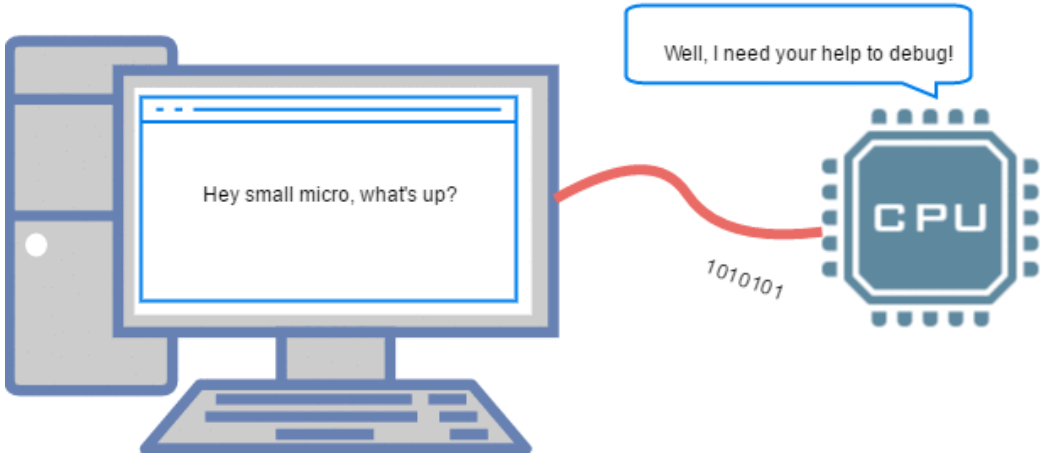


graphic source: http://exploreembedded.com/wiki/File:0_UART_main.gif

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

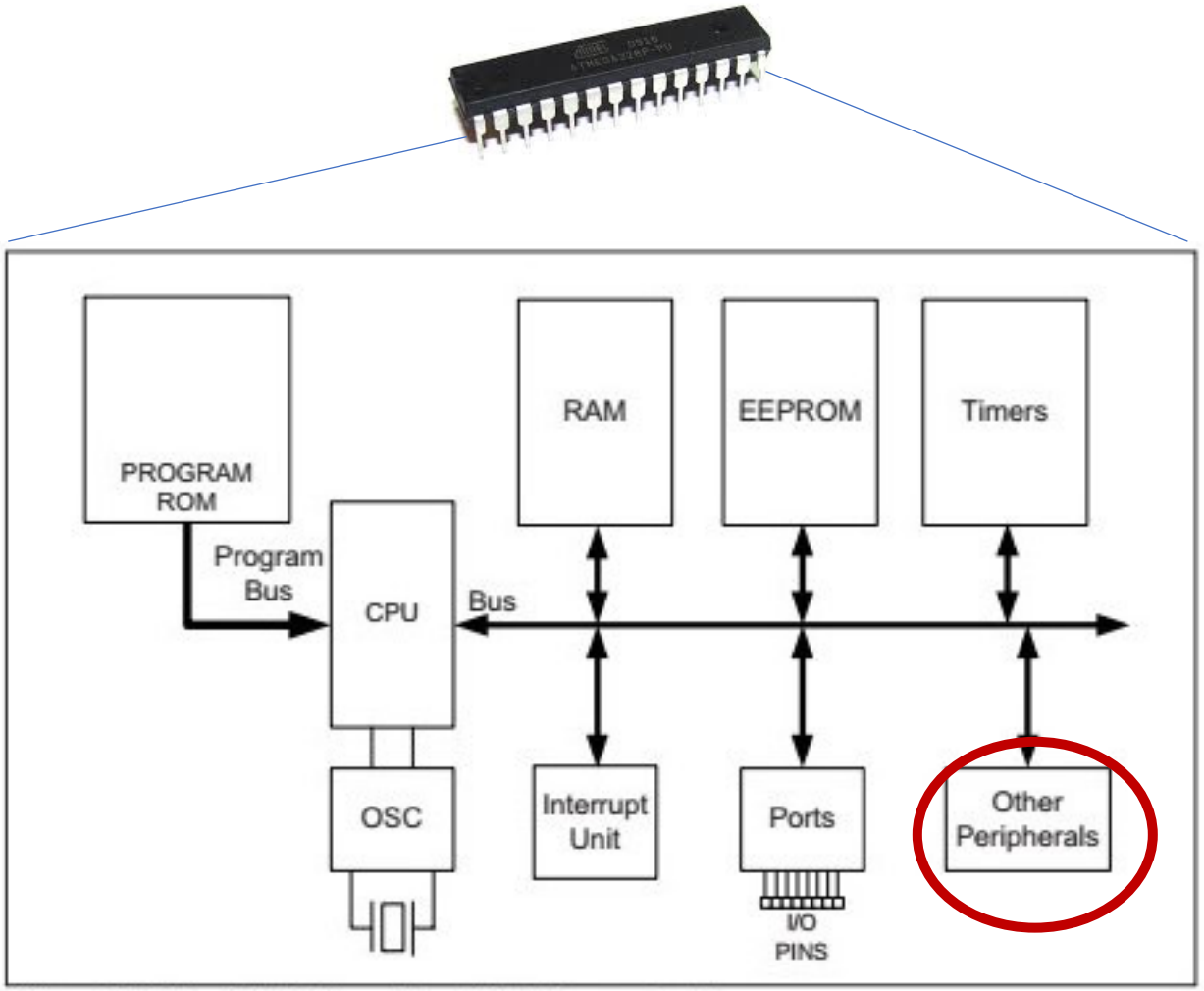
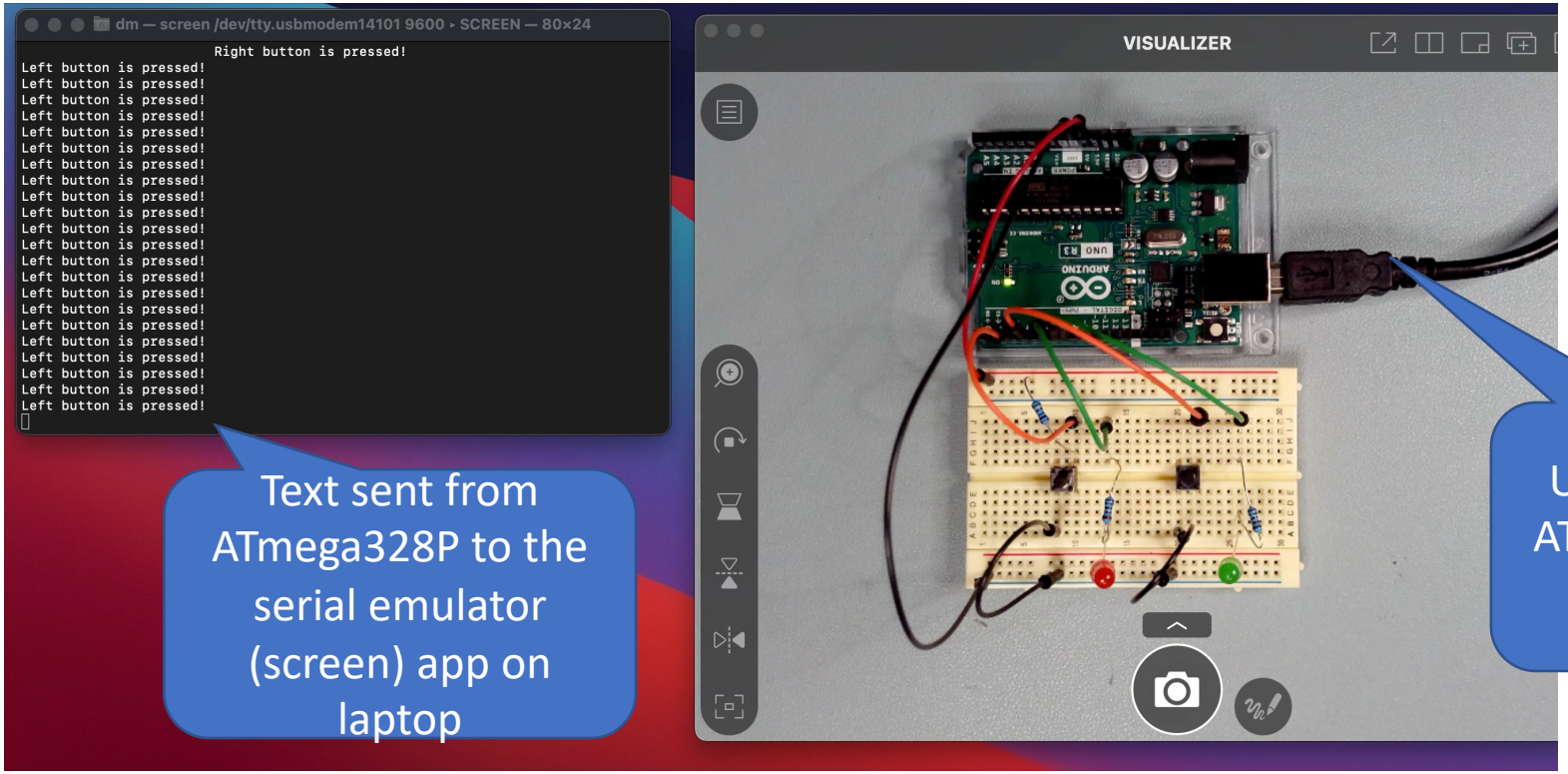


Figure 1-2. Simplified View of an AVR Microcontroller

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.

```c
/* switches2.c This code turns on LEDs when  momemtary pushbutton
switches are pressed. Input pins on PD3 and PD4. LEDS on PD6 on PD7
D. McLaughlin 2/21/22 */

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

int main(void){
    DDRD = 1<<DDD6|1<<DDD7;             // Set D6, D7 as output
    PORTD = 1<<PORTD3|1<<PORTD4;        // Set pullup on D3 & D4

    while(1){

        if  ((PIND & (1<<PIND3)) == 0){ // PIND3 is 0 when pressed
            PORTD |= (1<<PORTD6);       // P6 high when P3 is pressed
        } else {
            PORTD &= ~(1<<PORTD6);      // P6 low when P3 not pressed
        }

        if  ((PIND & (1<<PIND4)) == 0){ // PIND4=0 when pressed
            PORTD |= (1<<PORTD7);       // P7 high when P4 is pressed
        } else {
            PORTD &= ~(1<<PORTD7);      // P7 low when P4 not pressed
        }

    }
    return(0);
}

