CG1112 Engineering Principles and Practices II for CEG

Week 7 Studio 1 – Serial Communications

(UNGRADED)

1. INTRODUCTION

In this studio we will look at establishing serial communications on the AVR microcontroller. The particular AVR you are using – the Atmega328P – has just a single Universal Synchronous Asynchronous Receiver Transmitter (USART), but some AVRs like the Atmega2560P have 4 such ports.

The USART port can be configured to work in 3 ways:

- Normal asynchronous operations, where data is transmitted without reference to a clock signal. The receiver end derives clocking signals directly from the data being transmitted. Normal asynchronous mode is compatible with the UART devices on other CPUs and MCUs, and with voltage-shifting, with the old RS232 standard found on industrial and older computer systems. It is also compatible with the USB ports on today's computers, through special chipsets made by FTDI (Future Technology Devices International).
- Double speed asynchronous operations. Data is transmitted and received at twice the speed specified in the baud rate UBRR register. In this mode the receiver will use only half its regular number of samples to derive the clocking signal, resulting in less reliable transmissions that require more reliable clocking circuits for your MCU.
- Synchronous mode. In this mode the USART uses a third line called the XCK line to transmit square pulses that help the receiver to time and recover the data being transmitted.

For EPP we will use the USART only in normal asynchronous operations, since double-speed and synchronous operations will only work between two AVR MCUs, making them significantly less useful for data transmission.

We will have four activities today:

Activity 1: Setting up the Raspberry Pi to run Arduino IDE.

Activity 2: Setting up USART communications in polling mode.

Activity 3: Setting up USART communications in interrupt mode

Activity 4: Establishing USART communications with the Raspberry Pi

2. TEAM FORMATION

For this studio you will work in teams of 2 or 3. However for Activities 2 and 3, two teams will need to work together.

3. SUBMISSION INSTRUCTIONS

This studio is ungraded and no submissions are required. However please attempt the questions.

4. ACTIVITY 1 – SETTING UP ARDUINO IDE ON THE PI

Step 1.

Ensure that the Raspberry Pi has been set up to allow access via ssh. This means that the Pi should be connected either directly to your PC via LAN cable, or via a hotspot. Please see Week 1 Studio 2 for how to do this. Determine the IP address of your Pi.

Step 2.

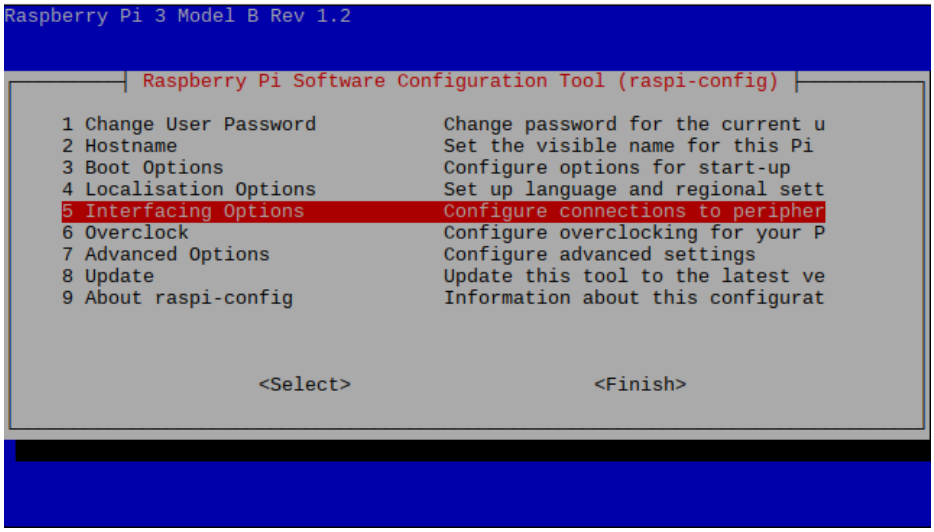
We will now enable Remote Desktop, allowing you to access the Pi's graphical interface remotely.

