, & occasional display)

ECE-231 Intro to Embedded Systems

Prof. David McLaughlin

ECE Dept

UMass Amherst

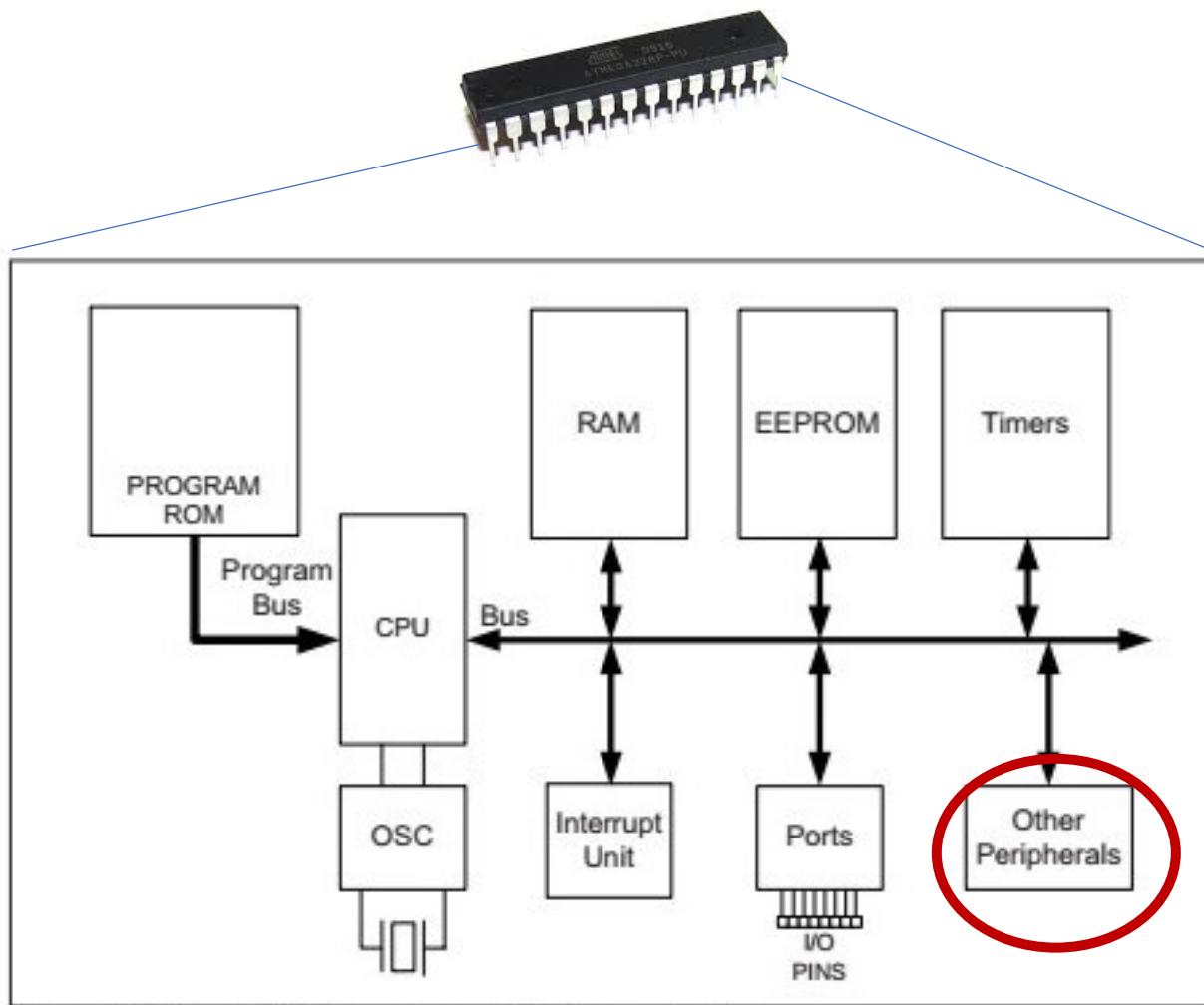
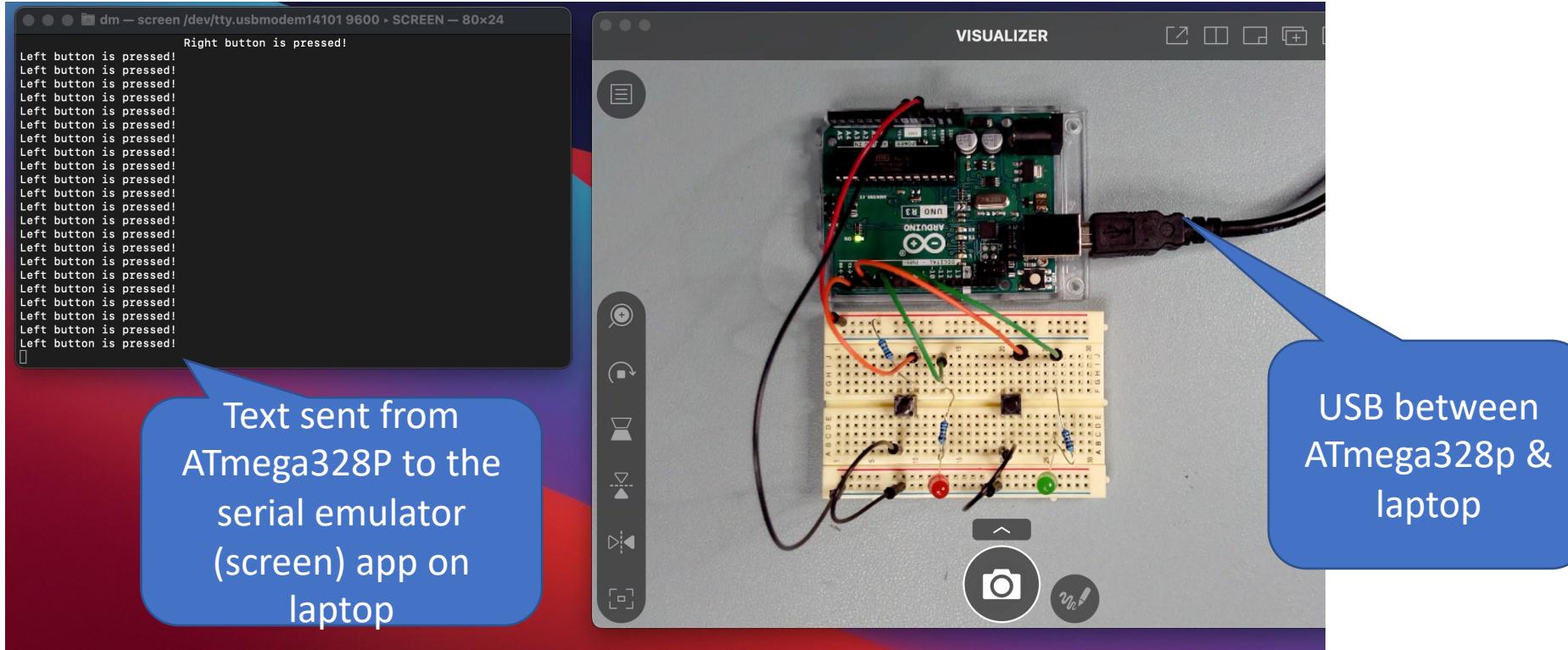


Figure 1-2. Simplified View of an AVR Microcontroller

Universal Synchronous/Asynchronous  
Receiver Transmitter peripheral  
(UART or USART)

# Demo: button status being sent to my laptop display as text messages.



We don't typically have a laptop display available as a display on an 8-bit embedded system. But it can be very helpful for debugging to be able to send text messages to a serial monitor display.

```
1  /* switches2.c This code turns on LEDs when momemtary pushbutton
2   switches are pressed. Input pins on PD3 and PD4. LEDS on PD6 on PD7
3   D. McLaughlin 2/21/22 */
4
5  #include <avr/io.h>
6  #include <util/delay.h>
7
8  int main(void){
9      DDRD = 1<<DDD6|1<<DDD7;           // Set D6, D7 as output
10     PORTD = 1<<PORTD3|1<<PORTD4;       // Set pullup on D3 & D4
11
12     while(1){
13
14         if  ((PIND & (1<<PIND3)) == 0){ // PIND3 is 0 when pressed
15             PORTD |= (1<<PORTD6);        // P6 high when P3 is pressed
16         } else {
17             PORTD &= ~(1<<PORTD6);       // P6 low when P3 not pressed
18         }
19
20         if  ((PIND & (1<<PIND4)) == 0){ // PIND4=0 when pressed
21             PORTD |= (1<<PORTD7);        // P7 high when P4 is pressed
22         } else {
23             PORTD &= ~(1<<PORTD7);       // P7 low when P4 not pressed
24         }
25
26     }
27     return(0);
28 }
29
```

```
1  /* switches_serial.c This code turns on LEDs when momentary pushbutton
2   switches are pressed. Input pins on PD3 and PD4. LEDS on PD6 on PD7
3   Code also sends text string to serial monitor via UART
4   D. McLaughlin 2/22/22 */
5
6  #include <avr/io.h>
7  #include <util/delay.h>
8  #include <string.h> //so that we can use the strlen() function
9  #include <stdlib.h> //need for itoa function
10 void uart_init(void);
11 void uart_send(unsigned char);
12 void send_string(char *stringAddress);
13
14 int main(void){
15     DDRD = 1<<DDD6|1<<DDD7;           // Set D6, D7 as output
16     PORTD = 1<<PORTD3|1<<PORTD4;      // Set pullup on D3 & D4
17     uart_init();                      // Initialize the UART
18     while(1{
19
20         if ((PIND & (1<<PIND3)) == 0){ // PIND3 is 0 when pressed
21             PORTD |= (1<<PORTD6);        // P6 high when P3 is pressed
22             send_string("Left button is pressed!\n");
23             uart_send(13);
24         } else {
25             PORTD &= ~(1<<PORTD6);       // P6 low when P3 not pressed
26         }
27
28         if ((PIND & (1<<PIND4)) == 0){ // PIND4=0 when pressed
29             PORTD |= (1<<PORTD7);        // P7 high when P4 is pressed
30             send_string("\t\t\tRight button is pressed!\n");
31             uart_send(13);
32         } else {
33             PORTD &= ~(1<<PORTD7);       // P7 low when P4 not pressed
34         }
35     }
36     return(0);
37 }
38
39 > void uart_init(void){...
40
41 > void uart_send(unsigned char ch){...
42
43 > void send_string(char *stringAddress){...
```

# Serial Comm & USART/UART

## Readings:

- <https://learn.sparkfun.com/tutorials/serial-communication>
- <https://learn.sparkfun.com/tutorials/terminal-basics/all>
- Textbook, Sections 11.1 – 11.4. OK to ignore 11.2. Don't be overly concerned about the details of 11.3. Try the examples in 11.4.
- ATmega328P data sheet Chapter 20
  - This is down-in-the-weeds details, but you do need practice reading data sheets.

# APPENDIX A

## ASCII CODES

Ctrl	Dec	Hex	Ch	Code
^@	0	00		NUL
^A	1	01	@	SOH
^B	2	02	@@	STX
^C	3	03	@@@	ETX
^D	4	04	@@@@	EOT
^E	5	05	@@@@@	ENQ
^F	6	06	@@@@@@	ACK
^G	7	07	@@@@@@@	BEL
^H	8	08	@@@@@@@@	BS
^I	9	09	@@@@@@@@@	HT
^J	10	0A	@@@@@@@@@@	LF
^K	11	0B	@@@@@@@@@@@	UT
^L	12	0C	@@@@@@@@@@@@	FF
^M	13	0D	@@@@@@@@@@@@@	CR
^N	14	0E	@@@@@@@@@@@@@@	SO
^O	15	0F	@@@@@@@@@@@@@@@	SI
^P	16	10	@@@@@@@@@@@@@@@@	DLE
^Q	17	11	@@@@@@@@@@@@@@@@@	DC1
^R	18	12	@@@@@@@@@@@@@@@@@@	DC2
^S	19	13	@@@@@@@@@@@@@@@@@@@	DC3
^T	20	14	@@@@@@@@@@@@@@@@@@@@	DC4
^U	21	15	@@@@@@@@@@@@@@@@@@@@@	NAK
^V	22	16	@@@@@@@@@@@@@@@@@@@@@@	SYN
^W	23	17	@@@@@@@@@@@@@@@@@@@@@@@	ETB
^X	24	18	@@@@@@@@@@@@@@@@@@@@@@@@	CAN
^Y	25	19	@@@@@@@@@@@@@@@@@@@@@@@@@	EM
^Z	26	1A	@@@@@@@@@@@@@@@@@@@@@@@@@@	SUB
^[_	27	1B	@@@@@@@@@@@@@@@@@@@@@@@@@@@	ESC
^`	28	1C	@@@@@@@@@@@@@@@@@@@@@@@@@@@@	FS
^]	29	1D	@@@@@@@@@@@@@@@@@@@@@@@@@@@@@	GS
^~	30	1E	@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@	RS
^_	31	1F	@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@	US

Dec	Hex	Ch
32	20	!
33	21	"
34	22	#
35	23	\$
36	24	%
37	25	>
38	26	&
39	27	'
40	28	<
41	29	>
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Ch
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

ASCII American Standard Code for Information Interchange, is a [character encoding](#) standard for electronic communication

'A' = 65 = 0x41 = 0b01000001

'B' = 66 = 0x42 = 0b01000010

'a' = 97 = 0x61 = 0b0110 0001

Carriage return = 13 = 0x0C = 0b00001101

'1' = 49 = 0x31 = 0b00110001

This is the standard coding for text messages.

