**Bản quyền phân phối tài liệu thuộc về Phùng Thái Sơn-arduino.vn**

**Ngày 10/4/2018**

[//https://github.com/NickChungVietNam/transmit\_class](//https:/github.com/NickChungVietNam/transmit_class)

<http://arduino.vn/users/nick-chung/>

//[phungthaison96@gmail.com](mailto:phungthaison96@gmail.com)

//[phungthaion2012@gmail.com](mailto:phungthaion2012@gmail.com)

**Hướng dẫn sử dụng**

**Thư viện truyền dữ liệu bất kỳ (byte ,long, float , double ) theo gói tin cho arduino**

Mở đầu :

Đây là bài viết chỉnh sửa lại thư viện truyền kiểu dữ liệu bất kỳ UART\_ARDUIINO.h.

Thư viện mới sẽ giúp truyền tin an toàn hơn,  hỗ trợ truyền theo mảng nhị phân (binary), cho phép sử dụng nhiều cổng giao tiếp khác nhau cùng lúc.

Link bài cũ : <http://arduino.vn/tutorial/1471-truyen-cac-so-kieu-long-int-float-trong-giao-tiep-serial-uart>

Giới thiệu.

Giả sử bạn có 2 arduino đang kết nối với nhau bằng 1 giao tiếp nào đó (I2C, SPI, UART, IR ..) , có thể là không dây hay có dây . Bạn có 1 biến dữ liệu trong arduino A và muốn nó sẽ nằm trong arduino B bằng cách gửi qua giao tiếp đó.

Trong thiết bị điện tử logic chỉ có 2 mức tín hiệu là High (1) và Low (0) tương ứng với 2 mức điện áp. Khi xuất ra đường truyền, dữ liệu cũng sẽ chỉ là các Byte chứa các Bit nhị phân 0-1 như thế này



Vậy làm sao để thiết bị nhận (arduino B) tìm ra chính xác dữ liệu mong muốn ?

Cách tốt nhất để giải quyết vấn đề này là đặt ra những quy ước chung cho dữ liệu muốn gửi.

Máy thu sẽ dựa vào những dấu hiệu đó để nhận biết và tách lấy thứ nó cần.

Ví dụ : Chúng ta có 1 byte dữ liệu A với giá trị bằng 11 . Phía trước của A sẽ là 1 byte Start (B10101010) , Phía sau của A là 1 byte Stop (B10011010), Như vậy A có thể sẽ bằng B00000101.



Ta nói “có thể sẽ bằng” là vì A có thể là 1 giá trị nhiễu (không giống với dữ liệu gửi), hoặc khi A trùng với Start sẽ khiến máy thu không thể phân biệt…

Như vậy , để xác suất nhận đúng A tăng lên thì ta phải nâng cao quy ước như :

+ Gía trị của A chỉ nằm trong khoảng quy ước.

+Tăng độ dài của Start và Stop..vv

Cũng giống như khi bạn mua hàng online , để hàng có thể đến tận tay thì bạn cần điền đầy đủ thông tin của người nhận , mỗi một thông tin điền vào sẽ giúp cho cửa hàng loại trừ và thu hẹp khu vực của bạn .

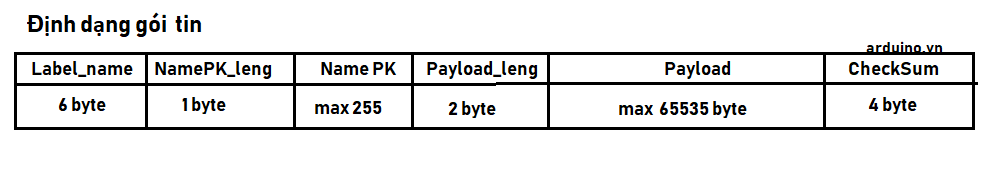
Quy ước càng chắc chắn thì dữ liệu càng đáng tin cậy.

Định dạng gói tin.

Nhiệm vụ của thư viện trong bài viết này gồm 2 phần cơ bản là Nén dữ liệu gửi và Giải nén dữ liệu nhận.