From your laptop, ssh into the Pi (note: Your laptop and Pi must be on the same hotspot, or connected via Ethernet cable). At the bash shell, type:

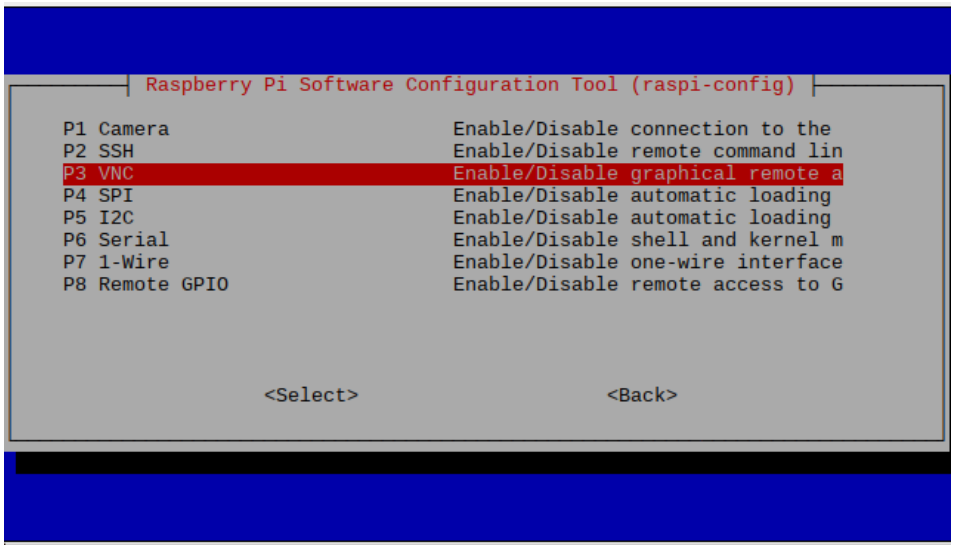
sudo raspi-config

Step 3.

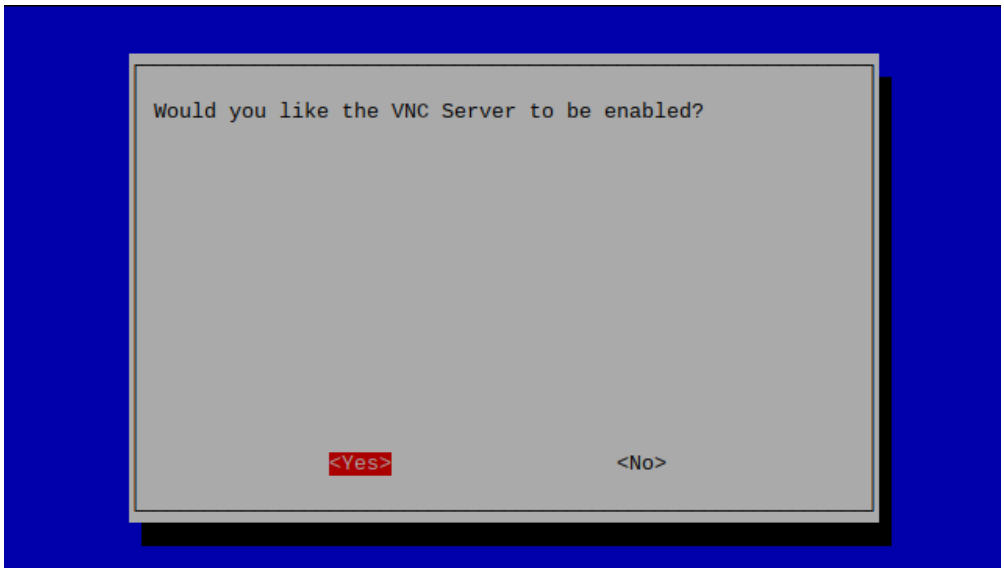
Choose "5 Interfacing Options".



