**Communication between UART and I2C**

**UART**

Universal Asynchronous Receiver Transmitter, is one of the most used serial protocols. Most controllers have a hardware UART on board. It uses a single data line for transmitting and one for receiving data. Most often 8-bit data is transferred, as follows: 1 start bit(low level), 8 data bits and 1 stop bit(high level). The low level start bit and high level stop bit mean that there's always a high to low transition to start the communication. That's what describes UART. No voltage level, so you can have it at 3.3 V or 5 V, whichever we microcontroller uses. Note that the microcontrollers which want to communicate via UART have to agree on the transmission speed, the bit-rate, as they only have the start bit's falling edge to synchronize. That's called asynchronous communication.

**I2C**

[I2C](http://www.i2c-bus.org/) is a serial communication protocol that only requires two signal lines. It was designed for communication between chips on a PCB. I2C was originally designed for 100kbps communication but [faster data transmission modes](http://dlnware.com/theory/I2C-Bus-Speed) have been developed over the years to achieve speeds of up to 3.4Mbit. The I2C protocol has been established as an official standard, which provides for good compatibility among I2C implementations and good backward compatibility.

The I2C Protocol uses only two bi-directional signal lines to communicate with all of the devices on the I2C bus. The two signals used are:

Serial Data Line (SDL)

Serial Data Clock (SDC)

The reason that I2C can use only two signals to communicate with a number of peripherals is in how communication along the bus is handled. Each I2C communication starts with a 7-bit (or 10-bit) address that calls out the address of the peripheral the rest of the communication is meant to receive the communication. This allows multiple devices on the I2C bus to play the role of the master device as the needs of the system dictate. To prevent communication collisions, the I2C protocol includes arbitration and collision detection capabilities which allow smooth communication along the bus.

With all of these advantages, I2C also has a [few limitations](http://www.i2c-bus.org/i2c-primer/common-problems/) that may need to be designed around. The most important I2C limitations include:

* Since only 7-bits (or 10-bits) are available for device addressing, devices on the same bus can share the same address. Some devices are capable of configuring the last few bits of the address, but this still imposes a limitation of devices on the same bus.
* Only a few limited communication speeds are available and many devices do not support the transmission at higher speeds. Partial support for each speed on the bus is required to prevent slower devices from catching partial transmissions that will result in operational glitches.
* The shared nature of the I2C bus can result in the entire bus hanging when a single device on the bus stops operating. Cycling the power to the bus can be used to restart the bus and restore proper operation.
* Since devices can set their communication speed, slower operational devices can delay the operation of faster speed devices.
* I2C draws more power than other serial communication busses due to the open-drain topology of the communication lines.
* The limitations of the I2C bus typically limit the number of devices on a bus to around a dozen devices.

Applications

The I2C bus is a great option for applications that require low cost and simple implementation rather than high speed. For example, reading certain memory ICs, accessing DACs and ADCs, [reading sensors](https://www.lifewire.com/applications-of-inductors-818816), transmitting and controlling user-directed actions, reading hardware sensors, and communicating with multiple microcontrollers are common uses of the I2C communication protocol.

**Ps2 Controller Interfacing**

Applications for PS2 Controllers

So where could you use a Playstation controller aside from on a Playstation console?

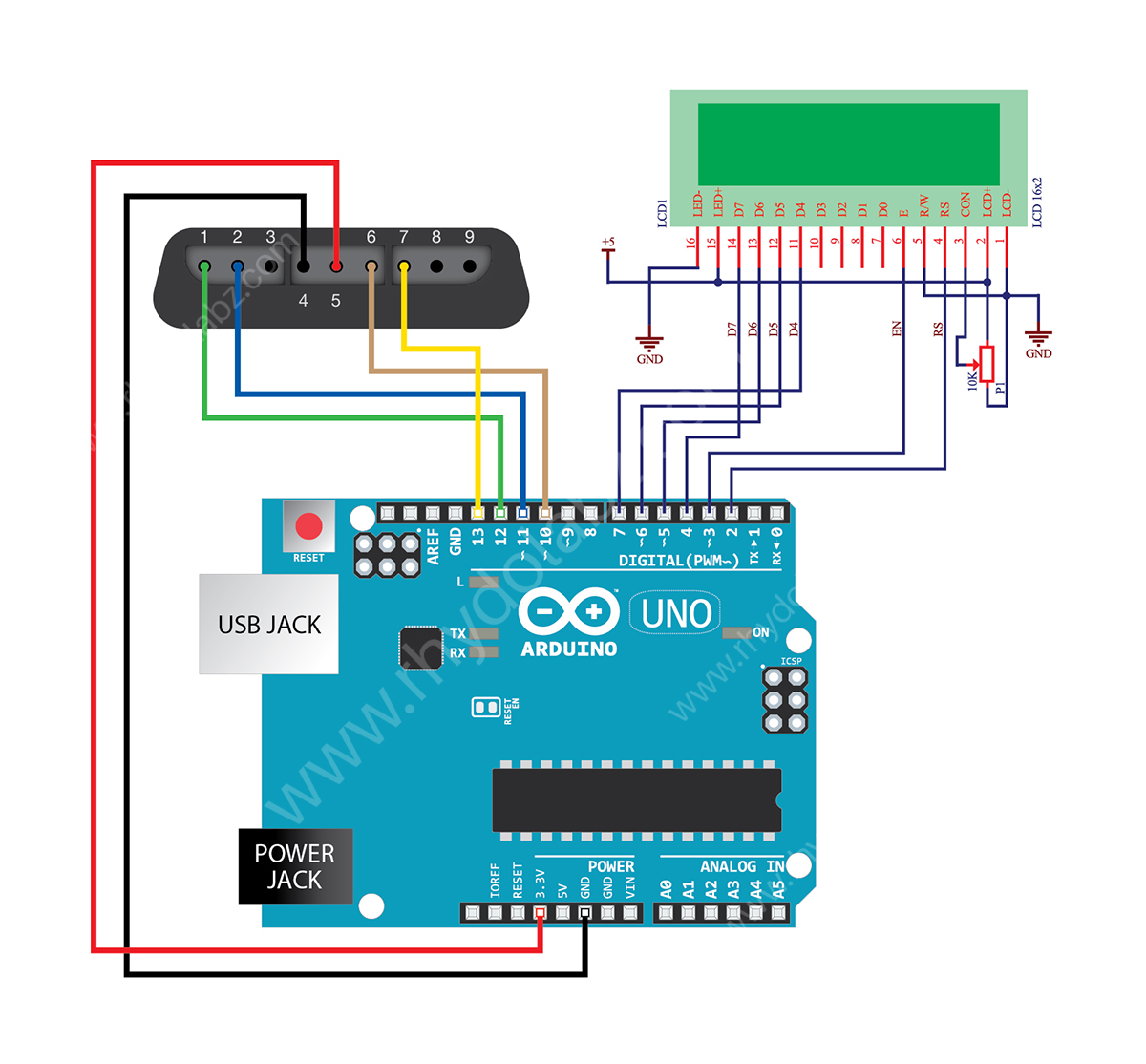
* Controlling a wheeled or tracked vehicle's motion.
* Controlling a robotic arm.
* Interacting with a computer.
* Controlling a pan and tilt camera mount.
* DIY radio control system by using a bluetooth module, or radio module to transmit the commands from the Playstation 2 controller to another Arduino which is controlling the vehicle.



1. **DATA:** This is the data line from Controller to PS2. This is an open collector output and requires a pull-up resistor (1 to 10k, maybe more). (A pull-up resistor is needed because the controller can only connect this line to ground; it can’t actually put voltage on the line).
2. **COMMAND:** This is the data line from PS2 to Controller.
3. **VIBRATION MOTOR POWER**
4. **GND:** Ground
5. **VCC:** VCC can vary from 5V down to 3V .
6. **ATT:** ATT is used to get the attention of the controller. This line must be pulled low before each group of bytes is sent / received, and then set high again afterwards. This pin consider as “Chip Select” or “Slave Select” line that is used to address different controllers on the same bus.
7. **CLK:**500kH/z, normally high on. The communication appears to be SPI bus.
8. **Not Connected**
9. **ACK:**Acknowledge signal from Controller to PS2. This normally high line drops low about 12us after each byte for half a clock cycle, but not after the last bit in a set. This is a [open collector output](http://en.wikipedia.org/wiki/Open_collector) and requires a pull-up resistor (1 to 10k, maybe more).

