Comunicación UART Teensy 4.1: ¿Cómo utilizar la comunicación serial por hardware?

samm_flynn · 26 de febrero de 2025

S

samm_flynn
Miembro conocido

26 de febrero de 2025 #1

EDITAR: Había un error en mi código anterior, que encontré justo después de publicar. Edito la pregunta porque tengo una nueva.

Hola a todos,

estoy trabajando en un proyecto con dos placas Teensy 4.1. Necesito enviar datos binarios estructurados al maestro por USB serie, y luego del maestro al esclavo por UART de hardware (Serie1 → Serie8). La configuración funciona a la perfección al enviar datos al Teensy esclavo por USB serie. Puedo enviar casi 512 KB a 20 MB/s, a veces incluso más.

Sin embargo, al cambiar a serie de hardware, no obtengo el mismo rendimiento (no lo he medido), pero el retraso es mucho mayor.

Me preguntaba si se debe a la codificación o si la serie de hardware es significativamente más lenta por diseño.

Código del Teensy Maestro -

C++:

```
int packetIndex = 0;
bool inPacket = false;

// Serial port initialization
void setup()
{
  Serial.begin(921600);
  while (!Serial) { delay(1); }
  Serial1.begin(921600);

}

void loop()
{
  if (Serial.available() >= 8)
  {
    byte startByte = Serial.read();
    byte commandByte = Serial.read();
```

```
    byte axisId = Serial.read();
    byte headerByte = Serial.read();

    uint32_t payloadSize;
    Serial.readBytes(reinterpret_cast<char *>(&payloadSize), sizeof(pay
    uint8_t payloadBuffer[payloadSize];
    Serial.readBytes(reinterpret_cast<char *>(payloadBuffer), payloadSi
    // Read the end byte
    byte endByte = Serial.read();
    Serial.println("\n--- Sending Packet Packet ---");
    Serial.printf("Start Byte: 0x%02X\n", startByte);
    Serial.printf("Command Type: 0x%02X\n", commandByte);
    Serial.printf("Axis ID: %d\n", axisId);
    Serial.printf("Header Byte: 0x%02X\n", headerByte);
    Serial.printf("Payload Size: %u \n", payloadSize);
    Serial.printf("End Byte: 0x%02X\n", endByte);
    Serial.println("----------------------\n");
    // Allocate buffer dynamically for the full packet
    uint32_t totalPacketSize = 8 + payloadSize + 1;  // Start + command
    uint8_t fullPacket[totalPacketSize];
```

Código de esclavos adolescentes -

C++:

```
#define N 63488  // Define max trajectory size
#define IRQ_LINE 5
#define ENABLE_LINE 32
#define ESTOP 33
#include <math.h>
#include "motor_axis.h"
// #include "utils.h"
#include <IntervalTimer.h>
IntervalTimer motor_timer;

// Define the buffer using DMA memory
DMAMEM union
{
  float thetas[N];
  uint8_t rawBytes[sizeof(float) * N];
} thetaBuffer1;

DMAMEM union
{
  float thetas[N];
  uint8_t rawBytes[sizeof(float) * N];
} thetaBuffer2;
// Function prototypes
void processPacket();
void handlePayload(byte headerByte, int payloadSize);
template<typename T> T readPayload(int payloadSize);

const int PROX_INT1 = 38;  // or 2
const int PROX_INT2 = 2;   // Different proximity interrupt for motor2
```

```
const int8_t LED_PIN = 13;
const unsigned long CAN_ID1 = 0x01;  // or 1
const unsigned long CAN_ID2 = 0x01;  // Different CAN ID for motor2

const int dir1 = -1;  // or -1
const int dir2 = 1;   // Opposite direction for second motor

MotorAxis<FlexCAN_T4<CAN1, RX_SIZE_8, TX_SIZE_8>> motor1(CAN_ID1, PROX_
```

Última edición:26 de febrero de 2(

do

Collin K
Miembro conocido

26 de febrero de 2025

La conexión serial por hardware es literalmente un orden o dos de magnitud más lenta que la conexión serial por USB. El enlace serial USB está limitado únicamente por el ancho de banda del bus. En USB2, esto es de unos 480 Mbps. No es probable que alcances esos 480 Mbps, pero como ya has comprobado, es posible alcanzar 20 megabytes por segundo.