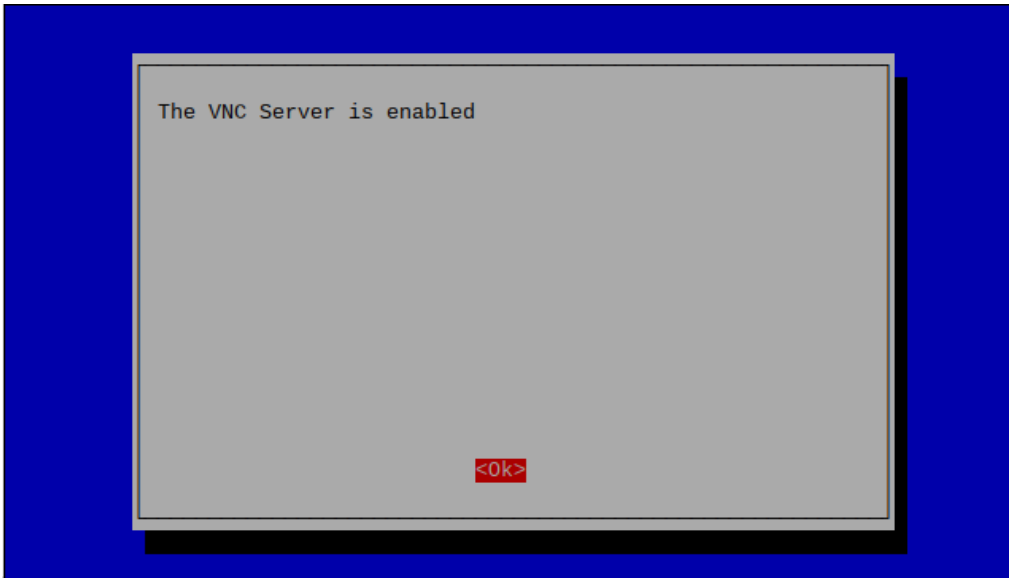
Now choose P3 VNC



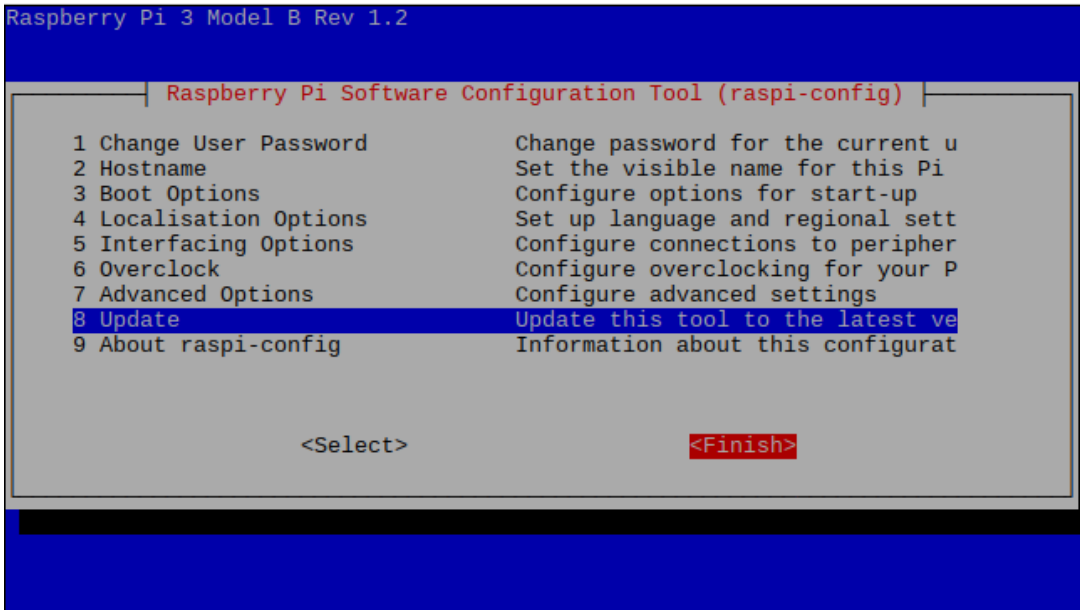
When asked if you would like VNC Server to be enabled, choose <Yes>:



You will get an acknowledgement:



Press <OK>, which will bring you back to the main screen. Finally select <Finish> to exit:

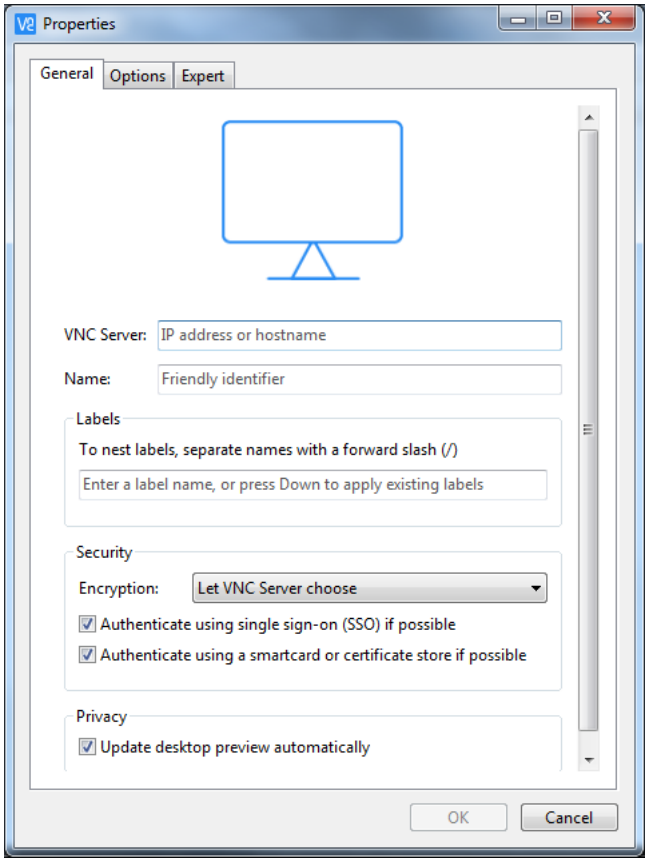


Step 4.

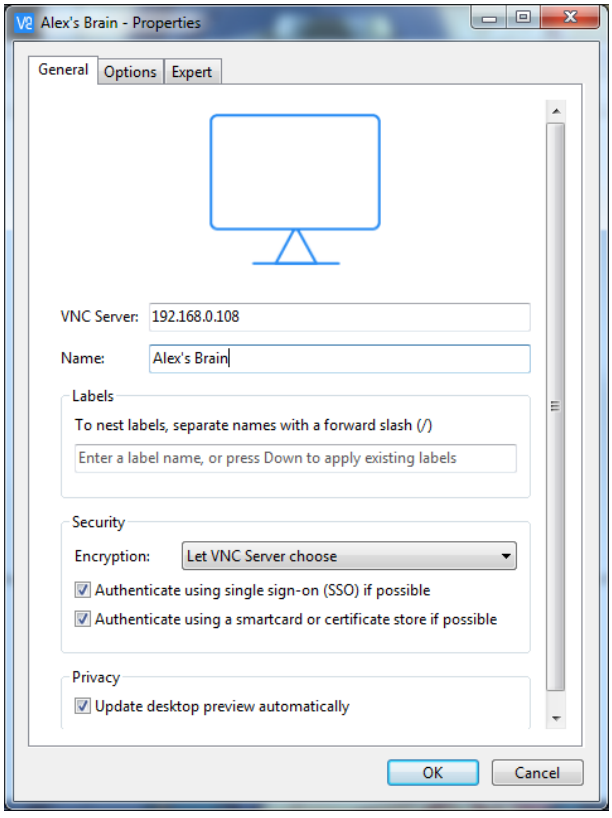
Download and install VNC Viewer from https://www.realvnc.com/en/connect/download/viewer/ on your laptop.

Step 5.

Start VNC Viewer. Select File->New Connection, giving you the following dialog box:

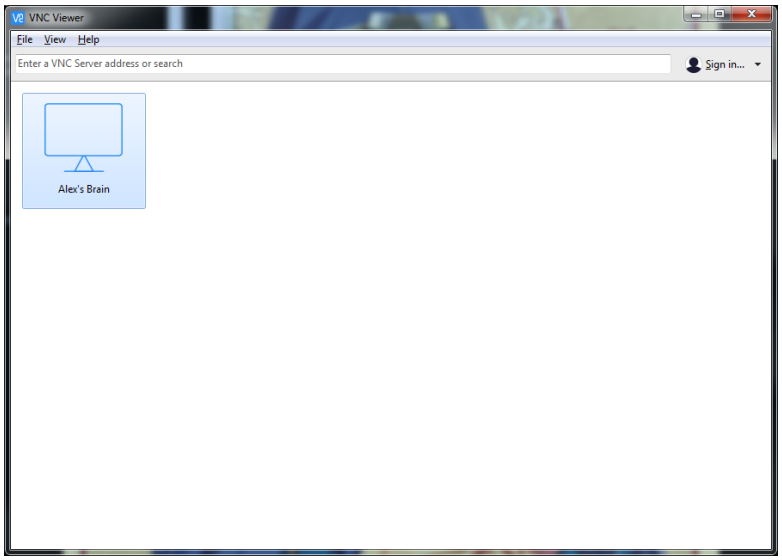


In VNC Server, fill in the IP address of the Pi. Enter a suitable name (e.g. “Vincen’ts Brain”) in the Name field. Leave the rest of the fields alone:

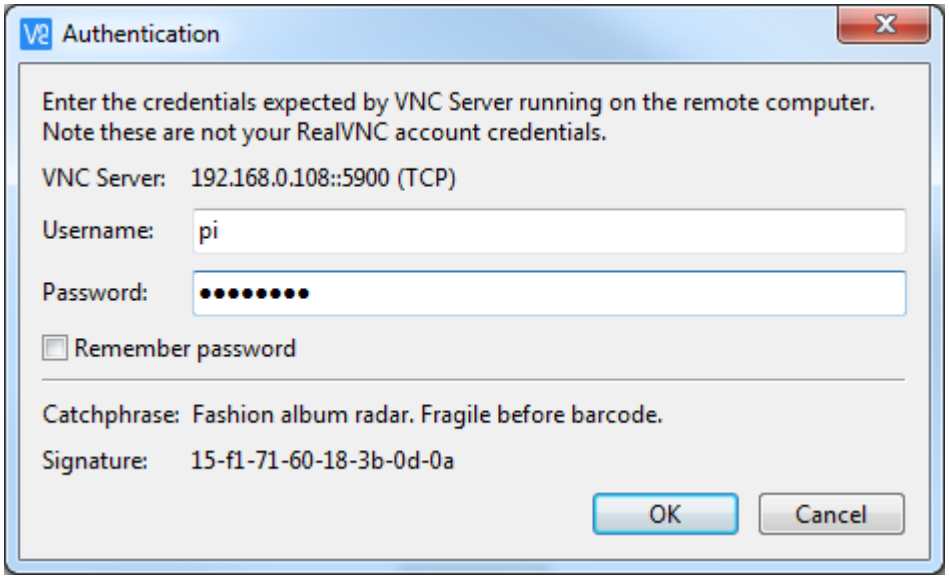


Click OK.

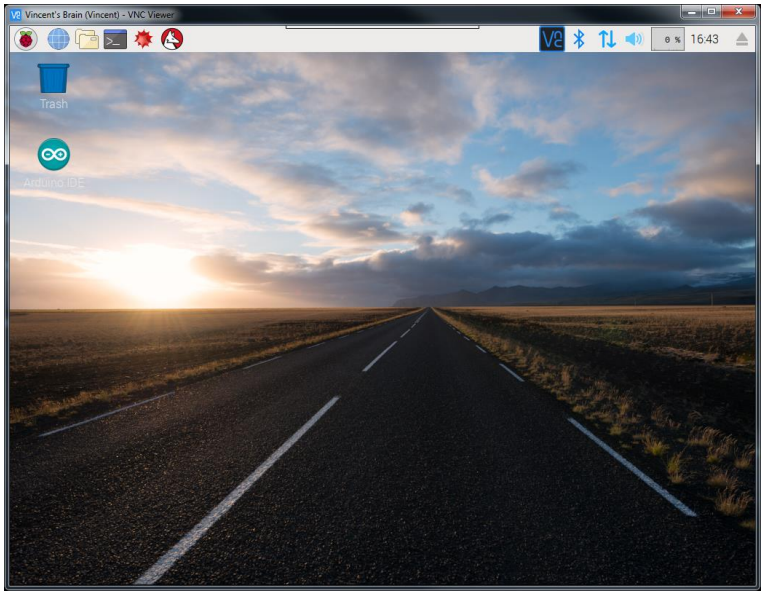
You will now see a new session in the VNC Viewer window.



Double click on the icon to start your session. You will be prompted for your username and password. Enter the same credentials you use for ssh:



If successful you should see the Pi's desktop on your laptop:



Step 6.

Using secure FTP or magic/telekinesis/telepathy, copy the Arduino-1.8.8-linuxarm.tar.xz over to the Pi. You can get this file from IVLE.

On Windows 10 Ubuntu Shell, MacOS or Linux, you can use sftp or scp. On Windows, you can use Cyberduck.

(You can also use the "Transfer files" function in VNC Viewer. Move your mouse to the top centre of the VNC View screen and a little box will drop down. Choose "Transfer files" and follow the instructions. It will transfer to your Desktop directory).

For magic/telekinesis/telepathy, please consult the Edgar Cayce Association for Research and Enlightenment or the Ordo Templi Orientis. I don't promise that anything they teach is actually usable, so use at your own risk.

