QSD Monitoring Network

Measuring Vacuum Pressures

T. Barrett

July 2018

Measuring Vacuum Pressures

The pressures in the different QSD lab vacuum chambers are typically measured with an *Agilent UHV-24P* ionisation gauge, which is attached to a spare DN40CF port on the vacuum system. The gauge will then be interfaced using an *Agilent XGS-600* controller, shown in the photograph in Fig. 1 a). The controller gives a reading of the pressure on its front screen, but as described on page 67 of the user manual the pressure can also be read by an external device communicating over a serial protocol. Page 37 of the manual describes that the protocol can be set on the controller specifically to the RS232 serial standard, and the baud rate to 9600, as shown in Fig. 1 b). Serial communication refers to the practice of sending data one bit at a time, and the baud rate is the number of bits per second that are transferred during communication. Figure 1 c) is taken from page 12 of the user manual, and indicates that the 9-pin D-sub connector (number 4) on the back panel of the controller is used for serial communication.

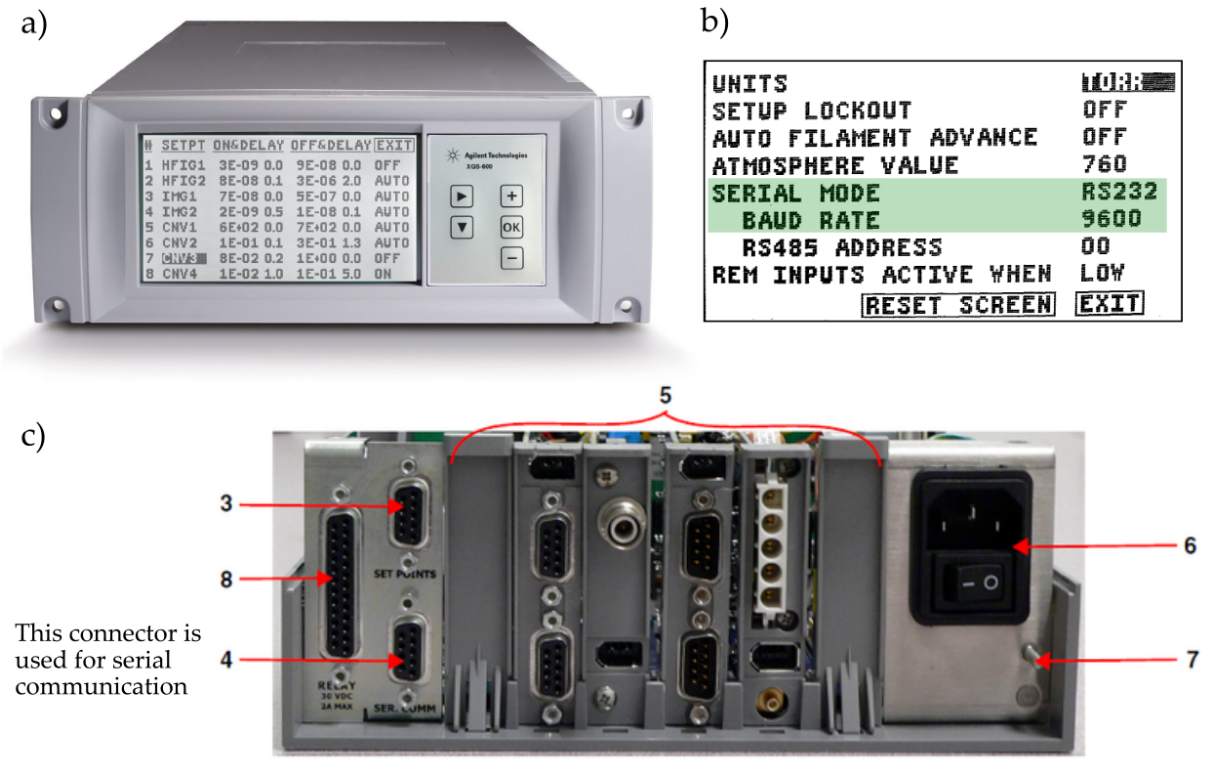


FIGURE 1: Photographs from the Agilent XGS-600 User Manual: a) The front of the gauge controller, b) The controller menu screen, showing settings for communication over RS232 serial protocol at 9600 baud rate, c) Back panel of the controller, indicating connector number 4 as the one used for serial communication with an external device.