La conexión serial por hardware, en cambio, probablemente solo funcionará hasta un par de Mbps, y básicamente le estás pidiendo poco menos de 1 Mbps. Por lo tanto, es 480 veces más lenta. La velocidad serial que estás pidiendo es de unos 90 kilobytes por segundo, lo que es MUCHO más lenta que la velocidad USB.

Deberás tener en cuenta esta diferencia de velocidad.

PaulStoffregen
Miembro conocido

26 de febrero de 2025

El puerto serie por hardware no solo es físicamente unas 500 veces más lento, sino que también tiene mucha más sobrecarga de CPU que el USB.

Los puertos serie de Teensy tienen al menos 4 bytes FIFO. Creo que actualmente configuramos la marca de agua de interrupción para que la mitad del FIFO reduzca la tasa de interrupción y la otra mitad proteja contra la latencia de interrupción inesperada. Pero incluso si se usaran los 4 bytes para mayor eficiencia (un diseño bastante arriesgado si se ejecuta a las velocidades de transmisión más altas), todavía se estaría viendo una interrupción y una sobrecarga de CPU cada 4 bytes.

El USB a 480 Mbit/s usa paquetes de 512 bytes. Ha pasado un tiempo desde que trabajé en el código de bajo nivel, pero creo que usamos un grupo de búferes de 2048 bytes. Pero incluso si hay una interrupción por cada paquete, se está consumiendo tiempo de CPU por una interrupción con mucha menos frecuencia. El hardware USB también es mucho más eficiente, utilizando DMA de bus maestro para colocar los datos entrantes directamente en el búfer de memoria.

El puerto serie por hardware se puede usar con el controlador DMA genérico, aunque no es el tipo de bus maestro más eficiente. Al menos un par de personas lo han logrado y han compartido código en este foro. Normalmente, esto implica

un uso más específico, ya que el DMA genérico necesita conocer el tamaño de los datos con antelación. El DMA no puede solucionar el hecho de que el puerto serie por hardware sigue siendo, en el mejor de los casos, 100 veces más lento, pero al menos puede mitigar gran parte de la sobrecarga de CPU que se produce al aumentar la velocidad del puerto serie por hardware a sus velocidades en baudios más altas. Si se ejecutan los 8 puertos serie simultáneamente a altas velocidades en baudios, probablemente se necesitará ese nivel de eficiencia (para alcanzar de forma fiable la capacidad de aproximadamente 100 veces más lento que USB).

S

Sami H
Miembro

26 de febrero de 2025 #4

¿Podría Ethernet ser una solución viable en estos casos? ¿Dónde se requiere una transferencia de datos de alta velocidad?

En cuanto al hardware subyacente, ¿tiene Ethernet más búfer? ¡
4 bytes es increíblemente pequeño!

PaulStoffregen
Miembro conocido

26 de febrero de 2025 #5

Sí, la Ethernet integrada en Teensy 4.1 utiliza un eficiente DMA de bus maestro y búferes de gran capacidad con la biblioteca QNEthernet. Muchos usuarios han reportado alcanzar casi el ancho de banda completo de 100 Mbit al comunicarse en una LAN. La velocidad en internet, con mayor latencia de paquetes, rara vez se acerca a la velocidad teórica total del hardware debido a la RAM limitada de los microcontroladores.

> Sami H

S

Shawn
Miembro conocido

26 de febrero de 2025 #6

Para más detalles, en la biblioteca *QNEthernet* hay cinco búferes de trama para RX y TX, cada uno con una capacidad de 1536 bytes. No he realizado ningún ajuste para esa cantidad (5).

S

samm_flynn
Miembro conocido

26 de febrero de 2025 #7

¡Rayos! Estaba pensando que voy a usar un Teensy como router serial.

Gracias por los comentarios, chicos.

26 de febrero de 2025 #8

Por eso siempre es recomendable explicar el contexto de tus preguntas. Si solo haces una pregunta específica sobre velocidad, probablemente te perderás respuestas como que el número de serie del hardware no está diseñado para datos de alta velocidad en Mbit/s (en cualquier placa, no solo en Teensy). Unas

PaulStoffregen
Miembro conocido

mejores respuestas pueden marcar la diferencia entre comenzar tu proyecto de forma viable o perder mucho tiempo solo para descubrir después que el enfoque no era viable.