Step 7.

Using the Remote Desktop or ssh, enter the following command to unpack the arduino-1.8.8-linuxarm.tar.xz file:

```
tar –xvf arduino-1.8.8-linuxarm.tar.xz
```

This will unpack the Arduino IDE into the arduino-1.8.8 directory.

Step 8.

To complete installation, change to the arduino-1.8.8 directory:

```
cd arduino-1.8.8
```

Now run the installer script:

```
./install.sh
```

It should say:

"Adding desktop shortcut and menu item for Arduino IDE... done!"

There may be a series of “touch: cannot touch… : No such file or directory” error messages. Ignore these.

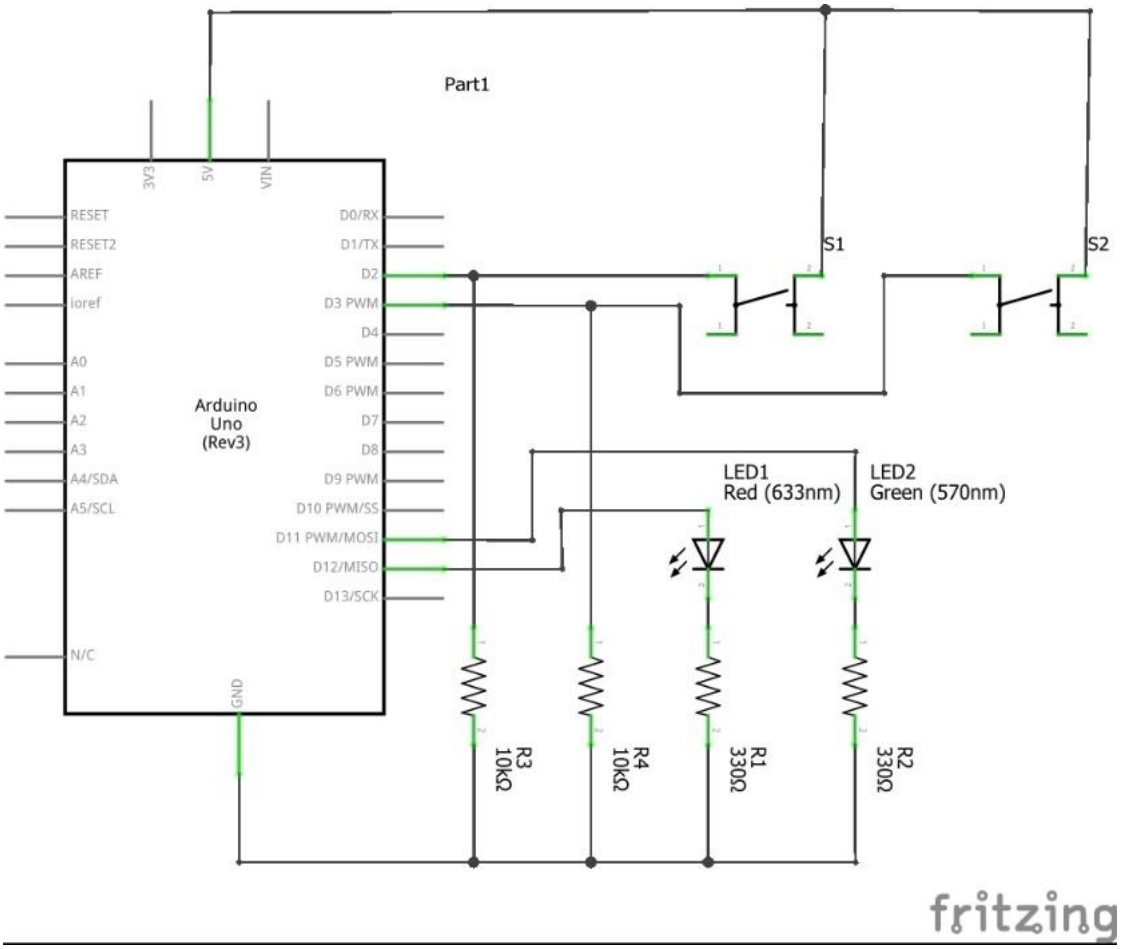
Now on the Remote Desktop you should see the Arduino IDE icon on the desktop. Double-click on this and use as per normal.

5. ACTIVITY 2 – USART IN POLLING MODE

In this activity two teams will be working together to control each other’s Arduino board. Do this part on your laptop.