128 standard ASCII symbols. Easy to represent using a 7 bit word ( $2^7=128$ )

# Extended ASCII

(256 different characters, 0-255; uses 8 bits)

Ctrl	Dec	Hex	Ch	Code	Dec	Hex	Ch															
^@	0	00	NUL		32	20	�	64	40	�	96	60	�	128	80	�	160	A0	�	192	C0	�
^A	1	01	�	SOH	33	21	�	65	41	�	97	61	�	129	81	�	161	A1	�	193	C1	�
^B	2	02	�	STX	34	22	�	66	42	�	98	62	�	130	82	�	162	A2	�	194	C2	�
^C	3	03	�	ETX	35	23	�	67	43	�	99	63	�	131	83	�	163	A3	�	195	C3	�
^D	4	04	�	EOI	36	24	�	68	44	�	100	64	�	132	84	�	164	A4	�	196	C4	�
^E	5	05	�	ENQ	37	25	�	69	45	�	101	65	�	133	85	�	165	A5	�	197	C5	�
^F	6	06	�	ACK	38	26	�	70	46	�	102	66	�	134	86	�	166	A6	�	198	C6	�
^G	7	07	�	BEL	39	27	�	71	47	�	103	67	�	135	87	�	167	A7	�	199	C7	�
^H	8	08	�	BS	40	28	�	72	48	�	104	68	�	136	88	�	168	A8	�	200	C8	�
^I	9	09	�	HT	41	29	�	73	49	�	105	69	�	137	89	�	169	A9	�	201	C9	�
^J	10	0A	�	LF	42	2A	�	74	4A	�	106	6A	�	138	8A	�	170	A0	�	202	C0	�
^K	11	0B	�	UT	43	2B	�	75	4B	�	107	6B	�	139	8B	�	171	A1	�	203	C1	�
^L	12	0C	�	FF	44	2C	�	76	4C	�	108	6C	�	140	8C	�	172	A2	�	204	C2	�
^M	13	0D	�	CR	45	2D	�	77	4D	�	109	6D	�	141	8D	�	173	A3	�	205	C3	�
^N	14	0E	�	SO	46	2E	�	78	4E	�	110	6E	�	142	8E	�	174	A4	�	206	C4	�
^O	15	0F	�	SI	47	2F	�	79	4F	�	111	6F	�	143	8F	�	175	A5	�	207	C5	�
^P	16	10	�	DLE	48	30	�	80	50	�	112	70	�	144	90	�	176	B0	�	208	D0	�
^Q	17	11	�	DC1	49	31	�	81	51	�	113	71	�	145	91	�	177	B1	�	209	D1	�
^R	18	12	�	DC2	50	32	�	82	52	�	114	72	�	146	92	�	178	B2	�	210	D2	�
^S	19	13	�	DC3	51	33	�	83	53	�	115	73	�	147	93	�	179	B3	�	211	D3	�
^T	20	14	�	DC4	52	34	�	84	54	�	116	74	�	148	94	�	180	B4	�	212	D4	�
^U	21	15	�	NAK	53	35	�	85	55	�	117	75	�	149	95	�	181	B5	�	213	D5	�
^V	22	16	�	SYN	54	36	�	86	56	�	118	76	�	150	96	�	182	B6	�	214	D6	�
^W	23	17	�	ETB	55	37	�	87	57	�	119	77	�	151	97	�	183	B7	�	215	D7	�
^X	24	18	�	CAN	56	38	�	88	58	�	120	78	�	152	98	�	184	B8	�	216	D8	�
^Y	25	19	�	EM	57	39	�	89	59	�	121	79	�	153	99	�	185	B9	�	217	D9	�
^Z	26	1A	�	SUB	58	3A	�	90	5A	�	122	7A	�	154	9A	�	186	BAA	�	218	DA	�
^_	27	1B	�	ESC	59	3B	;	91	5B	�	123	7B	�	155	9B	�	187	BB	�	219	DB	�
^`	28	1C	�	FS	60	3C	<	92	5C	�	124	7C	�	156	9C	�	188	BC	�	220	DC	�
^]	29	1D	�	GS	61	3D	=	93	5D	�	125	7D	�	157	9D	�	189	BD	�	221	DD	�
^~	30	1E	�	RS	62	3E	>	94	5E	�	126	7E	�	158	9E	�	190	BE	�	222	DE	�
^_	31	1F	�	US	63	3F	?	95	5F	�	127	7F	�	159	9F	�	191	BF	�	223	DF	�

## 3 Different Types of Communication between 2 devices

- Parallel communication
- Synchronous serial communication
- Asynchronous serial communication

# Synchronous parallel communication

Device sending data (transmitter)

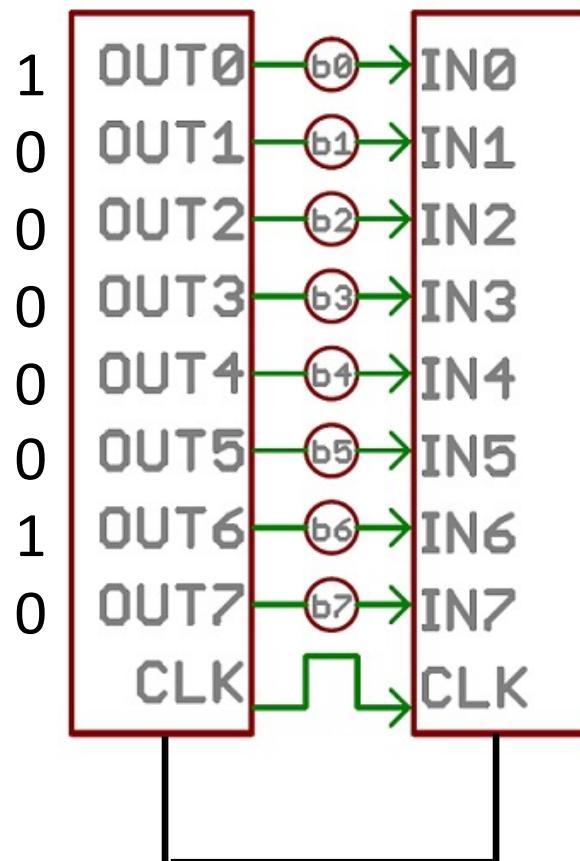
'A' = 01000001

## Protocol (agreement):

- 7 vs 8 bit ASCII
- transfer data on the rising edge of the clock (or the falling edge of the clock.)
- Hardware: 8 wires + clock + ground shared between devices

Device receiving data (transmitter)

'A' = 01000001

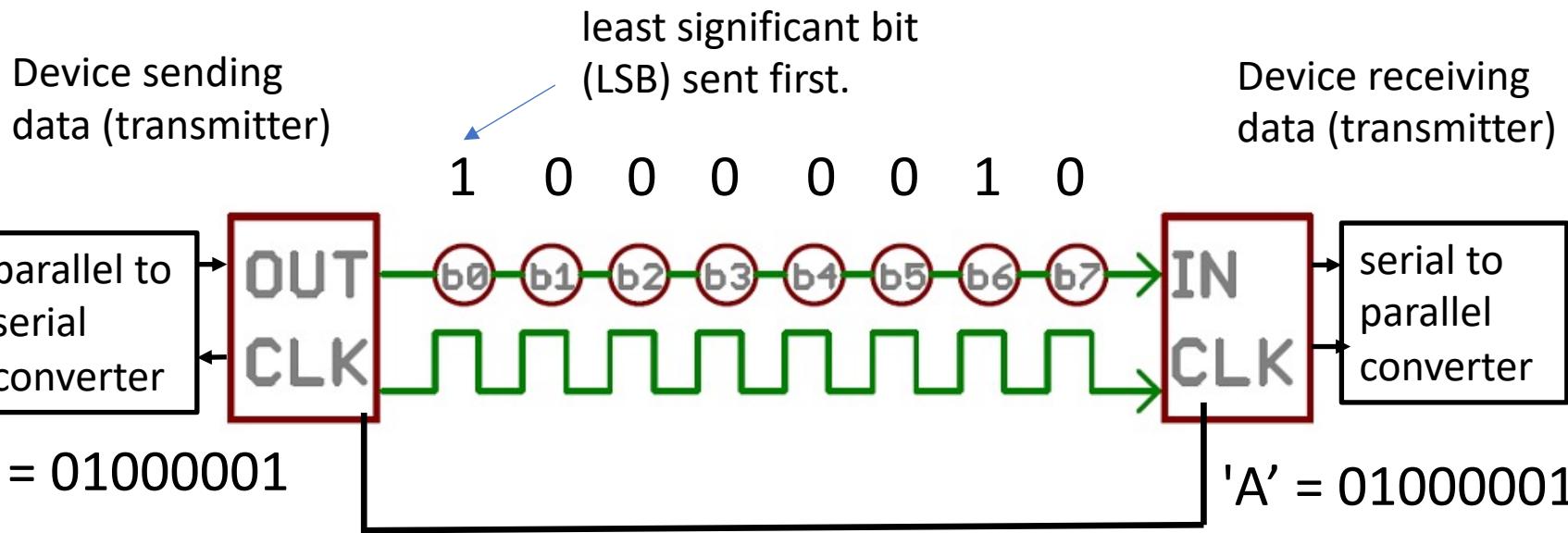


don't forget to connect the grounds!

Graphic from:

<https://learn.sparkfun.com/tutorials/serial-communication>

# Synchronous serial communication



## Protocol (agreement):

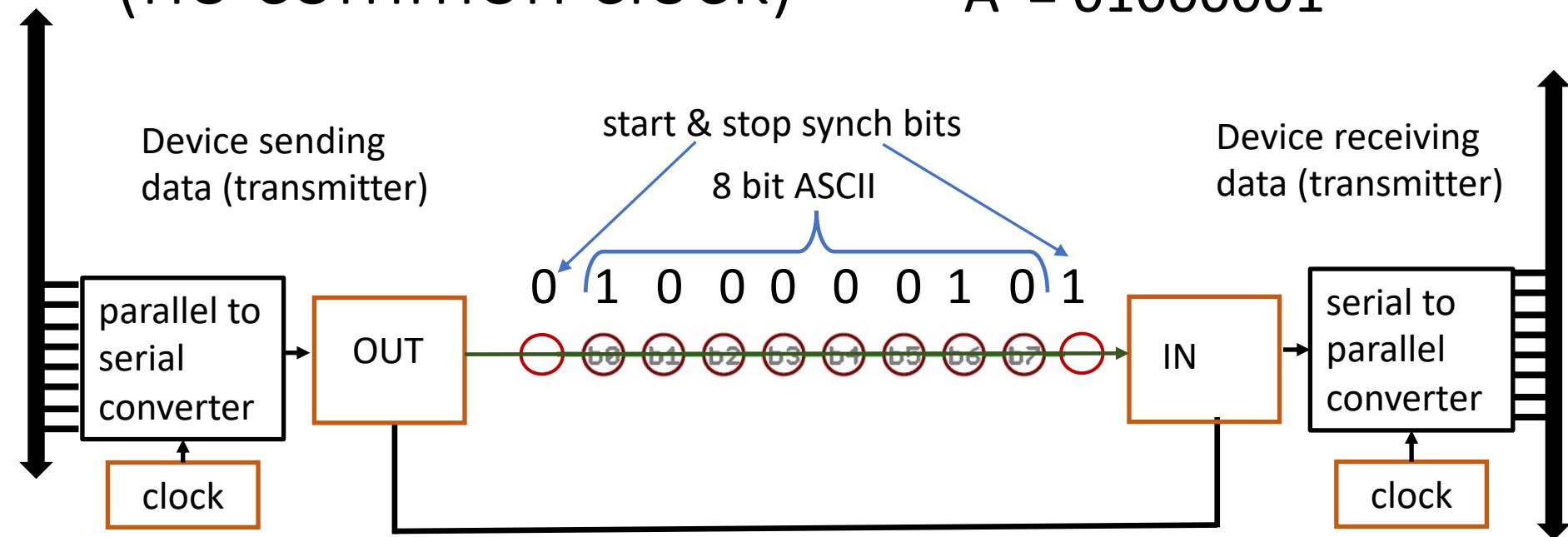
- 7 vs 8 bit ASCII
- transfer data on the rising edge of the clock (or the falling edge of the clock.)
- Hardware: 1 data line + clock + ground shared between devices

don't forget to connect the grounds!

Graphic from:  
<https://learn.sparkfun.com/tutorials/serial-communication>

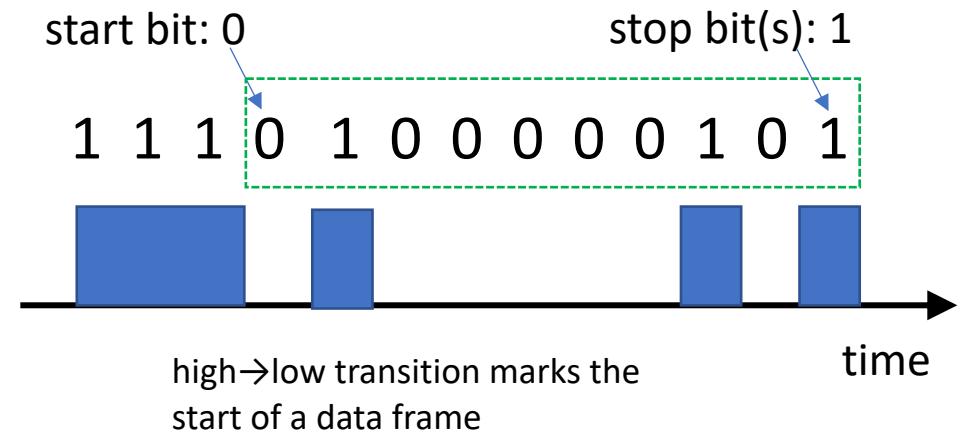
# asynchronous serial communication (no common clock)

'A' = 01000001

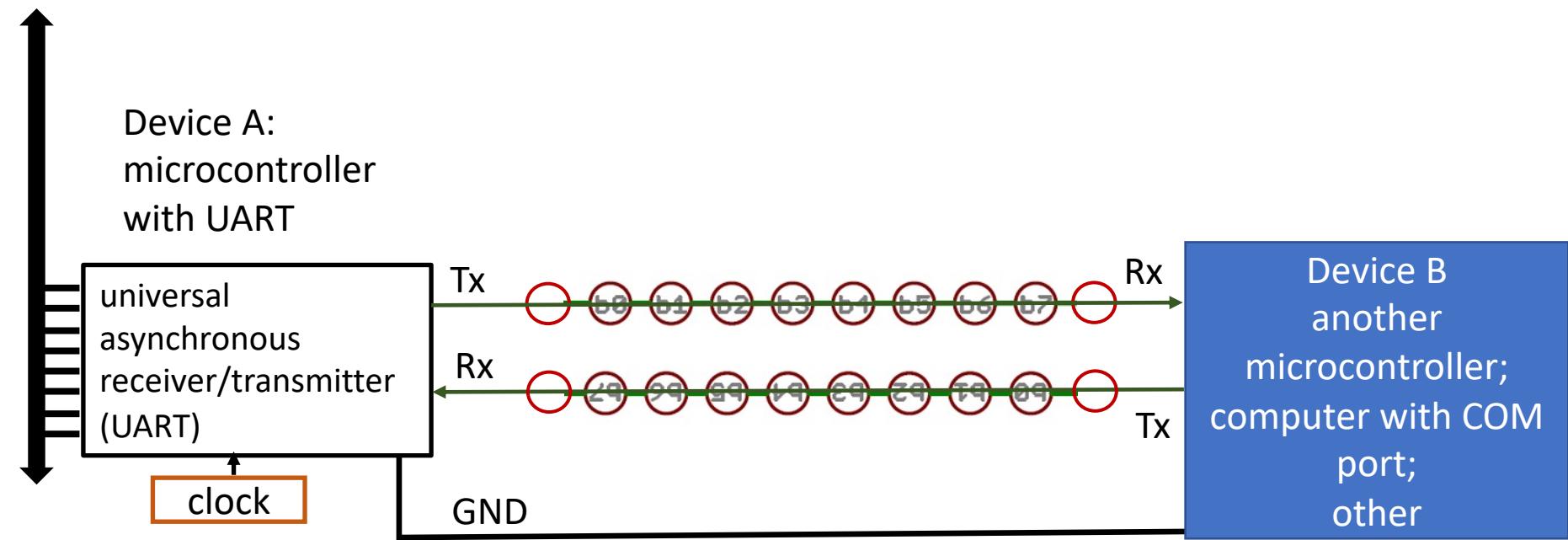


## Protocol (agreement):

- 7 vs 8 bit ASCII
- # synchronization bits
- 0 or 1 parity bits
- Hardware: 1 data line + ground shared between devices
- Data rate (bits/sec, baud)