Mình quy ước định dạng gói tin cho thư viện của mình như sau:

LABEL\_NAME +NamePK\_Leng + NamePK + Payload\_leng +PlayLoad +CheckSum



* -LABEL\_NAME : tên nhãn bắt đầu gói tin (6)byte (mặc định ={0,1,2,3,4,5}) ,
* có thể chỉnh sửa LABEL\_NAME bằng thành viên RESET\_LABEL\_NAME(x,x,x,x,x,x)
* -NamePK\_Leng (NamePacket\_length) : độ dài tên gói tin (1 byte)
* -NamePK (Namepacket) : tên gói tin (nhiều nhất là 255 kí tự)
* -Payload\_leng : kích thước của Payload (2 byte , 8bit thấp trước + 8 bit cao sau)
* -PayLoad: phần dữ liệu chính : là dữ liệu phân giải của kiểu dữ liệu tương ứng, xếp liên tục nhau , (bit thấp nhất viết đầu tiên)
* PayLoad chỉ được chứa 1 loại kiểu dữ liệu , PayLoad có thể lên tới 65535 byte dữ liệu
* -CheckSum : kiểm tra lần cuối (4 byte)

Cách hoạt động:

Khi gửi :

Đơn giản là gộp dữ liệu và xuất ra đường truyền theo định dạng.

Riêng phần Payload sẽ được thêm vào theo ý định của người dùng.

Khi nhận:

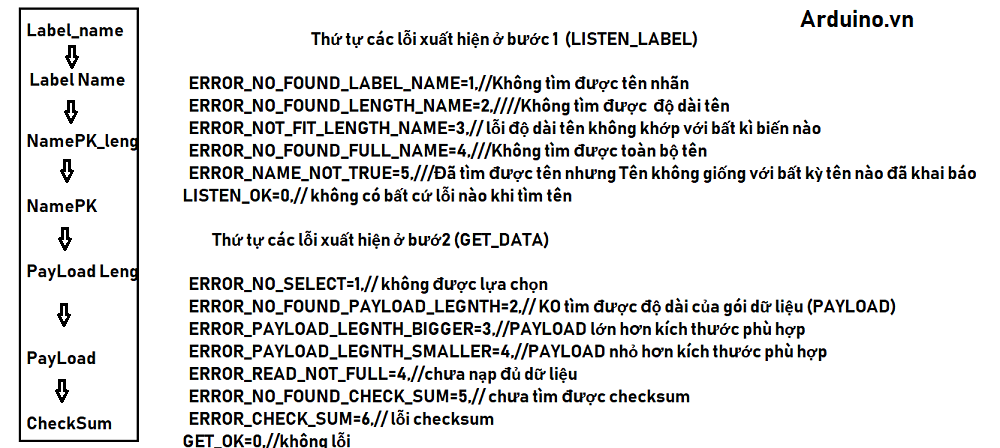
Bước 1 LISTEN\_LABEL: Phát hiện gói tin bằng cách kiểm tra 6 byte LABEL\_NAME, sau đó lấy chiều dài tên gói tin NamePK\_Leng và lọc lấy tên của gói NamePK. Cuối cùng sẽ kiểm tra lỗi và quyết định sẽ đổ dữ liệu PayLoad vào gói nào bằng câu lệnh rẽ nhánh.

Bước 2 GET\_DATA : Lấy chiều dài Payload\_leng và nạp vào biến lưu trữ (mảng hoặc con trỏ địa chỉ) cho đến khi đủ Payload\_leng thì dừng lại, Cuối cùng kiểm tra CheckSum để kiểm tra tính toàn vẹn của gói tin.

Việc nhận gói tin thành công chỉ xác nhận khi vượt qua tất cả các bước này.

Một lưu ý: Thư viện không dùng bộ nhớ đệm để lưu trữ dữ liệu PayLoad nhận được, dữ liệu nhận đến đâu sẽ ghi đè vào mảng hoặc biến đến đó ngay cả khi gói tin đó bị lỗi.

