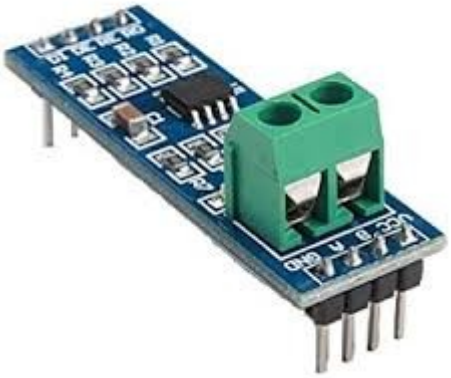
COMUNICAZIONE MASTER-SLAVE SU BUS RS485

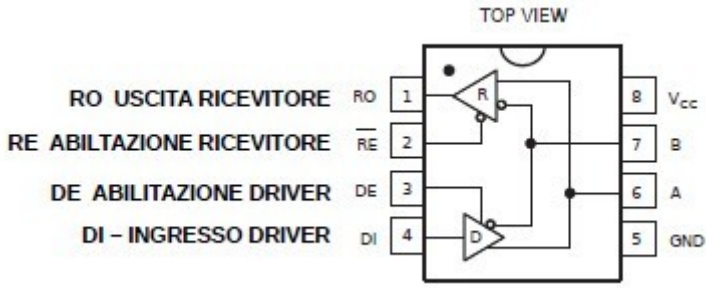
Con questa esperienza vogliamo realizzare una struttura che consenta ad un dispositivo Master di comunicare con più dispositivi Slave su un bus unico che sfrutta lo standard RS485.

Sia per il Master che per lo Slave utilizzeremo la scheda Arduino. Per comodità l'invio dei dati dal Master lo faremo utilizzando il monitor seriale di Arduino, lo stesso verrà utilizzato sugli Slave come debug per vedere i dati ricevuti.

La scheda Arduino non è dotata di un interfaccia RS485, per questo utilizzeremo la RS232 convertendo poì i segnali con un convertitore come quello in figura.

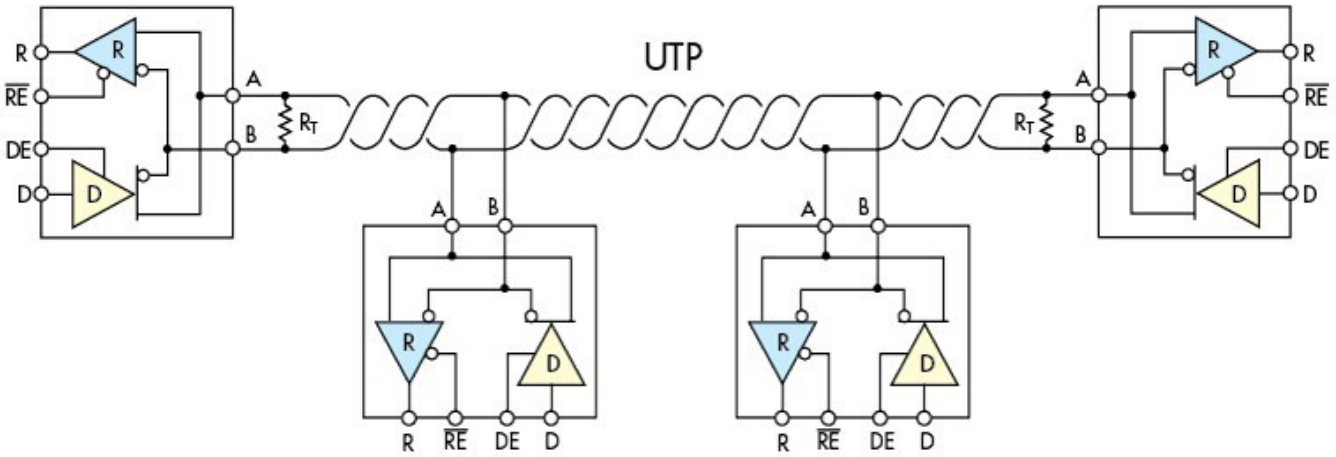
Questo circuito utilizza un integrato MAX485.
https://www.maximintegrated.com/en/products/interface/transceivers/MAX485.html