> Shawn

S

samm_flynn
Miembro conocido

27 de febrero de 2025 #9

> PaulStoffregen dijo:
>
> Por eso siempre es recomendable explicar el contexto de tus preguntas. Si solo haces una pregunta específica sobre velocidad, probablemente te perderás respuestas como que el número de serie del hardware no está diseñado para datos de alta velocidad en Mbit/s (en cualquier placa, no solo en Teensy). Unas mejores respuestas pueden marcar la diferencia entre comenzar tu proyecto de forma viable o perder mucho tiempo solo para descubrir después que el enfoque no era viable.

Buen
punto, un poco de contexto para cualquiera que se encuentre con esta publicación:

Tengo un robot con seis controladores de motor, y todo el sistema de control está desarrollado y ajustado para operar a una frecuencia de muestreo de 1 kHz.

Debido a la configuración, utilizo un Teensy por cada dos ejes, junto con un Teensy maestro que envía una señal de sincronización PWM a 1 kHz. Los Teensy esclavos ejecutan el algoritmo de control basándose en este pulso externo.

Para la comunicación, necesito enviar 6 coma flotante (24 bytes) + 8 bytes de datos adicionales, lo que suma un total de 32 bytes por ciclo a 1 kHz.
La retroalimentación de los sensores integrados consta de 16 bytes por eje para 6 ejes, lo que suma un total de 96 bytes por ciclo.

Dado que el robot está diseñado para funcionar durante períodos prolongados, los Teensy no pueden almacenar todos los datos de referencia y retroalimentación en la RAM. Por lo tanto, planeo implementar una subrutina de enrutador serie en el Teensy maestro para gestionar la transferencia de datos entre el PC y los Teensy esclavos para el control y el registro.

Con tres Teensies esclavos, el maestro necesita gestionar una transferencia de datos total de 384 kB/s.

Por cada Teensy esclavo, necesito recibir datos a 32 kB/s y enviarlos al maestro a 96 kB/s, lo que resulta en una velocidad de comunicación total de 128 kB/s por esclavo, lo que equivale a poco más de 1 Mb/s. Quizás necesite enviar retroalimentación por cada dos muestras promediadas. O podría probar la red Ethernet y un conmutador.

Bueno, tengo algo de trabajo por hacer.

Yo

joepasquariello
Miembro conocido

27 de febrero de 2025 #10

Entonces, por cada ciclo de control de 1 ms, el maestro envía 36 bytes a cada esclavo y recibe 92 bytes. Son al menos 10 bits por byte, lo que equivale a 1 280 000 bps. Si la comunicación es semidúplex, se necesitarán al menos 2 Mb/s. ¿Será solo una prueba por ahora o se construirá? ¿Será RS-232 o RS-485? Recomiendo usar la biblioteca SerialTransfer. Construye/deconstruye paquetes con CRC, con una sobrecarga bastante baja, por lo que se sabe con certeza, en ambos extremos, que se han recibido datos correctos. También permite un "ID de paquete", lo que permite tener diferentes paquetes de "datos" y posteriormente añadir paquetes de "configuración", que probablemente se necesiten. Con la comunicación serie, siempre intento escalar los números flotantes a enteros de 16 bits, que casi siempre ofrecen una resolución suficiente para el control. Eso podría reducir su velocidad de datos en 1/3 o más.

desfragmentador
Miembro Senior+

27 de febrero de 2025 #11

Este https://github.com/tonton81/SPI_MSTransfer_T4 se creó para usar T_3.6 y funcionó de maravilla en ese caso, ya que descargó datos a un segundo Teensy para su procesamiento y alimentación a una PC.

Parece haberse actualizado para T_4. ¿Suponiendo que los cuatro T_4 estén cerca?

S

samm_flynn
Miembro conocido

27 de febrero de 2025 #12

> joepasquariello dijo:
>
> Entonces, por cada ciclo de control de 1 ms, el maestro envía 36 bytes a cada esclavo y recibe 92 bytes. Son al menos 10 bits por byte, lo que equivale a 1 280 000 bps. Si la comunicación es semidúplex, se necesitarán al menos 2 Mb/s. ¿Están probando esto por ahora o ya lo están construyendo? ¿Será RS-232 o RS-485?