# two way asynchronous serial communication (no common clock)



## Protocol (agreement):

- # ASCII bits (7-8)
- #stop bits (1, 2)
- Parity bit (0, 1)
- Hardware: 2 wires + ground shared between devices
- Data rate (bits/sec, baud)

# USART – Universal Synchronous Asynchronous Receiver Transmitter (synchronous less frequently used, so UART)

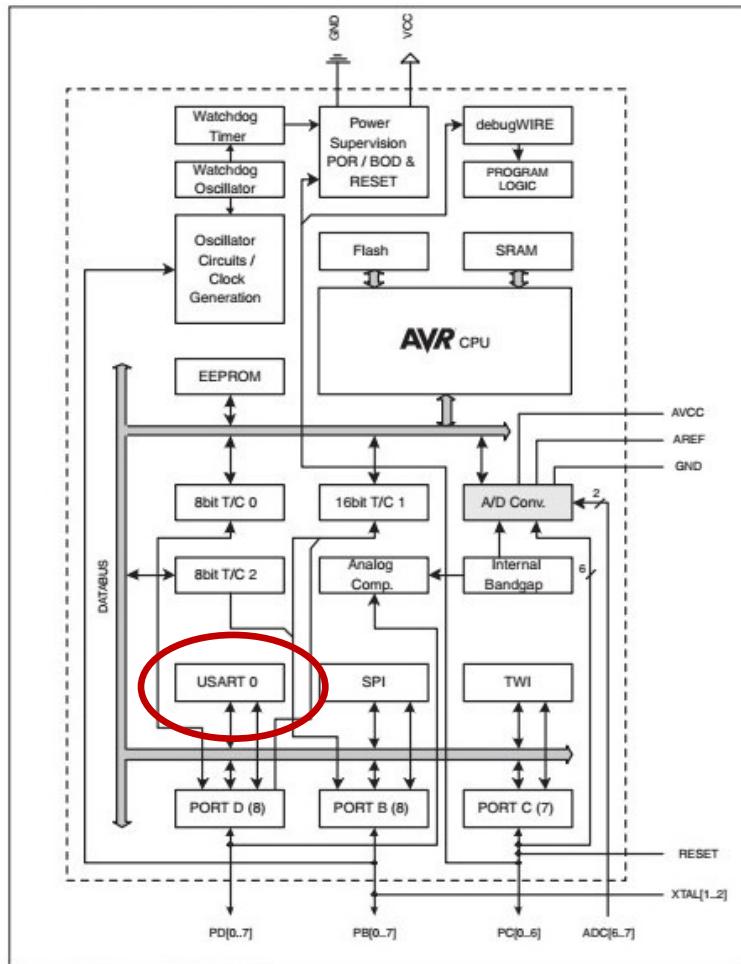


Figure 1-4. ATmega328 Block Diagram

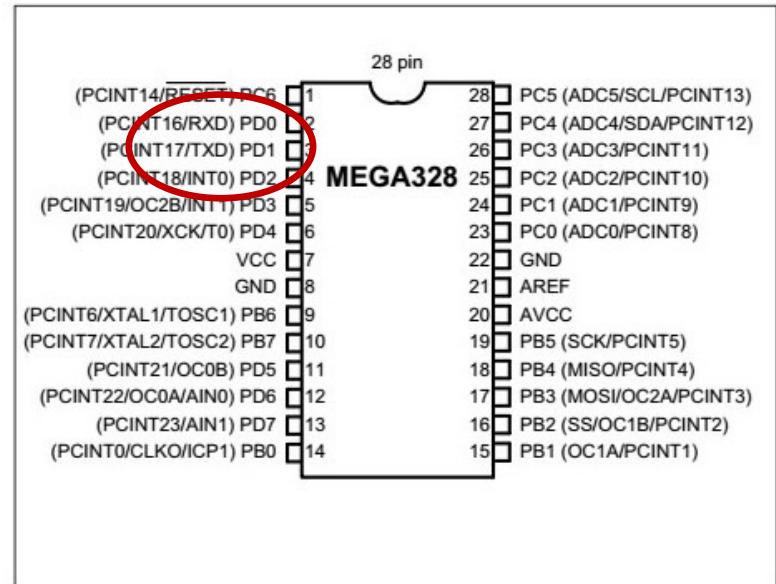
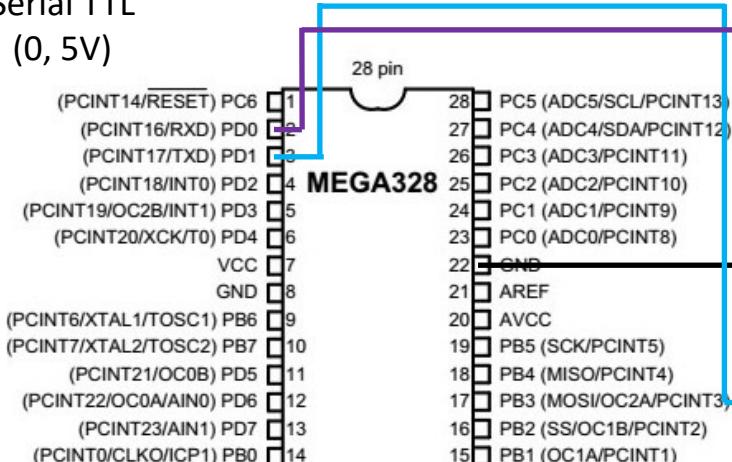


Figure 4-1. ATmega328 Pin Diagram

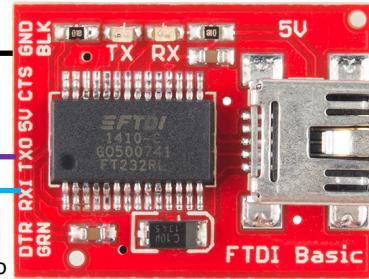
The USART is one of several peripherals built into the ATmega328P MCU

TXD, RXD  
Serial TTL  
(0, 5V)

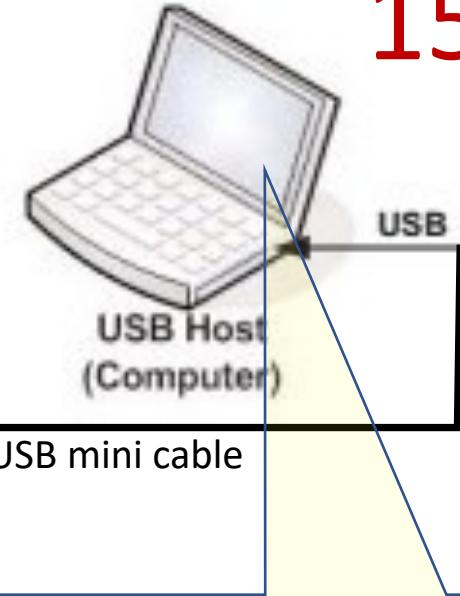


Note: TX0 to RXD; RX1 to TDX (cross the wires)

## Serial TTL to USB Converter



USB mini cable



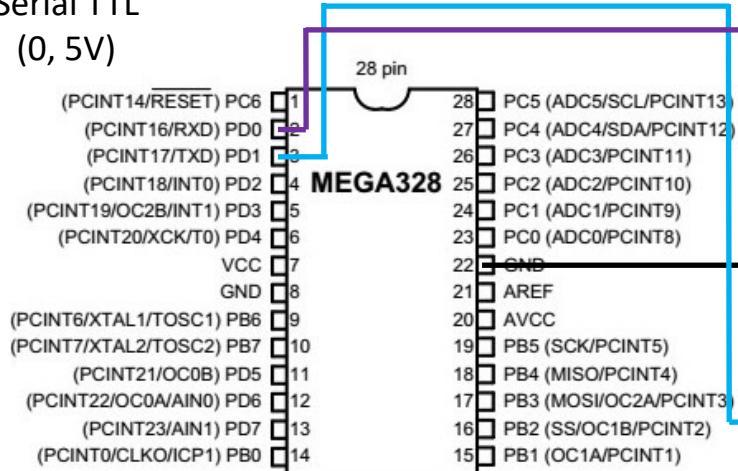
## Asynch Serial Comm between ATmega328P & USB host

- Hardware: Serial TTL to USB converter (FTDI interface)
- Terminal Emulator Software on Host Computer:
  - Windows -- use Putty (open source)
  - MacOS – use screen (UNIX command within MacOS terminal)
- Protocol: Configure the ATmega328P UART peripheral and the Host Computer Terminal Emulator (aka “Comm Software”) to same serial comm protocol elements
  - # bits (7 or 8, typically 8)
  - baud rate (bits/second) (1200, 9600, etc...)
  - synchronization: 1, 2 stop bits (typ. 1)
  - parity error checking: typically, none



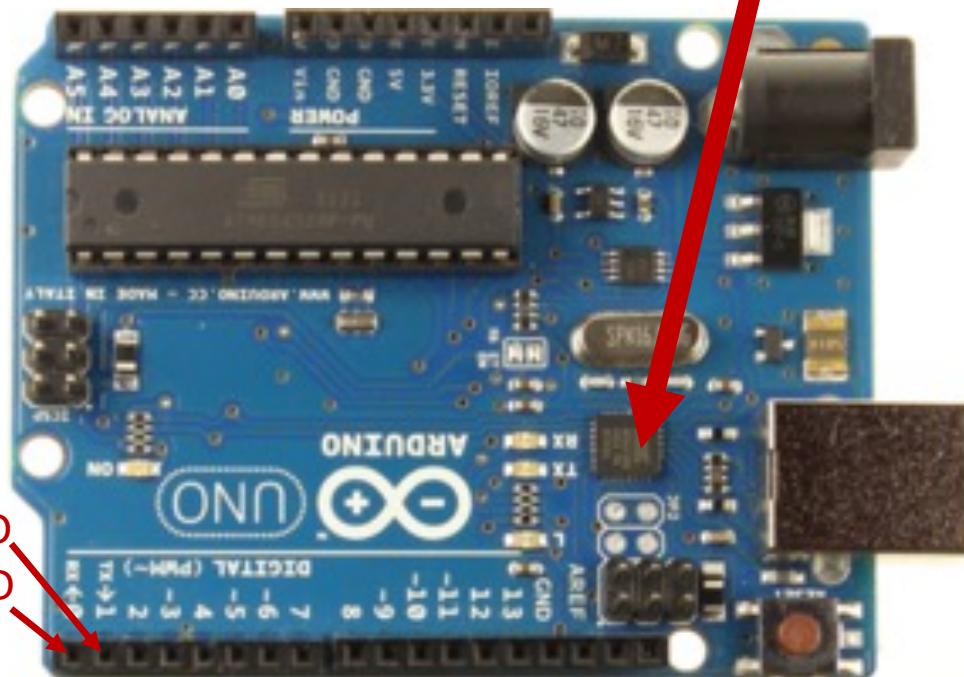
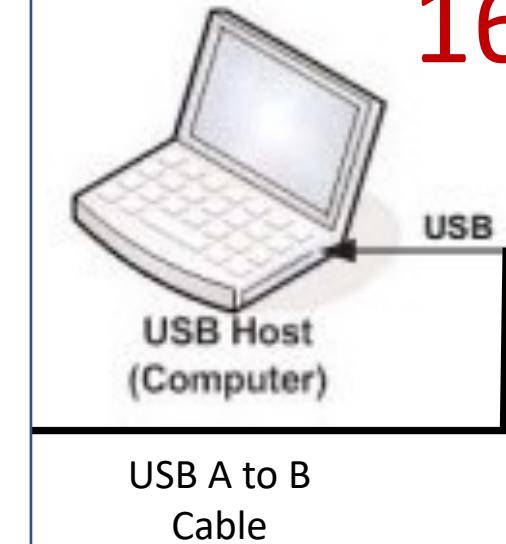
Run a Terminal Emulator App  
on the USB Host Computer  
(emulates VT100 or other  
video terminal)

TXD, RXD  
Serial TTL  
(0, 5V)



Arduino Uno Development Board

## Serial TTL to USB Converter

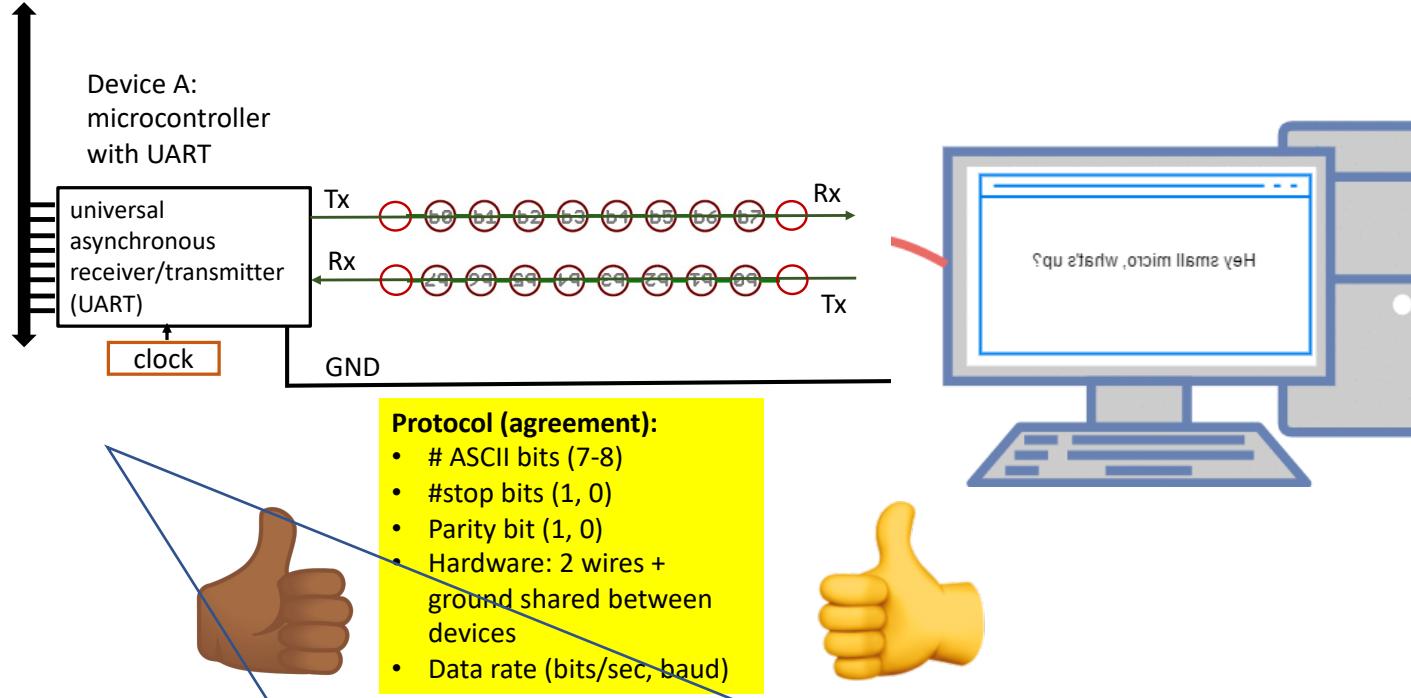


PD1, TXD  
PD0, RXD

Note:

- (1) The serial port on the ATmega328P uses PD0 & PD1 for RXD & TXD. Be aware of possible contention (conflict) if your code is using all of PORTD for one task while also engaging the UART.
- (2) The Arduino Uno has a pair of built-in LEDs, TX & RX, tied to PD1 & PD0

# Basic UART Idea:



- Enable UART
- Configure registers: UCSRA, UCSRB, USCRC, to set up the agreed-upon comm protocol: #data bits, #stop bits, use of parity bit
- Configure UBRR to set baud rate (# bits per second)
- Loop:
  - Load character byte to be transmitted into the UDR when this register is ready; repeat for additional characters to be sent
  - Internally, the UDR will shift the bits of the byte out, one-by-one and transmit them according to the agreed-upon protocol.