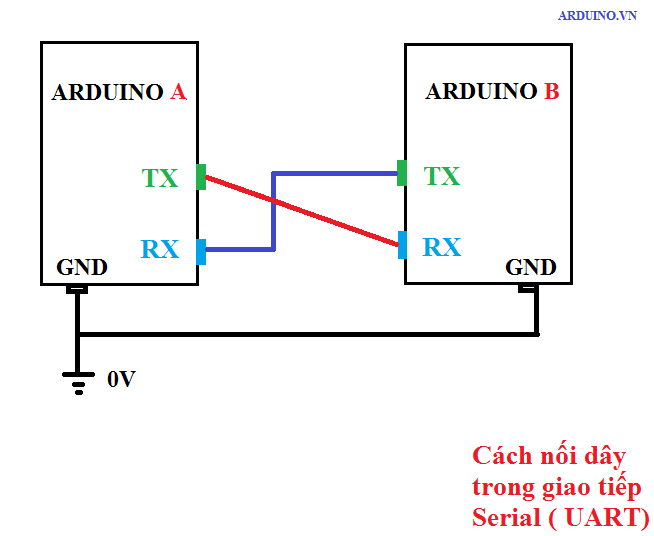
Tuy nhiên cả 2 hàm đều có khả năng trả về lỗi mà nó phát hiện, ta sẽ dựa vào đó đế quyết định xem có nên dùng dữ liệu đó hay không.

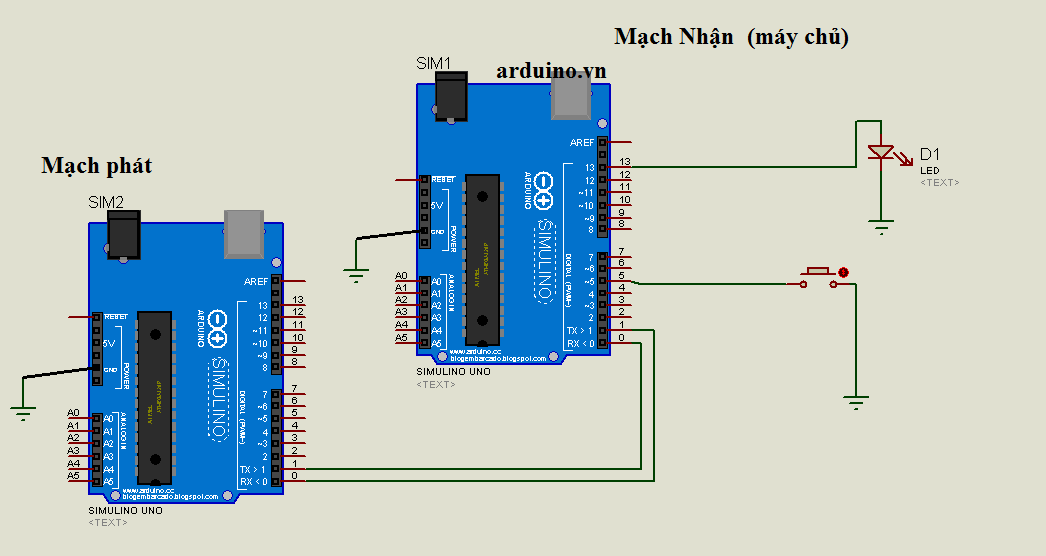


Cách sử dụng :

Toàn bộ ví dụ sử dụng giao tiếp UART để test.

Nối dây arduino RX của arduino này nối với TX của arduino kia..





Khai báo và sử dụng các hàm :

***Hàm dựng*** gói dữ liệu của class DATA\_TRANSMIT\_PACKET:

DATA\_TRANSMIT\_PACKET(String name\_);

Tạo 1 gói dữ liệu để định tên (NamePK ) cho gói .