Estoy usando cables muy cortos (<15 cm), con placas de conexión diminutas y conectores de tornillo para mantener los cables en su lugar, tengo la intención de usar RS 422 en el futuro, una vez que todo esté finalizado.

> Recomiendo usar la biblioteca SerialTransfer. Construye/deconstruye paquetes con CRC, con una sobrecarga bastante baja, por lo que sabrá con certeza, en ambos extremos, que se han recibido datos correctos. También permite un "ID de paquete", lo que permite tener diferentes paquetes de "datos" y posteriormente agregar paquetes de "configuración", que probablemente necesitará. Con la comunicación serial

Teniendo en cuenta SerialTransfer, podría modificarlo un poco para usar crc32 para paquetes muy grandes que pretendo enviar.

> Siempre intento escalar los flotantes a enteros de 16 bits, lo cual casi siempre ofrece suficiente resolución para el control. Esto podría reducir la velocidad de datos en un tercio o más.

Mi supervisor también mencionó eso y podría terminar haciéndolo.

S

samm_flynn
Miembro conocido

27 de febrero de 2025 #13

> defragster dijo:
>
> Este https://github.com/tonton81/SPI_MSTransfer_T4 se creó para usar T_3.6 y funcionó de maravilla en ese caso, ya que descargó datos a un segundo Teensy para su procesamiento y alimentación a una PC.
>
> Parece haberse actualizado para T_4. ¿Suponiendo que los cuatro T_4 estén cerca?

Lo investigaré, gracias por la sugerencia señor.

Yo

joepasquariello
Miembro conocido

27 de febrero de 2025 #14

> samm_flynn dijo:
>
> Teniendo en cuenta SerialTransfer, podría modificarlo un poco para usar crc32 para paquetes muy grandes que pretendo enviar.

Atención con respecto a SerialTransfer. La biblioteca actual tiene una carga útil máxima de 254 bytes por mensaje, por lo que funcionaría con los mensajes cortos que describiste, pero no con los muy largos.

> samm_flynn

S

samm_flynn
Miembro conocido

1 de marzo de 2025 #15

Acabo de darme cuenta de que no puedo usar Ethernet porque el Teensy 4.1 que tengo no tiene el chip soldado (creo que es la versión NE). ¿
Puedo usar mi Teensy maestro como host USB como reemplazo directo en ese caso?
Solo para que se vea, así es como se ve el código de mi Teensy maestro funcionando:

C++:

```
#define CHUNK_SIZE 128  // Define the byte chunk size
#define N 60000
byte packetBuffer[240000 + 9];  // Main buffer for packet storage
void processIncomingData(){
  //startByte(1), write/read(1), axis_id(1), header(1), payload_size(4)
  uint32_t payloadSize = 0;
  if (Serial.available() >= 8){
    Serial.readBytes(reinterpret_cast<char*>(packetBuffer), 8);  // Rea
    memcpy(&payloadSize, &packetBuffer[4], sizeof(payloadSize));
    Serial.readBytes(reinterpret_cast<char*>(packetBuffer + 8), payload
    Serial.readBytes(reinterpret_cast<char*>(&packetBuffer[8 + payloadS
    routePacket(packetBuffer[2], packetBuffer, 9 + payloadSize);
  }
}
void routePacket(byte axisID, uint8_t* fullPacket, uint32_t packetSize)
{
  HardwareSerial* targetSerial = nullptr;
  switch (axisID) {
    case 1:
```

```
    case 2:
      targetSerial = &Serial1;
      break;
    case 3:
    case 4:
      targetSerial = &Serial2;
      break;
    case 5:
    case 6:
      targetSerial = &Serial3;
      break;
    default:
      return;  // Invalid axisID, exit function
  }
  if (targetSerial){
    // uint32_t bytesSent = 0;
    // while (bytesSent < packetSize)
    // {
    //   uint32_t spaceAvailable = targetSerial->availableForWrite();
```

A

Andy A
Miembro conocido

3 de marzo de 2025

Volviendo a mi pregunta habitual cuando alguien tiene problemas para implementar algo complejo: ¿Estás seguro de que necesitas esto? ¿Por qué 1 kHz? ¿A qué velocidad se mueve el robot como para necesitar control y retroalimentación de 1 kHz hacia y desde cada nodo?
Muy poco movimiento físico requiere control a esa frecuencia; las cosas simplemente no se mueven tan rápido. Si se trata de algún tipo de bucle de control crítico para la sincronización, tendría más sentido (y probablemente más fiable) ejecutar ese bucle de control localmente y solo enviar parámetros/ubicaciones de destino a una velocidad mucho menor.