*UART Control & Status Registers A, B, C*



*UART Baud Rate & Data Register*

A common configuration 8 data bits; 1 stop bit; no parity checking; baud rate: 9600 bps

STEP 1.  
Set Transmit  
Enable in  
USCR0B

```
UCSR0B = 1<<TXEN0; //Enable Tx
```

STEP 2.  
Set configuration  
bits & baud rate

```
// Asych mode; 8 bit data; 1 stop bit  
// No parity checking  
UCSR0C = (1<< UCSZ01) | (1<<UCSZ00);  
// Divide 16MHz clock to give us 9600 bps  
UBRR0L = 103;
```

STEP 3.  
wait till tx data buffer  
is empty then

```
// Wait till Tx data buffer is empty  
while (( UCSR0A & (1<<UDRE0) )==0);  
or  
while (! ( UCSR0A & (1<<UDRE0) ));
```

STEP 4. send character

```
UDR0 = 'G'; // Send the letter 'G'
```

Write a C program for the ATmega328P on the Arduino Uno to transfer the letter 'G' serially at 9600 baud, continuously.

Protocol: Use 8-bit data, 1 stop bit, no parity bits

Part (A) write all the code in a single main function.

Part (B) organize the code into 3 functions: uart\_init() which initializes the UART baud rate and frame size, uart\_send() which sends the character, and main().

Part (C) Instead of sending 'G' send the letters 'WOW' repeatedly with carriage return & line feed after each WOW.

Part (D) Modify the code to send the following string of text  
"Shall I compare thee to a summer's day?"

**C serialnofunctions\_main.c > ...**

```
1  ****
2  * serialnofunctions_main.c - this program continuously sends
3  * the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
4  *
5  * Revision history:
6  * Date      Author      Revision
7  * 1/22/21    D. McLaughlin  Written using MPLABX for ECE-231 Spring 2021
8  * 12/29/21   D. McLaughlin  Improved comment style for ECE-231 Spring 2022
9  * 1/15/22    D. McLaughlin  using UART instead of USART functions
10 * 2/26/22   D. McLaughlin  everything in main instead of in functions
11 ****
12
13 #include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...
14
15 int main(void){
16
17     // Initialize the UART
18     UCSR0B = (1<<TXEN0);           // Enable Tx
19     UCSR0C = (1<< UCSZ01)|(1<<UCSZ00); // 8 ASCII bits, 1 stop bit, no parity
20     UBRR0L = 103;                  // Gives us 9600 baud from 16 MHz clock
21
22     while (1) {
23
24         while (! (UCSR0A&(1<<UDRE0))){ // Wait until Tx buffer is empty
25             ;
26         }
27
28         UDR0='G';                   // Transmit the ASCII character
29     }
30 }
```

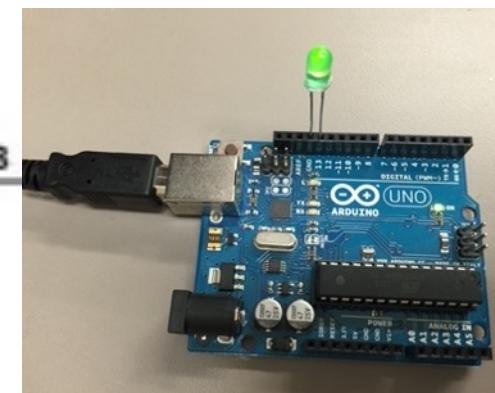
I am emulating (imitating) the function of an old-school VT100 terminal



*You need terminal emulator software to view the text data sent to your usb host by the ATmega328p UART ...*



USB Host  
(Computer)



# Windows Users

22



PuTTY (an open source serial comm program for Windows; TTY = “TeleTYpewriter”)

download from here...

<https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

helpful config instructions...

<https://pbxbook.com/voip/sputty.html>

Use built-in linux commands in Terminal ...

<https://geekinc.ca/using-screen-as-a-serial-terminal-on-mac-os-x/>

ls /dev/tty.\*

screen /dev/tty.usbserial-FTT3JMUZ 9600

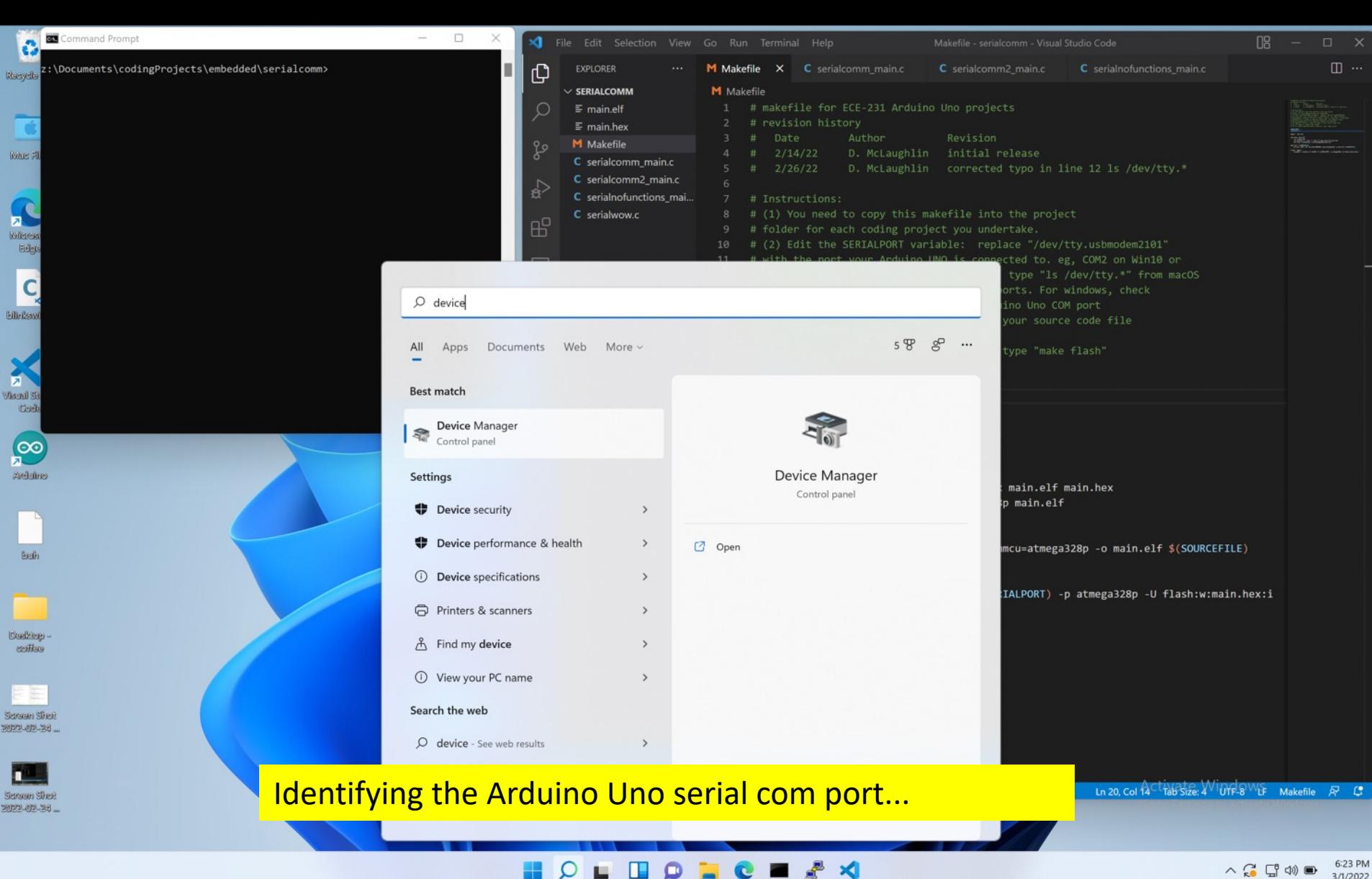
ctrl+a then k then y to detach the connection

# macOS & Linux Users



# Windows 11 Example

1. Make sure you have downloaded PuTTY using url from previous slide
2. Plug the Arduino Uno into an USB port
3. Identify the Arduino Uno Com port using the Windows Device Manager (use this port both for makefile and for serial comm using PuTTY)
4. Flash the code to the ATmega328p on the Arduino Uno using “make flash”
5. Launch PuTTY. Configure protocol: 8 data bits, 1 stop bit, no parity bits, 9600 baud
6. Observe “GGGGGGG....” on the display.
7. For fun: change ‘G’ to ‘H’ in the source code; re-compile & flash using: make, make flash. Do do this:
  - a) make. To be sure the code compiles correctly.
  - b) Close Putty. (Otherwise PuTTY’s serial comm connection will tie up the com port to the Arduino, preventing make flash from working.)
  - c) make flash. To download the modified code
  - d) Observe “HHHHH....” on the display



The screenshot shows a Windows desktop environment. On the left, the Start menu is open, displaying various icons for apps like Command Prompt, Device Manager, File Explorer, and Microsoft Edge. The main window is split into two parts: the left side shows the Windows Device Manager interface with a tree view of system components, and the right side shows a Visual Studio Code editor with an open 'Makefile' tab.

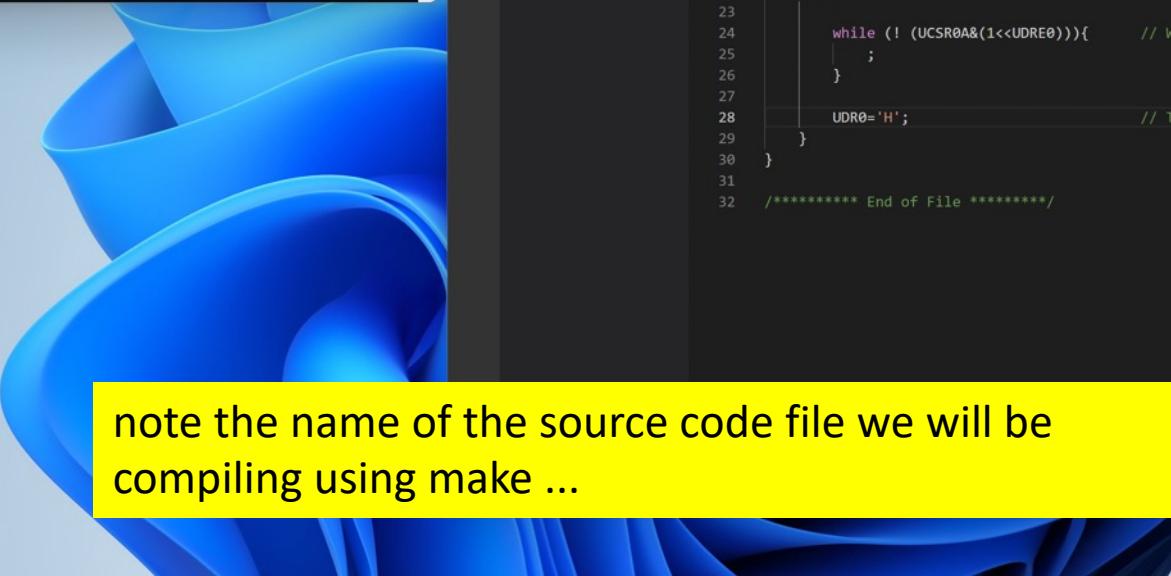
**Device Manager:**

- DAVE/MCLAUGH352D
  - Audio inputs and outputs
  - Batteries
  - Cameras
  - Computer
  - Disk drives
  - Display adapters
  - DVD/CD-ROM drives
  - Human Interface Devices
  - IDE ATA/ATAPI controllers
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - Ports (COM & LPT)
    - USB Serial Device (COM3) ←
  - Print queues
  - Processors
  - Security devices
  - Sensors
  - Software devices
  - Sound, video and game controllers
  - Storage controllers
  - System devices
  - Universal Serial Bus controllers

**Visual Studio Code - Makefile:**

```
1 # makefile for ECE-231 Arduino Uno projects
2 # revision history
3 # Date Author Revision
4 # 2/14/22 D. McLaughlin initial release
5 # 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/tty.*
6
7 # Instructions:
8 # (1) You need to copy this makefile into the project
9 # folder for each coding project you undertake.
10 # (2) Edit the SERIALPORT variable: replace "/dev/tty.usbmodem2101"
11 # with the port your Arduino UNO is connected to. eg, COM2 on Win10 or
12 # usbmodemxyz on macOS. If you're unsure, type "ls /dev/tty.*" from macOS
13 # Terminal for a listing of devices and ports. For windows, check
14 # Device Manager>Ports to find your Arduino Uno COM port
15 # (3) Replace "blink.c" with the name of your source code file
16 # (4) To compile, simply type "make"
17 # (5) To flash compiled code to Arduino, type "make flash"
18
19 SERIALPORT =
20 SOURCEFILE =
21
22 begin: main.hex
23
24 main.hex: main.elf
25 rm -f main.elf
26 avr-objcopy -j .text -j .data -O ihex main.elf main.hex
27 avr-size --format=avr --mcu=atmega328p main.elf
28
29 main.elf: $(SOURCEFILE)
30 avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf $(SOURCEFILE)
31
32 flash: begin
33 | avrdude -c arduino -b 115200 -P $(SERIALPORT) -p atmega328p -U flash:w:main.hex:i
```

**Yellow Callout:** Identifying the Arduino Uno serial com port...



z:\Documents\codingProjects\embedded\serialcomm>

File Edit Selection View Go Run Terminal Help

serialnofunctions\_main.c - serialcomm - Visual Studio Code

EXPLORER Makefile serialcomm\_main.c serialcomm2\_main.c serialnofunctions\_main.c

SERIALC... main.elf main.hex Makefile serialcomm\_main.c serialcomm2\_main.c serialnofunctions\_main.c serialwow.c

```
1 //*****
2 * serialnofunctions_main.c
3 * the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
4 *
5 * Revision history:
6 * Date Author Revision
7 * 1/22/21 D. McLaughlin Written using MPLABX for ECE-231 Spring 2021
8 * 12/29/21 D. McLaughlin Improved comment style for ECE-231 Spring 2022
9 * 1/15/22 D. McLaughlin using UART instead of USART functions
10 * 2/26/22 D. McLaughlin everything in main instead of in functions
11 ****
12
13 #include <avr/io.h> // Defines constants USCR0B, USCR0C, etc...
14
15 int main(void){
16
17     // Initialize the UART
18     UCSR0B = (1<<TXEN0); // Enable Tx
19     UCSR0C = (1<<UCSZ01)|(1<<UCSZ00); // 8 ASCII bits, 1 stop bit, no parity
20     UBRRL0L = 103; // Gives us 9600 baud from 16 MHz clock
21
22     while (1) {
23
24         while (! (UCSR0A&(1<<UDRE0))){ // Wait until Tx buffer is empty
25             ;
26         }
27
28         UDR0='H'; // Transmit the ASCII character
29     }
30
31 } //***** End of File *****/
32
```

note the name of the source code file we will be compiling using make ...