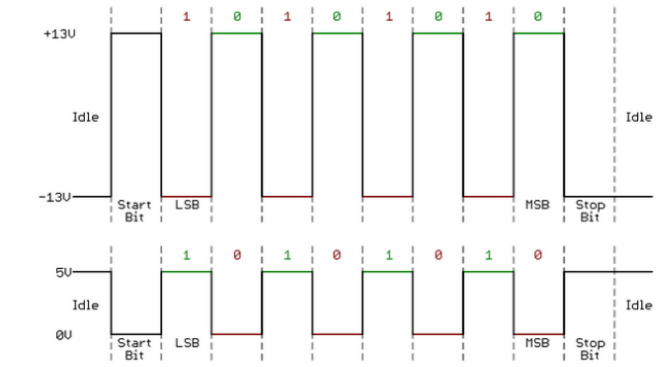
The pin assignments for the serial connector can be found on page 61 of the manual. The relevant ones for our purposes are:

- **TXD OUTPUT** (pin 2) - For data transmission from controller to device
- **RXD INPUT** (pin 3) - For data being received by controller from device
- **GND** (pin 5) reference for both pins.

The connector is a female socket type, and the pin numbering looking forwards directly at the back panel are given in Fig. 3 c).

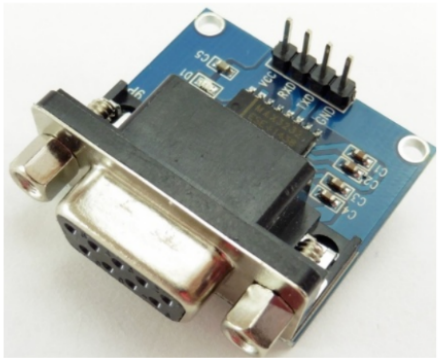
There is now a problem that the RS232 standard typically uses -13 V to represent a logic *HIGH* signal, and +13 V for a logic *LOW* signal. This is in contrast to transistor-transistor logic (TTL) that the Arudino can operate on, which often uses +5 V or +3.3 V for *HIGH* and 0 V for *LOW*, with the two cases being compared in Fig. 2 a). More information can be found here: `https://www.sparkfun.com/tutorials/215`. Not only are the voltage levels different (the RS232 voltages can damage the Arudino board), but there is also an inversion of the signal.

a)



This timing diagram shows both a TTL (bottom) and RS-232 signal sending 0b01010101

b)



c)

FIGURE 2: a) Difference in voltage levels and polarity between RS232 serial and TTL protocols. b) The MAX3232 adapter for converting between RS232 and TTL levels. c) The SP3232 adapter for converting between RS232 and TTL levels.

In order to allow the Arduino to communicate with the gauge controller, two batches of TTL-to-RS232 converters were purchased - one from *Proto-PIC* and one from *HobbyTronics*, and which ones to buy depends only on availability. The ones from Proto-PIC are based on the MAX3232 chip, while the ones from HobbyTronics are based on the SP3232 chip - both perform the necessary level shifting and signal inversion, and can be seen in Fig. 2 b) and c), respectively. When connecting up a device to communicate with the gauge, it is important to make sure that the RX (receiving) and TX (transmitting) lines are the correct way around. A TX or RX pin is always from the point of view of the device itself. So, for example, the TX line of the gauge must be connected to the RX line of the Arduino, allowing it to receive the transmitted data.