```

```c
/* switches_serial.c This code turns on LEDs when  momemtary pushbutton
switches are pressed. Input pins on PD3 and PD4. LEDS on PD6 on PD7
Code also sends text string to serial monitor via UART
D. McLaughlin 2/22/22 */

#include <avr/io.h>
#include <util/delay.h>
#include <string.h>  //so that we can use the  strlen() function
#include <stdlib.h>  //need for itoa function
void uart_init(void);
void uart_send(unsigned char);
void send_string(char *stringAddress);

int main(void){
    DDRD = 1<<DDD6|1<<DDD7;             // Set D6, D7 as output
    PORTD = 1<<PORTD3|1<<PORTD4;        // Set pullup on D3 & D4
    uart_init();                        // Initialize the UART
    while(1){

        if  ((PIND & (1<<PIND3)) == 0){ // PIND3 is 0 when pressed
            PORTD |= (1<<PORTD6);       // P6 high when P3 is pressed
            send_string("Left button is pressed!\n");
            uart_send(13);
        } else {
            PORTD &= ~(1<<PORTD6);      // P6 low when P3 not pressed
        }

        if  ((PIND & (1<<PIND4)) == 0){ // PIND4=0 when pressed
            PORTD |= (1<<PORTD7);       // P7 high when P4 is pressed
            send_string("\t\t\tRight button is pressed!\n");
            uart_send(13);
        } else {
            PORTD &= ~(1<<PORTD7);      // P7 low when P4 not pressed
        }
    }
    return(0);
}

> void uart_init(void){…

> void uart_send(unsigned char ch){…

> 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	♂	VT
^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	_

Dec	Hex	Ch
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

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
^@	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	♂	VT
^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	_

Dec	Hex	Ch
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

Dec	Hex	Ch
128	80	Ç
129	81	ü
130	82	é
131	83	â
132	84	ä
133	85	à
134	86	å
135	87	ç
136	88	ê
137	89	ë
138	8A	è
139	8B	ï
140	8C	î
141	8D	ì
142	8E	Ä
143	8F	Å
144	90	É
145	91	æ
146	92	Æ
147	93	ô
148	94	ö
149	95	ò
150	96	û
151	97	ù
152	98	ÿ
153	99	Ö
154	9A	Ü
155	9B	¢
156	9C	£
157	9D	¥
158	9E	₧
159	9F	ƒ

Dec	Hex	Ch
160	A0	á
161	A1	í
162	A2	ó
163	A3	ú
164	A4	ñ
165	A5	Ñ
166	A6	ª
167	A7	º
168	A8	¿
169	A9	⌐
170	AA	¬
171	AB	½
172	AC	¼
173	AD	¡
174	AE	«
175	AF	»
176	B0	░
177	B1	▒
178	B2	▓
179	B3	│
180	B4	┤
181	B5	╡
182	B6	╢
183	B7	╖
184	B8	╕
185	B9	╣
186	BA	║
187	BB	╗
188	BC	╝
189	BD	╜
190	BE	╛
191	BF	┐

Dec	Hex	Ch
192	C0	└
193	C1	┴
194	C2	┬
195	C3	├
196	C4	─
197	C5	┼
198	C6	╞
199	C7	╟
200	C8	╚
201	C9	╔
202	CA	╩
203	CB	╦
204	CC	╠