Activate Windows

Ln 28, Col 16 Tab Size: 4 UTF-8 LF C Win32

6:23 PM 3/1/2022

The screenshot shows a Windows desktop environment with Visual Studio Code open. On the left is a terminal window displaying a command-line session for building and flashing an AVR project. On the right is the Visual Studio Code interface with an Explorer sidebar, a Makefile tab, and a code editor tab for 'serialnofunctions\_main.c'.

```

z:\Documents\codingProjects\embedded\serialcomm>make
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf serialnofunctions_main.c
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf
AVR Memory Usage
Mac File
Device: atmega328p
Program: 166 bytes (0.5% Full)
(.text + .data + .bootloader)
MicrosData: 0 bytes (0.0% Full)
Edge(.data + .bss + .noinit)

z:\Documents\codingProjects\embedded\serialcomm>make flash
avrduude -c arduino -b 115200 -P com3 -p atmega328p -U flash:w:main.hex:i
avrduude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.02s
avrduude: Device signature = 0x1e950f
avrduude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrduude: erasing chip
avrduude: reading input file "main.hex"
avrduude: writing flash (166 bytes):
Writing | ##### | 100% 0.05s
avrduude: 166 bytes of flash written
avrduude: verifying flash memory against main.hex:
avrduude: load data flash data from input file main.hex:
avrduude: input file main.hex contains 166 bytes
avrduude: reading on-chip flash data:
Reading | ##### | 100% 0.03s
avrduude: verifying ...
avrduude: 166 bytes of flash verified
avrduude: safemode: Fuses OK
avrduude done. Thank you.

z:\Documents\codingProjects\embedded\serialcomm>

```

**Makefile Content:**

```

# makefile for ECE-231 Arduino Uno projects
# revision history
# Date Author Revision
# 2/14/22 D. McLaughlin initial release
# 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/tty.*
# Instructions:
# (1) You need to copy this makefile into the project
# folder for each coding project you undertake.
# (2) Edit the SERIALPORT variable: replace "/dev/tty.usbmodem2101"
# with the port your Arduino UNO is connected to. eg, COM2 on Win10 or
# usbmodemxyz on macOS. If you're unsure, type "ls /dev/tty.*" from macos
# Terminal for a listing of devices and ports. For windows, check
# Device Manager>Ports to find your Arduino Uno COM port
# (3) Replace "blink.c" with the name of your source code file
# (4) To compile, simply type "make"
# (5) To flash compiled code to Arduino, type "make flash"

SILENTPORT = com3
SOURCEFILE = serialnofunctions_main.c

begin: main.hex

main.hex: main.elf
    rm -f main.hex
    avr-objcopy -j .text -j .data -O ihex main.elf main.hex
    avr-size --format=avr --mcu=atmega328p main.elf

main.elf: $(SOURCEFILE)
    avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf $(SOURCEFILE)

flash: begin
    avrduude -c arduino -b 115200 -P $(SILENTPORT) -p atmega328p -U flash:w:main.hex:i

```

Two red arrows point to the 'SILENTPORT' variable assignment and the 'SOURCEFILE' variable assignment in the Makefile.

Modifying makefile: SERIALPORT & SOURCEFILE  
**make ---** to compile the source code  
**make flash** – to dowload the resulting hex file to the ATmega328P on the Arduino Uno

```

Select Command Prompt
z:\Documents\codingProjects\embedded\serialcomm>make
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf serialnofunctions_main.c
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf
AVR Memory Usage
Mac File
Device: atmega328p
Program: 166 bytes (0.5% Full)
(.text + .data + .bootloader)
MicrosData: 0 bytes (0.0% Full)
Edge(.data + .bss + .noinit)

z:\Documents\codingProjects\embedded\serialcomm>make flash
avrduude -c arduino -b 115200 -P com3 -p atmega328p -U flash:w:main.hex
avrduude: AVR device initialized and ready to accept instructions
Reading | #####| 100%
avrduude: Device signature = 0x1e950f
avrduude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrduude: erasing chip
avrduude: reading input file "main.hex"
avrduude: writing flash (166 bytes):
Writing | #####| 100%
avrduude: 166 bytes of flash written
avrduude: verifying flash memory against main.hex:
avrduude: load data flash data from input file main.hex:
avrduude: input file main.hex contains 166 bytes
avrduude: reading on-chip flash data:
Reading | #####| 100%
avrduude: verifying ...
avrduude: 166 bytes of flash verified
avrduude: safemode: Fuses OK
avrduude done. Thank you.

z:\Documents\codingProjects\embedded\serialcomm>

```

File Edit Selection View Go Run Terminal Help Makefile - serialcomm - Visual Studio Code

EXPLORER SERIALCOMM Makefile serialcomm\_main.c serialcomm2\_main.c serialnofunctions\_main.c

**Makefile**

```

1 # makefile for ECE-231 Arduino Uno projects
2 # revision history
3 # Date Author Revision
4 # 2/14/22 D. McLaughlin initial release
5 # 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/tty.*
6
7 # Instructions:
8 # (1) You need to copy this makefile into the project
9 # folder for each coding project you undertake.
10 # (2) Edit the SERIALPORT variable: replace "/dev/tty.usbmodem2101"
11 # with the port your Arduino UNO is connected to. eg, COM2 on Win10 or

```

Type here to search

All apps

< Back

- Paint
- Parallels Shared Applications
- Photos
- PuTTY (64-bit)
- Pageant
- PSFTP
- PuTTY
- PuTTY Manual
- PuTTY Web Site
- PuTTYgen
- S
- Settings
- Snipping Tool

davemclaughlin

a -O ihex main.elf main.hex  
atmega328p main.elf  
  
00000 -mmcu=atmega328p -o main.elf \$(SOURCEFILE)  
  
-P \$(SERIALPORT) -p atmega328p -U flash:w:main.hex:i

Activate Windows

Ln 20, Col 36 Tab Size: 4 UTF-8 LF Makefile

Open PuTTY...

6:25 PM 3/1/2022

z:\Documents\codingProjects\embedded\serialcomm>

PutTY Configuration

Category: Session

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM3 (highlighted by a red arrow)

Speed: 115200

Connection type: SSH (radio button) Serial (radio button, highlighted by a red arrow) Other: Telnet

Load, save or delete a stored session

Saved Sessions: Default Settings ArduinoUno

Close window on exit: Always (radio button) Never (radio button) Only on clean exit

Open Cancel

File Edit Selection View Go Run Terminal Help Makefile - serialcomm - Visual Studio Code

EXPLORER

Makefile

main.elf  
main.hex  
Makefile  
serialcomm\_main.c  
serialcomm2\_main.c  
serialnofunctions\_main.c  
serialwow.c

```

1 # makefile for ECE-231 Arduino Uno projects
2 # revision history
3 # Date Author Revision
4 # 2/14/22 D. McLaughlin initial release
5 # 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/tty.*
6
7 # Instructions:
8 # (1) You need to copy this makefile into the project
9 # folder for each coding project you undertake.
10 # (2) Edit the SERIALPORT variable: replace "/dev/tty.usbmodem2101"
11 # with the port your Arduino UNO is connected to. eg, COM2 on Win10 or
12 # usbmodemxyz on macOS. If you're unsure, type "ls /dev/tty.*" from macos
13 # Terminal for a listing of devices and ports. For windows, check
14 # Device Manager>Ports to find your Arduino Uno COM port
15 # (3) Replace "blink.c" with the name of your source code file
16 # (4) To compile, simply type "make"
17 # (5) To flash compiled code to Arduino, type "make flash"
18
19 SERIALPORT = com3
20 SOURCEFILE = serialnofunctions_main.c
21
22 begin: main.hex
23
24 main.hex: main.elf
25 rm -f main.elf
26 avr-objcopy -j .text -j .data -O ihex main.elf main.hex
27 avr-size --format=avr --mcu=atmega328p main.elf
28
29 main.elf: $(SOURCEFILE)
30 avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf $(SOURCEFILE)
31
32 flash: begin
33     avrdude -c arduino -b 115200 -P $(SERIALPORT) -p atmega328p -U flash:w:main.hex:i
34

```

Activate Windows

Ln 20, Col 36 Tab Size: 4 UTF-8 LF Makefile

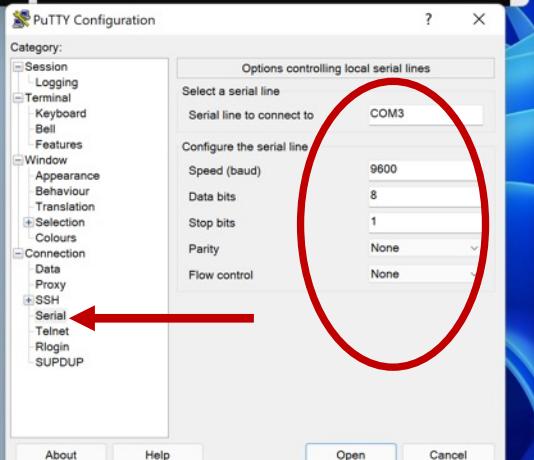
6:26 PM 3/1/2022

```
Select Command Prompt
avrduke: reading input file "main.hex"
avrduke: writing flash (166 bytes):
Writing | ##### | 100% 0.05s

avrduke: 166 bytes of flash written
avrduke: verifying flash memory against main.hex:
avrduke: load data flash data from input file main.hex:
avrduke: input file main.hex contains 166 bytes
avrduke: reading on-chip flash data:
Reading | ##### | 100% 0.03s

avrduke: verifying ...
avrduke: 166 bytes of flash verified
avrduke: safemode: Fuses OK
avrduke done. Thank you.

z:\Documents\codingProjects\embedded\serialcomm>
```



```
File Edit Selection View Go Run Terminal Help Makefile - serialcomm - Visual Studio Code

EXPLORER SERIALCOMM Makefile serialcomm_main.c serialcomm2_main.c serialnofunctions_main.c
M Makefile
1 # makefile for ECE-231 Arduino Uno projects
2 # revision history
3 # Date Author Revision
4 # 2/14/22 D. McLaughlin initial release
5 # 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/tty.*
6
7 # Instructions:
8 # (1) You need to copy this makefile into the project
9 # folder for each coding project you undertake.
10 # (2) Edit the SERIALPORT variable: replace "/dev/tty.usbmodem2101"
11 # with the port your Arduino UNO is connected to. eg, COM2 on Win10 or
12 # usbmodemxyz on macOS. If you're unsure, type "ls /dev/tty.*" from macOS
13 # Terminal for a listing of devices and ports. For windows, check
14 # Device Manager>Ports to find your Arduino Uno COM port
15 # (3) Replace "blink.c" with the name of your source code file
16 # (4) To compile, simply type "make"
17 # (5) To flash compiled code to Arduino, type "make flash"

SERIALPORT = com3
SOURCEFILE = serialnofunctions_main.c

begin: main.hex

main.hex: main.elf
rm -f main.elf
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf

main.elf: $(SOURCEFILE)
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf $(SOURCEFILE)

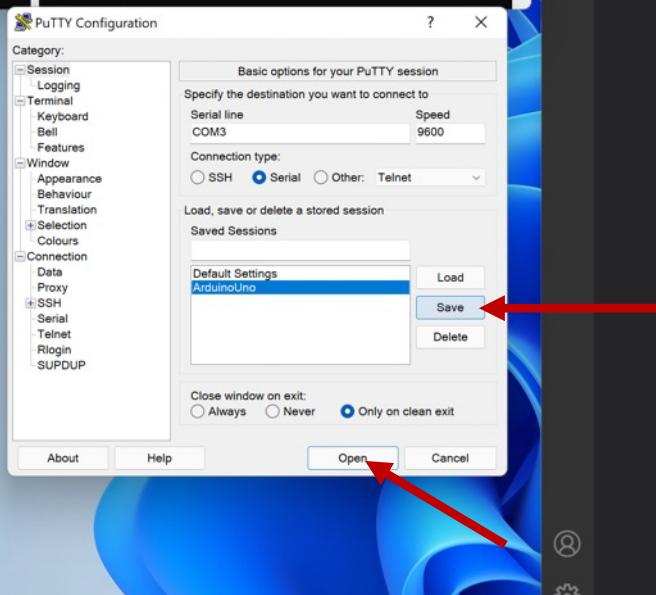
flash: begin
avrduke -c arduino -b 115200 -P $(SERIALPORT) -p atmega328p -U flash:w:main.hex:i
```

make sure PuTTY has the correct protocol details ...

