Tienda PJRC

- Teensy 4.1, $31.50
- Teensy 4.0, $23.80

Teensy

- Página principal
- **Hardware**
- **Primeros pasos**
- **Tutorial**
- **Consejos prácticos**
- **Biblioteca de códigos**
- Proyectos
- **Teensyduino**
  - Descarga principal
  - + Instalación
  - Uso básico
  - E/S digital
  - PWM y tono Código
  - **de temporización**

Seguridad Inicio
Informe de fallos USB
Serie Teclado USB
Ratón USB Joystick
USB MIDI USB
Simulador de vuelo
USB Serie **Bibliotecas**

- **Referencia**

Uso de los puertos serie del hardware

Las placas Teensy tienen de 1 a 8 puertos seriales de hardware, que pueden usarse para conectarse a dispositivos seriales, como receptores GPS, módulos Wifi XBee y ESP, controladores Modbus, pantallas de interfaz serial y muchos otros dispositivos seriales.

El problema más común con los puertos serie en Teensy es el uso de código diseñado para Arduino Uno con **Serial** dentro del código. En Teensy, **Serial** solo accede al USB. USB y Serial1 (pines 0 y 1) no se comparten en Teensy. Para los puertos serie de hardware, se deben usar **Serial1** , Serial2 , **Serial3** , **Serial4** , **Serial5** , **Serial6** , **Serial7** o **Serial8 .**

Hardware

Puerto serie	Señal	Teensy 1.0	Teensy 2.0	Teensy++ 1.0 y 2.0	Teensy LC	Teensy 3.0, 3.1, 3.2	Teensy 3.5 y 3.6	Teensy 4.0	Teensy 4.1
	Recibir	2	7	2	0, 3, 21, 25	0, 21	0, 21, 27	0, (xbarra)	0,52, (xbarra)
	Transmitir	3	8	3	1, 4, 5, 24	1, 5	1, 5, 26	1	1,53
Serie 1	Habilitar transmisión	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	18, 20	18, 20	(xbar *)	(xbar *)
	Recibir	-	-	-	9	9, 26	9	7,(x barra)	7,(x barra)
	Transmitir	-	-	-	10	10, 31	10	8	8
Serie 2	Habilitar transmisión	-	-	-	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	23	23	(xbar *)	(xbar *)
	Recibir	-	-	-	7, 6	7	7	15, (xbarra)	15, (xbarra)
	Transmitir	-	-	-	8, 20	8	8	14	14
Serie 3	Habilitar transmisión	-	-	-	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	Cualquier	Cualquier	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	14	14	19, (xbarra *)	19, (xbarra *)
Serie 4	Recibir	-	-	-	-	-	31	16,(x barra)	16,(x barra)
	Transmitir	-	-	-	-	-	32	17	17

	Habilitar transmisión	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	-	-	(xbar *)	(xbar *)
	Recibir	-	-	-	-	-	34	21,38, (xbarra)	21,46, (xbarra)
	Transmitir	-	-	-	-	-	33	20,39	20,47
Serie 5	Habilitar transmisión	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	-	24	35, (xbarra *)	43, (xbarra *)
	Recibir	-	-	-	-	-	47	25, (xbarra)	25, (xbarra)
	Transmitir	-	-	-	-	-	48	24	24
Serie 6	Habilitar transmisión	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	-	-	Cualquier	Cualquier	Cualquier
	CTS	-	-	-	-	-	56	(xbar *)	(xbar *)
	Recibir	-	-	-	-	-	-	28, (xbarra)	28, (xbarra)
	Transmitir	-	-	-	-	-	-	29	29
Serie 7	Habilitar transmisión	-	-	-	-	-	-	Cualquier	Cualquier
	Estrategia en tiempo real	-	-	-	-	-	-	Cualquier	Cualquier
	CTS	-	-	-	-	-	-	(xbar *)	(xbar *)
	Recibir	-	-	-	-	-	-	-	34,48, (xbarra)
	Transmitir	-	-	-	-	-	-	-	35
Serie 8	Habilitar transmisión	-	-	-	-	-	-	-	Cualquier
	Estrategia en tiempo real	-	-	-	-	-	-	-	Cualquier
	CTS	-	-	-	-	-	-	-	50, (xbarra *)

