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++:

```cpp
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_
MotorAxis<FlexCAN_T4<CAN2, RX_SIZE_8, TX_SIZE_8>> motor2(CAN_ID2, 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

Not only is hardware serial physically about 500 times slower, but it also has far more CPU overhead than USB.

The serial ports on Teensy do at least have 4 byte FIFO. I believe we currently configure the interrupt watermark so half the FIFO reduces interrupt rate and the other half protects against unexpected interrupt latency. But even if you used all 4 bytes for efficiency (a rather risk design if you run at the highest baud rates) you're still looking at an interrupt and CPU overhead every 4 bytes.

USB at 480 Mbit/sec uses 512 byte packets. It's been a while since I've worked on the low-level code, but I believe we use a group 2048 byte buffers. But even if there's an interrupt for each packet, you're taking CPU time for an interrupt far less often. The USB hardware is also far more efficient, using bus master DMA to put the incoming data directly into the memory buffer.

Hardware serial can be used with the generic DMA controller, though it's not the more efficient bus master type. At least a couple people have made it work and shared code on this forum. Usually this involves more specific usage, since generic DMA needs to know data sizes in advance. DMA can't solve the fact that hardware serial is still at best 100 times slower, but it can at least mitigate much of the excessive CPU overhead that occurs when you push hardware serial to its faster baud rates. If you're running all 8 serial ports concurrently at high baud rates, you'll probably need that sort of efficiency (to reliably achieve the ~100 slower than USB capability).

S

SamiH
Member

Feb 26, 2025 #4

Could Ethernet be a viable solutions in such cases? Where high speed data transfer is required?

I mean like underlying hardware wise , does Ethernet have more buffer?
4 bytes is unbelievably small !

PaulStoffregen
Well-known member

Feb 26, 2025 #5

Yes, the built in Ethernet on Teensy 4.1 uses efficient bus master DMA and large buffers with QNEthernet library. Many people have reported achieving nearly the full 100 Mbit bandwidth when communicating on a LAN. Speed over the internet with higher packet latency varies rarely comes close to full theoretical hardware speed due to limited RAM on microcontrollers.

SamiH

S

shawn
Well-known member

Feb 26, 2025 #6

To add more details, in the *QNEthernet* library, there are 5 frame buffers for RX and TX, and each buffer is 1536 bytes. I haven't done any tuning for that number (5).

S

samm_flynn
Well-known member

Feb 26, 2025 #7

Dang, here I was thinking, I am gonna use a teensy as a serial router.

Thanks for the feedback guys

PaulStoffregen
Well-known member

Feb 26, 2025 #8

This is why it's always wise explain the context of your questions. If you just ask a narrowly focused question about speed, you'll (probably) miss out regarding answers like hardware serial simply not being meant for many Mbit/sec speed data (on any boards, not just Teensy). Better answers can really make the difference between starting your project in a way that's feasible versus wasting a lot of time only to learn later that the approach wasn't viable.

shawn

S

samm_flynn
Well-known member

Feb 27, 2025 #9

> PaulStoffregen said:
>
> This is why it's always wise explain the context of your questions. If you just ask a narrowly focused question about speed, you'll (probably) miss out regarding answers like hardware serial simply not being meant for many Mbit/sec speed data (on any boards, not just Teensy). Better answers can really make the difference between starting your project in a way that's feasible versus wasting a lot of time only to learn later that the approach wasn't viable.

|
Fair point, A bit of background for anyone coming across this post:

I have a robot with six motor controllers, and the entire control system is developed and tuned to operate at a sampling frequency of 1 kHz.

Due to the setup, I am using one Teensy per two axes, along with a master Teensy that sends a PWM synchronization signal at 1 kHz. The slave Teensies run the control algorithm based on this external pulse.

For communication, I need to send 6 floats (24 bytes) + 8 bytes of additional data, totaling 32 bytes per cycle at 1 kHz.
The feedback from the onboard sensors consists of 16 bytes per axis for 6 axes, totaling 96 bytes per cycle.

Since the robot is intended to run for prolonged periods, the Teensies cannot store all reference and feedback data in RAM. Therefore, I plan to implement a serial router subroutine on the master Teensy to handle data transfer between the PC and the slave Teensies for control and logging.

With three slave Teensies, the master needs to handle an aggregate 384 kB/s of data transfer.

For each slave Teensy, I need to receive data at 32 kB/s and send data back to the master at 96 kB/s, resulting in a total communication rate of 128 kB/s per slave. which is just over 1M buad. I might need to send feedback over for every two sampled averaged. Or I might try out the ethernetnet and a switch.

Ok I have osme work to do.

J

joepasquariello
Well-known member

Feb 27, 2025 #10

So, for every 1-ms control cycle, the master sends 36 bytes to each slave and receives 92 bytes. It's at least 10-bits per byte, so that is 1,280,000 bps. If the communication is half duplex, you'll need at least 2M baud. Are you just bread-boarding this for now, or actually building it? Will it be RS-232 or RS-485? I recommend using the SerialTransfer library. It will construct/deconstruct packets with CRC, with fairly low overhead, so you'll know for sure, at both ends, that good data has been received. It also has provision for a "Packet ID", so you can have different "data" packets and later you can add "configuration" packets, which you'll probably need. With serial communication, I always try to scale floats to 16-bit integers, which is almost always enough resolution for control. That could reduce your data rate by 1/3 or more.

defragster
Senior Member+

Feb 27, 2025 #11

This https://github.com/tonton81/SPI_MSTransfer_T4 was made for T_3.6 usage worked wonders in that case where it offloaded data to a second Teensy for processing and feeding to a PC.

It seems to have been updated for T_4. Assuming the 4 T_4's are in close proximity?

S

samm_flynn
Well-known member

Feb 27, 2025 #12

> joepasquariello said:
>
> So, for every 1-ms control cycle, the master sends 36 bytes to each slave and receives 92 bytes. It's at least 10-bits per byte, so that is 1,280,000 bps. If the communication is half duplex, you'll need at least 2M baud. Are you just bread-boarding this for now, or actually building it? Will it be RS-232 or RS-485?

I am using very short wires(<15 cm), with a teensy breakout boards and screw connectors for holding wires in place, intend to use rs 422 in the future, once everything is final.

> I recommend using the SerialTransfer library. It will construct/deconstruct packets with CRC, with fairly low overhead, so you'll know for sure, at both ends, that good data has been received. It also has provision for a "Packet ID", so you can have different "data" packets and later you can add "configuration" packets, which you'll probably need. With serial communication

Considering SerialTransfer , I might mod it a little to use crc32 for very large packets I intend to send.

> I always try to scale floats to 16-bit integers, which is almost always enough resolution for control. That could reduce your data rate by 1/3 or more.

My supervisor mentioned that as well, might end up doing that.

S

samm_flynn
Well-known member

Feb 27, 2025 #13

> defragster said:
>
> This https://github.com/tonton81/SPI_MSTransfer_T4 was made for T_3.6 usage worked wonders in that case where it offloaded data to a second Teensy for processing and feeding to a PC.
>
> It seems to have been updated for T_4. Assuming the 4 T_4's are in close proximity?

Will look into it, thanks for the suggestion sire.

J

joepasquariello
Well-known member

Feb 27, 2025 #14

> samm_flynn said:
>
> Considering SerialTransfer , I might mod it a little to use crc32 for very large packets I intend to send.

Heads up regarding SerialTransfer. The library as-is has max payload of 254 bytes per message, so it would work for the short messages you described, but not for very large ones.

> samm_flynn

S

samm_flynn
Well-known member

Mar 1, 2025 #15

I just realized can't use Ethernet, because teensy 4.1 I have doesn't have chip soldered (NE version I believe).
Can I use my master teensy as USB host as a drop in replacement in that case?
just to give a picture, this is what my working master teensy code looks like -

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();
    //   Serial.printf("\nspaceAvailable : %d\n", spaceAvailable);
```

A

AndyA
Well-known member

Mar 3, 2025

Falling back to my default question when someone is having trouble implementing something tricky. Are you sure you need this? Why 1 kHz? How fast is the robot moving that you need 1kHz control and feedback to and from each node?
Very little physical motion requires control at that frequency, things simply don't move that fast. If it's for some form of timing critical control loop then it would make more sense (and probably be more reliable) to run that control loop locally and only send parameters/target locations at a far slower rate.

If it's purely for time sync reasons you can do that with a digital signal rather than with data. It's faster, more accurate and simpler.

Assuming you do need that much data how different are the values each time? Could you get away with only sending 32 absolute values some of the time and send 16 or 8 bits of delta the rest of the time.
Also keep in mind that while convenient you don't need to stick to 32 or 16 bit values. You can always send 24 bit values if that's all you need, it doesn't even need to be a multiple of 8 for each value, e.g. you could send 3 10 bit values in 4 bytes if needed. As long as you are consistent in how you pack and unpack the data you can pack the bits any way you want. Packing values like this is not normally worth the added software hassle but if you are data bandwidth limited then it's worth looking at.

S

samm_flynn
Well-known member

Mar 3, 2025

Hi @AndyA, thanks for taking the time to read my question.

> Falling back to my default question when someone is having trouble implementing something tricky. Are you sure you need this?

The need for 1 kHz control is actually beyond my control. If it were up to me, I would have stuck with something like 50 Hz. The reason for 1 kHz control is that the trajectory my robot (a Stewart platform) needs to follow involves impact-like events—that's the main goal of the project right now: simulating impacts. A lot of things influenced on why I chose 1KHz smapling frequency.

> Why 1 kHz? How fast is the robot moving that you need 1kHz control and feedback to and from each node?
> Very little physical motion requires control at that frequency, things simply don't move that fast.

A with a controller sampling rate of 1Khz, enables me to do position control at 200Hz, main goal of my project is to simulate imapcts. Higher the controller bandwidth, sharper the impacts I can make.

> If it's for some form of timing critical control loop then it would make more sense (and probably be more reliable) to run that control loop locally and only send parameters/target locations at a far slower rate.
>
> If it's purely for time sync reasons you can do that with a digital signal rather than with data. It's faster, more accurate and simpler.

For syncing different axes, I'm already using several digital signals. For example, the master Teensy generates a 2 kHz square wave, and the three slave Teensys detect the rising edge and only send a position command at that moment. There’s another pulse line that controls the rate at which data is read from the ring buffer, along with a catch-up line (open collector) that can override the pulse line. Also, there are standard lines for enable, emergency stop, etc.

The trajectories I need to run are very long, so they can’t fit in the Teensy's memory. My plan is to stream the data from the PC to the Teensys while using a ring buffer on the slave Teensys.

> Assuming you do need that much data how different are the values each time? Could you get away with only sending 32 absolute values some of the time and send 16 or 8 bits of delta the rest of the time.

Considering this, but my motor drivers take in absolute values. so gotta do this in software.

> Also keep in mind that while convenient you don't need to stick to 32 or 16 bit values. You can always send 24 bit values if that's all you need, it doesn't even need to be a multiple of 8 for each value, e.g. you could send 3 10 bit values in 4 bytes if needed. As long as you are consistent in how you pack and unpack the data you can pack the bits any way you want. Packing values like this is not normally worth the added software hassle but if you are data bandwidth limited then it's worth looking at.

The feedback, I am planning to exactly what you said, might use 16 bit values.

A

AndyA
Well-known member

Mar 3, 2025 #18

You can always add 8 / 16 MB of extra RAM by adding a PSRAM chip or two. That allow you to buffer a lot more data on the board either before or after. That's still only a few seconds worth at your rates so it may not be enough for everything but means you only have to worry about average data rate rather than peak.

Could you use a variable rate? Lower update rate for the initial operation and then higher during the collision? In that situation the higher rate during the collision could be buffered and then offloaded at a lower speed after the event.

CAN-FD offers a higher data rate than the uart but will require extra drivers. Since CAN is designed to be a common bus between all nodes and is broadcast based this may be a good option if you need to send the same data to all the nodes but probably not a good choice if each node needs to send it's own unique data back to the master node. Running independent busses to each device isn't an option since there is only 1 FD interface rather than 8 hardware uarts.

How about a daisy chain of USB serial ports? Each teensy has a USB device and a USB host port. The host port supports serial devices. So can you run USB the whole way? That would give you far more bandwidth than physical uarts.

edit - sorry, that ended up being a bit of a random thoughts collection didn't it.

S

samm_flynn
Well-known member

Mar 3, 2025 #19

> You can always add 8 / 16 MB of extra RAM by adding a PSRAM chip or two. That allow you to buffer a lot more data on the board either before or after. That's still only a few seconds worth at your rates so it may not be enough for everything but means you only have to worry about average data rate rather than peak.

Agreed, already ordered the RAM chips.

> Could you use a variable rate? Lower update rate for the initial operation and then higher during the collision? In that situation the higher rate during the collision could be buffered and then offloaded at a lower speed after the event.

By design, the system runs at constant rate. and the impacts are continuous as well, they contain high frequency content, but repeat at 3 Hz.

> CAN-FD offers a higher data rate than the uart but will require extra drivers. Since CAN is designed to be a common bus between all nodes and is broadcast based this may be a good option if you need to send the same data to all the nodes but probably not a good choice if each node needs to send it's own unique data back to the master node. Running independent busses to each device isn't an option since there is only 1 FD interface rather than 8 hardware uarts.

For the reason mentioned, CAN is not an option in my opinion as well.

> How about a daisy chain of USB serial ports? Each teensy has a USB device and a USB host port. The host port supports serial devices. So can you run USB the whole way? That would give you far more bandwidth than physical uarts.

This seems like the only viable solution at the moment. I just tired the USBHost_t36 example with a pair of spare teensy 4.1, seems promising.

Even if I add three of them, the master teensy, connected to 3 slave teensy via, idk how the USB bus spilts the speed between the nodes, but the max 480 mbit/s results in 160mbit/s or 80mbit/s one way, should be more than enough to run things at full speed.

I just need to find out, how to connect 3 teensy to one master teensy as a usb serial device, and do I need a hub? I will play with the USBHost_t36 library.

> edit - sorry, that ended up being a bit of a random thoughts collection didn't it.

I am beyond grateful for taking the time to even read my post, much appriciated.

defragster
Senior Member+

Mar 3, 2025 #20

@samm_flynn Have tried the USB_Host Serial some - not sure and don't recall how extensively.

For 'normal use' (or a test to be performed) use the 'master' would attach USB_Host to a powered USB Hub.
Any desired 'devices' would plug the device USB port into that Hub.
The devices would see the master HOST for data transfer when it connects to them

S

samm_flynn
Well-known member

Mar 3, 2025 #21

> defragster said:
>
> @samm_flynn Have tried the USB_Host Serial some - not sure and don't recall how extensively.
>
> For 'normal use' (or a test to be performed) use the 'master' would attach USB_Host to a powered USB Hub.
> Any desired 'devices' would plug the device USB port into that Hub.
> The devices would see the master HOST for data transfer when it connects to them

I have ran the examples, will share a verison of the serial router soon, differnce will be, instead of hadware serial, will use the USBHost_t36 to route packets, Ethernet was another attractive options, for my use case at least, but I bought 12 teensy 4.1 without the chip unfortunately.

S

samm_flynn
Well-known member

Mar 4, 2025 #22

> samm_flynn said:
>
> I have ran the examples, will share a verison of the serial router soon, differnce will be, instead of hadware serial, will use the USBHost_t36 to route packets, Ethernet was another attractive options, for my use case at least, but I bought 12 teensy 4.1 without the chip unfortunately.

C++:

```cpp
#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
{
```

If anyone's interested.

Gotta figure out how to detect a master teensy USB connection drop, based on interrupt instead of polling for Serial.dtr()

defragster
Senior Member+

Mar 4, 2025

with 2-3 devices the number of messages/sec the Host can receive should be <*edit*> tested/observed as it is using scheduled packets on a shared connection to and from host.

Last edited: Mar 4, 2025

samm_flynn

Home > Forums > Main Category > **General Discussion** >

You must log in or register to reply here.

Terms and rules Privacy policy Help Home