PJRC Store

- Teensy 4.1, $26.85
- Teensy 4.0, $19.95
- Teensy 3.6, $29.25
- Teensy 3.5, $24.25
- Teensy 3.2, $19.80
- Teensy LC, $11.65
- Teensy 2.0, $16.00
- Teensy++ 2.0, $24.00

Teensy

- Main Page
- **Hardware**
- **Getting Started**
- **Tutorial**
- **How-To Tips**
- **Code Library**
- Projects
- **Teensyduino**
  - Main
  - Download+Install
  - Basic Usage
  - Digital I/O
  - PWM & Tone
  - **Timing**
  - USB Serial
  - USB Keyboard
  - USB Mouse
  - USB Joystick
  - USB MIDI
  - USB Flight Sim
  - Serial
  - **Libraries**
- **Reference**

Using the Hardware Serial Ports

The Teensy boards have 1 to 8 hardware serial ports, which may be used to connect to serial devices, like GPS receivers, XBee & ESP Wifi modules, Modbus controllers, serial interface displays and many other serial devices.

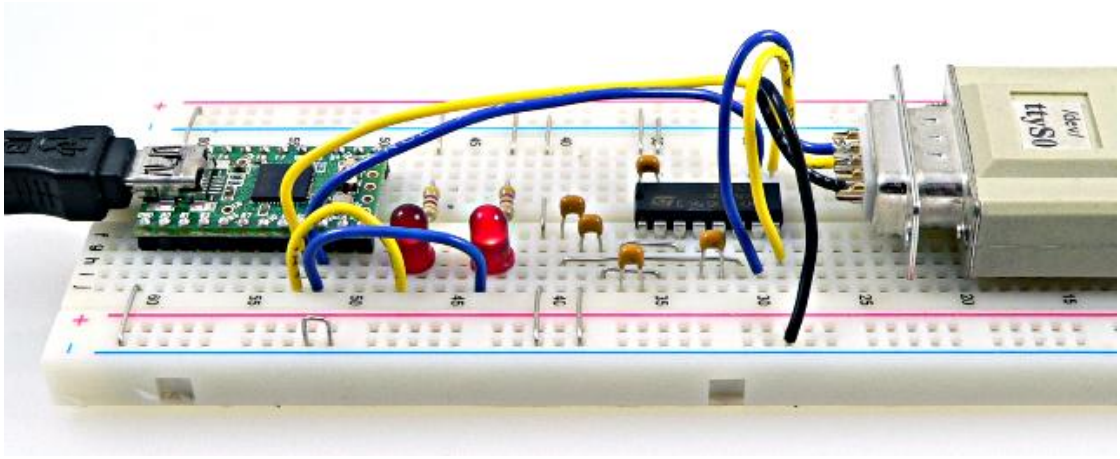
The most common issue with serial ports on Teensy is use of code designed for Arduino Uno with **Serial** within the code. On Teensy, **Serial** accesses the USB only. USB and Serial1 (pins 0 & 1) are not shared on Teensy. For hardware serial ports, **Serial1**, **Serial2**, **Serial3**, **Serial4**, **Serial5**, **Serial6**, **Serial7** or **Serial8** must be used.