***Hàm dựng ( constructor ) của class TRANSMIT\_CLASS :***

***TRANSMIT\_CLASS ( uint8\_t (\*rb)(), uint32\_t (\*av)() , void (\*cl)() ,void (\*wb)(uint8\_t));***

* Truyền vào 4 hàm với 4 nhiệm vụ :
* Rb: hàm đọc 1 byte dữ liệu
* Av: hàm đọc kích thước sẵn có của dữ liệu
* Cl:hàm xóa toàn bộ dữ liệu sẵn có
* Wb: hàm ghi 1 byte dữ liệu ra đường truyền.

Bạn có thể sửa lại các hàm này trong tùy chọn thành viên :

* **void ADD\_READ\_BYTE( uint8\_t (\*fc)() ;**
* **void ADD\_AVAILABLE( uint32\_t (\*fc)() ;**
* **void ADD\_CLEAR( void (\*fc)() ;**
* **void ADD\_WRITE\_BYTE( void(\*fc)(uint8\_t) ;**

**RESET\_LABEL\_NAME( uint8\_t n0, uint8\_t n1,uint8\_t n2,uint8\_t n3,uint8\_t n4,uint8\_t n5);**

* Khi cần thiết, bạn có thể sửa lại giá trị của LABEL\_NAME bằng hàm trên.

**uint8\_t LISTEN\_LABEL(uint32\_t timeout, DATA\_TRANSMIT\_PACKET \*\*array\_ );**

* Dùng để lắng nghe gói tin (bước 1) .
* Timeout: thời gian đợi lâu nhất (milli giây)
* array\_ : mảng chứa gói tin tổng hợp

**template <class Type> uint8\_t GET\_DATA(uint32\_t time\_out,DATA\_TRANSMIT\_PACKET x ,Type c, uint16\_t Count, ... );**

* Lấy dữ liệu nhận được và ghi đè vào từng BIẾN.
* Timout: thời gian đợi lâu nhất (milli giây)
* X: gói tin nhận biết .
* Type c: Kiểu dữ liệu của biến sẽ phân giải.
* Count: số lượng biến sẽ phân giải.
* … : liệt kê biến cần ghi đè vào đây.

**template <class Type> uint8\_t GET\_DATA\_ARRAY(uint32\_t time\_out,DATA\_TRANSMIT\_PACKET x ,Type c, uint16\_t Count, Type \*array\_ );**

* Lấy dữ liệu biến và ghi đè vào mảng chứa biến dữ liệu.
* Timeout: thời gian đợi lau nhất (mili s)
* X: gói tin nhận biết
* Type c: Kiểu dữ liệu của biến sẽ phân giải
* Count: số lượng biến sẽ phân giải (bằng với kích thước mảng)
* Array\_: lưu dữ liệu vào mảng này

**uint8\_t GET\_BINARY\_ARRAY(uint32\_t time\_out,DATA\_TRANSMIT\_PACKET x , uint16\_t SIZE\_ARRAY, uint8\_t \*array\_ , uint16\_t \*number\_get);**

* Lấy dữ liệu nhị phân và ghi đè vào mảng (từng byte 1).
* Timeout: thời gian đợi lau nhất (mili s)
* X: gói tin nhận biết
* SIZE\_ARRAY : kích thước lớn nhất của mảng lưu trữ.
* uint8\_t \*array\_ : mảng (kiểu dữ liệu bắt buộc)
* \*number\_get: hàm trả về kích thước Payload nhận được (số byte)

**template <class Type> uint8\_t SEND\_DATA( DATA\_TRANSMIT\_PACKET x , Type c, uint16\_t Count , ...);**

* Gửi dữ liệu theo từng biến riêng rẽ
* X: gói tin lựa chọn
* Type c: kiểu dữ liệu của biến
* Count; số lượng biến
* …: liệt kê biến cần gửi vào đây

**template <class Type> uint8\_t SEND\_DATA\_ARRAY( DATA\_TRANSMIT\_PACKET x , Type c, uint16\_t Count, Type \*array\_);**

* Gửi 1 mảng dữ liệu
* X: gói tin lựa chọn
* C: kiểu dữ liệu của mảng
* Count: số lượng phần tử sẽ gửi
* Array\_: mảng chứa dữ liệu gửi.