A schematic of the routing connections on the MAX3232 breakout board has been sketched

a) b)

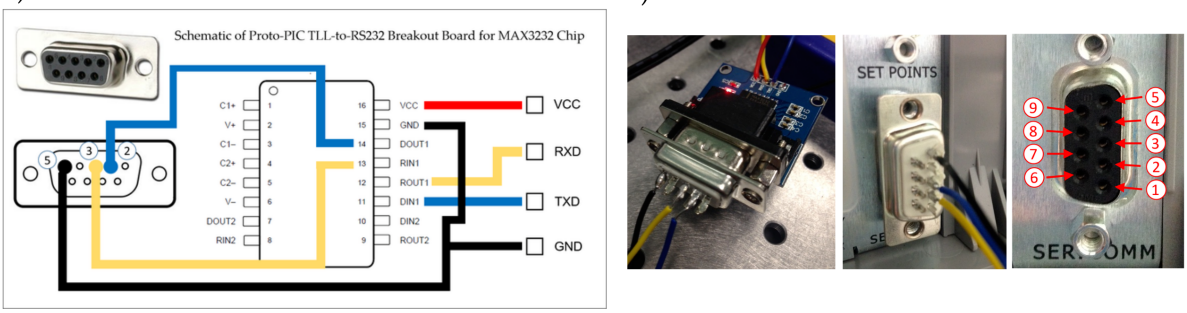


FIGURE 3: a) Pin connections on the MAX3232 adapter board's PCB. b) Connecting the MAX3232 adapter to a 9 pin subD connector, for interfacing with the gauge controller.

in Fig. 3 a). The connections that must be made are as follows:

For Transmitting Data From Gauge Controller to Arduino using MAX3232 Adapter Board

Gauge controller **TXD OUTPUT**, pin 2 controller SubD
(transmits data from gauge in RS232 level)
↳
MAX3232 pin 13, **RIN1 RS232 Data Input** (goes to pin 3 adapter SubD)
(receives transmitted data in RS232 level)
↳
MAX3232 pin 12, **ROUT1 TTL Data Output** (goes to pin labelled **RXD** on board)
(outputs transmitted data in TTL level)
↳
Arduino Software Serial **RX** pin
(receives data in TTL level)

For Transmitting From Arduino to Gauge Controller using MAX3232 Adapter Board

Arduino Software Serial **TX** pin
(transmits data in TTL level)
↳
MAX3232 pin 11, **DIN1 TTL Data Input** (goes to pin labelled **TXD** on board)
(receives transmitted data in TTL level)
↳
MAX3232 pin 14, **DOUT1 RS232 Data Output** (goes to pin 2 adapter SubD)
(outputs transmitted data in RS232 level)
↳
Gauge controller **RXD INPUT**, pin 3 controller SubD
(receives data from Arduino in RS232 level)

If using the SP3232 adapter board, the connections that must be made are as follows:

For Transmitting Data From Gauge Controller to Arduino using SP3232 Adapter Board

Gauge controller **TXD OUTPUT**, pin 2 controller SubD
(transmits data from gauge in RS232 level)
↳
SP3232 pin 9, **R2OUT RS-232 Receiver Input** (goes to pin 3 adapter SubD)
(receives transmitted data in RS232 level)
↳
SP3232 pin 10, **R2OUT TTL Reciever Output** (goes to pin labelled **TXD** on board)
(outputs transmitted data in TTL level)
↳
Arduino Software Serial **RX** pin
(receives data in TTL level)

For Transmitting From Arduino to Gauge Controller using SP3232 Adapter Board

Arduino Software Serial **TX** pin
(transmits data in TTL level)
↳
SP3232 pin 11, **T2IN TTL Driver Input** (goes to pin labelled **RXD** on board)
(receives transmitted data in TTL level)
↳
SP3232 pin 8, **T2OUT RS-232 Driver Output** (goes to pin 2 adapter SubD)
(outputs transmitted data in RS232 level)
↳
Gauge controller **RXD INPUT**, pin 3 controller SubD
(receives data from Arduino in RS232 level)

Once the necessary connections have been made, we can use the Arduino *SoftwareSerial()* library to achieve the serial communication. This can take place on regular digital pins, as opposed to the USB serial communication that is normally done (the tutorial here is useful: `http://forum.arduino.cc/index.php?topic=58035.0`). Note that the *SoftwareSerial()* documentation states that not all digital pins are capable of change interrupts, and so cannot all be used for RX functionality - check before use.