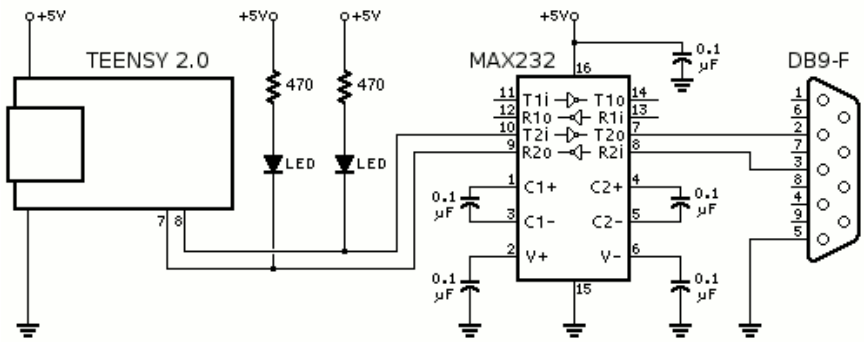
Hardware

Serial Port	Signal	Teensy 1.0	Teensy 2.0	Teensy++ 1.0 & 2.0	Teensy LC	Teensy 3.0, 3.1, 3.2	Teensy 3.5 & 3.6	Teensy 4.0	Teensy 4.1
Serial1	Receive	2	7	2	0, 3, 21, 25	0, 21	0, 21, 27	0	0
	Transmit	3	8	3	1, 4, 5, 24	1, 5	1, 5, 26	1	1
	Transmit Enable	Any	Any	Any	Any	Any	Any	Any	Any
	RTS	-	-	-	Any	Any	Any	Any	Any
	CTS	-	-	-	-	18, 20	18, 20	-	-
Serial2	Receive	-	-	-	9	9, 26	9	7	7
	Transmit	-	-	-	10	10, 31	10	8	8
	Transmit Enable	-	-	-	Any	Any	Any	Any	Any
	RTS	-	-	-	Any	Any	Any	Any	Any
	CTS	-	-	-	-	23	23	-	-
Serial3	Receive	-	-	-	7, 6	7	7	15	15
	Transmit	-	-	-	8, 20	8	8	14	14
	Transmit Enable	-	-	-	Any	Any	Any	Any	Any
	RTS	-	-	-	Any	Any	Any	Any	Any
	CTS	-	-	-	-	14	14	19	19
Serial4	Receive	-	-	-	-	-	31	16	16
	Transmit	-	-	-	-	-	32	17	17
	Transmit Enable	-	-	-	-	-	Any	Any	Any
	RTS	-	-	-	-	-	Any	Any	Any
	CTS	-	-	-	-	-	-	-	-
Serial5	Receive	-	-	-	-	-	34	21	21
	Transmit	-	-	-	-	-	33	20	20
	Transmit Enable	-	-	-	-	-	Any	Any	Any
	RTS	-	-	-	-	-	Any	Any	Any

	CTS	-	-	-	-	-	24	35	43
Serial6	Receive	-	-	-	-	-	47	25	25
	Transmit	-	-	-	-	-	48	24	24
	Transmit Enable	-	-	-	-	-	Any	Any	Any
	RTS	-	-	-	-	-	Any	Any	Any
	CTS	-	-	-	-	-	56	-	-
Serial7	Receive	-	-	-	-	-	-	28	28
	Transmit	-	-	-	-	-	-	29	29
	Transmit Enable	-	-	-	-	-	-	Any	Any
	RTS	-	-	-	-	-	-	Any	Any
	CTS	-	-	-	-	-	-	-	-
Serial8	Receive	-	-	-	-	-	-	-	34
	Transmit	-	-	-	-	-	-	-	35
	Transmit Enable	-	-	-	-	-	-	-	Any
	RTS	-	-	-	-	-	-	-	Any
	CTS	-	-	-	-	-	-	-	50

The serial signals on the Rx and Tx pins are "TTL level". To convert to RS-232 level a MAX232 or similar chip is needed.





On Teensy 3.2, 3.5, 3.6, Serial1 and Serial2 have 8 byte transmit and receive FIFOs, which allow for higher speed baud rates, even when other libraries create interrupt latency. All serial ports on Teensy 4.0 and 4.1 have 4 byte transmit and receive FIFOs.

All serial ports on all Teensy boards use interrupt-based transmit and receive buffering, to allow your program to avoid waiting when writing short messages, and to allow data to be reliably received even if your program must spend time performing other tasks.

On Teensy 3.6 when running faster than 120 MHz, writing to EEPROM may momentarily interfere with Serial1 and Serial2 baud rates. Reading EEPROM does not interfere. Serial3 to

Serial6 are not affected.