Si se trata únicamente de sincronización temporal, se puede hacer con una señal digital en lugar de con datos. Es más rápido, más preciso y más sencillo.

Suponiendo que se necesiten tantos datos, ¿cuán diferentes son los valores cada vez? ¿Podrías enviar solo 32 valores absolutos algunas veces y 16 u 8 bits de delta el resto del tiempo?
También ten en cuenta que, aunque es conveniente, no es necesario limitarse a valores de 32 o 16 bits. Siempre puedes enviar valores de 24 bits si solo necesitas eso; ni siquiera es necesario que cada valor sea múltiplo de 8; por ejemplo, podrías enviar 3 valores de 10 bits en 4 bytes si fuera necesario. Siempre que seas consistente al empaquetar y desempaquetar los datos, puedes empaquetar los bits como quieras. Empaquetar valores de esta manera no suele justificar la complejidad del software, pero si tienes un ancho de banda de datos limitado, vale la pena considerarlo.

S

samm_flynn
Miembro conocido

3 de marzo de 2025 #17

Hola @AndyA , gracias por tomarte el tiempo de leer mi pregunta.

> Volviendo a mi pregunta habitual cuando alguien tiene problemas para implementar algo complejo: ¿Estás seguro de que necesitas esto?

La necesidad de un control de 1 kHz escapa a mi control. Si por mí fuera, me habría quedado con unos 50 Hz. El motivo del control de 1 kHz es que la trayectoria que mi robot (una plataforma Stewart) debe seguir implica eventos similares a impactos; ese es el objetivo principal del proyecto actualmente: simular impactos. Muchos factores influyeron en la elección de una frecuencia de muestreo de 1 kHz.

> ¿Por qué 1 kHz? ¿A qué velocidad se mueve el robot como para necesitar control y retroalimentación de 1 kHz hacia y desde cada nodo?
> Muy poco movimiento físico requiere control a esa frecuencia; las cosas simplemente no se mueven tan rápido.

Con una frecuencia de muestreo de 1 kHz, puedo controlar la posición a 200 Hz. El objetivo principal de mi proyecto es simular impactos. Cuanto mayor sea el ancho de banda del controlador, más nítidos serán los impactos.

> Si se trata de algún tipo de bucle de control crítico para la sincronización, sería más
> únicamente parametros/ubicaciones de destino a una velocidad mucho menor.
>
> Si se trata únicamente de sincronización temporal, se puede hacer con una señal digital en lugar de datos. Es más rápido, preciso y sencillo.

Para sincronizar los diferentes ejes, ya utilizo varias señales digitales. Por ejemplo, el Teensy maestro genera una onda cuadrada de 2 kHz, y los tres Teensys esclavos detectan el flanco ascendente y solo envían un comando de posición en ese momento. Hay otra línea de pulsos que controla la velocidad de lectura de los datos del búfer de anillo, junto con una línea de recuperación (colector abierto) que puede anular la línea de pulsos. También hay líneas estándar para habilitación, parada de emergencia, etc.

Las trayectorias que necesito ejecutar son muy largas, por lo que no caben en la memoria del Teensy. Mi plan es transmitir los datos desde el PC al Teensys mientras uso un búfer de anillo en el Teensys esclavo.

> Suponiendo que se necesitan tantos datos, ¿cuán diferentes son los valores cada vez? ¿Podría bastar con enviar solo 32 valores absolutos algunas veces y enviar 16 u 8 bits de delta el resto del tiempo?

Teniendo esto en cuenta, mis controladores de motor toman valores absolutos, por lo que tengo que hacer esto en el software.