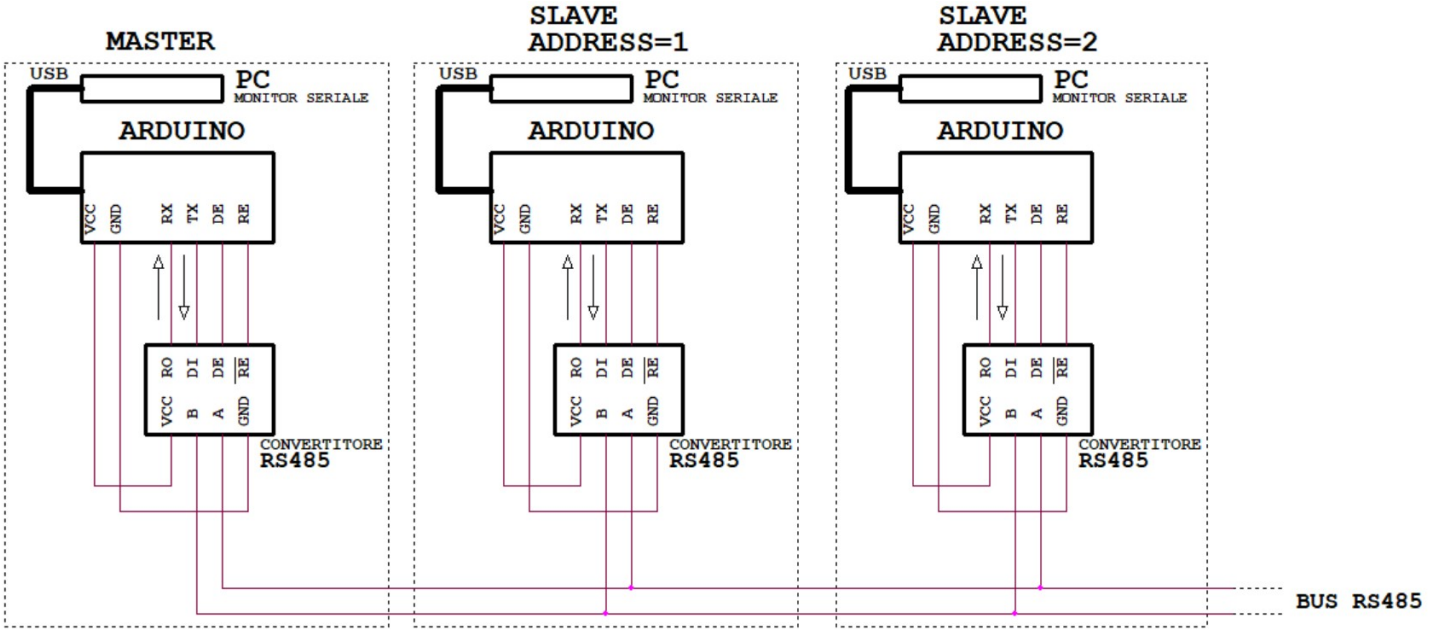


L'integrato va alimentato a 5Volt, e converte i dati dallo standard RS232 TTL, allo standard RS485.
Il pin TX della seriale RS232 di Arduino andrà collegato al pin 4 denominato DI-Ingresso Driver,
il pin RX della seriale RS232 di Arduino andrà collegato al pin 1 denominato RO-Uscita Ricevitore.
I due pin RE (2) e DE (3) permettono di abilitare la ricezione o la trasmissione, e più esattamente se RE=0 si attiva la ricezione sul pin RO, se DE=1 si attiva la trasmissione sul pin DI.