Các ví dụ

Mình đã chú thích khá chi tiết trong mỗi ví dụ, hi vọng bạn sẽ sớm làm quen với thư viện.

**Các lưu ý :**

* Các ví dụ của Thư viện sử dụng cổng UART trên arduino để test truyền nhận. Do đó khi bạn sử dụng Serial để làm chung vào việc khác (thường là Serial.print() gì đó để bugcode ) sẽ gây chồng chéo dữ liệu.
* Kiểu float và double trên từng dòng vi điều khiển (8bit / 32bit..) sẽ có cỡ lưu trữ khác nhau.
* Kiểu float hiện không được sử dụng để khai báo trong **SEND\_DATA() và** **GET\_DATA();** nhưng có thể sử dụng bình thường trong **SEND\_DATA\_ARRAY()** **và GET\_DATA\_ARRAY();**
* Như đã nói ở phần đầu , Thư viện không dùng bộ nhớ đệm để lưu trữ dữ liệu PayLoad nhận được,  dữ liệu nhận đến đâu sẽ ghi đè vào mảng hoặc biến đến đó ngay cả khi gói tin đó bị lỗi.
* Tuy nhiên cả 2 hàm đều có  khả năng trả về lỗi mà nó phát hiện, ta sẽ dựa vào đó đế quyết định xem có nên dùng dữ liệu đó hay không. Cách tốt nhất là chỉ xem những biến bị ghi đè như những biến đệm, khi nhận đúng thì mới truyền vào biến thực sự sử dụng.
* Bạn hoàn toàn có thể mở rộng thêm cổng giao tiếp (xem trong ví dụ ) hoặc thay đổi sang kiểu giao tiếp khác (UART, SPI, I2C, IR , sóng RF , ...) bằng việc thay đổi lại 4 hàm khi truyền vào hàm dựng.
* Trong khoảng thời gian Timeout ,máy thu sẽ đợi và ghép các gói tin (ngay cả khi có trễ trên đường truyền ) giúp kích thước nhận gửi có thể lên đến 65535 byte cho 1 gói tin.
* Độ dài tên gói tin (lớn nhất 255 kí tự ) càng lớn sẽ giúp gói có độ tin cậy cao hơn, tuy nhiên nó lại khiến băng thông tốn nhiều hơn, hãy cân nhắc.

Tạm kết :

**Ngày 10/4/2018**

Trước kia mình cũng đã viết 1 thư viện để gửi kiểu số bất kỳ trong truyền dẫn data(UART\_ARDUINO.h) <http://arduino.vn/tutorial/1471-truyen-cac-so-kieu-long-int-float-trong-giao-tiep-serial-uart>

Khi xây dựng nó mình cũng đã nhận ra một vài điểm yếu của UART\_ARDUINO.h

Cũng thật bất ngờ khi thấy bài viết giới thiệu thư viện có tới trên 11k view.

Nhận thấy nhu cầu không hề nhỏ từ phía cộng đồng, mặc dù bận rộn những mình cũng cố gắng dành thời gian phát triển lại (từ đầu) 1 cách cẩn thận hơn với hi vọng gỡ gạc lại phần nào những thiếu sót của thư viện cũ.

TRANSMIT\_CLASS.h có ưu điểm dễ sử dụng, trực quan, dễ nâng cấp và rẽ nhánh có thể sử dụng linh hoạt ở bất kỳ dạng truyền nào đòi hỏi tính đóng gói và bảo toàn dữ liệu (UART, SPI, I2C, IR , sóng RF , ...).

Mình cũng rất mong thư viện sẽ được mọi người đóng góp xây dựng thêm . Mọi phản hồi các bạn có thể gửi qua phần bình luận hoặc qua email của mình

[//https://github.com/NickChungVietNam/transmit\_class](//https:/github.com/NickChungVietNam/transmit_class)

<http://arduino.vn/users/nick-chung/>

//[phungthaison96@gmail.com](mailto:phungthaison96@gmail.com)

//[phungthaion2012@gmail.com](mailto:phungthaion2012@gmail.com)

**Ngày 10/4/2018**