205	CD	═
206	CE	╬
207	CF	╧
208	D0	╨
209	D1	╤
210	D2	╥
211	D3	╙
212	D4	╘
213	D5	╒
214	D6	╓
215	D7	╫
216	D8	╪
217	D9	┘
218	DA	┌
219	DB	█
220	DC	▄
221	DD	▌
222	DE	▐
223	DF	▀

Dec	Hex	Ch
224	E0	α
225	E1	ß
226	E2	Γ
227	E3	π
228	E4	Σ
229	E5	σ
230	E6	µ
231	E7	τ
232	E8	Φ
233	E9	Θ
234	EA	Ω
235	EB	δ
236	EC	∞
237	ED	ø
238	EE	∈
239	EF	∩
240	F0	≡
241	F1	±
242	F2	≥
243	F3	≤
244	F4	⌠
245	F5	⌡
246	F6	÷
247	F7	≈
248	F8	°
249	F9	∙
250	FA	·
251	FB	√
252	FC	ⁿ
253	FD	²
254	FE	■
255	FF	

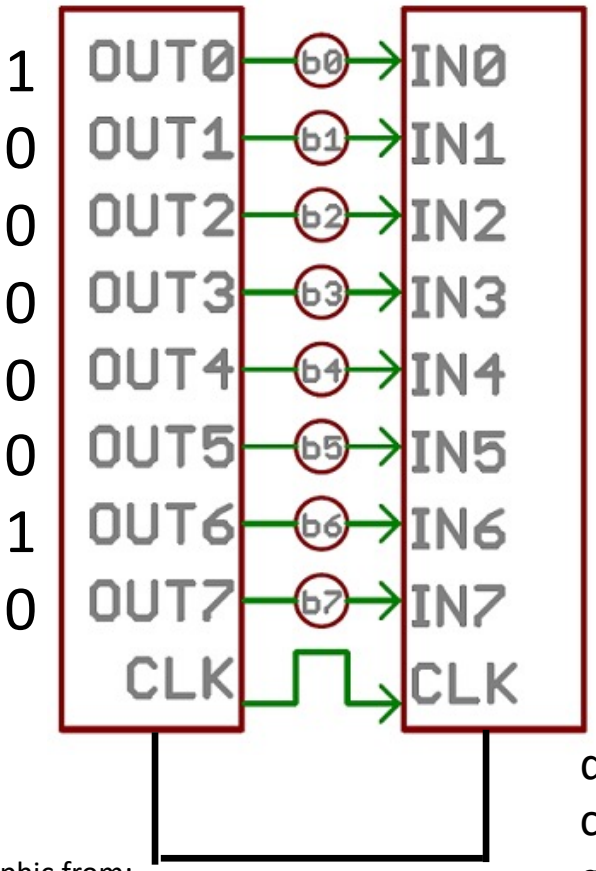
3 Different Types of Communication between 2 devices

- Parallel communication
- Synchronous serial communication
- Asyshronous serial communication

Synchronous parallel communication

Device sending data (transmitter)

'A' = 01000001



Device receiving data (transmitter)

'A' = 01000001

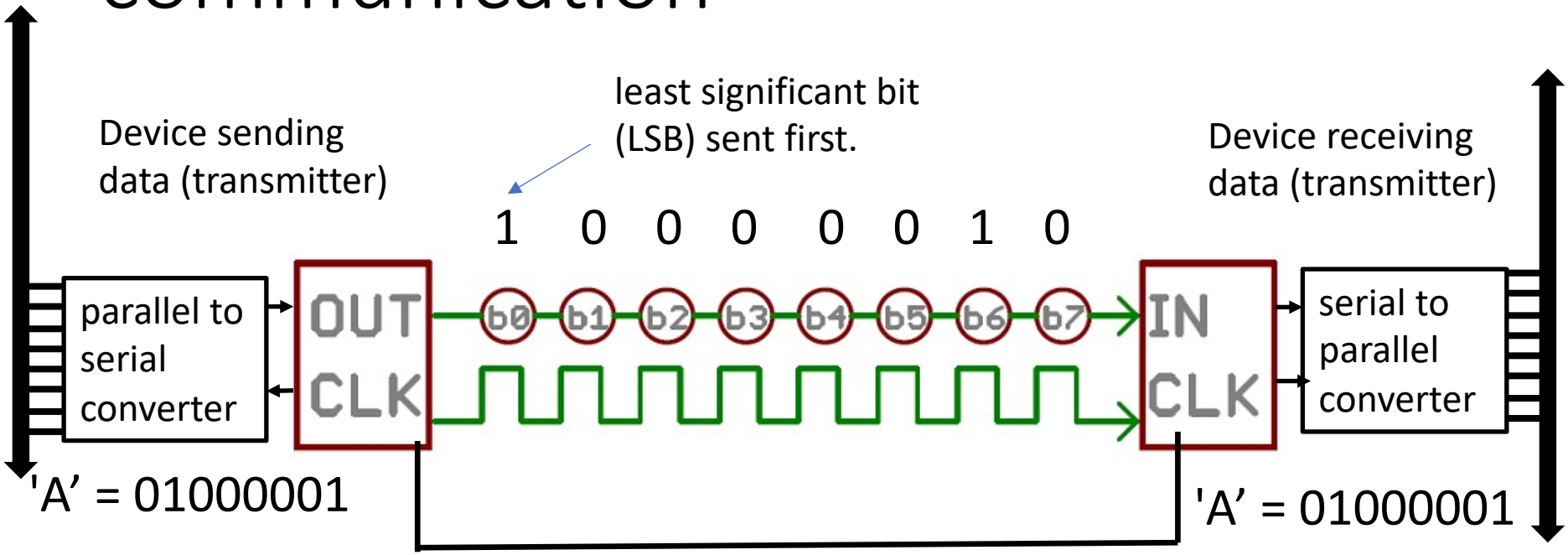
don't forget to connect the grounds!

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

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

Synchronous serial communication



Protocol (agreement):

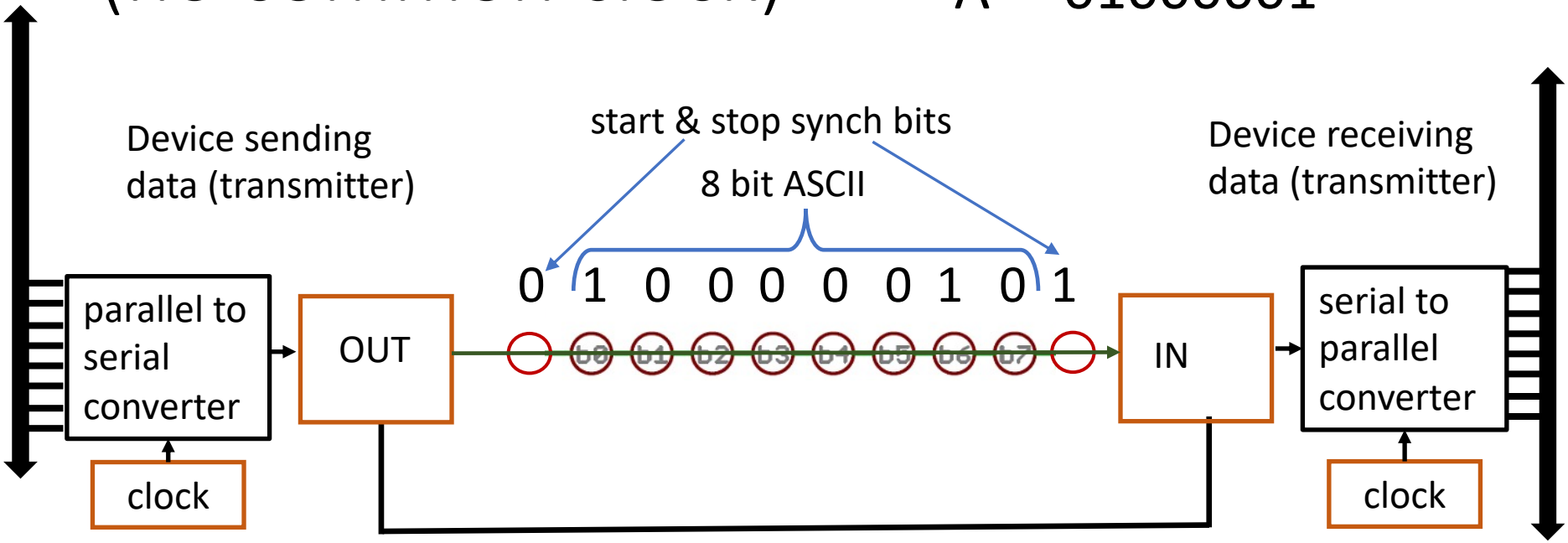
- 7 vs 8 bit ASCII
- transfer data on the rising edge of the clock (or the falling edge of the clock.)
- Hardware:1 data line + clock + ground shared between devices

don't forget to connect the grounds!

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

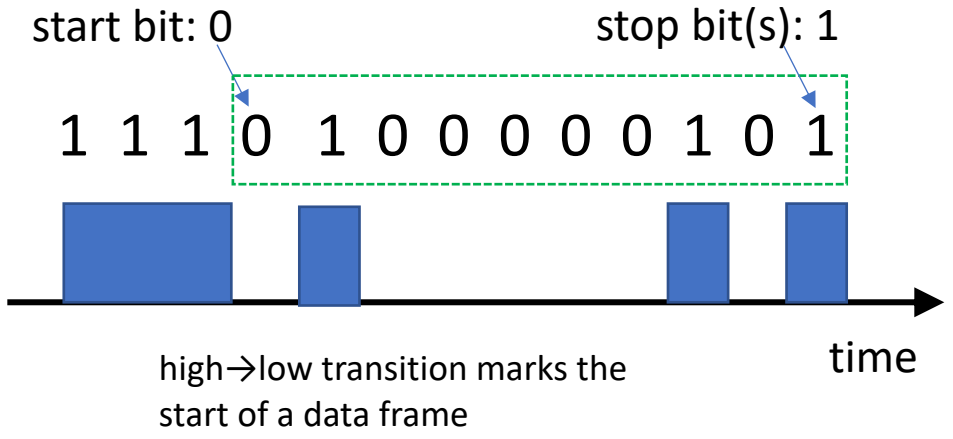
asynchronous serial communication (no common clock)

'A' = 01000001

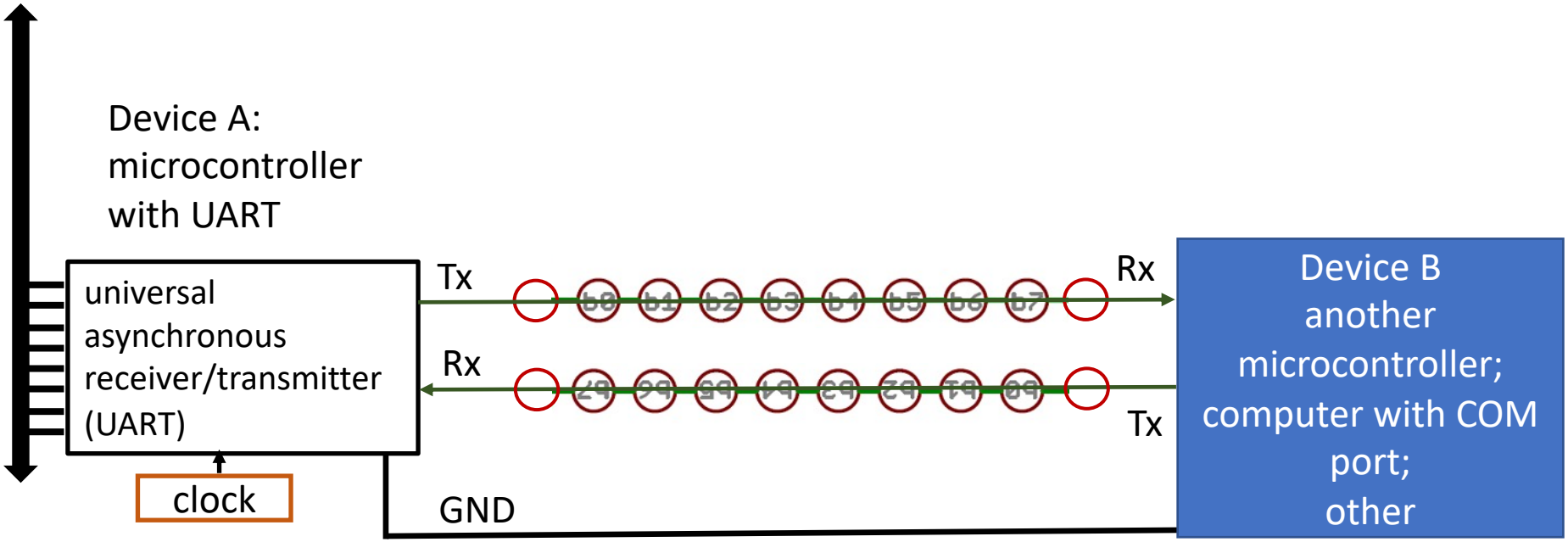


Protocol (agreement):

- 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 Asychronous Receiver Transmitter (synchronous less frequently used, so UART)

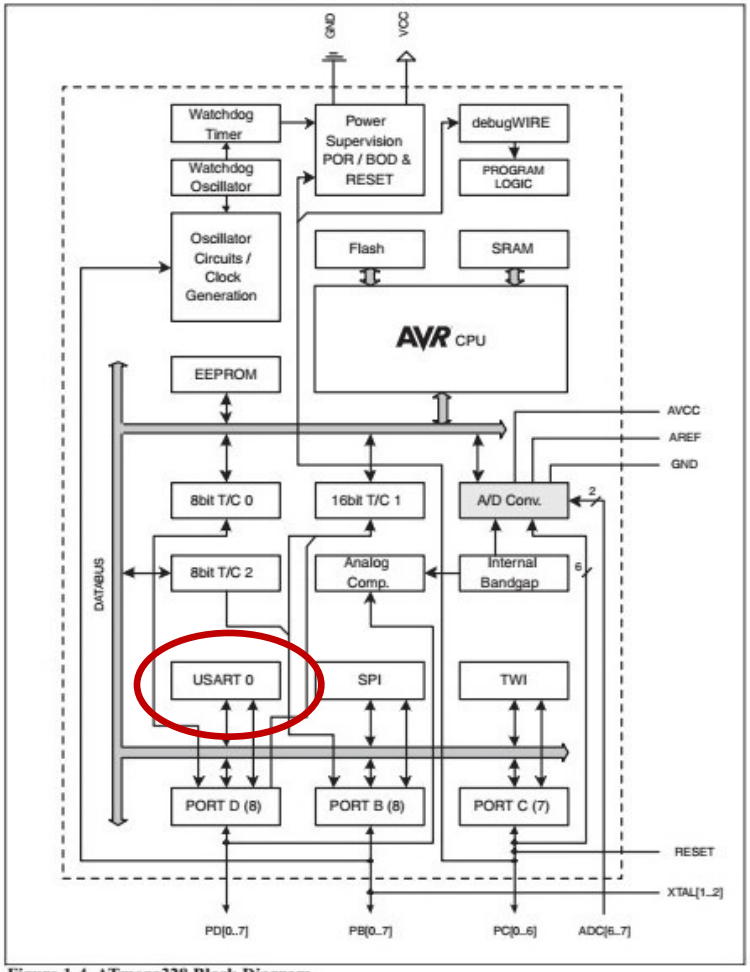


Figure 1-4. ATmega328 Block Diagram

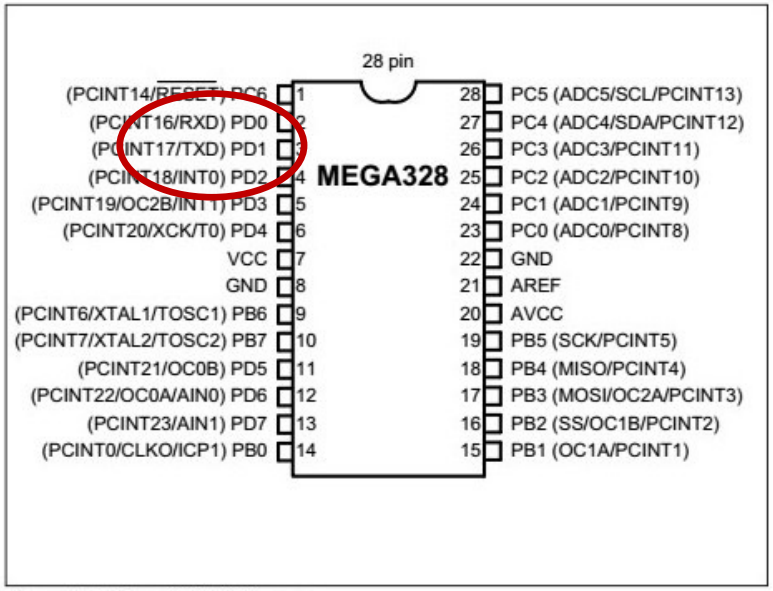
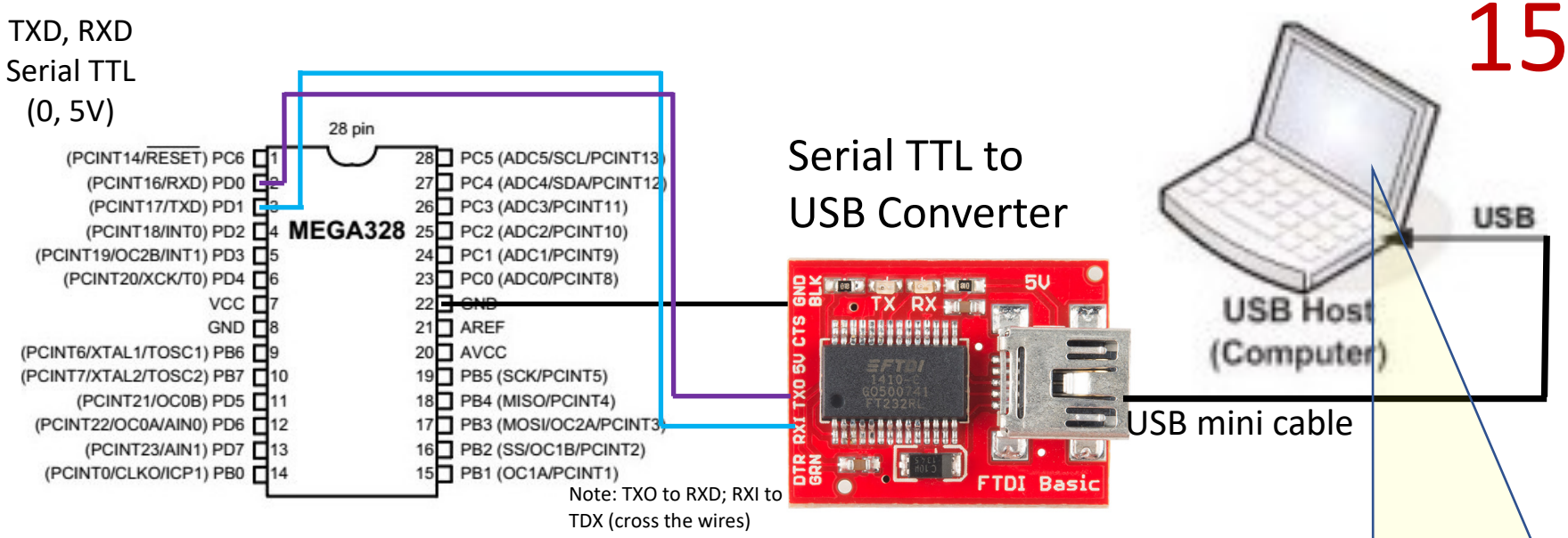


Figure 4-1. ATmega328 Pin Diagram

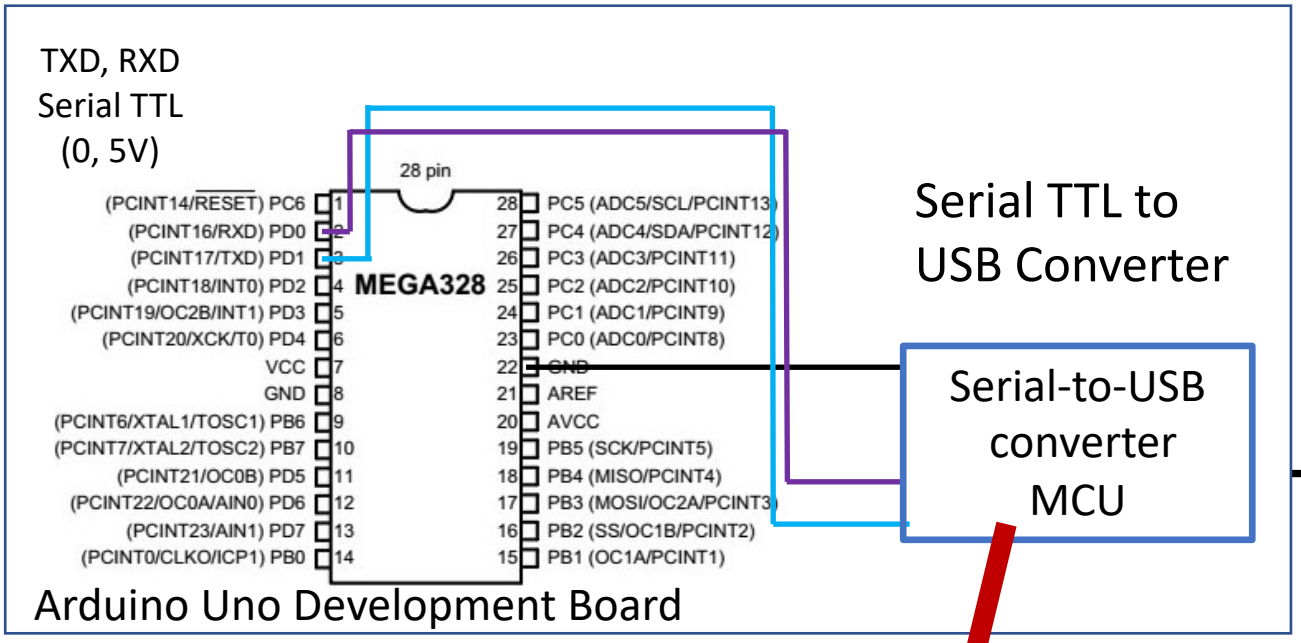
The UART is one of several peripherals built into the ATmega328P MCU



Asynch Serial Comm between ATmega328P & USB host

- Hardware: Serial TTL to USB converter (FDTI 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)



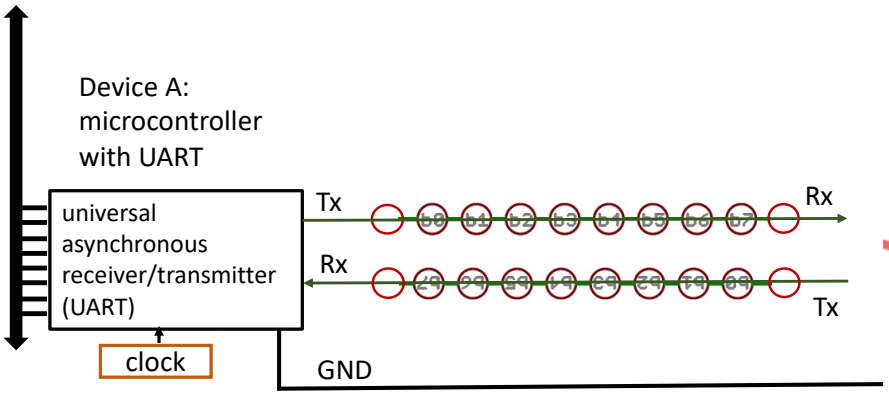
USB A to B
Cable

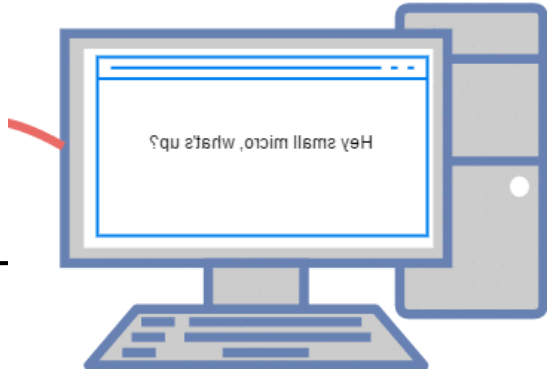
Note:

(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

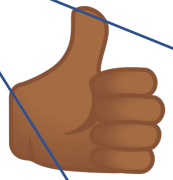
Basic UART Idea:





Protocol (agreement):

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



- 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

STEP 4. send character

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