> Tenga en cuenta que, si bien es conveniente, no es necesario limitarse a valores de 32 o 16 bits. Siempre puede enviar valores de 24 bits si solo necesita uno; ni siquiera es necesario que cada valor sea múltiplo de 8; por ejemplo, podría enviar 3 valores de 10 bits en 4 bytes si es necesario. Siempre que sea consistente al empaquetar y

> desempaquetar los datos, puede empaquetar los bits como desee. Empaquetar valores de esta manera no suele justificar la complejidad del software, pero si tiene un ancho de banda de datos limitado, vale la pena considerarlo.

La retroalimentación, estoy planeando exactamente lo que dijiste, podría utilizar valores de 16 bits.

A

Andy A
Miembro conocido

3 de marzo de 2025 #18

Siempre puedes añadir 8/16 MB de RAM extra añadiendo uno o dos chips PSRAM. Esto te permite almacenar en búfer muchos más datos en la placa, ya sea antes o después. A tu velocidad, esto solo supone unos pocos segundos, así que puede que no sea suficiente para todo, pero significa que solo tienes que preocuparte por la velocidad de datos media en lugar de los picos. ¿

Podrías usar una velocidad variable? ¿Una velocidad de actualización más baja para la operación inicial y luego más alta durante la colisión? En ese caso, la velocidad más alta durante la colisión podría almacenarse en búfer y luego descargarse a una velocidad menor después del evento.

CAN-FD ofrece una velocidad de datos más alta que el UART, pero requerirá controladores adicionales. Dado que CAN está diseñado para ser un bus común entre todos los nodos y se basa en la difusión, puede ser una buena opción si necesitas enviar los mismos datos a todos los nodos, pero probablemente no sea una buena opción si cada nodo necesita enviar sus propios datos únicos al nodo maestro. Ejecutar buses independientes para cada dispositivo no es una opción, ya que solo hay una interfaz FD en lugar de 8 UART de hardware.

¿Qué tal una conexión en cadena de puertos USB serie? Cada Teensy tiene un dispositivo USB y un puerto host USB. El puerto host admite dispositivos serie. Entonces, ¿se puede conectar USB en todo el trayecto? Eso ofrecería mucho más ancho de banda que los UART físicos.

Edito: lo siento, al final fue una recopilación de ideas un poco aleatoria, ¿no?

S

samm_flynn
Miembro conocido

3 de marzo de 2025 #19

> Siempre puedes añadir 8 o 16 MB de RAM adicional añadiendo uno o dos chips PSRAM. Esto te permite almacenar en búfer muchos más datos en la placa, ya sea antes o después. A tu velocidad, esto solo supone unos pocos segundos, así que puede que no sea suficiente para todo, pero significa que solo tienes que preocuparte por la velocidad de datos promedio en lugar de por la máxima.

De acuerdo, ya pedí los chips RAM.

> ¿Podría utilizarse una tasa variable? ¿Una tasa de actualización más baja para la operación inicial y luego más alta durante la colisión? En ese caso, la tasa más alta durante la colisión podría almacenarse en búfer y luego descargarse a una velocidad menor después del evento.

Por diseño, el sistema funciona a una velocidad constante y los impactos también son continuos, contienen contenido de alta frecuencia, pero se repiten a

3 Hz.

> CAN-FD ofrece una mayor velocidad de datos que el UART, pero requiere controladores adicionales. Dado que CAN está diseñado para ser un bus común entre todos los nodos y se basa en la transmisión, puede ser una buena opción si se necesitan enviar los mismos datos a todos los nodos, pero probablemente no sea una buena opción si cada nodo necesita enviar sus propios datos al nodo maestro. Utilizar buses independientes para cada dispositivo no es una opción, ya que solo hay una interfaz FD en lugar de ocho UART de hardware.

Por la razón mencionada, CAN tampoco es una opción en mi opinión.

> ¿Qué tal una conexión en cadena de puertos USB serie? Cada Teensy tiene un dispositivo USB y un puerto host USB. El puerto host admite dispositivos serie. Entonces, ¿se puede conectar USB en todo el trayecto? Eso ofrecería mucho más ancho de banda que los UART físicos.

Esta parece ser la única solución viable por el momento. Acabo de probar el ejemplo USBHost_t36 con un par de Teensy 4.1 de repuesto y parece prometedor.