Commonly Used Serial Protocols

- MIDI
- ESP8266 (with fast baud rate and RTS/CTS flow control)
- SBUS
- Modbus RTU (see RS-485 transmitterEnable function)
- M-Bus
- Robotis Dynamixel AX and MX-T
- GPS NMEA0183
- DMX Lighting
- EasyTransfer library
- BlueSMiRF
- XBee

Example Code

This simple example shows how to use both the UART and USB Serial at the same time. Both are monitored for incoming bytes, and when either receives data, the results are printed to both.

```
// set this to the hardware serial port you wish to use
#define HWSERIAL Serial1

void setup() {
        Serial.begin(9600);
        HWSERIAL.begin(9600);
}

void loop() {
        int incomingByte;

        if (Serial.available() > 0) {
                incomingByte = Serial.read();
                Serial.print("USB received: ");
                Serial.println(incomingByte, DEC);
                HWSERIAL.print("USB received:");
                HWSERIAL.println(incomingByte, DEC);
        }
        if (HWSERIAL.available() > 0) {
                incomingByte = HWSERIAL.read();
                Serial.print("UART received: ");
                Serial.println(incomingByte, DEC);
                HWSERIAL.print("UART received:");
                HWSERIAL.println(incomingByte, DEC);
        }
}
```

Standard Serial Functions

All of the standard Serial functions are supported.

Serial1.begin(baud)

Initialize the Uart object. The baud rate must be given.

Serial1.print(data) and Serial1.println(data)

Print a number or string. Serial1.print() prints only the number or string, and Serial1.println() prints it with a newline character.

```
  // Serial1.print() can print many different types
```

```
int number = 1234;
Serial1.println("string");     // string
Serial1.println('a');          // single character
Serial1.println(number);       // number (base 10 if 16 or 32 bit)
Serial1.println(number, DEC);  // number, base 10 (default)
Serial1.println(number, HEX);  // number, base 16/hexidecimal
Serial1.println(number, OCT);  // number, base 8/octal
Serial1.println(number, BIN);  // number, base 2/binary
Serial1.println(3.14);         // number in floating point, 2 digits
```

Serial1.availableForWrite()

Returns the number of bytes that may be written without significant delay. If you write more than this number, Serial1.write() may need to wait for data to transmit before it can place your data into the transmit buffer.

Serial1.write(byte)

Transmit a byte.

Serial1.available()

Returns true if at least 1 byte is available, or false if nothing has been received.

Serial1.read()

Read 1 byte (0 to 255), if available, or -1 if nothing available. Normally Serial1.read() is used after Serial1.available(). For example:

```
if (Serial1.available()) {
  incomingByte = Serial1.read();  // will not be -1
  // actually do something with incomingByte
}
```

In 9 bit mode, the return value is 0 to 511, or -1 if nothing available.

Serial1.flush()

Wait for any transmitted data still in buffers to actually transmit. If no data is waiting in a buffer to transmit, flush() returns immediately.

Serial1.clear()

Discard any received data that has not been read.

Serial1.setTX(pin)

Configure the serial port to use an alternate transmit pin. Teensy LC and 3.x support only specific alternate pins. This function may be called before Serial1.begin(baud), to pre-configure the pin used. It may also be used while the serial port is active. Physical pullup resistors are recommended for applications which change pins while running.

Serial1.setRX(pin)

Configure the serial port to use an alternate receive pin. Teensy LC and 3.x support only specific alternate pins. This function may be called before Serial1.begin(baud), to pre-configure the pin used. It may also be used while the serial port is active. Physical pullup resistors are recommended for applications which change pins while running.

Serial1.transmitterEnable(pin)

Use a pin to automatically enable a RS-485 transceiver chip. This pin outputs a logic HIGH when transmitting data. It goes LOW after the last stop bit, to allow reception when not transmitting.

Most RS-485 chips have 2 control signals, an active-high transmitter enable and an active-low receiver enable. Normally those 2 signals are wired together and connected to the transmitter enable output.