Ln 20, Col 36 Tab Size: 4 UTF-8 LF Makefile

```
z:\Documents\codingProjects\embedded\serialcomm>avrdude -v -p m328p -c usbtiny -P COM3 -b 115200 -U flash:w:main.hex
avrdude: reading input file "main.hex"
avrdude: writing flash (166 bytes):
Writing | #####| ################################################## | 100% 0.05s
avrdude: 166 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 166 bytes
avrdude: reading on-chip flash data:
Reading | #####| ################################################## | 100% 0.03s
avrdude: verifying ...
avrdude: 166 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.

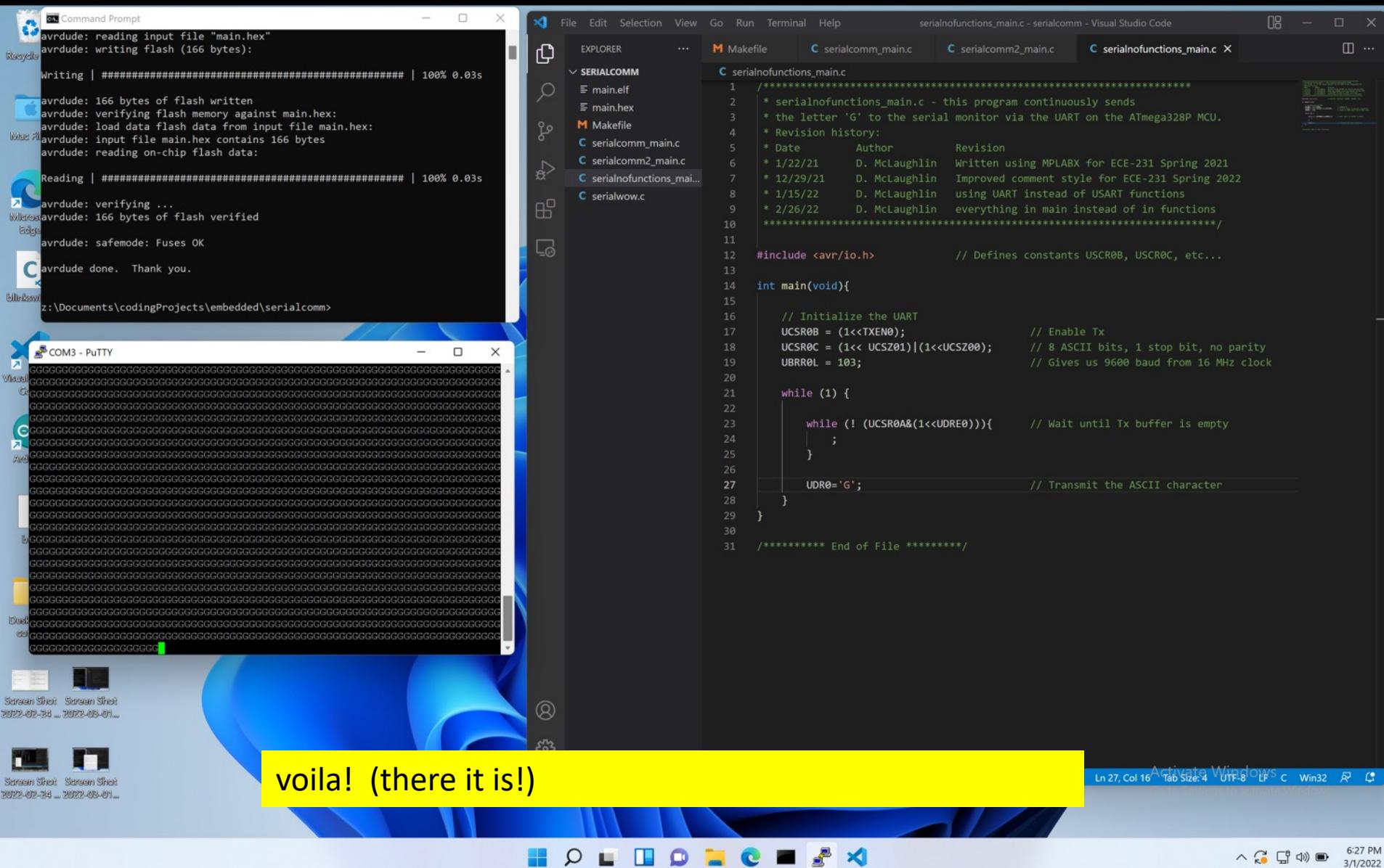
z:\Documents\codingProjects\embedded\serialcomm>
```

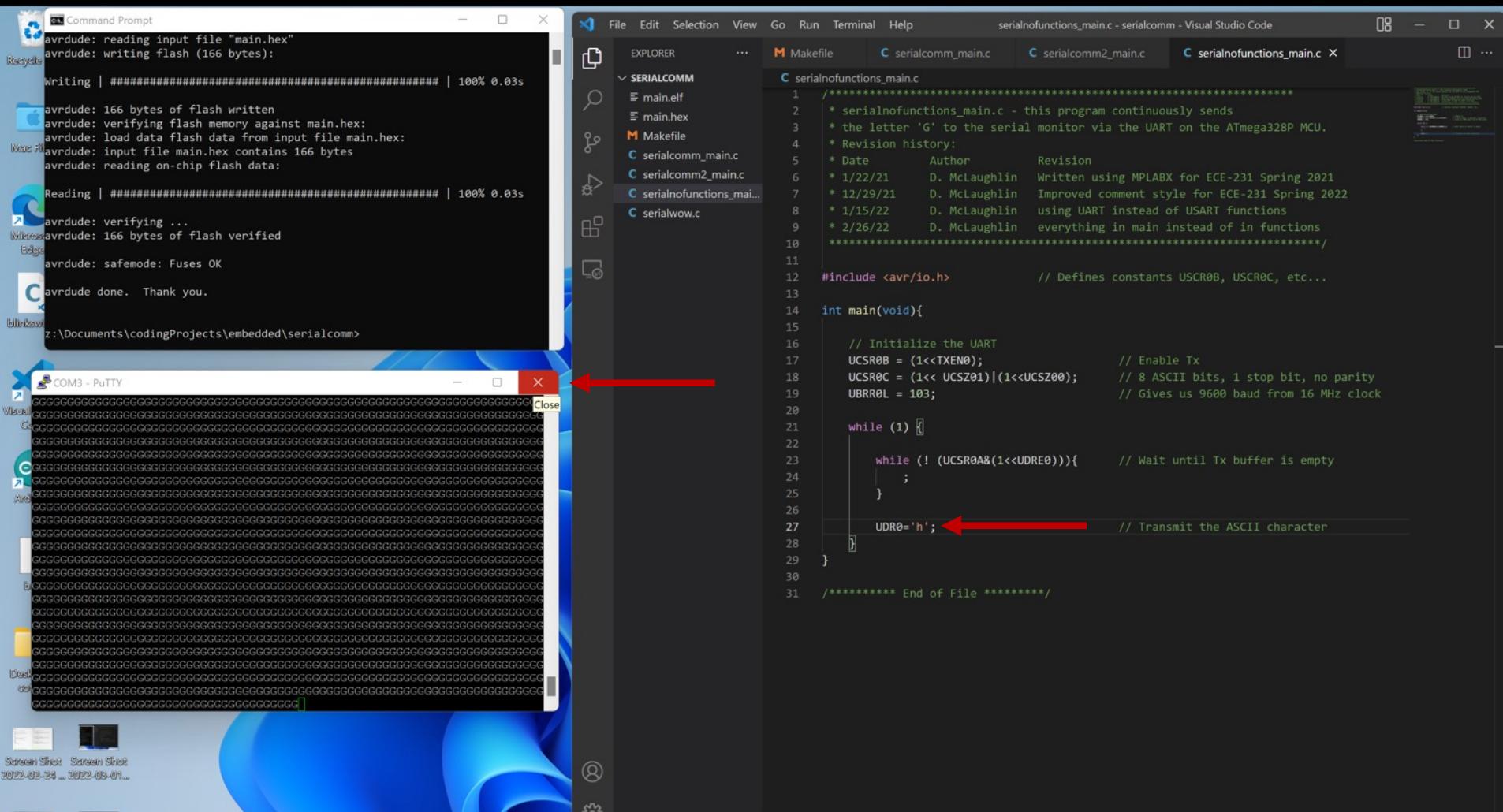


Save this configuration for later use  
Now open up the port ...

The screenshot shows the Visual Studio Code interface with the following details:

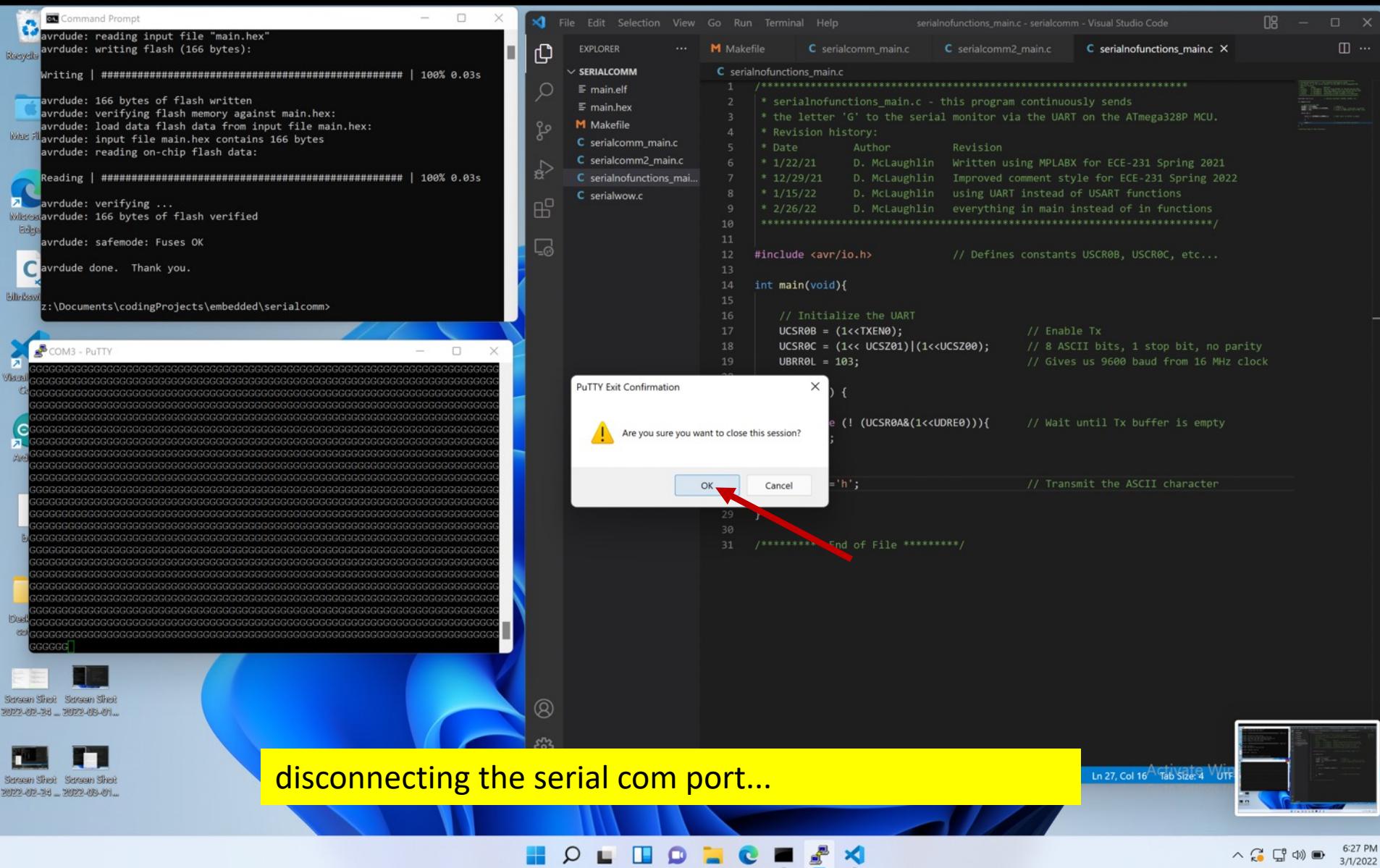
- File Explorer:** Shows a folder named "SERIALCOMM" containing files: main.elf, main.hex, Makefile, serialcomm\_main.c, serialcomm2\_main.c, serialnofunctions\_main.c, and serialwow.c. The "Makefile" file is currently selected.
- Editor:** Displays the content of the "Makefile" file. The code defines a makefile for ECE-231 Arduino Uno projects, specifying SERIALPORT as com3 and SOURCEFILE as serialnofunctions\_main.c. It includes rules for generating hex files from elf files using avr-objcopy and avr-size, and for compiling the source file with AVR-GCC. A flash rule uses avrdude to program an Atmega328P at 115200 baud via the specified port.
- Terminal:** A small terminal window is visible in the bottom right corner, showing the command "make flash" being run.

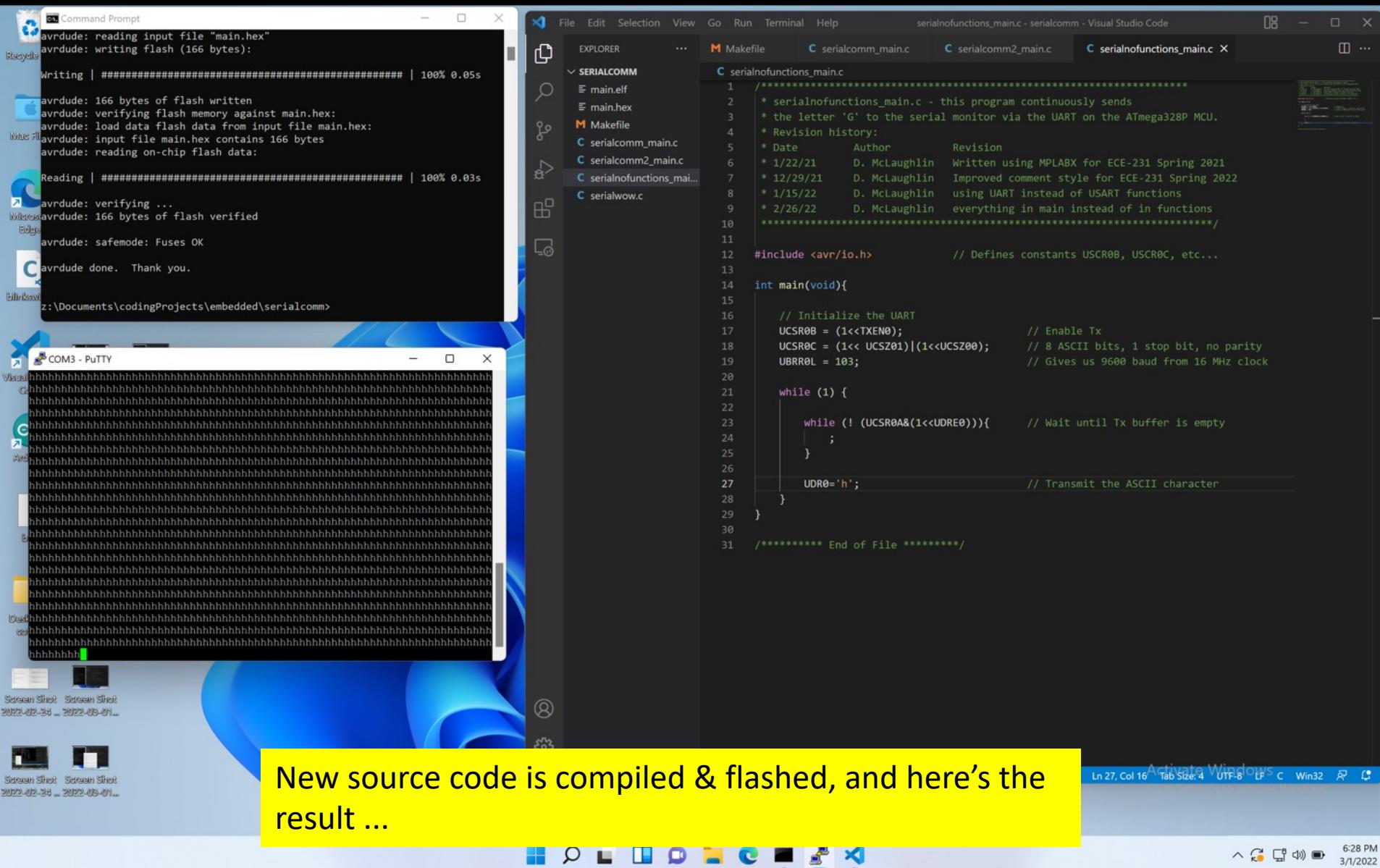




Now we modify the source code a bit.