Incluso si añado tres, el Teensy maestro, conectado a tres Teensy esclavos, no sé cómo el bus USB distribuye la velocidad entre los nodos, pero los 480 Mbit/s máximos resultan en 160 Mbit/s o 80 Mbit/s en un solo sentido, lo que debería ser más que suficiente para funcionar a plena velocidad.

Solo necesito averiguar cómo conectar tres Teensy a un Teensy maestro como dispositivo USB serie y si necesito un concentrador. Experimentaré con la biblioteca USBHost_t36.

> editar - lo siento, terminó siendo una colección de pensamientos un tanto aleatorios, ¿no?

Estoy más que agradecido por tomarte el tiempo de leer mi publicación, te lo agradezco mucho.

desfragmentador

Miembro Senior+

3 de marzo de 2025 #20

@samm_flynn He probado el USB_Host Serial un poco, pero no estoy seguro y no recuerdo con cuánta frecuencia.

Para un uso normal (o para realizar una prueba), usar el "maestro" conectaría el USB_Host a un concentrador USB con alimentación.
Cualquier dispositivo deseado conectaría el puerto USB del dispositivo a ese concentrador.
Los dispositivos verían el host maestro para la transferencia de datos cuando se conectara a ellos.

S

samm_flynn
Miembro conocido

3 de marzo de 2025 #21

> defragster dijo:
>
> @samm_flynn He probado el USB_Host Serial un poco, pero no estoy seguro y no recuerdo con cuánta frecuencia.
>
> Para un uso normal (o para realizar una prueba), usar el "maestro" conectaría el USB_Host a un concentrador USB con alimentación.
> Cualquier dispositivo deseado conectaría el puerto USB del dispositivo a ese concentrador.
> Los dispositivos verían el host maestro para la transferencia de datos cuando se

He ejecutado los ejemplos, pronto compartiré una versión del enrutador serial, la diferencia será que, en lugar del serial hardware, utilizaré USBHost_t36 para enrutar paquetes, Ethernet fue otra opción atractiva, al menos para mi caso de uso, pero desafortunadamente compré 12 teensy 4.1 sin el chip.

S

samm_flynn
Miembro conocido

4 de marzo de 2025 #22

> samm_flynn dijo:
>
> He ejecutado los ejemplos, pronto compartiré una versión del enrutador serial, la diferencia será que, en lugar del serial hardware, utilizaré USBHost_t36 para enrutar paquetes, Ethernet fue otra opción atractiva, al menos para mi caso de uso, pero desafortunadamente compré 12 teensy 4.1 sin el chip.

C++:

```
#include <USBHost_t36.h>
byte packetBuffer[240009];

USBHost myusb;

USBHub hub1(myusb);

USBSerial_BigBuffer slaveSerial1(myusb, 1);
USBSerial_BigBuffer slaveSerial2(myusb, 2);
// USBSerial_BigBuffer slaveSerial3(myusb, 3);
// Global buffers for accumulating partial lines
String buffer1 = "";
String buffer2 = "";
String buffer3 = "";

void setup()
{
  while (!Serial) { delay(1); }
  if (CrashReport)
  {
    Serial.println(CrashReport);
  }

  myusb.begin();
  slaveSerial1.begin(480000000);
  slaveSerial2.begin(480000000);
  // slaveSerial3.begin(480000000);
```

```
  Serial.println("<ROUTER READY>");
}

void loop()
{
  myusb.Task();
  processSerial(slaveSerial1, buffer1, "1");
  processSerial(slaveSerial2, buffer2, "2");
  // processSerial(slaveSerial3, buffer3, "3");
}
void processSerial(USBSerial_BigBuffer& serial, String& buffer, const c
{
```

Si a alguien le interesa,

necesito averiguar cómo detectar una caída de la conexión USB Master Teensy, basándome en una interrupción en lugar de consultar Serial.dtr().

desfragmentador

Miembro Senior+

4 de marzo de 2025

con 2-3 dispositivos, se debe probar /observar la cantidad de mensajes/seg que puede recibir el host, ya que utiliza paquetes programados en una conexión compartida hacia y desde el host.

Última edición:4 de marzo de 2025

oría principal › Discusión general ›

Debes iniciar sesión o registrarte para responder ac

Compartir: