

Features

- Connects your Arduino Uno or Arduino Mega to an RS-485 network
- Isolation protects your device and you
- True fail-safe RS-485 transceiver copes with common network configuration problems
- Driven or automatically generated transmission direction control
- Suits 3.3V and 5V Arduinos
- Open Source Hardware

Isolated RS-485 Shield

RS-485 is a long distance serial communication interface commonly used in industrial equipment, appliances and control systems. Protocols including Modbus, DMX and Profibus can all be carried on RS-485.

The Ocean Controls Isolated RS-485 shield allows a 5V Arduino to connect to an RS-485 bus.

RS-485 is often used by high power equipment (like heating and cooling controllers, inverters and motor drives), which for safety reasons you wouldn't want to directly wire an Arduino to. RS-485 allows for long distances but long distances increase the risk of ground imbalances (different devices on the bus not being at the same potential) and induced currents and noise. These issues pose a risk to your Arduino and to yourself.

The shield uses optoisolators and an isolated DC-DC converter to keep dangerous voltages away from your Arduino and your fingers.

RS-485 vs. RS-232

Why use RS-485 instead of the more common RS-232? There are two main reasons: multidrop capability and distance achievable.

RS-232 is a point to point link. It connects one device to one device. RS-485 is a bus that many devices can connect to and share. This makes it suitable for use in monitoring, control and automation configurations. Often a single master controller talks to many different devices on a bus.

RS-485 can run much longer distances than RS-232. Networks up to 1,200 metres long can be implemented. RS-232 typically reaches about 15 metres before special techniques are required to go farther.

The main disadvantage of RS-485 is that it's half-duplex. Data can go both ways but only one way at a time. Devices that use RS-485 have to take this into account as part of the communication protocol design.

RS-485 As a Bus and Transmission Line

The multidrop capability and long distances of RS-485 are achievable because RS-485 uses differential signalling on a transmission line.

In RS-485, two wires carry the signal. One wire carries the mirror image of the signal on the other wire. That is, when one wire has a high voltage on it, the other has a low voltage and vice versa. A receiver is measuring both signal wires and looking for the difference between them.

When the difference is more than 200mV positive, the receiver outputs a high voltage. When the difference is less than -200mV, the receiver outputs a low voltage.

This greatly reduces the effect of electrical noise. The signal wires are run next to each other from one device to another. If those wires pass near a source of electrical noise, both wires are affected in the same way. The receiver is looking for signals that are a difference between the lines, so the noise, which is common to both wires, is ignored.

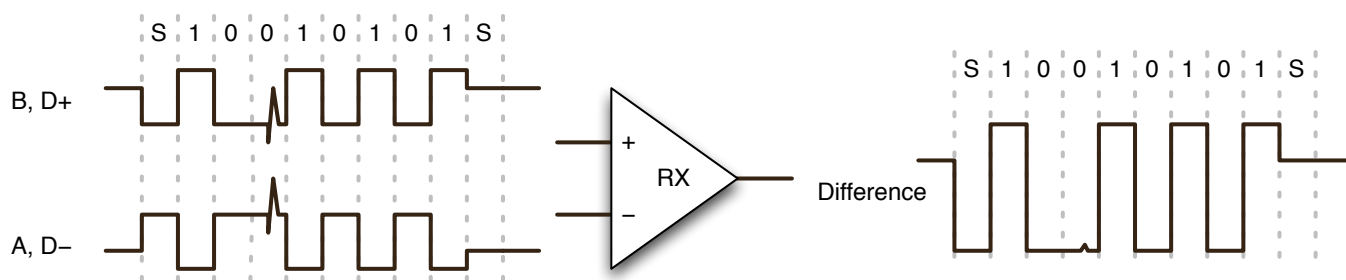


Figure 1 - A differential receiver greatly reduces the effect of noise

RS-485 is typically carried on a transmission line with 120Ω characteristic impedance. Unshielded twisted pair for Ethernet (Category 5e and similar) is also often used; though its characteristic impedance is 100Ω. Applications less than a few hundred metres and at low speeds (like 9600 baud) usually work perfectly fine with just about any type of paired wire.

A transmission line relies on matching termination at both ends. A termination resistor is placed at the first and last devices on the bus. The shield can terminate a 120Ω characteristic impedance line by soldering jumpers J1 and J2. This should be done on if the Arduino is at one end of the bus, but not if the Arduino is in the middle.