```
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.

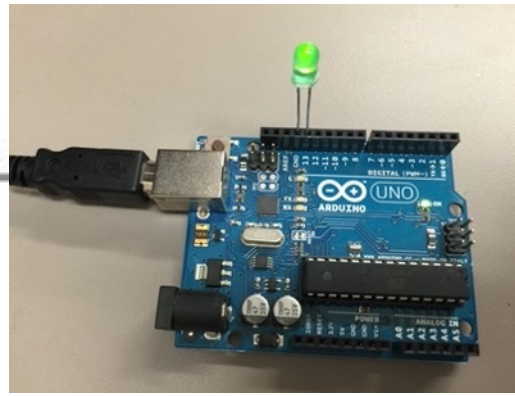
Part (D) Modify the code to send the following string of text “Shall I compare thee to a summer’s day?”

C serialnofunctions_main.c > ...

```c
/*****************************************************************
* 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
    }
}

```

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 ...



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

Windows Users

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

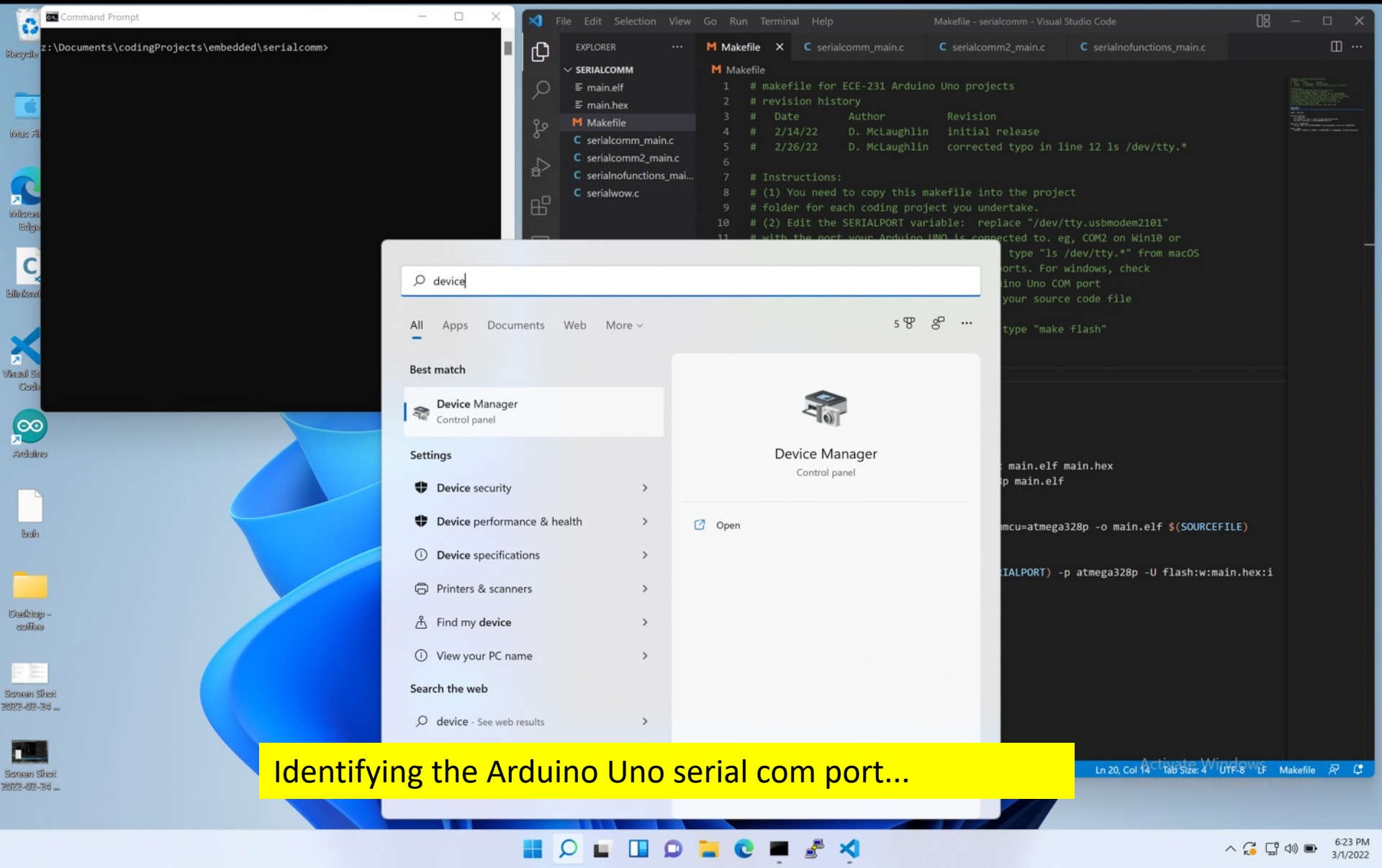
3. Identify the Arduino Uno Com port using the Windows Device Manager (use this port both for makefile and for serial comm using PuTTY)

4. Flash the code to the ATmega328p on the Arduino Uno using “make flash”

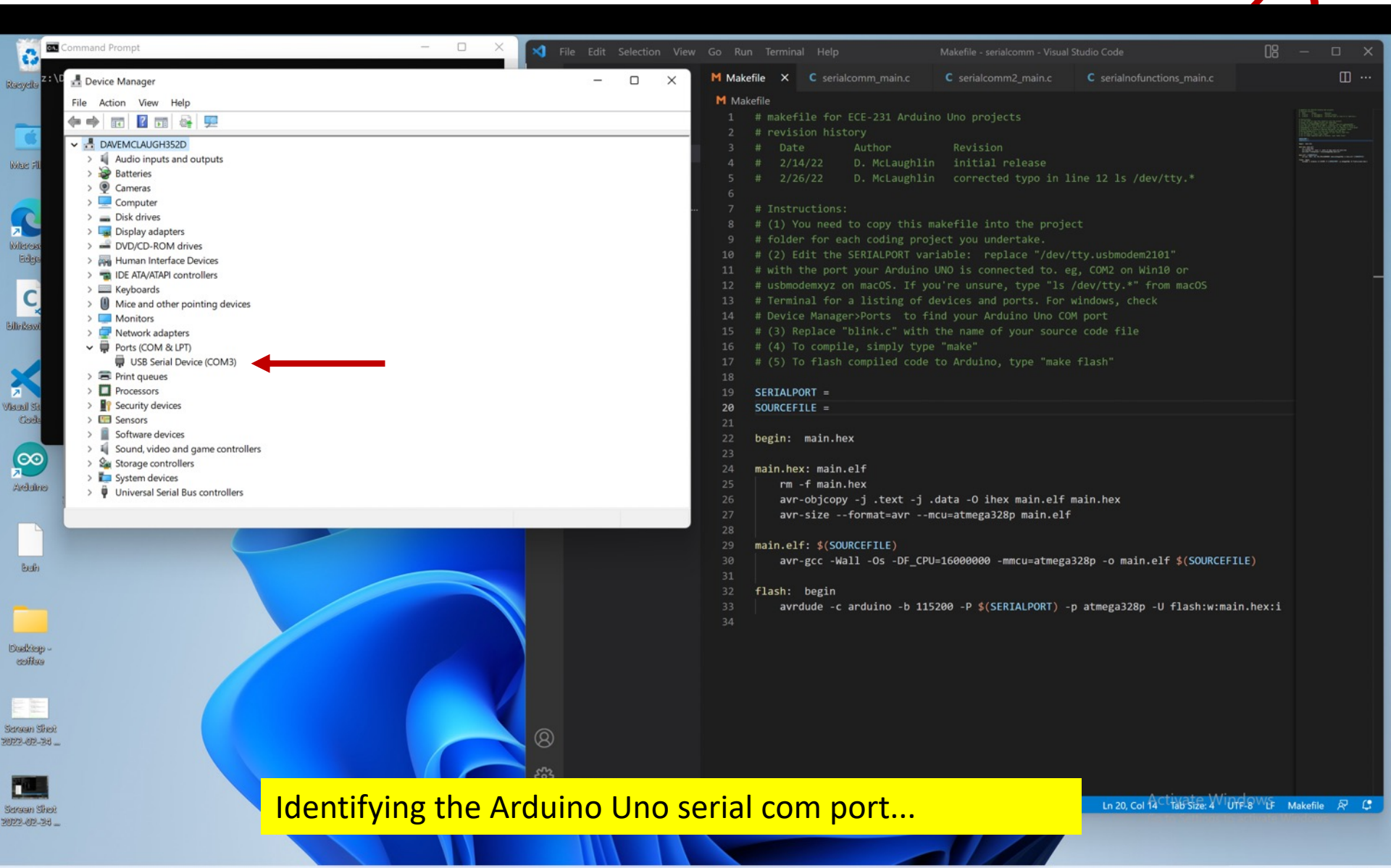
5. Launch PuTTY. Configure protocol: 8 data bits, 1 stop bit, no parity bits, 9600 baud

6. Observe “GGGGGGG....” on the display.

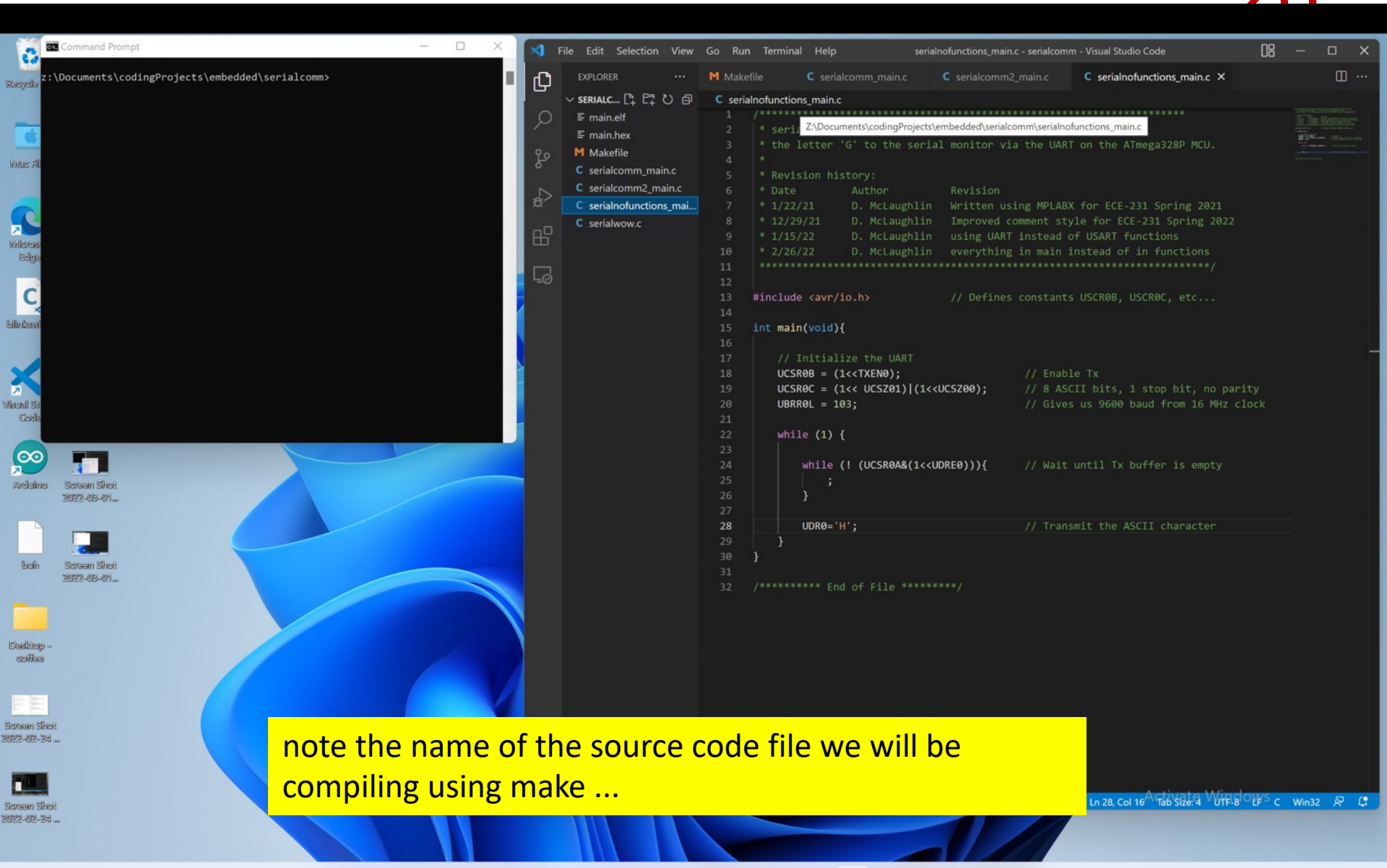
7. For fun: change ‘G’ to ’H’ in the source code; re-compile & flash using: make, make flash. Do do this:
   a) make. To be sure the code compiles correctly.
   b) Close Putty. (Otherwise PuTTY’s serial comm conneftion will tie up the com port to the Arduino, preventing make flash from working.)
   c) make flash. To download the modified code
   d) Observe “HHHHH....” on the display