Step 1.

Construct the following circuit:



Step 2.

Open the w7s1p1 sketch in Arduino IDE. Write in the body for setupEINT so that we trigger the INT0 and INT1 interrupts when pressing on buttons S1 and S2 respectively.

Set up the ISRs for the respective interrupt to set buttonVal to 1 when S1 is pressed, and to 2 when S2 is pressed.

Step 3.

We will now set up the USART to work at 57600 bps in 8N1 frame format. Consult the tables below to fill in the proper values into the UCSR0C register:

Question 1a.

UCSR0C							
UMSEL01	UMSEL00	UPM01	UPM00	USB0	UCSZ01	UCSZ00	UCPOL0

UMSEL0[1:0]	Mode
00	Asynchronous USART
01	Synchronous USART
10	Reserved
11	Master SPI (MSPIM)(1)

UPM0[1:0]	ParityMode
00	Disabled
01	Reserved
10	Enabled, Even Parity
11	Enabled, Odd Parity

USBS0	Stop Bit(s)
0	1-bit
1	2-bit

UCSZ0[2:0]	Character Size
000	5-bit
001	6-bit
010	7-bit
011	8-bit
100	Reserved
101	Reserved
110	Reserved
111	9-bit

Question 1b.

Write down the C statement to set USRC0C:

Question 1c.

We will now set the baud rate registers UBRR0L and UBRR0H. Calculate b for 57600 bps:

b =

Now write down the C statements to set UBRR0H and UBRR0L.

Question 1d.

Now implement setupSerial in your main.c program based on your answers above. **REMEMBER TO SET UCSR0A to 0!** (WHY?)

Step 4.

We will now set up startSerial. We are using polling mode. Fill in the appropriate values and write the C statements for startSerial below that.

UCSR0B							
RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB0	TXB0

Step 5.

We will now implement the sendData function. Open up the w7s1p2 project and implement the following:

Question 2a.

Using the register diagram for UCSR0A below as a guide, implement sendData by polling UDRE0 and writing to UDR0 when UDRE0 is 1:

Name: UCSR0A
Offset: 0xC0
Reset: 0x20
Property: -

Bit	7	6	5	4	3	2	1	0
	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Access	R	R/W	R	R	R	R	R/W	R/W
Reset	0	0	1	0	0	0	0	0

Question 2b.

Similarly, implement recvData by polling RXC0 and reading UDR0.

Step 6.

Implement the calls to setupEINT, setupUART and startUART, remembering to disable and re-enable interrupts before and after setting up (again, see lecture if you haven't the foggiest idea what I am saying).

Now Using the recvData and sendData functions declared above, implement the following in main.

- When we receive a 1, we flash the red LED continuously by calling flashRed.
- When we receive a 2, we flash the green LED continuously by called flashGreen.
- When you press button 1 (decide which button is button 1), write a 1 to the serial port.
- When you press button 2 (the button that isn't button 1), write a 2 to the serial port.

Note that this will run infinitely, therefore implement inside the while(1) loop.

Step 7.

Use Serial Monitor to verify that your program works. Ensure that your program is set up for 57600 bps 8N1, and that Serial Monitor is set up for 57600 bps communications.

- Sending a 1 from serial monitor will cause the red LED to flash.
- Sending a 2 will cause the green LED to flash.
- When you press button 1, you will receive a 1 on serial monitor.
- When you press button 2, you will receive a 2 on serial monitor.

Step 8.

Now find your friend at the same bench, and connect the two Arduinos together using the serial port.

- TX must connect to RX
- RX must connect to TX
- GND connect to GND

Now use the buttons on one board to control the LEDS on the other board.

ISN'T THAT COOL??

6. ACTIVITY 3 – USART IN INTERRUPT MODE

We will now repeat Activity 2 using interrupts instead of polling. Open up the w7s1p2 sketch in Arduino IDE.

Step 1.

Verify that setupUART for w7s1p2 is identical to w7s1p1, and implement setupUART. For now implement for 57600 8N1.

Step 2.

Now write the code for startUART, using the following register layout and tables as guides. You will use the UDR0 empty interrupt for transmitting, but FOR NOW set UDRIE0 to 0.