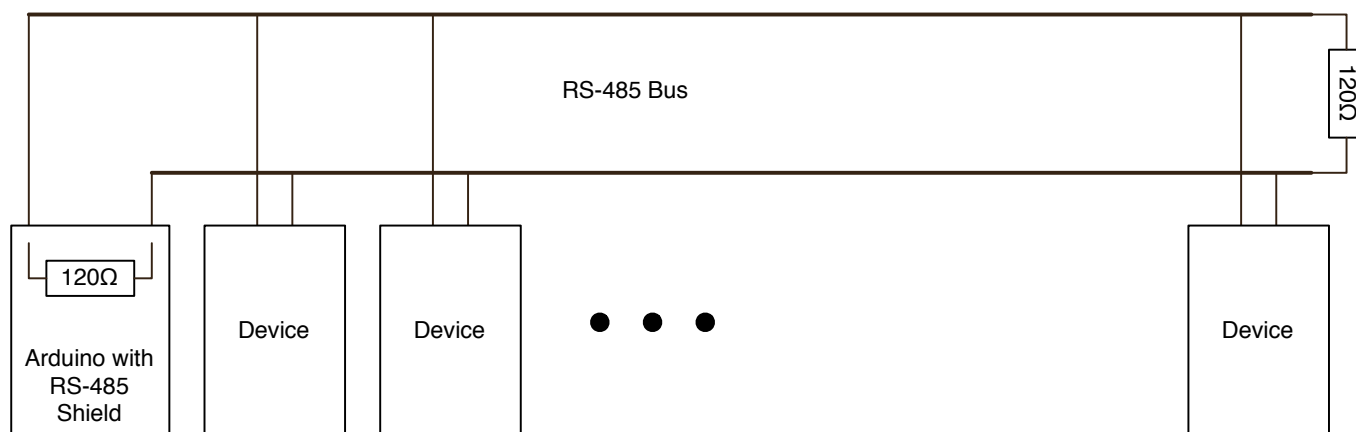


Figure 2 RS-485 bus with termination at both ends

TTL Serial Port

The Isolated RS-485 Shield is compatible with 3.3V and 5V Arduinos, including the Uno, Mega, Leonardo and Due. It takes power from the 5V pin on the host Arduino. Transistors Q1 and Q2 (see the schematic on the last page) level shift the data input and data direction input signals to 5V.

The Arduino's serial port is used for programming the microcontroller, so if the shield drove the output when the bus was idle, you wouldn't be able to program the Arduino without removing the shield. Instead, the shield's data output is open-collector (actually open-drain as the transistor Q3 is a MOSFET). This means the microcontroller sees the serial data from the USB port and the serial data from the RS-485 port superimposed on top of each other.

An open collector output relies on a pull-up resistor. In most Arduinos a resistor between the microcontroller and the USB interface, and the fact that the serial line from the interface is high while idle, provides this pull-up function. In the case of an Arduino Pro, which doesn't have a USB interface, you'll need to add an external resistor or enable the pin's pull-up in your code.

The jumpers J5 and J6 connect the data in and out of the shield to Arduino pins D0 and D1. In most cases you'd want to solder these short. You can also leave them open and use the vias on the board nearby to connect the shield to different pins (like a different serial port on the Arduino Mega.)

Serial Mark and Space Levels

Serial data from the Arduino is 3.3V or 5V TTL. In TTL serial, the line is sitting high at 3.3V or 5V while idle. When a byte is transmitted, the byte is sent in binary with the least significant bit transmitted first.¹ If the bit is a 1, the voltage on the line drops low to near 0V. This signal condition is called a “mark”. If the bit is a 0, the voltage goes high. This is referred to as a “space”.

On RS-485 there are two signal wires. The specification calls these “A” and “B”. “A” is the inverting signal. It’s like TTL, and is pulled low when transmitting logic 1. “B” is the non-inverting signal. When transmitting logic 1, B is pulled high.

Some RS-485 transceiver manufacturers used the label “A” for the non-inverting signal and “B” for the inverting signal, leading to some confusion. In products, the signal lines are often labelled “D+” for non-inverting and “D-” for the inverting signal to avoid the confusion.