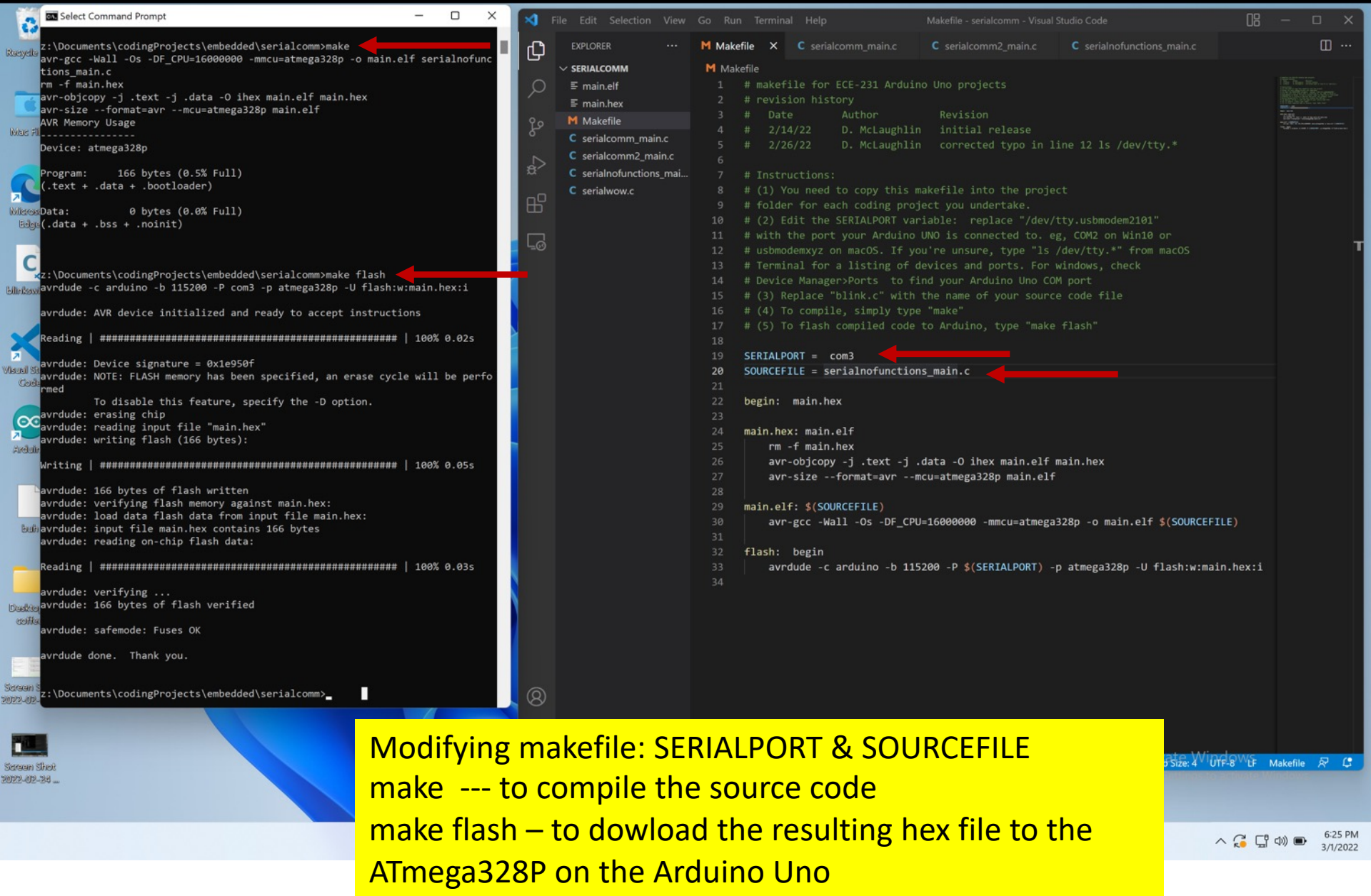
Identifying the Arduino Uno serial com port...



Identifying the Arduino Uno serial com port...



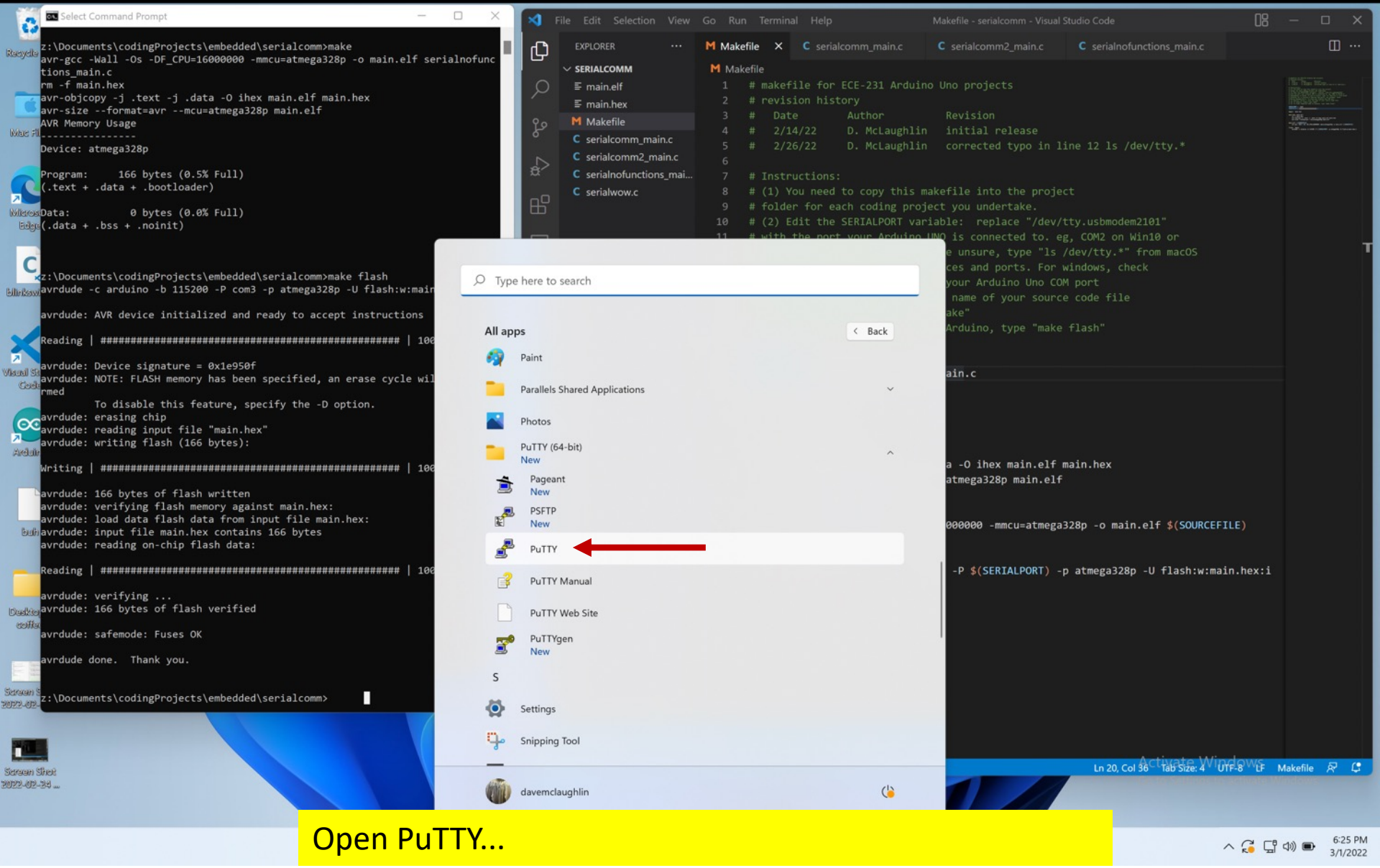
note the name of the source code file we will be compiling using make ...



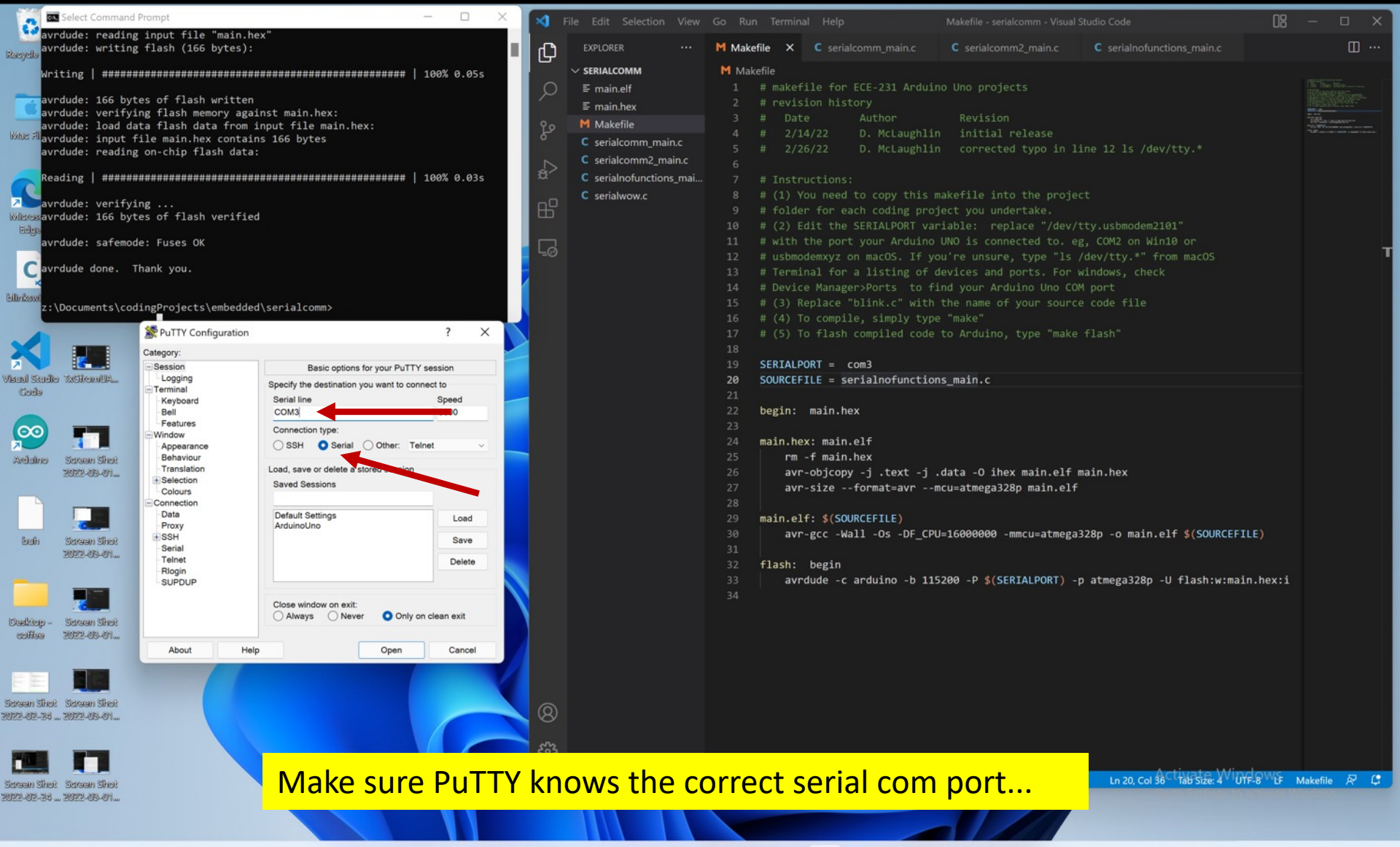
Modifying makefile: SERIALPORT & SOURCEFILE

make --- to compile the source code

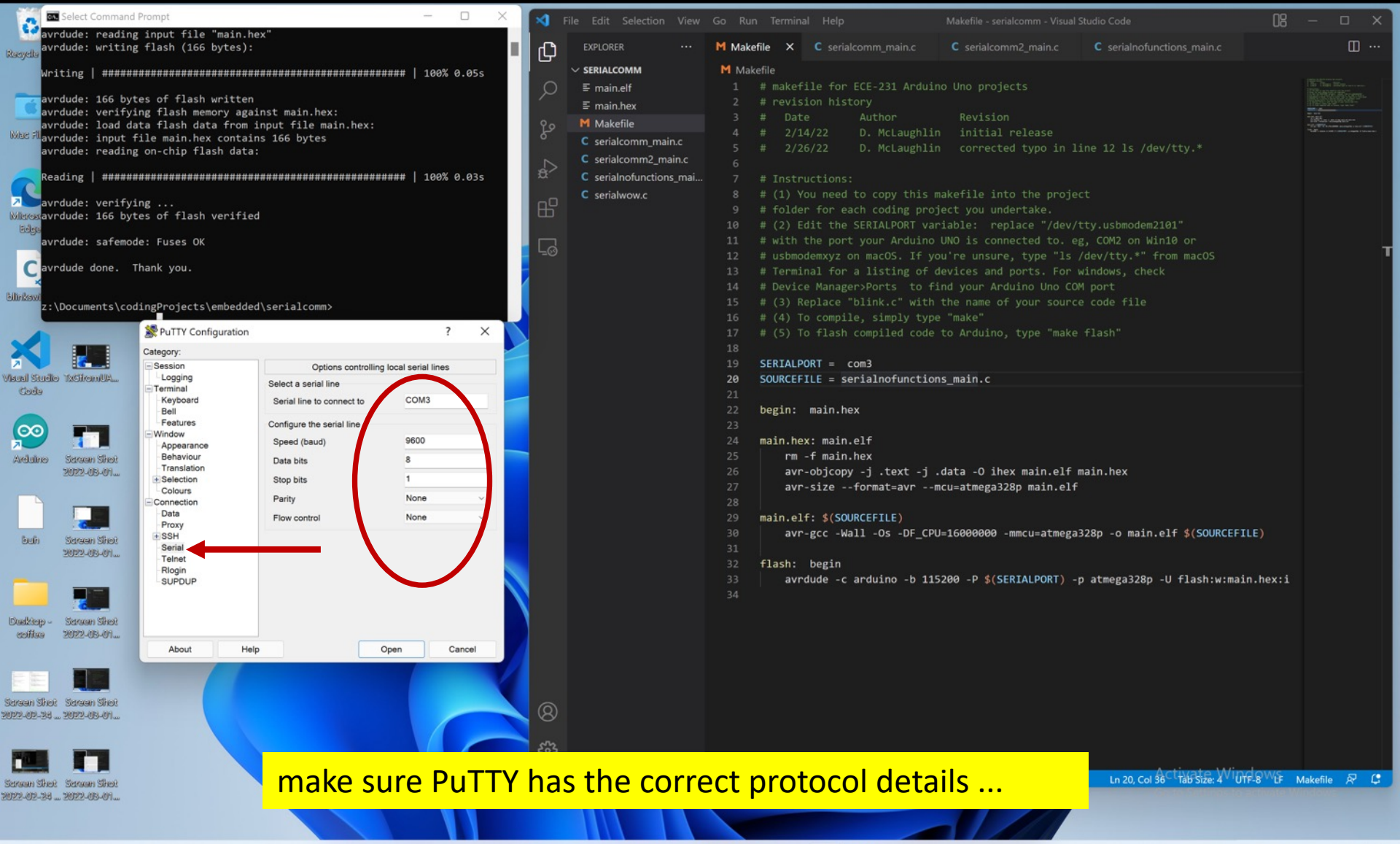
make flash – to dowload the resulting hex file to the ATmega328P on the Arduino Uno