Serial1.attachRts(pin)

Enable RTS flow control. The specified pin becomes an output. Low indicates the Teensy is ready to recieve data. High is a request for the other device to pause transmission. RTS is driven high before Teensy's receive buffer is full, to allow tolerance for devices which do not immediately pause transmission in response to RTS.

This function must be called after Serial1.begin();

The RTS output from Teensy should be connected to the CTS input on the other serial device, similarly to the TX output connecting to the other device's RX input. Do not simply connect same-named signals with serial.

Supported only on Teensy LC, 3.0 - 3.6. Any digital pin may be used.

Serial1.attachCts(pin)

Enable CTS flow control. When this pin is driven high, Teensy will pause transmission after the currently transmitting byte (if any) is completed.

This function must be called after Serial1.begin();

The CTS input from Teensy should be connected to the RTS output on the other serial device, similarly to the RX input connecting to the other device's TX output. Do not simply connect same-named signals with serial.

Supported only on Teensy 3.0 - 3.6. Only specific pins may be used.

Serial1.begin(baud, format)

Initialize the serial object with a baud rate and data format. Supported only on Teensy LC, 3.0, 3.1, 3.2, 3.5, 3.6.

Format Name	Data Bits	Parity	Stop Bits	RX Polarity	TX Polarity
SERIAL_7E1	7	Even		Normal	Normal
SERIAL_7O1	7	Odd			
SERIAL_8N1	8	None			
SERIAL_8N2	8	None			
SERIAL_8E1	8	Even			
SERIAL_8O1	8	Odd			
SERIAL_7E1_RXINV	7	Even		Inverted	Normal
SERIAL_7O1_RXINV	7	Odd			
SERIAL_8N1_RXINV	8	None			
SERIAL_8N2_RXINV	8	None			
SERIAL_8E1_RXINV	8	Even			
SERIAL_8O1_RXINV	8	Odd			
SERIAL_7E1_TXINV	8	Even		Normal	Inverted

SERIAL_7O1_TXINV	8	Odd			
SERIAL_8N1_TXINV	8	None			
SERIAL_8N2_TXINV	8	None			
SERIAL_8E1_TXINV	8	Even			
SERIAL_8O1_TXINV	8	Odd			
SERIAL_7E1_RXINV_TXINV	7	Even		Inverted	Inverted
SERIAL_7O1_RXINV_TXINV	7	Odd			
SERIAL_8N1_RXINV_TXINV	8	None			
SERIAL_8N2_RXINV_TXINV	8	None			
SERIAL_8E1_RXINV_TXINV	8	Even			
SERIAL_8O1_RXINV_TXINV	8	Odd			
The following formats are available when HardwareSerial.h is edited with SERIAL_9BIT_SUPPORT					
SERIAL_9N1	9	None		Normal	Normal
SERIAL_9E1	9	Even			
SERIAL_9O1	9	Odd			
SERIAL_9N1_RXINV	9	None		Inverted	Normal
SERIAL_9E1_RXINV	9	Even			
SERIAL_9O1_RXINV	9	Odd			
SERIAL_9N1_TXINV	9	None		Normal	Inverted
SERIAL_9E1_TXINV	9	Even			
SERIAL_9O1_TXINV	9	Odd			
SERIAL_9N1_RXINV_TXINV	9	None		Inverted	Inverted
SERIAL_9E1_RXINV_TXINV	9	Even			
SERIAL_9O1_RXINV_TXINV	9	Odd			

Serial1.write9bit(word)

Transmit 9 bit data. The "word" may have the 9th bit set or cleared.

Teensy 3.0, 3.1, 3.2 support 9 bit mode on Serial1, Serial2 and Serial3.
Teensy 3.5, 3.6 support 9 bit mode on Serial1, Serial2, Serial3, Serial4, Serial5 and Serial6.
Teensy LC supports 9 bit mode only on Serial1. Serial2 and Serial3 do not have 9 bit mode.

In 7 and 8 bit modes, this function is the same as write(byte).

Usable Baud Rates

Serial communcation can tolerate about 2.5% error. Because the baud rates are created using the CPU clock, which is not an exact multiple of standard baud rates, higher speed baud rates can not be used if the clock is set (in the Tools > CPU Speed menu) to slower speeds.