After setting up the serial port, it is time to read the pressure from the gauge controller. From page 67 of the XGS-600 manual, it can be seen that the command format for communication is of the form:

```
1 # {XGS-600 address} {command number} {optional data} {carriage return}
```

So, to read the pressure, one should send the command:

```
#000F\r
```

Here, the first **00** is the default for RS232 protocol, the second **0F** is the command for reading pressure, and the final **\r** is a carriage return. The pressure value will then be returned in the scientific format **X.XXXE-XX**. The user manual says do not attempt to read pressure more frequently than once every 100 ms.

Running the code shown in the section **Arduino Code** below prints the current vacuum pressure from the gauge to the Arudino serial USB port. Figure 4 shows the reading on gauge front panel matching the values in the serial port, with the latter offering more digits of precision.

a)

b)

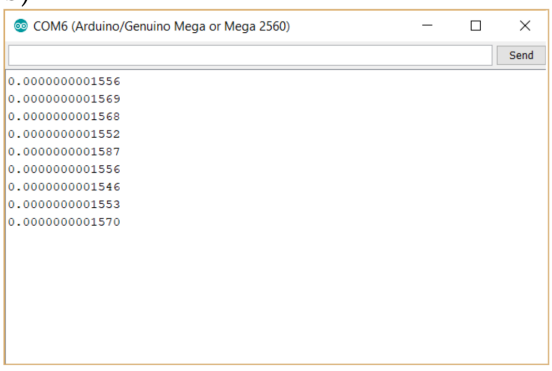


FIGURE 4: a) Photograph of gauge controller's front panel. b) The output from the Arduino USB serial monitor, showing the values matching those from the controller screen.

Arduino Code

```
/* 03_VacuumPressureReader
 * Code for reading pressure from XGS-600 Agilent gauge controller
*/

#include <SoftwareSerial.h>

String payloadStr;
String pressure_string = "";
String exponent_string = "";
float pressure_val;
float exponent_val;
float total_val;

#define RX  10 //Define the pins for SoftwareSerial
#define TX  11
SoftwareSerial mySerial(RX, TX);

void setup() {
  mySerial.begin(9600);  // set the baud rate for the SoftwareSerial port
  Serial.begin(9600);    // set the baud rate for USB serial port
}

void loop() {

    payloadStr = "";

    mySerial.write("#000F\r"); // Send command to controller for pressure readings
    delay(100);                // Give controller chance to respond

    pressure_string = "";
    exponent_string = "";

    // Read data transmitted from controller, character at a time
    while (mySerial.available()) {
      char a  = mySerial.read();
      if (isDigit(a) || a == '.')
        pressure_string += a;
      else if (a == 'E')
        break;
      delay(10);
    }
    delay(10);

    // Read exponent of pressure
    while (mySerial.available()) {
      char a  = mySerial.read();
      if (isDigit(a))
        exponent_string += a;
      delay(10);
    }

    // Convert engineering notation into floating point for sending back
    if (pressure_string.length() > 0) {
      pressure_val = pressure_string.toFloat();
      exponent_val = exponent_string.toFloat();
      total_val = pressure_val * pow(10, -exponent_val);
      payloadStr = String(total_val, 13);
    }
    else {
      payloadStr = "0.0";
    }

    Serial.println(payloadStr);
    delay(1000);
}
```