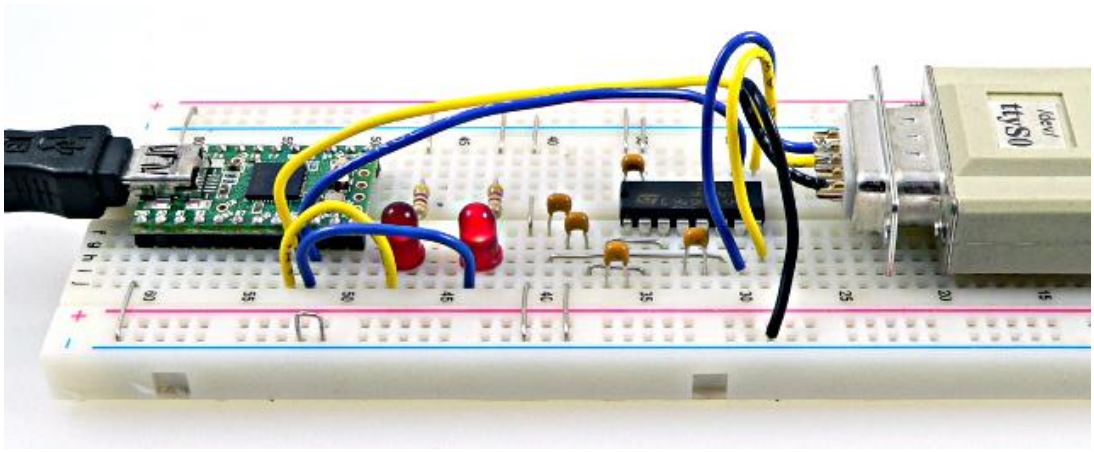
Los pines XBAR se pueden usar con las funciones "attachCts()" o "setRX()". Cada puerto serie solo puede establecer una conexión con el XBAR para un enrutamiento flexible de pines. Por lo tanto, en cada puerto se puede asignar un pin XBAR a CTS o RX, pero no a ambos. Los pines alternativos de XBAR anulan el uso de uno de los pines principales, por lo que se debe tener cuidado para evitar conflictos al usar pines alternativos.

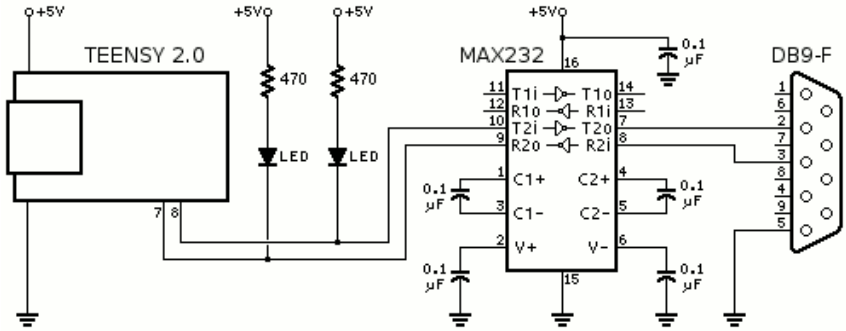
- **Teensy 4.0** : Pines XBAR principales: 1, 2, 3, 4, 5, 7, 8, 30, 31, 32, 33 - Pines alternativos: 0, 34, 35, 36, 37, 38, 39
- **Teensy 4.1** : Pines XBAR principales: 1, 2, 3, 4, 5, 7, 8, 30, 31, 32, 33 - Pines alternativos: 0, 36, 37, 42, 43, 44, 45, 46, 47

- **MicroMod Teensy** : Pines XBAR principales: 1, 2, 3, 4, 5, 7, 8, 30, 31, 32, 33 - Pines alternativos: 0, 34, 35, 36, 37, 38, 39

* La polaridad de CTS se invierte al conectarse mediante XBAR. Normalmente, CTS está activo en baja, lo que significa que Teensy puede transmitir datos cuando CTS está en baja y debe pausar la transmisión cuando CTS está en alta. Sin embargo, al usar XBAR, Teensy transmite cuando el pin está en alta lógica y pausa la transmisión cuando el pin está en baja.