In uscita ci sono i due terminali A e B, dove viaggerà il segnale differenziale previsto dallo standard RS485, e potremo collegare tutti i dispositivi Master e Slave allo stesso bus come in figura.

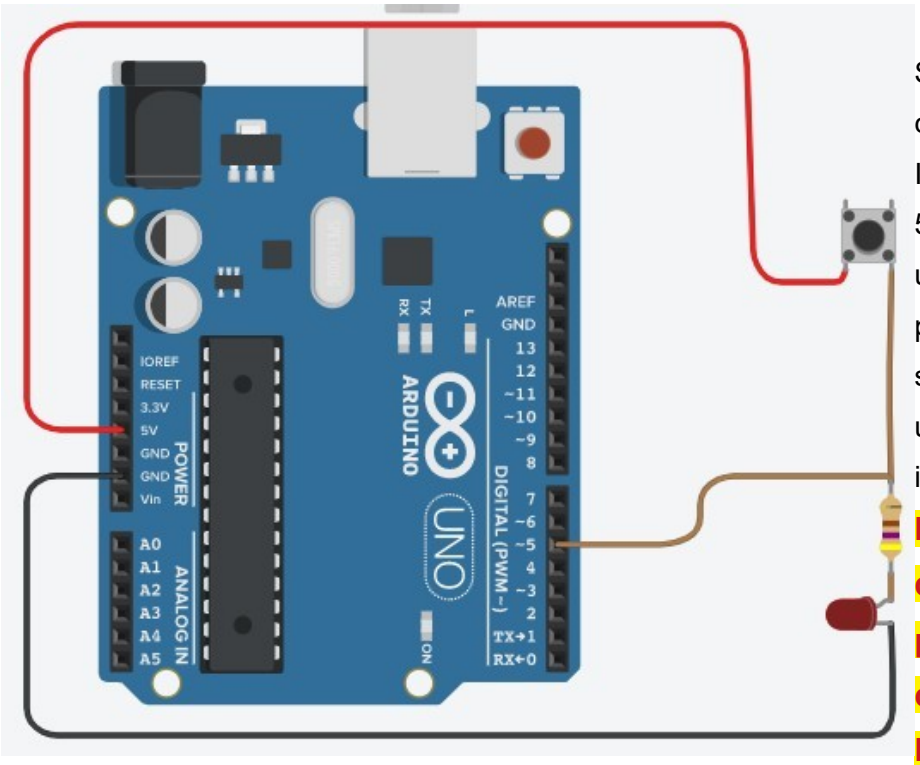


Sulla scheda oltre all'intergrato MAX485, troviamo anche tutte le resistenze di polarizzazione necessarie, pertanto il collegamento che dovremo realizzare sarà il seguente:



Nello schema è indicato un Master e 2 Slave, ma il programma che faremo, consentirà a tutti i dispositivi di svolgere entrambi le funzioni, ovviamente quando uno dei dispositivi viene configurato come Master, gli altri saranno configurati come Slave.