If you can’t figure out which wire is meant to be which on your device, you can try both ways (incorrect polarity won’t harm the shield or your device, it just won’t communicate.)

Bus Keeper Resistors

RS-485 is a shared bus, which means that only one device can be transmitting on the bus at a time. When a device wants to transmit, it drives the signal lines high or low as required to transmit mark or space bits. When not transmitting, the driver is turned off and the signal lines are left free for another transmitter on the bus can control them.

Without a transmitter driving the bus, the lines are free to float. If the transmission line is terminated, the termination resistor pulls the voltage difference down to 0V, an undefined state for RS-485.

The shield has a full fail-safe transceiver, which keeps the receiver output high in these cases, but other devices on the network may not be similarly equipped.

On an unterminated, floating bus, charge can build up on the lines and cause the receivers to trigger, leading to random bits on the serial data input of the Arduino.

Bus keeper resistors are used to hold the RS-485 bus to the idle state and dissipate any weak charge building up on the signal lines.

You only need one pair of resistors for the entire bus. You can have more than one pair, though each pair added increases the load on the bus. If there are too many resistors or the resistors’ value is too small transmitters may not be strong enough to transmit clear data. If the resistors are too large they won’t hold the correct idle voltage (that is, at least 200 mV difference) for non fail-safe receivers.

The shield is fitted with a pair of 680Ω resistors. This is a good value for a 5V bus with 120Ω terminating resistors. Shorting jumpers J3 and J4 enables the bus keeping resistors.

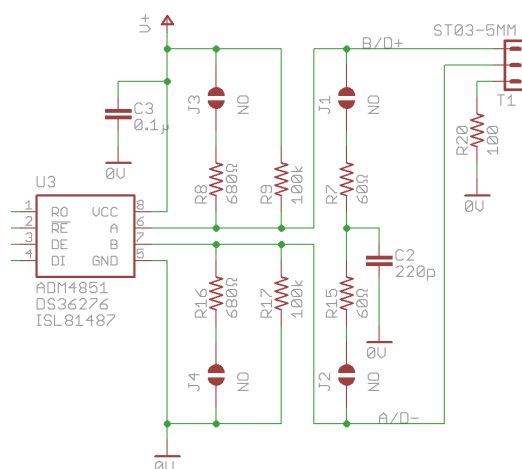


Figure 3 Transceiver, termination and bus keeper resistors with jumpers

¹ After a start bit; you may want to check Wikipedia for a more detailed explanation of the start bit.

Data Direction

The TTL serial out of the Arduino is full duplex – data can go both directions at the same time. RS-485 is half duplex – data can go both directions but only one direction at a time. The RS-485 transceiver needs to know whether it should have control of the bus and it should be transmitting 1s and 0s, or if it should be a receiver and disable the transmitter.

The most robust way to do this is to support it in your code and use a digital pin to signal the transceiver. The shield has jumpers that allow any one of Arduino pins D2 to D7 to be easily connected as the TX direction control (J7 through J12.)

In your code, manual control of the direction pin involves setting it high at the beginning of a transmission and resetting it low when the last byte has finished transmitting. The ATmega328P microcontroller has an interrupt (UDRE) that triggers when a byte has finished transmitting.

Automatic Data Direction Control

The shield can create an automatic data direction signal. Shorting jumper J13 enables the automatic direction control circuitry.

The automatic direction control works by enabling the transmitter whenever the transmit line is low and for a short period after the transmit line returns high.

This system relies on sufficient bus keeping resistance to hold the line in the idle (space) state while the Arduino is transmitting the high bits of the serial data. The received signal ends up with three distinct levels: a low level, a driven high level and a high level established by the bus keeper resistors.