Las señales serie en los pines Rx y Tx son de nivel TTL. Para convertirlas a nivel RS-232, se requiere un chip MAX232 o similar.





En Teensy 3.2, 3.5 y 3.6, Serial1 y Serial2 tienen FIFO de transmisión y recepción de 8 bytes, lo que permite velocidades de transmisión más altas, incluso cuando otras bibliotecas generan latencia de interrupción. Todos los puertos serie en Teensy 4.0 y 4.1 tienen FIFO de transmisión y recepción de 4 bytes.

Todos los puertos seriales en todas las placas Teensy usan buffering de transmisión y recepción basado en interrupciones, para permitir que su programa evite esperar cuando escribe mensajes cortos y para permitir que los datos se reciban de manera confiable incluso si su programa debe dedicar tiempo a realizar otras tareas.

En Teensy 3.6, al funcionar a más de 120 MHz, la escritura en la EEPROM puede interferir momentáneamente con las velocidades de transmisión de las series Serial 1 y Serial 2. La lectura de la EEPROM no interfiere. Las series Serial 3 a Serial 6 no se ven afectadas.

La confusión con las conexiones seriales es tan común que Shawn Hymel diseñó esta placa y Drew Fustini creó una PCB morada . No te preocupes si te confundes. ¡No estás solo!

Protocolos seriales comúnmente usados

- MIDI
- ESP8266 (con velocidad en baudios rápida y control de flujo RTS/CTS)
- SBUS
- Modbus RTU (ver función de habilitación del transmisor RS-485)
- M-Bus

- Robotis Dynamixel AX y MX-T
- GPS NMEA0183
- Iluminación DMX
- Biblioteca EasyTransfer
- BlueSMiRF
- XBee

Código de ejemplo

Este sencillo ejemplo muestra cómo usar simultáneamente el UART y el puerto USB serie. Ambos se monitorizan para detectar bytes entrantes y, cuando alguno recibe datos, los resultados se imprimen en ambos.

```
// Establezca esto en el puerto serie de hardware que desea utilizar
#define  HWSERIAL  Serial1

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



 bucle vacío ( )  {
  int  byte entrante ;

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

Funciones seriales estándar

Se admiten todas las funciones seriales estándar.

Serial1.begin(baud)

Inicialice el objeto Uart. Se debe proporcionar la velocidad en baudios.

Si se llama a begin(baud) después de que ya se estén recibiendo datos en serie, cualquier bit de datos 0 podría detectarse erróneamente como el primer bit de inicio, lo que resultaría en una recepción corrupta. Si se inicia desincronizado, el receptor necesita un tiempo de inactividad de 9 bits entre un bit de parada y el siguiente bit de inicio para sincronizarse. Ciertos patrones de utilización del 100% del ancho de banda podrían no sincronizarse nunca , lo cual constituye una limitación fundamental de la comunicación en serie.

Serial1.print(datos) y Serial1.println(datos)

Imprime un número o una cadena. Serial1.print() solo imprime el número o la cadena, y Serial1.println() lo imprime con un carácter de nueva línea.