We need to close the serial comm session in order to use the USB/serial line to flash the ATmega328P ...





The screenshot shows a Windows desktop environment with several windows open:

- Command Prompt:** Shows the output of a `make flash` command for an AVR project. It includes compilation steps like `avr-gcc` and `avr-objcopy`, and a memory usage check for the ATmega328P.
- Visual Studio Code:** An IDE window showing the file structure of a project named "SERIALCOMM". The "serialfunctions\_main.c" file is open, displaying C code for a serial communication application. A red arrow points to the line `UDR0='P';` which is highlighted in yellow.
- Putty:** A terminal window titled "COM3 - Putty" showing a continuous stream of 'h' characters, indicating data being transmitted from the serial port.

See what happens if we modify the source code then try to re-flash without disconnecting the puTTY terminal ...

Activate Windows  
Ln 27, Col 16 Tab Size: 4 UTF-8 LF C Win32

Write a C program for the ATmega328P on the Arduino Uno to transfer the letter 'G' serially at 9600 baud, continuously with 1/2 second delay between transmissions.

Protocol: Use 8-bit data, 1 stop bit, no parity bits

Part (A) write all the code in a single main function.

Part (B) organize the code into 3 functions: uart\_init() which initializes the UART baud rate and frame size, uart\_send() which sends the character, and main().

Part (C) Instead of sending 'G' send the letters 'WOW!' repeatedly with carriage return & line feed after each WOW.

Part (D) Modify the code to send the following string of text  
"Shall I compare thee to a summer's day?"

```
*****
* serialnofunctions_main.c - this program continuously sends
* the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
*
* Revision history:
* Date      Author      Revision
* 1/22/21    D. McLaughlin  Written using MPLABX for ECE-231 Spring 2021
* 12/29/21   D. McLaughlin  Improved comment style for ECE-231 Spring 2022
* 1/15/22    D. McLaughlin  using UART instead of USART functions
* 2/26/22    D. McLaughlin  everything in main instead of in functions
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...
```

```
int main(void){
```

```
    // Initialize the UART
    UCSR0B = (1<<TXEN0);           // Enable Tx
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00); // 8 ASCII bits, 1 stop bit, no parity
    UBRR0L = 103;                  // Gives us 9600 baud from 16 MHz clock
```

```
    while (1) {
```

```
        while (! (UCSR0A&(1<<UDRE0))){ // Wait until Tx buffer is empty
            ;
```

```
        }
```

```
        UDR0='G';                      // Transmit the ASCII character
```

```
    }  
***** End of File *****
```

```
*****
* serialcomm2_main.c - this program continuously sends
* the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
* Note that this version gives the user-defined function definitions
* up top.
*
* Revision history:
* Date      Author      Revision
* 3/1/22    D. McLaughlin  initial writing of code
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...
```

```
// This function initializes the UART peripheral
// enable; 8 data bits; 1 stop bit, no parity
// 9600 baud from 16 MHz clock
void uart_init(void) {
    UCSR0B = (1<<TXEN0);
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock
}
```

```
// This function sends a single character to the serial comm port
void send_char(char letter){
    while (! (UCSR0A&(1<<UDRE0))); // Wait until Tx buffer is empty
    UDR0=letter;
}
```

```
// Here is main
int main(void){                // Main function definition

    uart_init();                // Initialize the UART

    while (1) {
        send_char('G');         // Send over and over again
    }
}***** End of File *****
```

```
*****
* serialnofunctions_main.c - this program continuously sends
* the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
*
* Revision history:
* Date      Author      Revision
* 1/22/21    D. McLaughlin  Written using MPLABX for ECE-231 Spring 2021
* 12/29/21   D. McLaughlin  Improved comment style for ECE-231 Spring 2022
* 1/15/22    D. McLaughlin  using UART instead of USART functions
* 2/26/22    D. McLaughlin  everything in main instead of in functions
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...
```

```
int main(void){
```

```
    // Initialize the UART
    UCSR0B = (1<<TXEN0);           // Enable Tx
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00); // 8 ASCII bits, 1 stop bit, no parity
    UBRR0L = 103;                  // Gives us 9600 baud from 16 MHz clock
```

```
    while (1) {
```

```
        while (! (UCSR0A&(1<<UDRE0))){ // Wait until Tx buffer is empty
            ;
        }
```

```
        UDR0='G';                   // Transmit the ASCII character
    }
```

```
***** End of File *****
```

```
*****
* serialcomm2_main.c - this program continuously sends
* the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
* Note that this version gives the user-defined function definitions
* up top.
*
* Revision history:
* Date      Author      Revision
* 3/1/22    D. McLaughlin  initial writing of code
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...
```

```
// This function initializes the UART peripheral
// enable; 8 data bits; 1 stop bit, no parity
// 9600 baud from 16 MHz clock
void uart_init(void) {
    UCSR0B = (1<<TXEN0);
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
    UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock
}
```

```
// This function sends a single character to the serial comm port
void send_char(char letter){
    while (! (UCSR0A&(1<<UDRE0))); // Wait until Tx buffer is empty
    UDR0=letter;
}
```

```
// Here is main
int main(void){                // Main function definition
```

```
    uart_init();                // Initialize the UART
```

```
    while (1) {
        send_char('G');         // Send over and over again
    }
}
```

```
***** End of File *****
```

```
*****
 * serialcomm_main.c - this program continuously sends
 * the letter 'G' to the serial monitor via the UART on the ATmega328P MCU.
 * This version uses forward function declarations, main() defined up top.
 * Revision history:
 * Date      Author      Revision
 * 1/22/21    D. McLaughlin Written using MPLABX for ECE-231 Spring 2021
 * 12/29/21   D. McLaughlin Improved comment style for ECE-231 Spring 2022
 * 1/15/22    D. McLaughlin using UART instead of USART functions; split braces for structures
*****
```

```
#include <avr/io.h>          // Defines constants USCR0B, USCR0C, etc...

void uart_init(void);        // Function prototype (declaration)
void send_char(char);       // Function prototype (declaration)

int main(void){              // Main function definition
    uart_init();             // Initialize the UART

    while (1) {
        send_char('G');      // Send over and over again
    }
}

// This function initializes the UART peripheral
// enable; 8 data bits; 1 stop bit, no parity
// 9600 baud from 16 MHz clock
void uart_init(void) {
    UCSRB = (1<<TXEN0);
    UCSRC = (1<< UCSZ01)|(1<<UCSZ00);
    UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock
}

// This function sends a single character to the serial comm port
void send_char(char letter){
    while (! (UCSR0A&(1<<UDRE0))); // Wait until Tx buffer is empty
    UDR0=letter;
}
***** End of File *****
```

Putting main() up top and the user-defined functions below is a preferred programming style.

The user can quickly see what's happening in main() without needing to search for it

Write a C program for the ATmega328P on the Arduino Uno to transfer the letter 'G' serially at 9600 baud, continuously with 1/2 second delay between transmissions.

Protocol: Use 8-bit data, 1 stop bit, no parity bits

Part (A) write all the code in a single main function.

Part (B) organize the code into 3 functions: uart\_init() which initializes the UART baud rate and frame size, uart\_send() which sends the character, and main().

Part (C) Instead of sending 'G' send the letters 'WOW' repeatedly with carriage return & line feed after each WOW.

Part (D) Modify the code to send the following string of text  
"Shall I compare thee to a summer's day?"

serialwow.c &gt; ...

```
*****  
 * serialcomm_main.c - this program continuously sends  
 * the letter 'WOW!' to the serial monitor via the UART on the ATmega328P MCU.  
 * Date          Author      Revision  
 * 3/1/22        D. McLaughlin initial release. based on serialcomm_main.c  
*****  
  
#include <avr/io.h>           // Defines constants USCR0B, USCR0C, etc...  
  
void uart_init(void);          // Function prototype (declaration)  
void send_char(char);          // Function prototype (declaration)  
  
int main(void){                // Main function definition  
  
    uart_init();                // Initialize the UART  
  
    while (1) {                  // Send repeatedly  
        send_char('W');  
        send_char('o');  
        send_char('W');  
        send_char(10);            // Carriage Return  
        send_char(13);            // Line feed  
    }  
}  
  
// This function initializes the UART peripheral  
// enable; 8 data bits; 1 stop bit, no parity  
// 9600 baud from 16 MHz clock  
void uart_init(void) {  
    UCSR0B = (1<<TXEN0);  
    UCSR0C = (1<< UCSZ01)|(1<<UCSZ00);  
    UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock  
}  
  
// This function sends a single character to the serial comm port  
void send_char(char letter){  
    while (! (UCSR0A&(1<<UDRE0))); // Wait until Tx buffer is empty  
    UDR0=letter;  
}  
***** End of File *****
```

Easy enough to make this change when we only want to send a few characters to the terminal...

# This example done under macOS

The screenshot shows a macOS desktop environment with a terminal window and a code editor window.

**Terminal Window:**

```

serialcomm -- zsh - 70x64
davemclaughlin@wine serialcomm % rm main.*
davemclaughlin@wine serialcomm % clear
davemclaughlin@wine serialcomm % ls /dev/tty.*
/dev/ttys000
davemclaughlin@wine serialcomm % make
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf serial
main.c
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf
AVR Memory Usage
-----
Device: atmega328p

Program: 202 bytes (0.6% Full)
(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)

davemclaughlin@wine serialcomm % make flash
avrdude -c arduino -b 115200 -P /dev/ttys000 -p atmega328p -U flash:w:main.hex:i
avrdude: AVR device initialized and ready to accept instructions

Reading | 0% 0.00
Reading | ##### 100% 0.
0s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "Flash" memory has been specified, an erase cycle will
be performed
To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (202 bytes):
Writing | 0% 0.00
Writing | ##### 50% 0.0
Writing | ##### 100% 0.
0s

avrdude: 202 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 202 bytes
avrdude: reading on-chip flash data:
Reading | 0% 0.00
Reading | ##### 50% 0.0
Reading | ##### 100% 0.
0s

avrdude: verifying ...
avrdude: 202 bytes of flash verified
avrdude: safemode: Fuses OK (E:00, H:00, L:0)
avrdude done. Thank you.
davemclaughlin@wine serialcomm %

```

**Code Editor Window:**

The code editor shows a project structure in the Explorer panel and a Makefile in the main panel.

**Makefile Content:**

```

# makefile for ECE-231 Arduino Uno projects
# revision history
# Date Author Revision
# 2/14/22 D. McLaughlin initial release
# 2/26/22 D. McLaughlin corrected typo in line 12 ls /dev/ttys000

# Instructions:
# (1) You need to copy this makefile into the project
# folder for each coding project you undertake.
# (2) Edit the SERIALPORT variable: replace "/dev/ttys000" or
# ttys000 with the port your Arduino UNO is connected to. eg, COM2 on Win10 or
# usbmodemxyz on macOS. If you're unsure, type "ls /dev/ttys000" from macos
# Terminal for a listing of devices and ports. For windows, check
# Device Manager>Ports to find your Arduino Uno COM port
# (3) Replace "blink.c" with the name of your source code file
# (4) To compile, simply type "make"
# (5) To flash compiled code to Arduino, type "make flash"

SERIALPORT = /dev/ttys000
SOURCEFILE = serialwow.c

begin: main.hex

main.hex: main.elf
rm -f main.elf
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
avr-size --format=avr --mcu=atmega328p main.elf

main.elf: $(SOURCEFILE)
avr-gcc -Wall -Os -DF_CPU=16000000 -mmcu=atmega328p -o main.elf $(SOURCEFILE)

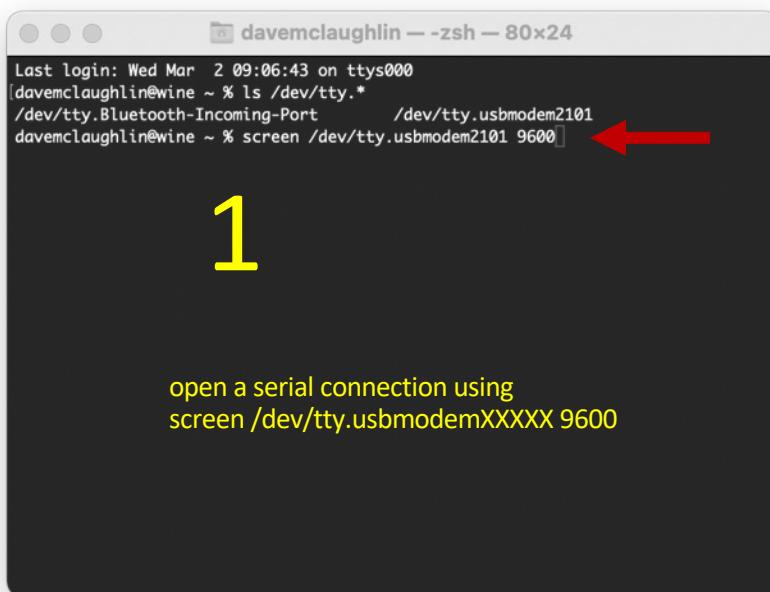
flash: begin
avrdude -c arduino -b 115200 -P $(SERIALPORT) -p atmega328p -U flash:w:main.hex:i

```

**Listed Steps:**

- (1) determine the serial connection using ls /dev/ttys000
- (2) modify Makefile
- (3) make
- (4) make flash

44



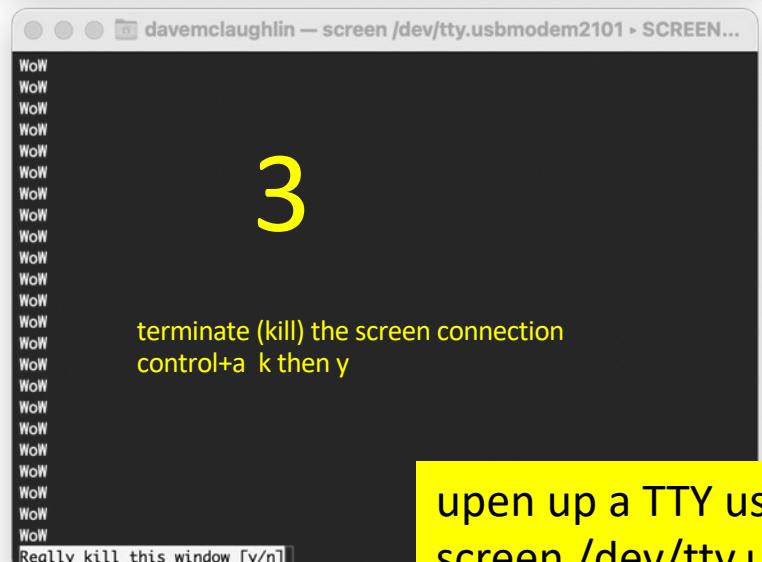
1

open a serial connection using  
screen /dev/tty.usbmodemXXXXX 9600



2

here is the result...