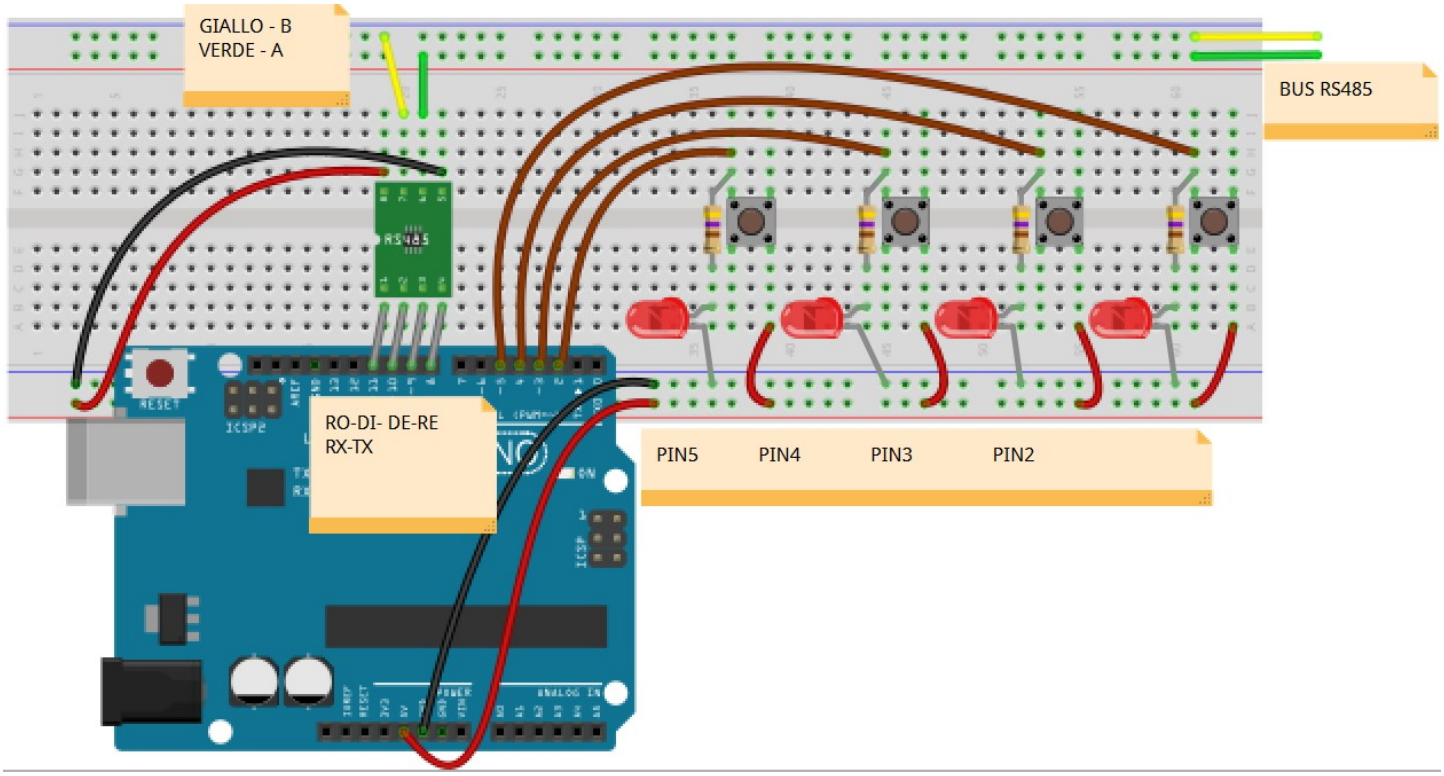
Le schede Arduino oltre che essere connesse al bus tramite il convertitore RS485, dovranno gestire 4 PULSANTI e 4 LED, che il Master comanderà da remoto.
Il circuito che realizzeremo per ognuno di queste 4 linee sarà il seguente:



Sia il pulsante che il LED sono connessi al pin 5 di Arduino.
In questo modo se configureremo il pin 5 come ingresso con resistenza di pull-up, potremo leggere lo stato del pulsante (1=premuto 0=non premuto) se invece configureremo il pin 5 come uscita possiamo accendere o spegnere il led.

In questo secondo caso dobbiamo evitare di premere il pulsante quando il pin 5 è configurato come uscita e sta a livello basso.

Dovendo fare un analogo discorso su 4 piedini di Arduino avremo il seguente schema di montaggio:



Nello schema sopra abbiamo la scheda Arduino connessa al bus tramite il convertitore RS485, e la stessa connessa a 4 PULSANTI e 4 LED.
Ora occorre definire le specifiche per la comunicazione, proviamo perciò ad inventarci un protocollo per la comunicazione tra il Master e gli Slave.
Dobbiamo considerare che i dati arrivano ad ogni Slave, che pertanto avrà un indirizzo, inoltre dobbiamo dare la possibilità al Master di decidere cosa chiedere allo Slave, se attivare o disattivare un uscita o se leggere lo stato di un ingresso. Perciò le informazioni che dovranno transitare dal Master allo Slave sono le seguenti:

- indirizzo dello slave
- funzione (*cosa si chiede di fare allo slave)*
- valore (*se si chiede di gestire un'uscita, occorre indicare il valore)*

Lo Slave dovrà poi rispondere eseguendo quanto richiesto, e rispondendo con un messaggio il cui contenuto va sempre definito nel nostro protocollo.