```
  // Serial1.print() puede imprimir muchos tipos diferentes
  int  number  =  1234 ;
  Serial1 . println ( "string" ) ;      // string
  Serial1 . println ( 'a' ) ;           // carácter único
  Serial1 . println ( number ) ;        // número (base 10 si es de 16 o 32 bits)
  Serial1 . println ( number ,  DEC ) ;   // número, base 10 (predeterminado)
  Serial1 . println ( number ,  HEX ) ;   // número, base 16/hexadecimal
  Serial1 . println ( number ,  OCT ) ;   // número, base 8/octal
  Serial1 . println ( number ,  BIN ) ;   // número, base 2/binario
  Serial1 . println ( 3.14 ) ;          // número en punto flotante, 2 dígitos
```

Serial1.availableForWrite()

Devuelve el número de bytes que se pueden escribir sin un retraso significativo. Si escribe más de este número, Serial1.write() podría tener que esperar a que se transmitan los datos antes de poder colocarlos en el búfer de transmisión.

Serial1.write(byte)

Transmitir un byte.

Serial1.disponible()

Devuelve verdadero si hay al menos 1 byte disponible, o falso si no se ha recibido nada.

Serial1.read()

Leer 1 byte (0 a 255), si está disponible, o -1 si no hay ninguno disponible. Normalmente, Serial1.read() se usa después de Serial1.available(). Por ejemplo:

```
if  ( Serial1 . available ( ) )  {
  incomingByte  =  Serial1 . read ( ) ;   // no será -1
  // realmente hará algo con incomingByte
}
```

En el modo de 9 bits, el valor de retorno es de 0 a 511, o -1 si no hay nada disponible.

Serial1.flush()

Espere a que los datos transmitidos que aún se encuentren en los búferes se transmitan. Si no hay datos esperando en un búfer para transmitirse, la función flush() regresa inmediatamente.

Serial1.clear()

Descarte cualquier dato recibido que no haya sido leído.

Serial1.setTX(pin)

Configure el puerto serie para usar un pin de transmisión alternativo. Teensy LC y 3.x solo admiten pines alternativos específicos. Esta función puede ejecutarse antes de Serial1.begin(baud) para preconfigurar el pin utilizado. También puede usarse mientras el puerto serie está activo. Se recomiendan resistencias pullup físicas para aplicaciones que cambian de pin durante la ejecución.

Serial1.setRX(pin)

Configure el puerto serie para usar un pin de recepción alternativo. Teensy LC y 3.x solo admiten pines alternativos específicos. Esta función puede llamarse antes de Serial1.begin(baud) para preconfigurar el pin utilizado. También puede usarse mientras el puerto serie está activo. Se recomiendan resistencias pullup físicas para aplicaciones que cambian de pin durante la ejecución.

Serial1.transmitterEnable(pin)

Use un pin para habilitar automáticamente un chip transceptor RS-485. Este pin emite un nivel lógico ALTO al transmitir datos. Se vuelve BAJO después del último bit de parada para permitir la recepción cuando no se transmite.

La mayoría de los chips RS-485 tienen dos señales de control: una de habilitación del transmisor activa-alta y otra de habilitación del receptor activa-baja. Normalmente, estas dos señales se conectan a la salida de habilitación del transmisor.

Serial1.addMemoryForRead(búfer, tamaño)

Aumente la cantidad de memoria intermedia entre la recepción de bytes por parte del hardware serial y las funciones available() y read(). Esto resulta útil cuando el programa debe dedicar mucho tiempo a realizar otras tareas, como escribir en una tarjeta SD, antes de poder volver a leer los datos seriales entrantes.

La matriz de búfer debe ser una variable global o estática. El tamaño máximo del búfer es de 64 KB.

Serial1.addMemoryForWrite(búfer, tamaño)

Aumente la cantidad de memoria intermedia entre print(), write() y la transmisión serial por hardware. Esto puede ser útil cuando su programa necesita imprimir o escribir una gran cantidad de datos sin espera.

La matriz de búfer debe ser una variable global o estática. El tamaño máximo del búfer es de 64 KB.

Serial1.attachRts(pin)

Habilitar el control de flujo RTS. El pin especificado se convierte en una salida. Bajo indica que el Teensy está listo para recibir datos. Alto indica que el otro dispositivo pausa la transmisión. El RTS se activa antes de que el búfer de recepción del Teensy se llene, para permitir la tolerancia de los dispositivos que no pausan la transmisión inmediatamente en respuesta al RTS.

Esta función debe llamarse después de Serial1.begin();

La salida RTS de Teensy debe conectarse a la entrada CTS del otro dispositivo serial, de forma similar a como la salida TX se conecta a la entrada RX del otro dispositivo. No conecte señales con el mismo nombre a la serial.

Compatible solo con Teensy LC, 3.x y 4.x. Se puede usar cualquier PIN digital.

Serial1.attachCts(pin)

Habilitar el control de flujo CTS. Cuando este pin se activa, Teensy pausará la transmisión una vez completado el byte que se está transmitiendo (si lo hay).

Esta función debe llamarse después de Serial1.begin();

La entrada CTS de Teensy debe conectarse a la salida RTS del otro dispositivo serial, de forma similar a como la entrada RX se conecta a la salida TX del otro dispositivo. No conecte señales con el mismo nombre a la serial.

Compatible solo con Teensy 3.x y 4.x. Solo se pueden usar pines específicos.

Serial1.begin(baud, formato)

Inicialice el objeto serial con una velocidad en baudios y un formato de datos. Compatible solo con Teensy LC 3.0, 3.1, 3.2, 3.5 y 3.6.

Nombre del formato	Bits de datos	Paridad	Bits de parada	Polaridad RX	Polaridad TX
SERIE_7E1	7	Incluso		Normal	Normal
SERIE_7O1	7	Extraño			
SERIE_8N1	8	Ninguno			
SERIE_8N2	8	Ninguno			
SERIE_8E1	8	Incluso			
SERIE_8O1	8	Extraño			

SERIE_7E1_RXINV	7	Incluso		Invertido	Normal
SERIE_7O1_RXINV	7	Extraño			
SERIE_8N1_RXINV	8	Ninguno			
SERIE_8N2_RXINV	8	Ninguno			
SERIE_8E1_RXINV	8	Incluso			
SERIE_8O1_RXINV	8	Extraño			