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
Device: atmega328p
Program: 166 bytes (0.5% Full)
(.text + .data + .bootloader)
Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)
z:\Documents\codingProjects\embedded\serialcomm>make flash
avrdude: AVR device initialized and ready to accept instructions
avrdude: Device signature = 0x1e950f
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (166 bytes):
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:
avrdude: verifying ...
avrdude: 166 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.
z:\Documents\codingProjects\embedded\serialcomm>
Makefile - serialcomm - Visual Studio Code
EXPLORER
SERIALCOMM
main.elf
main.hex
Makefile
serialcomm_main.c
serialcomm2_main.c
serialnofunctions_mai...
serialwow.c
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"
Type here to search
All apps
Back
Paint
Parallels Shared Applications
Photos
PuTTY (64-bit)
New
Pageant
New
PSFTP
New
PuTTY
PuTTY Manual
PuTTY Web Site
PuTTYgen
New
S
Settings
Snipping Tool
davemclaughlin
Ln 20, Col 36
Tab Size: 4
UTF-8
LF
Makefile
6:25 PM
3/1/2022

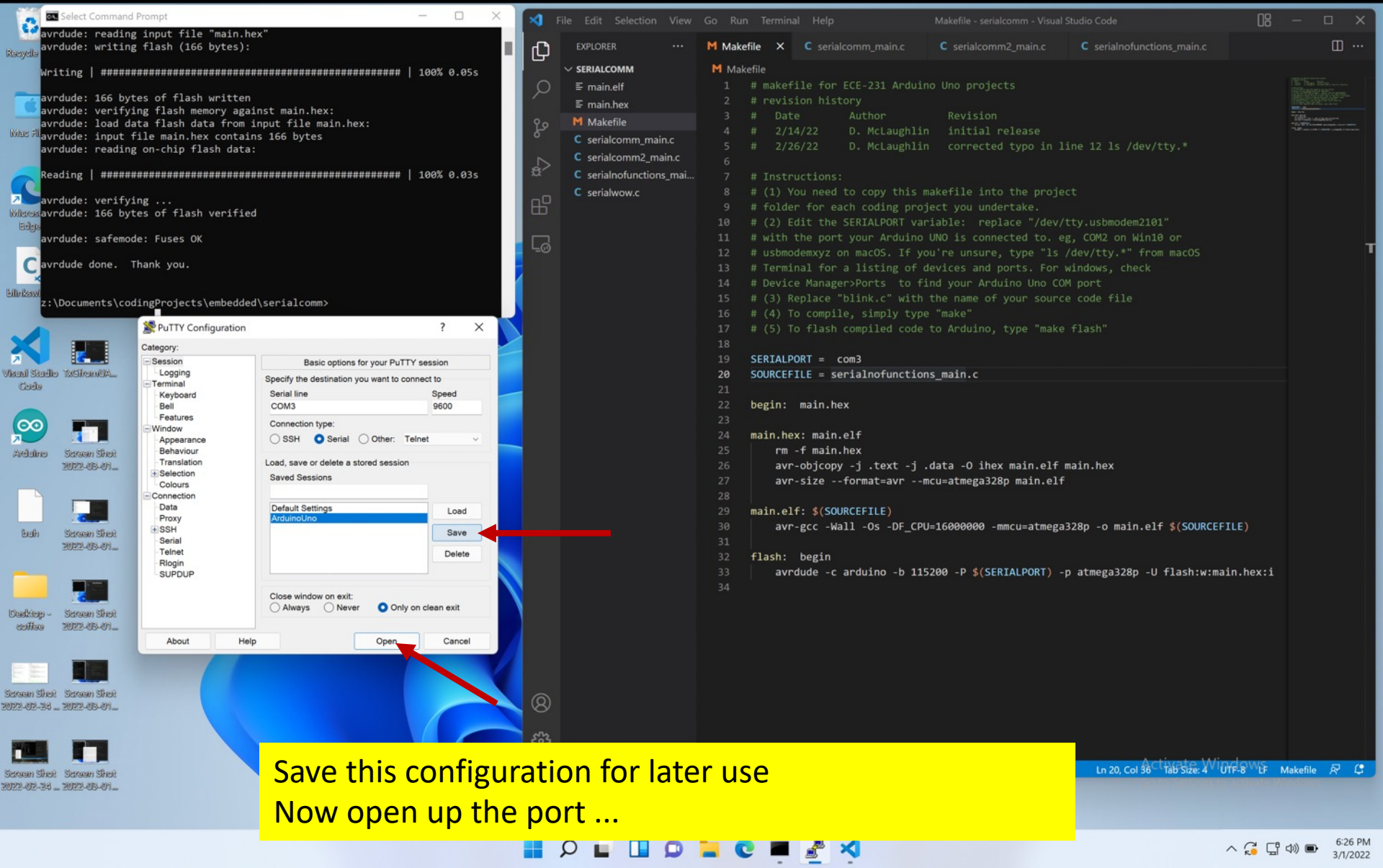
Open PuTTY...



Make sure PuTTY knows the correct serial com port...

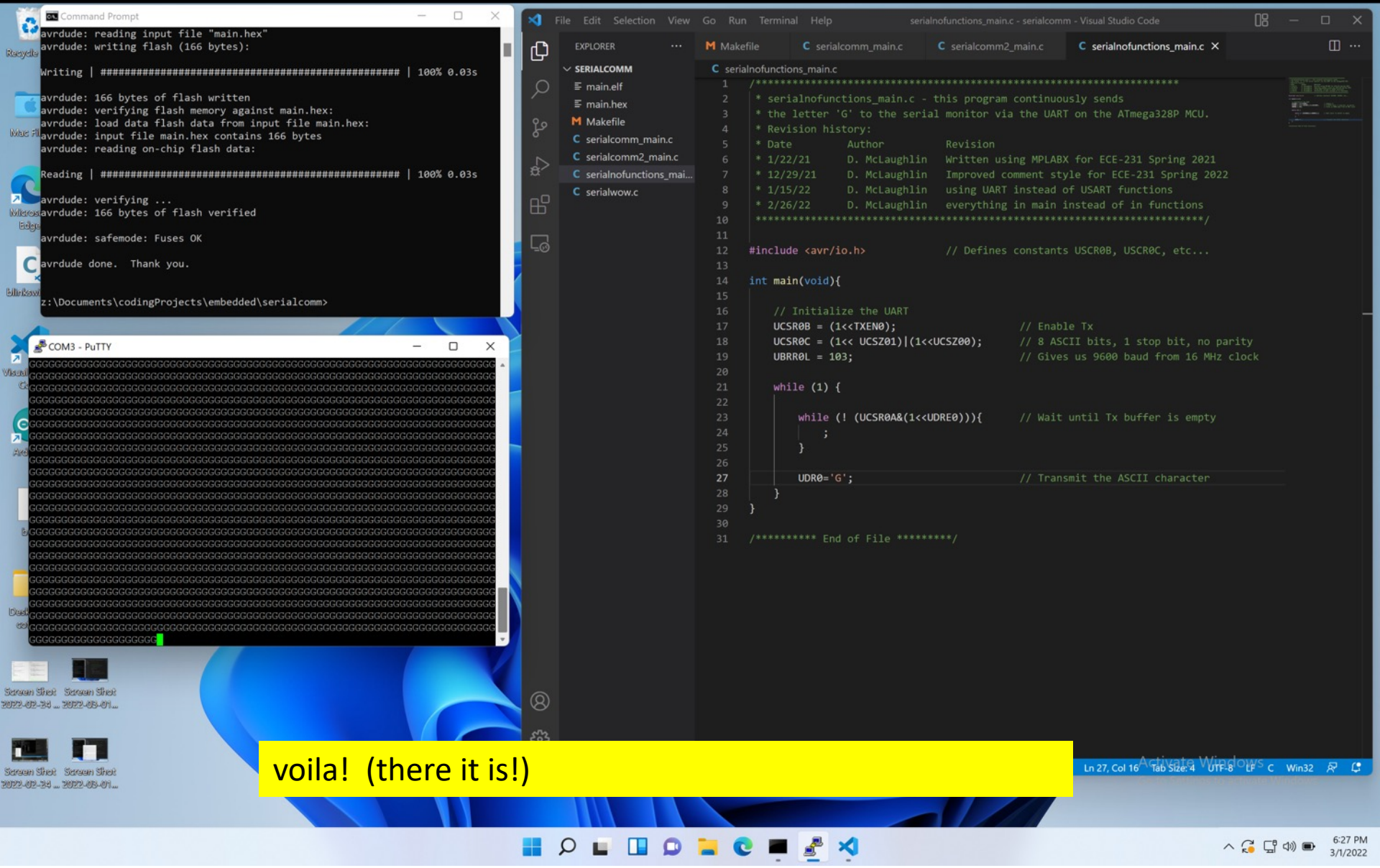


make sure PuTTY has the correct protocol details ...

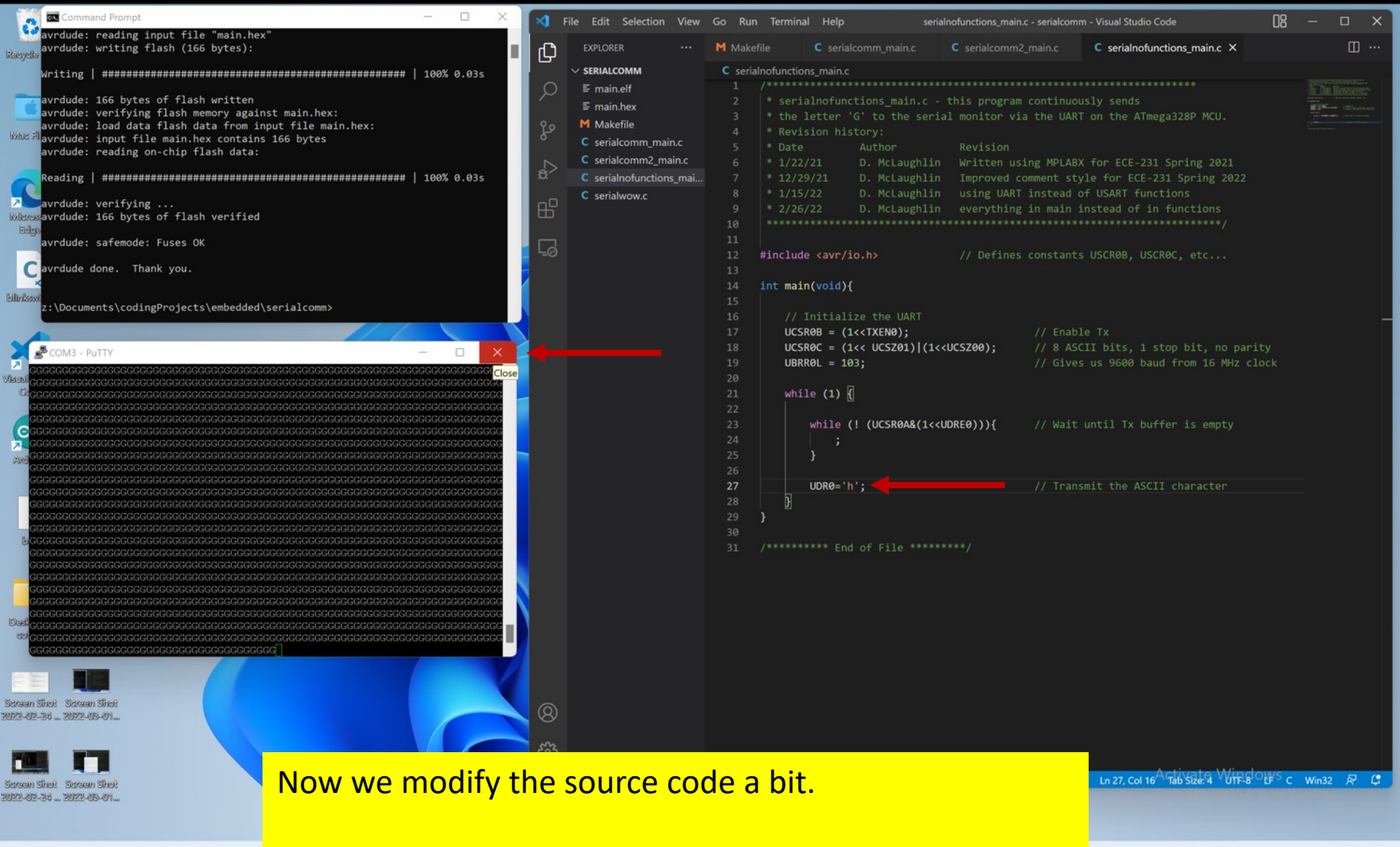


Save this configuration for later use

Now open up the port ...