3

terminate (kill) the screen connection  
control+a k then y



4

Open up a TTY using the screen command:  
screen /dev/tty.usbmodem2101 9600

terminate (kill) the screen using control+A then k then y

Write a C program for the ATmega328P on the Arduino Uno to transfer the letter 'G' serially at 9600 baud, continuously with 1/2 second delay between transmissions.

45

Protocol: Use 8-bit data, 1 stop bit, no parity bits

Part (A) write all the code in a single main function.

Part (B) organize the code into 3 functions: uart\_init() which initializes the UART baud rate and frame size, uart\_send() which sends the character, and main().

Part (C) Instead of sending 'G' send the letters 'WOW' repeatedly with carriage return & line feed after each WOW.

Part (D) Modify the code to send the following string of text  
"Shall I compare thee to a summer's day?"

The screenshot shows a code editor window with the following details:

- EXPLORER** panel on the left showing project files: main.elf, main.hex, Makefile, serialcomm\_main.c, serialcomm2\_main.c, serialnofunctions\_ma..., and serialwow.c (2 changes).
- Makefile** tab in the top center.
- serialwow.c 2** tab in the top center.
- Code Editor**:
  - File: serialwow.c
  - Content:

```
1  ****
2  * serialcomm_main.c - this program continuously sends
3  * the letter 'WOW!' to the serial monitor via the UART on the ATmega328P MCU.
4  * Date      Author      Revision
5  * 3/1/22    D. McLaughlin  initial release. based on serialcomm_main.c
6  ****
7
8 #include <avr/io.h>           // Defines constants USCR0B, USCR0C, etc...
9
10 void uart_init(void);         // Function prototype (declaration)
11 void send_char(char);        // Function prototype (declaration)
12
13 int main(void){              // Main function definition
14     uart_init();              // Initialize the UART
15
16     while (1) {                // Send repeatedly
17         send_char('W');       // send character 'W'
18         send_char('O');       // send character 'O'
19         send_char('W');       // send character 'W'
20         send_char(10);        // Carriage return
21         send_char(13);        // Line feed
22     }                          // End of while loop
23 }
24
25 // This function initializes the UART peripheral
26 // enable; 8 data bits; 1 stop bit, no parity
27 // 9600 baud from 16 MHz clock
28 void uart_init(void) {         ...
29     UCSR0B = (1<<TXEN0);
30     UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
31     UBRR0L = 103; // Gives us 9600 baud from 16 MHz clock
32 }
```
- Bottom Status Bar**: Ln 34, Col 1 | Tab Size: 4 | UTF-8 | LF | C | Mac | ⌂ | ⌂

we can type “send\_char();” for every character, but that is obviously tedious, clumsy, inefficient. And impractical for longer messages.

## Arrays in C:

An array is a series of elements of one data type with common name

Array declaration: tells compiler how many elements array contains & the data type for these elements

examples of array declarations:

```
int candy[365];           // 365 ints (2 bytes; -32,768 to + 32,767)
char candy[100];          // 100 chars (1 byte each; -128 to +127)
unsigned char candy[100];  // 100 unsigned chars (values 0 to 255)
```

examples of declaration, initialization, assignments:

```
int a[5]={1, 32, 45, 5, 4};
char b[]={2,4,6,8,10};    // ok to leave out array size
a=b;                      // This matlab command NOT ALLOWED IN C
a[1]=b[3];                //OK
a[5] = 4;                 //NOT OK SINCE INDEX GOES FROM 0 to 4
for (int i=0; i<5; i++) {
    b[i]=a[i]*i;
}
```

## Characters in C:

Characters are just 1 byte ints

```
char thisletter = 'a';           // stored as 01000001 (ASCII for 'a')
char thisletter = 65;            // Same thing
char thisletter = 0x41;          // Same thing
char thisletter = 0b01000001     // Still the same
```

## Strings in C:

Strings are just arrays of chars

examples of declaration, initialization, assignments:

```
char candy[9] = { 'c', 'h', 'o', 'c', 'o', 'l', 'a', 't', 'e' };
char candy[9] = "chocolate";    // C lets us do this with strings
char candy[] = "chocolate";    // Compiler figures out size
char wow[] = "WOW!";
char mystring[] = "Shall I compare thee to a summer's day?";
```

## C uart\_string.c &gt; ...

```
1  *****/  
2  * uart_string.c - This code sends strings to com port via uart.  
3  * Date      Author      Revision  
4  * 12/16/21 D. McLaughlin Initial writing of the code  
5  * 1/15/22  D. McLaughlin Tested on Arduino Uno w/ Apple M1 pro  
6  *                      host running Monterey  
7  * 2/27/22  D. McLaughlin tested on Windows 11 on Parallels VM  
8  *****/  
9  #include <avr/io.h>  
10 #include <util/delay.h>  
11 #include <string.h> //so that we can use the strlen() function  
12  
13 void uart_init(void);  
14 void uart_send(unsigned char);  
15 void send_string(char *stringAddress);  
16  
17 int main(void){  
18     char mystring[] = "Shall I compare thee to a summer's day?";  
19     uart_init(); // initialize the USART  
20     while (1) {  
21         send_string(mystring);  
22         uart_send(13); // Carriage return (goto beginning of line)  
23         uart_send(10); //line feed (new line)  
24         _delay_ms(500);  
25     }  
26 }  
27  
28 // Send a string, char by char, to uart via uart_send()  
29 // Input is pointer to the string to be sent  
30 void send_string(char *stringAddress){  
31     for (unsigned char i = 0; i < strlen(stringAddress); i++)  
32         uart_send(stringAddress[i]);  
33 }  
34  
35 > void uart_init(void){...  
36  
37 > void uart_send(unsigned char ch){...  
38  
39  
40  
41  
42  
43  
44  
45  
46 **** End of file ****/
```

the name of the array without an index refers to the whole array. It is a pointer to the starting address of the array in memory



Write a C program for the ATmega328P on the Arduino Uno to transfer the letter 'G' serially at 9600 baud, continuously with 1/2 second delay between transmissions.

51

Protocol: Use 8-bit data, 1 stop bit, no parity bits

Part (E) Let's do this right: Create a user-defined library out of the 3 UART functions having an API (.h) and an implementation (.c) file. Create a main function to create a line of text.

```
de / uart_via_API > C dart_sourcecode.c > ...
✓ /*****t_sourcecode.c
Changed lines
    * Written 4/21/22 D.McLaughlin
    * here are the necessary includes and function prototypes
    * to create a simple library out of our 3 uart functions
    *****/
✓ #include <avr/io.h>
#include <string.h>
void uart_init(void);
void uart_send(unsigned char letter);
void send_string(char *stringAddress);

// here are the function definitions
✓ void send_string(char *stringAddress){
    for (unsigned char i = 0; i < strlen(stringAddress); i++)
        uart_send(stringAddress[i]);
}

✓ void uart_init(void){
    UCSR0B = (1 << TXEN0); //enable the UART transmitter
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //set 8 bit character size
    UBRR0L = 103; //set baud rate to 9600 for 16 MHz crystal
}

✓ void uart_send(unsigned char ch){
    while (!(UCSR0A & (1 << UDRE0))); //wait til tx data buffer empty
    UDR0 = ch; //write the character to the USART data register
```

# 2 parts to a user-defined library:

mylib.c

```
int function1(int){  
    instruction;  
    instruction;  
    instruction;  
    return (result);  
}
```

```
int function2(int){  
    instruction;  
    return (result);  
}
```

etc..

This is the Application Programming Interface(API). It tells YOU how to use the functions in the library

This is the implementation of the functions in the library

mylib.h

```
// This is what function1 does  
// Inputs are ...  
// Outputs are...  
int function1(int);
```

```
// This is what function2 does  
// Inputs are...  
// Outputs are...  
int function2(int);
```

etc..

```
// my_uart_lib.h
// header file to accompany my_uart_lib.c
// Version 1.0 4/21/22 D.McLaughlin
// Version 1.1 4/7/24 D. McLaughlin

#include <avr/io.h>
#include <string.h>

/* Initialize the UART: Enables the UART transmitter; Sets 8 bit character size
 * Sets baud rate to 9600 for 16 MHz crystal
 * Arguments: none
 * Returns: none */
void uart_init(void);

/* Transmit a single character via UART
 * Arguments:
 *      letter - ASCII character to be transmitted
 * Returns: none */
void uart_send(unsigned char letter);

/* Transmit a character string via UART.
 * Sends the string, char by char, to the UART
 * via uart_send()
 * Arguments:
 *      *stringAddress - pointer to the string
 * Returns: none */
void send_string(char *stringAddress);
```

```
/* my_uart_lib.c */

#include "my_uart_lib.h"

// Initialize the UART
void uart_init(void){
    UCSR0B = (1 << TXEN0); //enable the UART transmitter
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); //set 8 bit character size
    UBRR0L = 103; //set baud rate to 9600 for 16 MHz crystal
}

// Send a single character
void uart_send(unsigned char ch){
    while (!(UCSR0A & (1 << UDRE0))); //wait til tx data buffer empty
    UDR0 = ch; //write the character to the USART data register
}

// Send a string of characters using uart_send
void send_string(char *stringAddress){
    for (unsigned char i = 0; i < strlen(stringAddress); i++)
        uart_send(stringAddress[i]);
}
```

✓ /\*\*\*\*\*

Click to collapse the range. uart\_send.c - This code sends strings to com port via uart.

Version	Date	Author	Revision
* 1.0	12/16/21	D. McLaughlin	Initial writing of the code
* 1.1	1/15/22	D. McLaughlin	Tested on Arduino Uno w/ Apple M1 pro host running Monterey
* 1.2	2/27/22	D. McLaughlin	tested on Windows 11 on Parallels VM
* 1.3	4/21/22	D. McLaughlin	re-written using uart.c/uart.h API
* 2.0	4/7/24	D. McL	improved formatting

\*\*\*\*\*

// \_\_\_\_\_ preamble \_\_\_\_\_  
✓ #include <util/delay.h>  
#include "my\_uart\_lib.h"

✓ int main(void){  
  
 // \_\_\_\_\_ inits \_\_\_\_\_  
 char mystring[] = "This is pretty cool stuff!";  
 uart\_init(); // initialize the USART  
  
 // \_\_\_\_\_ event loop \_\_\_\_\_  
 while (1) {  
 send\_string(mystring);  
 uart\_send(13); // Carriage return (goto beginning of line)  
 uart\_send(10); //line feed (new line)  
 \_delay\_ms(50);  
 } // end of event loop  
}

\*\*\*\*\* End of file \*\*\*\*\*

# makefile V2.0. This is a makefile for AVR ATmega328P projects using Arduino Uno Dev bo  
# ECE-231 Spring 2024.

#  
# Instructions:  
# Put this file (makefile, no extension) into the source code folder for each  
# programming project. You should only need to change the SERIALPORT and SOURCEFILE.  
# Options for running this makefile from Command Prompt (Windows) or Terminal (macOS):  
# "make" compiles code & creates a hex file for uploading to the MCU  
# "make -B" will force compiling even if the source code hasn't changed  
# "make flash" will compile, create hex file, and upload the hex file to the MCU  
# "make clean" will delete intermediate files make.elf and make.hex

#\_\_\_\_\_ MODIFY SERIALPORT AND SOURCEFILE\_\_\_\_\_

# Specify the com port (windows) or USB port (macOS)  
# Use Device Manager to identify COM port number for Arduino Uno board in Windows  
# In Terminal, type ls /dev/tty.usb\* to determine USB port number in macOS

SERIALPORT = /dev/tty.usbmodem2101

# Specify the name of your source code here:

SOURCEFILE = uart\_string.c my\_uart\_lib.c

#\_\_\_\_\_

# Don't change anything below unless you know what you're doing....

CLOCKSPEED = 16000000

PROGRAMMER = arduino

begin: main.hex

main.hex: main.elf  
    rm -f main.hex  
    avr-objcopy -j .text -j .data -O ihex main.elf main.hex  
    avr-size --format=avr --mcu=atmega328p main.elf

For Windows users, to use PuTTY for your serial COM connections, follow these steps:

1. Figure out the COM port you'll be using.
2. Run PuTTY.
3. Switch the Connection Type to Serial.
4. Edit the Serial Line to match the COM port you want to use.
5. Edit the Speed to match the BAUD Rate you want to use.
6. Select the Serial category from the menu on the left.
7. Make sure all of the settings are correct (8 bit ASCII, 1 stop bit, etc....).
8. Select the Open button to start the session.

Your connection should now be fully functional.

For macOS users, to use screen for your serial COM connections, follow these steps:

1. Lanch the Terminal App
2. Determine your serial port identification:

```
ls /dev/tty.*
```

You will see a result something like: /dev/tty.usbmodemXYZ

3. Type the following to make a TTY connection

```
screen /dev/tty.usbmodemXYZ 9600
```

(XYZ refers to your device; 9600 is the baud rate)

Note that this will invoke screen with the default 8N1 protocol (8 data bits, no parity checking, 1 stop bit). This is the same as typing

```
screen /dev/tty.usbmodemXYZ 9600, cs8, -parenb, -stopb
```

4. To terminate the active screen, type CTRL+a then k (kill) then y