SERIE_7E1_TXINV	8	Incluso		Normal	Invertido
SERIE_7O1_TXINV	8	Extraño			
SERIE_8N1_TXINV	8	Ninguno			
SERIE_8N2_TXINV	8	Ninguno			
SERIE_8E1_TXINV	8	Incluso			
SERIE_8O1_TXINV	8	Extraño			
SERIE_7E1_RXINV_TXINV	7	Incluso		Invertido	Invertido
SERIE_7O1_RXINV_TXINV	7	Extraño			
SERIE_8N1_RXINV_TXINV	8	Ninguno			
SERIE_8N2_RXINV_TXINV	8	Ninguno			
SERIE_8E1_RXINV_TXINV	8	Incluso			
SERIE_8O1_RXINV_TXINV	8	Extraño			
Los siguientes formatos están disponibles cuando HardwareSerial.h se edita con SERIAL_9BIT_SUPPORT					
SERIE_9N1	9	Ninguno		Normal	Normal
SERIE_9E1	9	Incluso			
SERIE_9O1	9	Extraño			
SERIE_9N1_RXINV	9	Ninguno		Invertido	Normal
SERIE_9E1_RXINV	9	Incluso			
SERIE_9O1_RXINV	9	Extraño			
SERIE_9N1_TXINV	9	Ninguno		Normal	Invertido
SERIE_9E1_TXINV	9	Incluso			
SERIE_9O1_TXINV	9	Extraño			
SERIE_9N1_RXINV_TXINV	9	Ninguno		Invertido	Invertido
SERIE_9E1_RXINV_TXINV	9	Incluso			
SERIE_9O1_RXINV_TXINV	9	Extraño			

Serial1.write9bit(palabra)

Transmite datos de 9 bits. La palabra puede tener el noveno bit activado o desactivado.

Teensy 3.0, 3.1 y 3.2 admiten el modo de 9 bits en Serial1, Serial2 y Serial3.
Teensy 3.5 y 3.6 admiten el modo de 9 bits en Serial1, Serial2, Serial3, Serial4, Serial5 y Serial6.
Teensy LC solo admite el modo de 9 bits en Serial1. Serial2 y Serial3 no lo admiten.

En los modos de 7 y 8 bits, esta función es la misma que write(byte).

Velocidades en baudios utilizables

La comunicación serial puede tolerar un error de aproximadamente el 2,5 %. Dado que las velocidades en baudios se crean utilizando la frecuencia del reloj de la CPU, que no es un múltiplo exacto de las velocidades estándar, no se pueden utilizar velocidades más altas si la frecuencia está configurada (en el menú Herramientas > Velocidad de CPU) a velocidades más bajas.

Velocidad en baudios prevista	Error de Teensy 3.6 a 180 MHz Serial 1-2	Error de Teensy 3.6 a 180 MHz Serial 3-6	Error de Teensy 3.5 a 120 MHz Serial 1-2	Error de Teensy 3.5 a 120 MHz Serie 3-6	Error de Teensy 3.2 a 96 MHz Serial 1-2	Error de Teensy 3.2 a 96 MHz Serial 3	Error de LC de Teensy a 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%

Velocidad en baudios prevista	Error de Teensy 3.0 a 48 MHz	Error de Teensy 3.0 a 24 MHz	Error de Teensy 2.0 a 16 MHz	Error de Teensy 2.0 a 8 MHz	Error de Teensy 2.0 a 4 MHz	Error de Teensy 2.0 a 2 MHz	Error de Teensy 2.0 a 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%

Altas tasas de transmisión en baudios

Los puertos serie 1 y serie 2 de Teensy 3.2, 3.5 y 3.6 cuentan con FIFO, lo que reduce la sobrecarga de interrupciones al usar altas velocidades de transmisión. Estos dos primeros puertos también funcionan a una velocidad de reloj más alta, lo que proporciona mayor precisión a velocidades de transmisión muy altas. Todos los puertos serie de Teensy 4.0, Teensy 4.1 y MicroMod Teensy cuentan con FIFO.

A velocidades de transmisión muy altas, incluso pequeños retrasos de software pueden provocar que el búfer de recepción se desborde, lo que provoca la pérdida de datos entrantes. El control de flujo RTS/CTS ofrece la mejor solución para la pérdida de datos, pero requiere dos señales adicionales y la compatibilidad adecuada con RTS/CTS por parte del otro dispositivo que se comunica con Teensy. La alternativa es usar addMemoryForRead() para aumentar el tamaño del búfer de recepción lo suficiente como para capturar todos los datos entrantes cuando el programa esté ocupado con otras tareas.

En las placas Teensy 4, los puertos serie suelen estar limitados a una velocidad de transmisión de 6 Mbit/s. Sin embargo, la velocidad máxima puede aumentarse a 20 Mbit/s reconfigurando la fuente de reloj UART.

Arduino Uno a 57600 Baud - En realidad son 58824

Arduino Uno genera las mismas velocidades de transmisión que Teensyduino, excepto por 57600, donde el error de Arduino es de +2,12 %. Al comunicarse con una placa Arduino, el error combinado de +2,12 % y -0,79 % es excesivo. Para una comunicación exitosa con Arduino se requieren 58824 baudios.

```
void  setup ( )  {
  Serial1.begin ( 58824 ) ; // Arduino 57600 es en realidad 58824 }
```

Alternativamente, si edita el código de Arduino para usar "Serial.begin(57601)", Arduino creará una tasa de baudios con solo un error de -0,79 %, que coincide exactamente con Teensy.

El gestor de arranque de Arduino en Duemilanove y Diecimila, y el chip 8u2 o 16u2 en Arduino Uno, al configurarse a 57600, siempre funcionan a 58824 baudios. No son fáciles de reconfigurar, por lo que configurar Teensy a 58824 baudios es la mejor solución si necesita comunicarse con ellos .