Si potrebbe perciò pensare ad un messaggio che parte dal Master, composto dai seguenti 5 byte espressi tutti in formato ASCII:

- 1°byte - indirizzo dello Slave a cui è indirizzato il messaggio
- 2°byte – funzione richiesta
- 3° e 4° byte - numero del dispositivo da gestire, con due byte in ASCII possiamo sceglere uno dei 13 pin di Arduino anche se nello schema ne abbiamo solo 4 collegati.
- 5° byte - valore da scrivere sul pin richiesto, essendo nel nostro caso pin digitali, potremo scrivere 0 oppure 1, sempre in formato ASCII.

La risposta dello Slave invece potrebbe essere l'eco dei 5 caratteri, ed in caso di richiesta di lettura, potrebbe contenere nel 5°byte il valore del pulsante cioè 0 o 1.

Uno schema della struttura della comunicazione potrebbe essere il seguente.

STRUTTURA VETTORE SCAMBIO DATI CHAR DATI[5];

INDICE	CONTENUTO	VALORI POSSIBILI IN ASCII
BYTE 0	ADDRESS	1-2
BYTE 1	FUNCTION	R-W
BYTE 2	HIGH	0-1
BYTE 3	LOW	1-2-3-4-5-6-7-8-9-0
BYTE 4	VALUE	0-1

esempi

RICHIESTA	**RISPOSTA**	**AZIONE**
1W051	1W051	SLAVE 1 LED 05 ON
2W120	2W120	SLAVE 2 LED 12 OFF
1R08X	1R080	SLAVE 1 LEGGI BIT 08 nell'esempio il bit è a 0
2R14X	2R141	SLAVE 1 LEGGI BIT 14 nell'esempio il bit è a 1

Come si può vedere negli esempi, se il Master decide di leggere lo stato di uno dei 4 pulsanti, nel 5°byte non servirà indicare un valore, questo campo potrebbe perciò contenere un qualsiasi valore indicato nell'esempio con la x.
Inoltre come abbiamo detto, sono stati dedicati due byte per la scelta del pin da gestire, ma nel nostro caso abbiamo solo 4 LED collegati e non servirebbero due caratteri. Ma essendo stata fatta questa scelta per poter aggiungere altri dispositivi su tutti i pin di Arduino, dobbiamo lasciare questa struttura, ed dovendo gestire il LED collegato al pin 5 nei due byte dovremo mettere '0' e '5'.

In pratica nel prendere queste decisioni abbiamo definito un protocollo di comunicazione su un bus che utilizza lo standard RS485.