Intended Baud Rate	Teensy 3.6 Error at 180 MHz Serial 1-2	Teensy 3.6 Error at 180 MHz Serial 3-6	Teensy 3.5 Error at 120 MHz Serial 1-2	Teensy 3.5 Error at 120 MHz Serial 3-6	Teensy 3.2 Error at 96 MHz Serial 1-2	Teensy 3.2 Error at 96 MHz Serial 3	Teensy LC Error at 48 MHz Serial 1-3
4608000	0.16%	0.16%	0.16%	0.16%	-0.79%	-0.79%	-65.45%
2000000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-25.00%

1000000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-25.00%
921600	-0.10%	0.16%	0.16%	0.16%	0.16%	0.16%	-18.62%
500000	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
460800	+0.03%	0.16%	-0.03%	0.16%	-0.08%	0.16%	8.51%
250000 (DMX)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
230400	-0.03%	-0.03%	-0.03%	-0.03%	0.04%	-0.08%	-6.99%
115200	0.00%	-0.03%	0.02%	-0.03%	-0.02%	0.04%	0.16%
57600	0.00%	0.02%	0.01%	0.02%	0.01%	-0.02%	0.16%
38400	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.16%
31250 (MIDI)	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
19200	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.16%
9600	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.16%
4800	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	-0.16%
2400	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
1200	14.44%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
300	357.77%	52.59%	205.18%	52.59%	144.14%	22.07%	0.00%

Serial1 and Serial2 on Teensy 3.2, 3.5 and 3.6 have FIFOs, which reduce interrupt overhead when used at high baud rates. These first 2 ports also run from a higher clock speed, for higher accuracy at very fast baud rates.

Intended Baud Rate	Teensy 3.0 Error at 48 MHz	Teensy 3.0 Error at 24 MHz	Teensy 2.0 Error at 16 MHz	Teensy 2.0 Error at 8 MHz	Teensy 2.0 Error at 4 MHz	Teensy 2.0 Error at 2 MHz	Teensy 2.0 Error at 1 MHz
115200	+0.04%	-0.08%	+2.12%	-3.55%	+8.51%	+8.51%	+8.51%
57600	-0.02%	+0.04%	-0.79%	+2.12%	-3.55%	+8.51%	+8.51%
38400	+0.00%	+0.00%	+0.16%	+0.16%	+0.16%	-6.99%	+8.51%
31250 (MIDI)	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%	+0.00%
19200	+0.00%	+0.00%	+0.16%	+0.16%	+0.16%	+0.16%	-6.99%
9600	+0.00%	+0.00%	+0.16%	+0.16%	+0.16%	+0.16%	+0.16%
4800	+0.00%	+0.00%	-0.08%	+0.16%	+0.16%	+0.16%	+0.16%
2400	+0.00%	+0.00%	+0.04%	-0.08%	+0.16%	+0.16%	+0.16%
1200	+0.00%	+0.00%	-0.02%	+0.04%	-0.08%	+0.16%	+0.16%
300	+22.07%	+0.00%	+0.01%	+0.01%	-0.02%	+0.04%	-0.08%

Arduino at 57600 Baud - It's Actually 58824

Arduino creates the same baud rates as Teensyduino, except at 57600, where Arduino's error is +2.12%. When communicating with an Arduino board, the combined +2.12% and -0.79%

error is too much. Successful communication with Arduino requires 58824 baud.

```
void setup() {
  Serial1.begin(58824);  // Arduino 57600 is actually 58824
}
```

Alternately, if you edit the Arduino code to use "Serial.begin(57601)", Arduino will create a baud rate with only -0.79% error, which exactly matches Teensy.

The Arduino bootloader on Duemilanove and Diecimila, and the "8u2" chip on Arduino Uno when set to 57600, are always 58824 baud. These are not easily reconfigured, so setting Teensy to 58824 baud is the best solution if you must communicate with these.