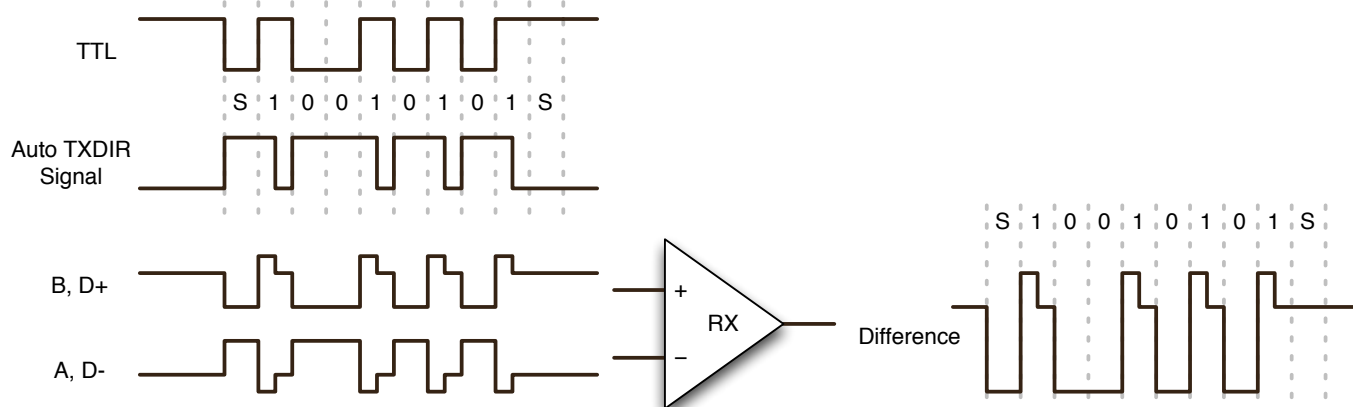


Figure 4 Automatic transmission direction control

The disadvantage of the automatic direction control system is that you're relying on the bus keeper resistors to provide the high level for some of the time. This is a much weaker level than the driver's high output, and is more susceptible to noise or transmission faults.

The length of time the driver can be kept enabled has to be set manually. The shorter the time is, the more you're relying on the weak high level. Make the time too long though, and you'll leave the transmitter enabled past the end of the byte and could interfere with a response coming from another device trying to use the bus.

The shield has three jumpers (J14, J15 and J16) that select the timing of the pulse. Slower baudrates allow for longer pulse widths. The table below shows the recommended jumper setting for different baudrates.

Baudrate	Jumper Configuration
Up to 4200 baud	All jumpers open
9600 to 28.8k	Short J14 only
38.4k to 115.2k	Short J15 only
230.4k and up	Short J16 only

Open Source Hardware

The KTA-286 Isolated RS-485 Shield is open source hardware. It's released under the Creative Commons Attribution Share-Alike 3.0 License. Among other things, this means all the design files are available to you (check the Ocean Controls website) and you can take them, change them and make your own versions of the shield so long as you give Ocean Controls some credit for the original work, and that you also release your changes under the same licence.

Circuit Diagram

The circuit diagram is shown on the next page.

Transistors Q1 and Q2 and resistors R1 and R18 form level shifters to allow the shield to work with 3.3V or 5V TTL serial ports. Transistor Q3 presents the data output as an open collector, allowing the shield to work with 3.3V or 5V Arduinos and preventing interference with the Arduino programming scheme. Jumpers J7 through to J12 select a pin to use for manual direction control. Diodes D1 and D2 protect the inverter in case the auto-direction jumper has also been soldered. Resistor R2 holds the direction input low if none of the jumpers are soldered.

The automatic data direction is enabled by J13. The serial data in is inverted by the 74AC14's gates D and B. Low levels (mark bits) on the data in become 5V high levels at D1. These charge capacitor C1. C1 discharges through R10 in parallel with one of R3, R4 or R5 depending on the speed selected. The inverters have Schmitt trigger inputs. This combines with the capacitor discharge curve to create predictable timing for the direction input.

The data signals and data direction signal are transferred across the isolation barrier by three optocouplers. Power gets across via a 5V to 5V DC-DC converter module.

The power and data signals connect to the RS-485 transceiver. The output of the transceiver is connected to screw terminals. Off the bus are an arrangement of resistors for optional bus keeping and bus termination.

The terminating resistor in RS-485 circuits is usually a simple 120Ω resistor. In this circuit a pair of 60Ω resistors is used with a capacitor down to ground in the centre. This improves common mode noise rejection.

100kΩ resistors are permanently attached to the bus as very weak bus keeping resistors. These, in combination with the true fail-safe transceiver keep the Arduino from receiving junk data if the network doesn't have bus keeping resistors or the wiring is broken.

The screw terminals also connect the signal ground to the transceiver ground via a 100Ω, 1W resistor (as recommended by the RS-485 specification.)