La parte sicuramente più complessa sarà quella relativa alla parte software.
Dobbiamo prevedere che la scheda che funge da Master, può diventare anche uno Slave. Per questo conviene nel programma mettere la scheda in modalità "lettura" dal bus, e ciò si può fare mettendo a 0 i pin RE e DI (vedi la descrizione del convertitore sopra). La modalità invio dei dati va attivata solo quando la scheda deve inviare il dato, ed al termine dell'invio deve tornare in modalità "lettura".
Di seguito il programma che dovremo analizzare in ogni sua parte.

```
#include <SoftwareSerial.h>

SoftwareSerial rs485(11, 8); // RX, TX

#define sel_DE 9  //sel=0 lettura sel=1 scrittura
#define sel_RE 10  //sel=0 lettura sel=1 scrittura

char tipo;        //0=master 1=slave
char cont;
char ricevi[5];
char myaddress='1';
unsigned int attesa,cicli_attesa=60000;

//-------------------------------------------------------
//in base al valore di tipo R o W
//accendo o spengo l'uscita selezionata in base al valore
//presente sull'array[4]
//leggo lo stato dell'ingresso indicato
//e scrivo il valore nell'array[4]
void esegui(char pin, char tipo){
       int lettura;

       //SCRITTURA
       if(tipo=='W')
        {
          pinMode(pin,OUTPUT);
          if(ricevi[4]=='0') digitalWrite(pin,LOW);
          if(ricevi[4]=='1') digitalWrite(pin,HIGH);
        }//if
       //LETTURA
       if(tipo=='R')
        {
          pinMode(pin,INPUT);
          lettura=digitalRead(pin);
          if(lettura==0) ricevi[4]='0';
          if(lettura==1) ricevi[4]='1';
        }//if
}//esegui



//-------------------------------------------------------
//legge e restituisce il numero del bit selezionato nel messaggio
char decodifica_bit()
{
    char valore=0;
    char temp;

    temp=ricevi[2]-48; //dal codice ASCII ricavo le decine
    valore=temp*10;  //calcolo le decine
    temp=ricevi[3]-48; //dal codice ASCII ricavo le unità
    valore=valore+temp; //sommo le decine alle unità
    return(valore);
}//decodifica_esegui
```

```
//-------------------------------------------------------------
//invia array ricevuto con risposta
void invia_eco()
{
  char cont_char=0;

  digitalWrite(sel_DE,HIGH); //modo scrittura
  digitalWrite(sel_RE,HIGH);

  while(cont_char<5)
   {
     rs485.write(ricevi[cont_char]);
     cont_char++;
   }//while
  digitalWrite(sel_DE,LOW); //modo lettura
  digitalWrite(sel_RE,LOW);
}//invia_eco



//-------------------------------------------------------------
void setup() {
   Serial.begin(9600);  //seriale per monitor seriale
   rs485.begin(9600);   //seriale per bus RS485
   pinMode(sel_DE,OUTPUT); //imposto il pin direzione come uscita
   pinMode(sel_RE,OUTPUT); //imposto il pin direzione come uscita
   digitalWrite(sel_DE,LOW); //modo lettura
   digitalWrite(sel_RE,LOW); //modo lettura
   attesa=cicli_attesa;
}
```

```
//-------------------------------------------------------
void loop() {

    //se ricevo un carattere dal monitor seriale lo invio sulla RS485
    if(Serial.available())
      {
        digitalWrite(sel_DE,HIGH); //modo scrittura
        digitalWrite(sel_RE,HIGH);
        rs485.write(Serial.read());
        digitalWrite(sel_DE,LOW);  //modo lettura
        digitalWrite(sel_RE,LOW);
      }//if

    //ricezione caratteri
    if(rs485.available())
        {
           ricevi[cont]=rs485.read();
           Serial.println(ricevi[cont]);
           cont++;
           attesa=cicli_attesa;
        }//if
    else
        {
          delayMicroseconds(1);
          attesa--;
          if(!attesa)
            {
              cont=0;
              attesa=cicli_attesa;
            }//if
        }//else

      //analisi messaggio al 5° carattere arrivato
      if(cont==5)
          {
            cont=0;
            //verifica se corrisponde all'indirizzo della scheda
            if(ricevi[0]==myaddress)
             {
                esegui(decodifica_bit(),ricevi[1]);// richiamo le funzioni decodifica ed esegui
                invia_eco();  //invio la risposta
                cont=0;
             }//if
           }//if
    }//loop
```

Una volta realizzato il circuito e programmato le schede, occorre verificare il funzionamento inviando i comandi per la gestione dei GPIO sul bus, visualizzando con un oscilloscopio i segnali che transitano sul bus.
Successivamente provare a modificare il programma per inserire la funzione “lampeggia” **L** per far lampeggiare il numero di volte indicato nell’ultimo byte uno dei 4 led.