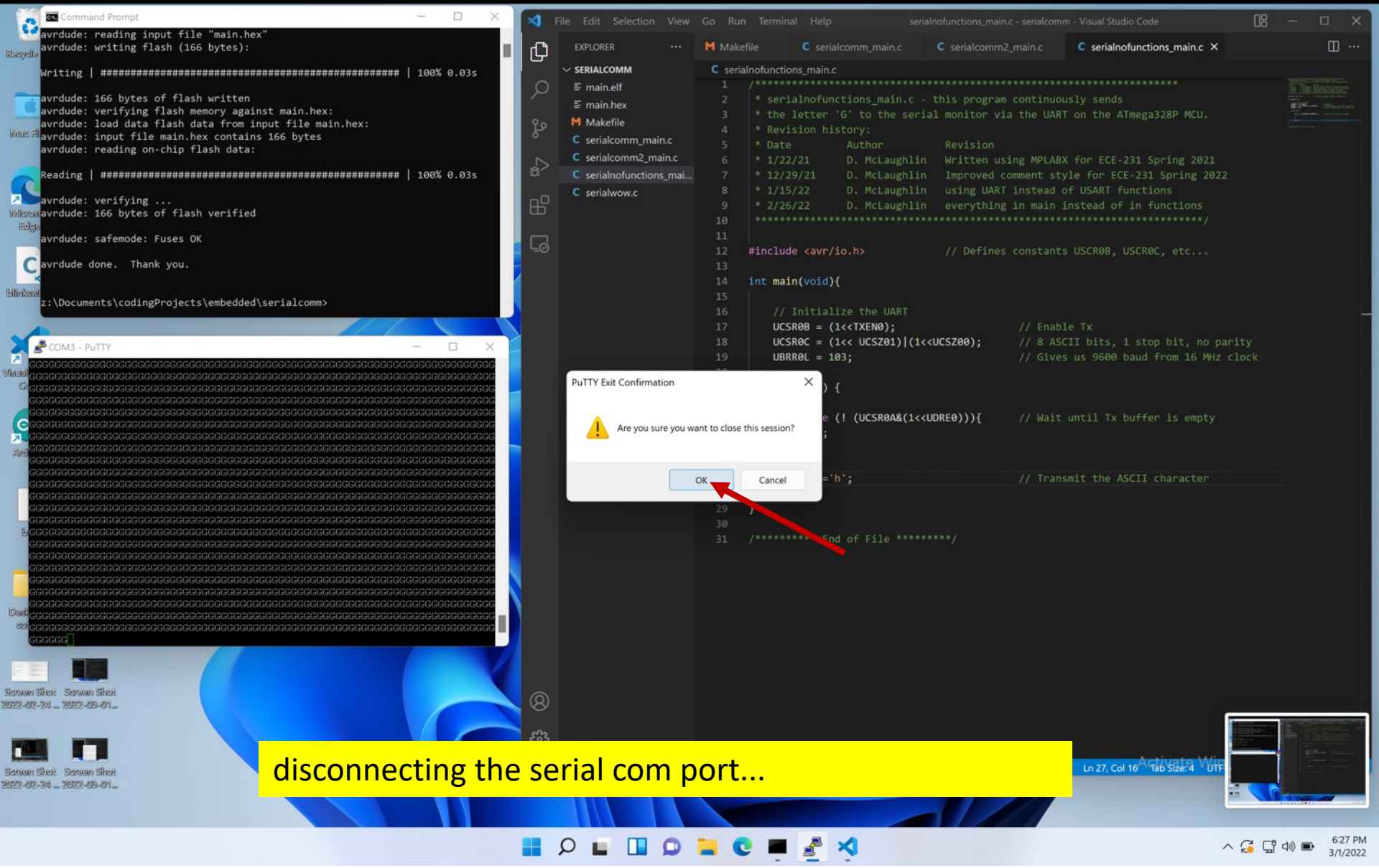


voila! (there it is!)

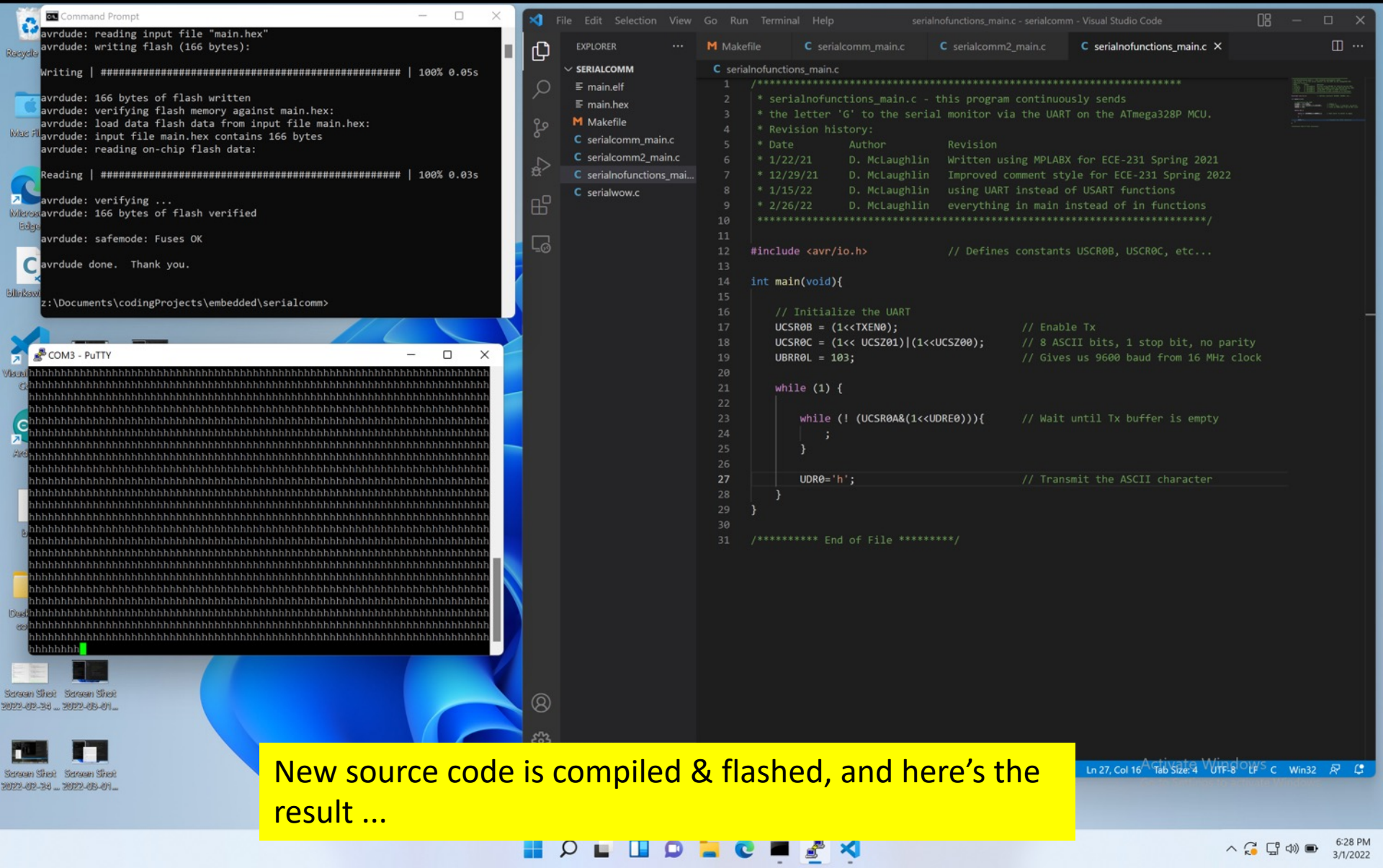


Now we modify the source code a bit.

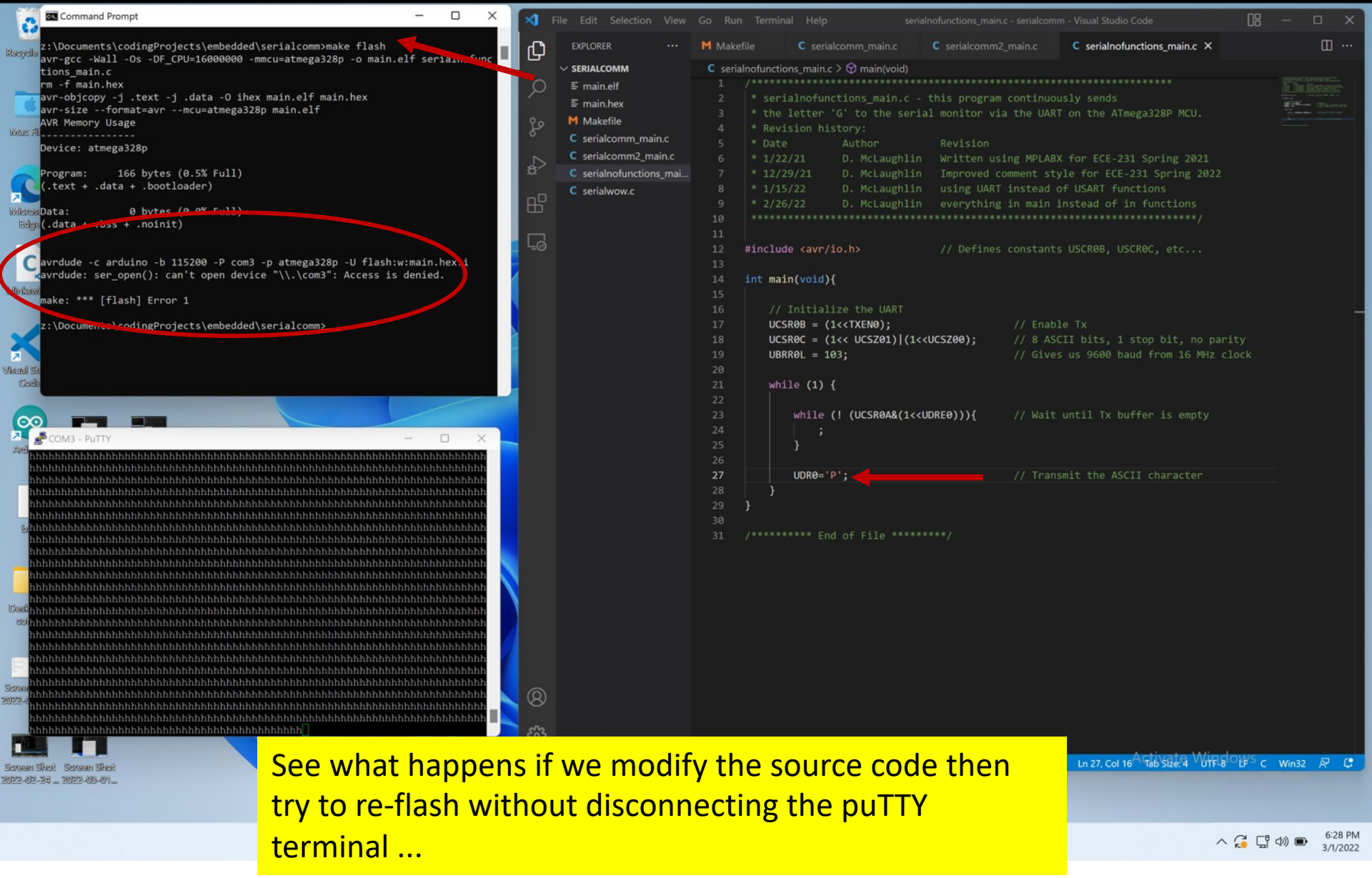
We need to close the serial comm session in order to use the USB/serial line to flash the ATmega328P ...



disconnecting the serial com port...



New source code is compiled & flashed, and here's the result ...