UCSR0B							
RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB0	TXB0

Bit	Label	Comment
7	RXCIE0	Set to 1 to trigger USART_RX_vect interrupt when a character is RECEIVED.
6	TXCIE0	Set to 1 to trigger USART_TX_vect interrupt when finish sending a character.
5	UDRIE0	Set to 1 to trigger USART_UDRE_vect interrupt when sending data register is empty.
4	RXEN0	Enable USART receiver.
3	TXEN0	Enable USART transmitter.
2	UCSZ02	Used with UCSZ01 and UCSZ00 (bits 2 and 1) bits in UCSR0C to specify data size.
1	RXB0	9th bit received when operating in 9-bit word size mode.
0	TXB0	9th bit to be transmitted when operating in 9-bit word size mode.

Use what you have above to write startUART.

Step 3.

Implement the ISRs for transmitting and receiving. We are NOT going to be using buffers. So you will need to get creative. Some hints:

- There is a dataRecv global variable that you can use to store the data you've read. Remember that you will read from the UDR0 register in the USART_TX_vect ISR.
- For the INT0 and INT1 handlers:
    - Set the dataSend global variable accordingly.
    - Set UDRIE0 to 1.
    - In the UDR Empty (USART_UDRE_vect) ISR:
        - Copy dataSend to UDR0.
        - Clear UDRIE0.

Using the hints above, implement the ISRs.

Step 4.

Use Serial Monitor as above to verify that your program works. Ensure that your program is set up for 57600 8N1, and that Serial Monitor is set up for 57600 bps communications.

7. ACTIVITY 4 – USART COMMUNICATIONS WITH RASPBERRY PI

We will now learn how to talk to the Arduino using the Raspberry Pi.

Step 1.

Connect to your Pi either using ssh or VNC Viewer. Using the file transfer function in VNC Viewer, Cyberduck, sftp, scp, telekinesis or magic, transfer the w7s2p3pi.c, serial.c and serial.h files to your Pi.

Step 2.

Ensure that your completed and working program from Activity 3 has been uploaded to the Arduino using your laptop. Ensure that it is set for 57600 8N1.

You can use the Arduino IDE to figure out the port name for the Arduino, or you can follow these steps:

1. With the Arduino NOT plugged into the Pi, on the Pi, type:

   ls /dev/tty*

   Take note of the list of devices that appear.

2. Plug in the Arduino into the Pi, and wait for a moment. Again type:

   ls /dev/tty*

   Look out for the new device that appears (for most of you it will be /dev/ttyACM0, /dev/ttyACM1, or /dev/ttyUSB0).

Step 3.

Open the w7s1p3pi.c program, and see how to use the serial.c library. There are 3 main functions you need to know:

void startSerial(portName, baudRate, byteSize, parity, stopBits, maxAttempts)	portName = name of port, e.g. /dev/ttyACM0 baudRate = Bit rate. You must specified it as Bxxxx, e.g. B9600 for 9600 bps, B57600 for 57600 ps, B115200 for 115200 bps, etc. byteSize = Size of each character. Legal values are 5, 6, 7, 8 or 9. parity = 'N' for no parity, 'E' for even parity, 'O' for odd parity. stopBits = # of stop bits. Either 1 or 2. maxAttempts = Maximum number of attempts to connect to the serial port before giving up.
int serialRead(char *buffer)	buffer = Array to write data read from the serial port to. Returns: # of characters received from serial port.
void serialWrite(char *buffer, int len)	buffer = Array of characters to write to serial port. len = Length of buffer in bytes.

Look at how startSerial, serialRead and serialWrite are being used in the program.

Question 3.

Notice that there is a fork() statement in w7s1p3pi.c. What does a fork do? Why do we need to use fork? (Hint: serialRead is a "blocking call" – i.e. If you call serialRead, the call will not return until there's a character being received).

Now edit the PORT_NAME macro to put in the name of the port you found earlier.

Step 4.

Compile the program with:

```
gcc w7s1p3pi.c serial.c –o w7s1
```

Run it with:

```
./w7s1
```

Following the instructions on the screen, you can enter “1” to flash the red LED, or “2” to flash the green LED. The program will then return the last button pressed on the Arduino. Press any button, then enter “1” or “2”, and see the latest button pressed on the Pi.

(NOTE: IF YOU WANT TO UPLOAD A SKETCH TO YOUR ARDUINO, YOU MUST EXIT THE w7s1 PROGRAM BY PRESSING CTRL-C. IF YOU DON’T THE UPLOAD WILL FAIL.)