**PS2 Signals:**

PS2 wireless controller communicates with Arduino using a protocol that is basically SPI. The play station sends a byte at the same time as it receives one (full duplex) via serial communication. There’s a clock (SCK) to synchronize bits of data across two channels: DATA and CMD. Additionally, there’s a “Attention” (ATT) channel which tells the slave whether or not it is “active” and should listen to data bits coming across the CMD channel, or send data bits across the DATA channel (Reasonably, only one slave device should be active at a time) . The PlayStation 2 actually uses this plus an additional line that is not specifically part of the SPI protocol – an “Acknowledge” (ACK) line.  
The clock is held high until a byte is to be sent. It then drops low (active low) to start 8 cycles during which data is simultaneously sent and received. The logic level on the data lines is changed by the transmitting device on the falling edge of clock. This is then read by the receiving device on the rising edge allowing time for the signal to settle. After each Command is received from the controller, that controller needs to pull ACK low for at least one clock cycle. If a selected controller does not ACK the PS2 will assume that there is no controller present. LSBs (least significant bits) are transmitting first.

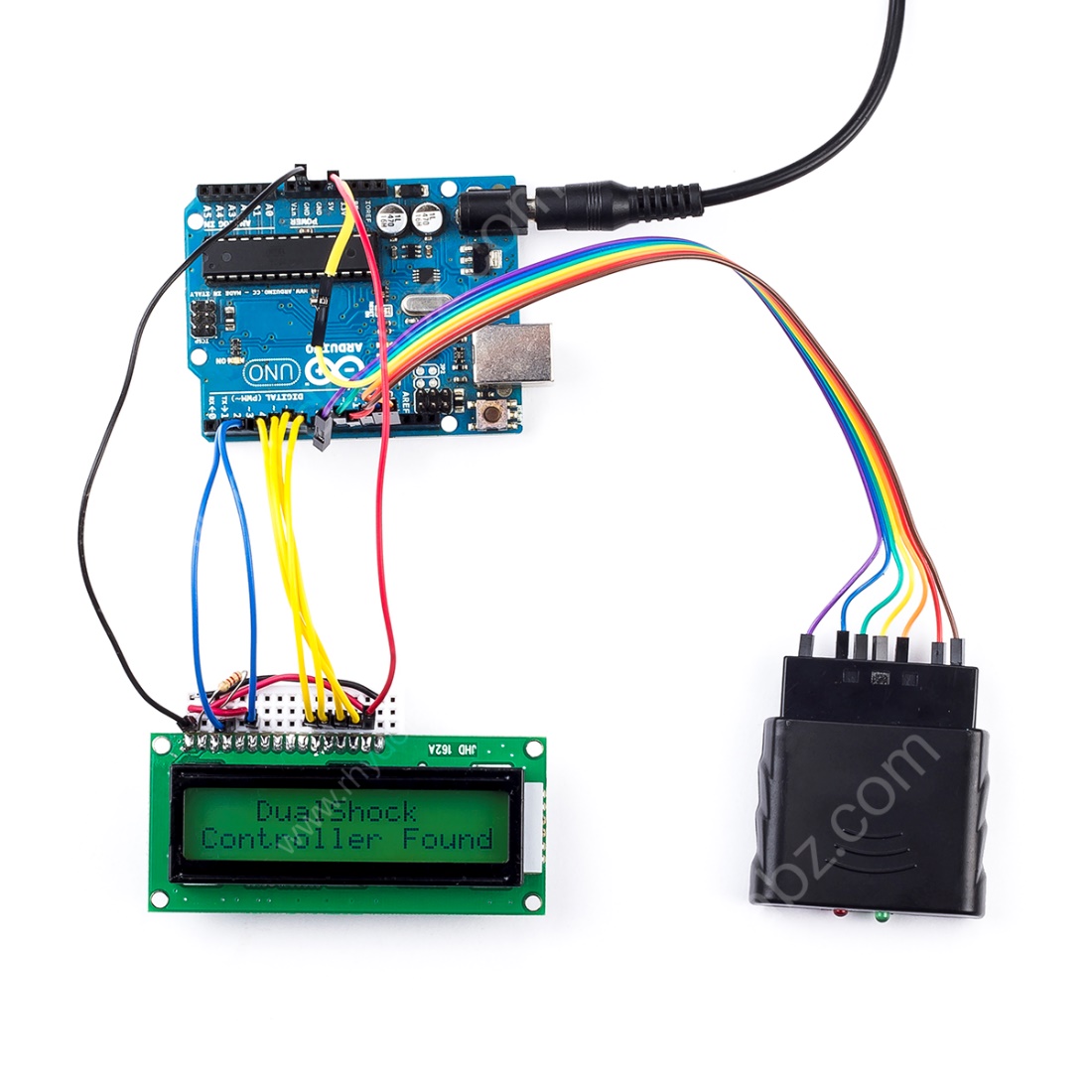


Here we have interfaced the PS2 Wireless Controller with an Arduino. Upon each button press the Arduino receives the RF signal on the PS2 receiver and displays the it on the alphanumeric LCD module. We followed the standard PS2 protocol for realizing the communication algorithm, identical to the SPI protocol. Our program on the arduino detects and reads the button presses only, pressure values are not read. Analog stick state values are continuously displayed on the LCD module.

**Connection Details:**

The PS2 receiver CLK line and ATT lines are held normally high. The  ATT operates like the Slave Select line under SPI. You pull it low to tell the controller you are talking to it and then send it back high once a communications cycle is complete. CMD is the data line to the controller and DATA is the data coming from the controller. Here in our application we are not using the acknowledge pin.

Well that was so much until here, so here it is the photograph of what we actually did with that really awesome PS2 controller and the project code run on Arduino:



Arduino Code:

|  |
| --- |
| #include <PS2X\_lib.h>                         /\* PS2 Controller Library \*/  #include <LiquidCrystal.h>                    /\* LiquidCrystal Library \*/  PS2X ps2x;                                    /\* create PS2 Controller Class\*/  byte Type = 0;  byte vibrate = 0;  int RX=0,RY=0,LX=0,LY=0;  LiquidCrystal lcd(2,3,7, 6, 5, 4);            /\* initialize the library with the numbers of the interface pins\*/  void setup(){     lcd.begin(16, 2);                             /\* 16X2 lcd display \*/     ps2x.config\_gamepad(13,11,10,12, true, true); /\* setup pins and settings:  GamePad(clock, command, attention, data, Pressures?, Rumble?) check for error\*/     Type = ps2x.readType();                       /\* Reading type of the PS2 Ccontroller \*/     if(Type==1){                                  /\* Type 1 is Duel shock controller \*/        lcd.setCursor(0, 0);                       /\* Setting display position\*/        lcd.print("   DualShock    ");             /\* display if the controller is duel shock\*/        lcd.setCursor(0, 1) ;        lcd.print("Controller Found");        delay(1000);        lcd.clear();     }  }  void loop(){       ps2x.read\_gamepad(false, vibrate);   /\* read controller and set large motor to spin at 'vibrate' speed \*/     lcd.setCursor(0, 0);                 /\* Position the LCD cursor \*/     lcd.print("Stick values:   ");       /\* Display analog stick values \*/     lcd.setCursor(0, 1);     LY = ps2x.Analog(PSS\_LY);          /\* Reading Left stick Y axis \*/     LX = ps2x.Analog(PSS\_LX);          /\* Reading Left stick X axis \*/     RY = ps2x.Analog(PSS\_RY);          /\* Reading Right stick Y axis \*/     RX = ps2x.Analog(PSS\_RX);          /\* Reading Right stick X axis \*/     if((LY <= 9))                      /\* standardize to 3 digit by checking less than 10 \*/        lcd.print("00");                /\* eg: if LY= 5 then it display as "005" in lcd \*/     if((LY >= 9 &&LY <= 99))           /\* standardize to 3 digit by checking between 10-99 \*/        lcd.print("0");                 /\* eg: if LY= 55 then it display as "055" in lcd \*/     lcd.print(LY,DEC);                 /\* display left analog stick Y axis \*/     lcd.print(",");                    /\* separate values using comma \*/     if((LX <= 9))                      /\* standardize to 3 digit by checking less than 10 \*/        lcd.print("00");                /\* eg: if LX= 5 then it display as "005" in lcd \*/     if((LX >= 9 && LX<=99))            /\* standardize to 3 digit by checking between 10-99 \*/        lcd.print("0");                 /\* eg: if LX= 55 then it display as "055" in lcd \*/     lcd.print(LX,DEC);                 /\* display left analog stick X axis \*/     lcd.print(",");                    /\* separate values using comma \*/     if((RY <= 9))                      /\* standardize to 3 digit by checking less than 10 \*/        lcd.print("00");                /\* eg: if RY= 5 then it display as "005" in lcd \*/     if((RY >= 9 &&RY<=99))             /\* standardize to 3 digit by checking between 10-99 \*/        lcd.print("0");                 /\* eg: if RY= 55 then it display as "055" in lcd \*/     lcd.print(RY,DEC);                 /\* display Right analog stick Y axis               \*/     lcd.print(",");                    /\* separate values using comma \*/     if((RX <= 9))                      /\* standardize to 3 digit by checking less than 10 \*/        lcd.print("00");                /\* eg: if RX= 5 then it display as "005" in lcd \*/     if((RX >= 9 &&RX <= 99))           /\* standardize to 3 digit by checking between 10-99 \*/        lcd.print("0");                 /\* eg: if RX= 55 then it display as "055" in lcd \*/     lcd.print(RX,DEC);                 /\* display Right analog stick X axis          \*/     lcd.print(" ");     if(ps2x.NewButtonState()) {        /\* will be TRUE if any button changes state  \*/        lcd.setCursor(0, 0);        if(ps2x.Button(PSB\_START))      /\* will be TRUE as long START button is pressed \*/           lcd.print("START PRESSED   ");        if(ps2x.Button(PSB\_SELECT))             /\* will be TRUE as long SELECT button is pressed \*/           lcd.print("SELECT PRESSED  ");        if(ps2x.Button(PSB\_PAD\_UP))             /\* will be TRUE as long as UP button is pressed \*/           lcd.print("UP PRESSED      ");        if(ps2x.Button(PSB\_PAD\_RIGHT))          /\* will be TRUE as long as UP button is pressed \*/           lcd.print("RIGHT PRESSED   ");        if(ps2x.Button(PSB\_PAD\_LEFT))           /\* will be TRUE as long as LEFT button is pressed \*/           lcd.print("LEFT PRESSED    ");        if(ps2x.Button(PSB\_PAD\_DOWN))           /\* will be TRUE as long as DOWN button is pressed \*/           lcd.print("DOWN PRESSED    ");        if(ps2x.Button(PSB\_L1))                 /\* will be TRUE as long as L1 button is pressed \*/           lcd.print("L1 pressed      ");        if(ps2x.Button(PSB\_R1))                 /\* will be TRUE as long as R1 button is pressed \*/           lcd.print("R1 pressed      ");        if(ps2x.Button(PSB\_L2))                 /\* will be TRUE as long as L2 button is pressed \*/           lcd.print("L2 pressed      ");        if(ps2x.Button(PSB\_R2))                 /\* will be TRUE as long as R2 button is pressed \*/           lcd.print("R2 pressed      ");        if(ps2x.Button(PSB\_L3))                 /\* will be TRUE as long as L3 button is pressed \*/           lcd.print("L3 pressed      ");        if(ps2x.Button(PSB\_R3))                 /\* will be TRUE as long as R3 button is pressed \*/           lcd.print("R3 pressed      ");        if(ps2x.Button(PSB\_GREEN))              /\* will be TRUE as long as GREEN/Triangle button is pressed \*/           lcd.print("Triangle pressed");        if(ps2x.Button(PSB\_BLUE))                /\* will be TRUE as long as BLUE/CROSS/X button is pressed \*/           lcd.print("X pressed       ");         if(ps2x.Button(PSB\_RED))               /\* will be TRUE as long as RED/Circle button is pressed \*/           lcd.print("Circle pressed  ");        if(ps2x.Button(PSB\_PINK))               /\* will be TRUE as long as PINK/Squre button is pressed \*/           lcd.print("Square pressed  ");        delay(700);     }     else;  } |