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

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 *******/
```

```c
/*****************************************************************
 * 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 *********/
```

```c
/*****************************************************************
 * 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() definined 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)  {
    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 *************/
```

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.

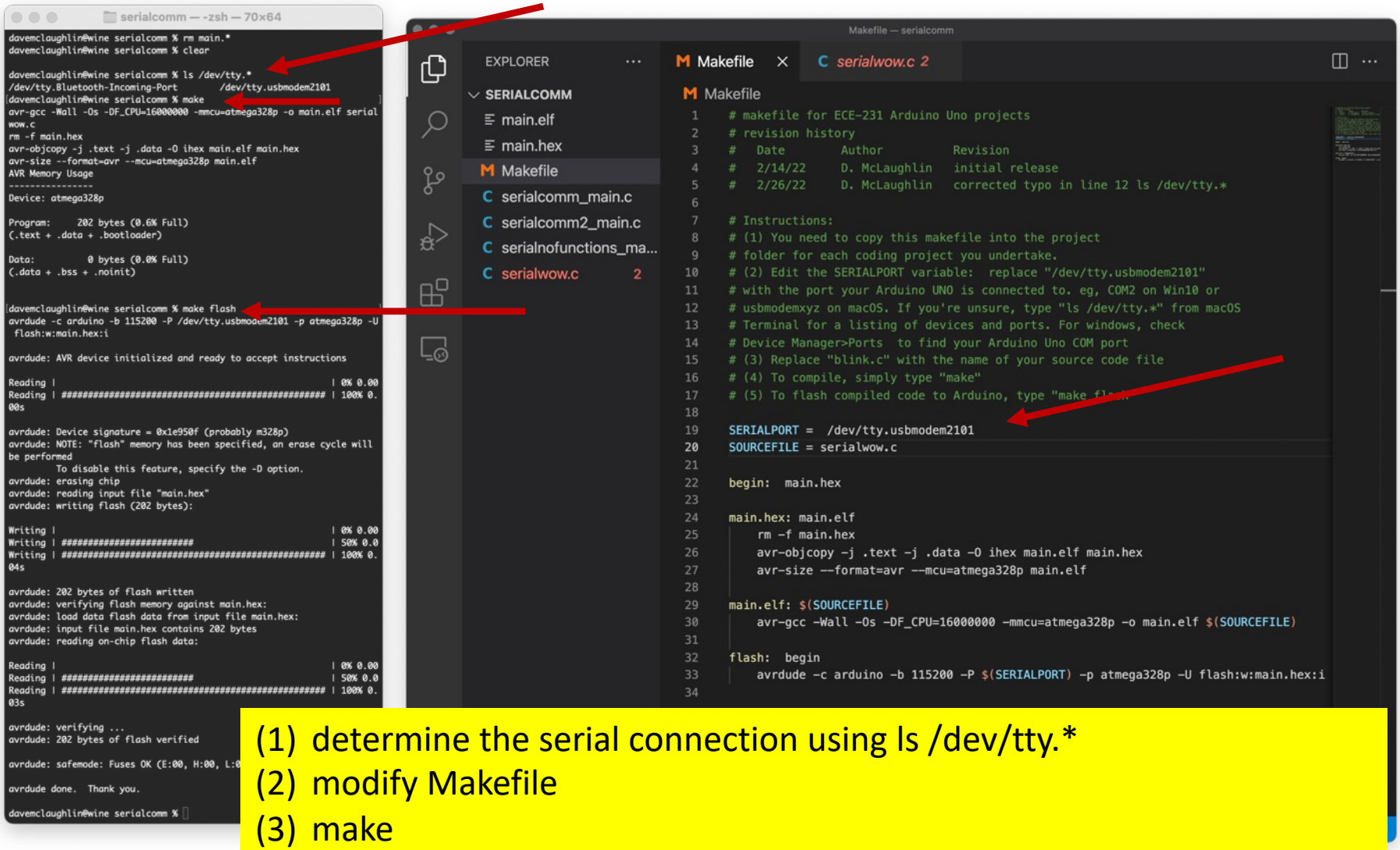
Part (D) Modify the code to send the following string of text “Shall I compare thee to a summer’s day?”

serialwow.c > ...

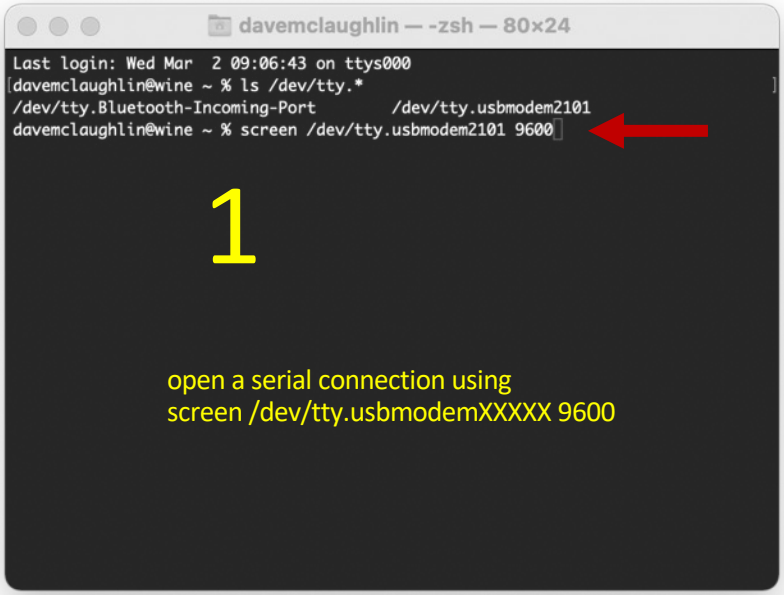
```
/*****************************************************************************
 * serialcomm_main.c - this program continuously sends
 * the letter 'WOW!' to the serial monitor via the UART on the ATmega328P MCU.
 * Date         Author          Revision
 * 3/1/22       D. McLaughlin   initial release. 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 *************/
```

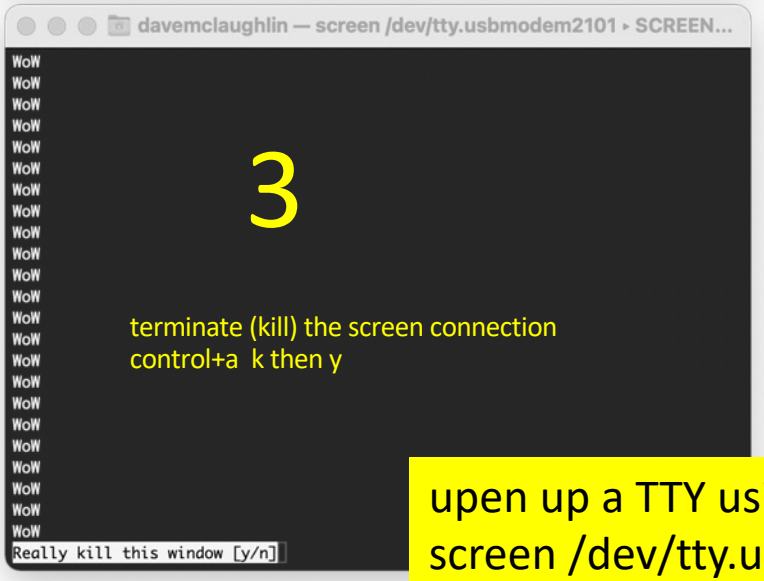
Easy enough to make this change when we only want to send a few characters to the terminal...

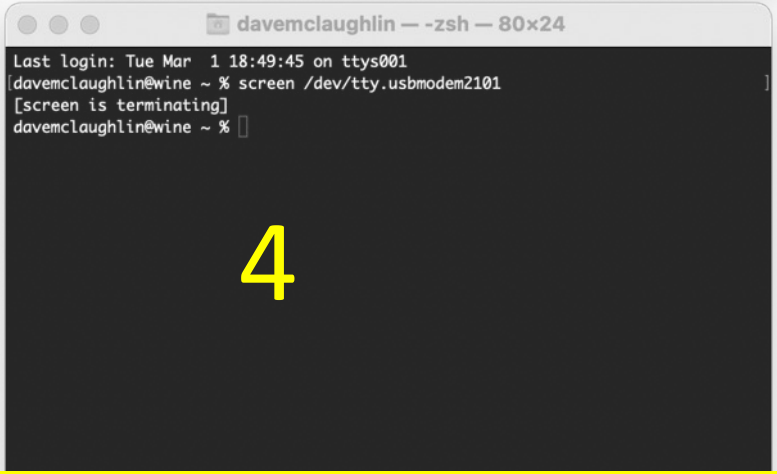
This example done under macOS



(1) determine the serial connection using ls /dev/tty.*
(2) modify Makefile
(3) make
(4) make flash







upen 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.

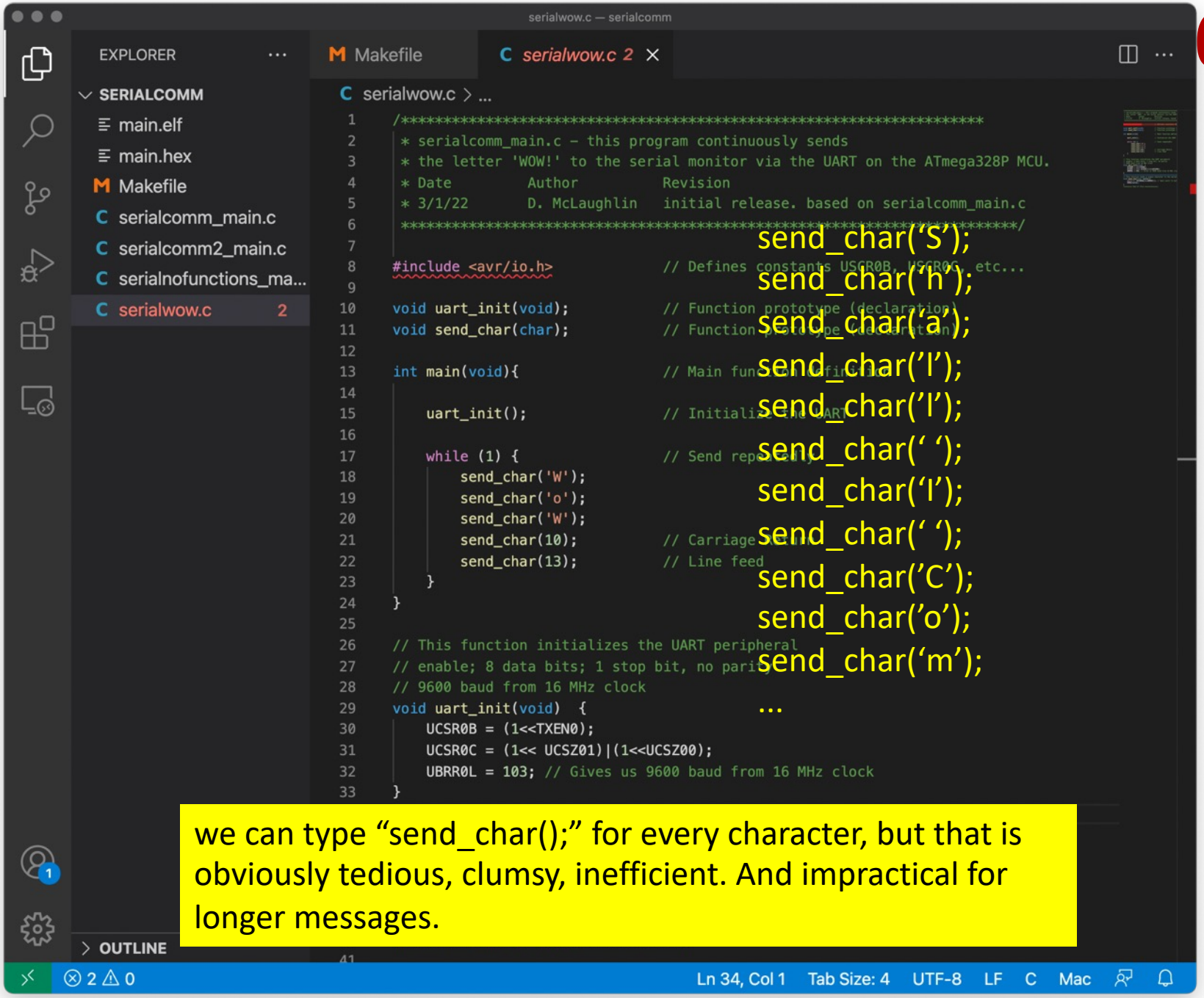
Protocol: Use 8-bit data, 1 stop bit, no parity bits

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 — serialcomm
EXPLORER
SERIALCOMM
main.elf
main.hex
Makefile
serialcomm_main.c
serialcomm2_main.c
serialnofunctions_ma...
serialwow.c 2
Makefile
serialwow.c 2
serialwow.c > ...
/***
* 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 UCSR0B, UCSR0C, 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
}
send_char('S');
send_char('h');
send_char('a');
send_char('l');
send_char('l');
send_char(' ');
send_char('I');
send_char(' ');
send_char('C');
send_char('o');
send_char('m');
...
we can type "send_char();" for every character, but that is obviously tedious, clumsy, inefficient. And impractical for longer messages.
OUTLINE
Ln 34, Col 1 Tab Size: 4 UTF-8 LF C Mac

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?";
```

uart_string.c > ...

```c
/*****************************************************************
 * uart_string.c - This code sends strings to com port via uart.
 * Date     Author          Revision
 * 12/16/21 D. McLaughlin   Initial writing of the code
 * 1/15/22  D. McLaughlin   Tested on Arduino Uno w/ Apple M1 pro
 *                          host running Monterey
 * 2/27/22  D. McLaughlin   tested on Windows 11 on Parallels VM
 *****************************************************************/
#include <avr/io.h>
#include <util/delay.h>
#include <string.h>  //so that we can use the  strlen() function

void uart_init(void);
void uart_send(unsigned char);
void send_string(char *stringAddress);

int main(void){
    char mystring[]= "Shall I compare thee to a summer's day?";
    uart_init(); // initialize the USART
    while (1) {
        send_string(mystring);
        uart_send(13); // Carriage return (goto beginning of line)
        uart_send(10); //line feed (new line)
        _delay_ms(500);
    }
}

// Send a string, char by char, to uart via uart_send()
// Input is pointer to the string to be sent
void send_string(char *stringAddress){
    for (unsigned char i = 0; i < strlen(stringAddress); i++)
        uart_send(stringAddress[i]);
}

> void uart_init(void){…

> void uart_send(unsigned char ch){…

/***** 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

davemclaughlin — screen /dev/tty.usbmodem2101 9600 ▸ SC...

```
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
Shall I compare thee to a summer's day?
```

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 (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.

```
/*************************************************************
Changed lines
t_sourcecode.c
 * 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 funtion2(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 funtion2(int);

etc..
```

```c
// 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);
```

```c
/* 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]);
}
```

```c
/*****************************************************************
Click to collapse the range.
 * uart_string.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.

https://www.techwalla.com/articles/how-to-use-putty-for-a-serial-connection

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