



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato **Computer Systems Design**

Progetto di sistemi embedded

Anno Accademico 2020/21

Studenti:

Michele Maresca M63/1151

Vincenzo Riccardi M63/1146

Marco Feliciano M63/1136

INDICE

PROGETTO DI SISTEMI EMBEDDED	I
1. ESERCIZIO UART TAPPO	3
1.1 Specifiche di Progetto	3
1.2 Architettura del Sistema	3
1.3 Configurazione periferiche e pin	4
1.4 Descrizione del programma implementato	5
2. SERIALE ST – SERIALE SOFTWARE ARDUINO – SERIALE COMPUTER	8
2.1 Specifiche di progetto	8
2.2 Architettura del sistema	8
2.3 Configurazione periferiche e pin	9
2.4 Descrizione del programma implementato	11

1. Esercizio UART TAPPO

1.1 Specifiche di Progetto

Il primo esercizio proposto riguarda l'uso dell'UART, presente sulla board di sviluppo STM32F303 Discovery, in configurazione TAPPO. L'esercizio simula due nodi in comunicazione dove uno svolge il ruolo di transmitter, l'altro di receiver.

Il trasmettitore invia al più M messaggi di lunghezza N, in particolare il protocollo di comunicazione è asincrono ed il singolo messaggio è inviato se e solo se è avvenuta la pressione del pulsante USER presente sulla board di sviluppo, gestendo tale evento con una apposita interruzione.

Il ricevitore verifica se l'ultimo byte del messaggio in arrivo è diverso da zero: in caso affermativo accende il led relativo al numero del messaggio ricevuto (es. i-esimo messaggio ricevuto accendi il led in posizione $i \bmod \text{numero_di_led}$), altrimenti non accende il led.

Anche per la fase di ricezione è previsto l'uso delle interrupt per l'accensione progressiva dei led di diagnostica.

1.2 Architettura del Sistema

L'architettura complessiva è rappresentata in figura 1, in particolare la figura 1.1 raffigura la vista logica e la figura 1.2 raffigura la vista fisica.

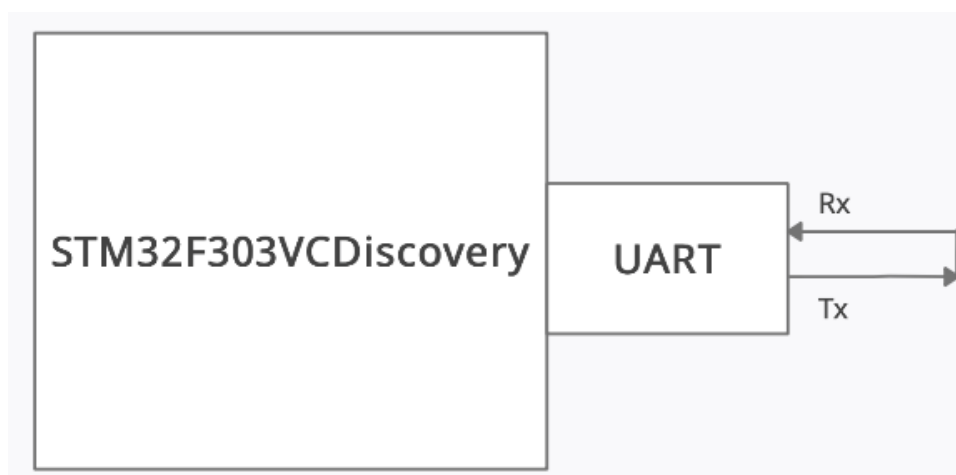


Figura 1.1: Architettura logica

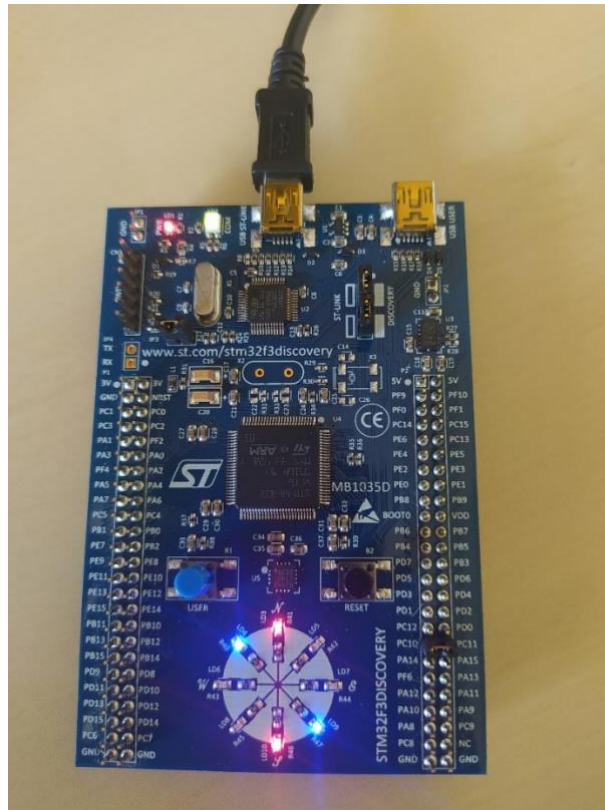


Figura 1.2: Architettura fisica

1.3 Configurazione periferiche e pin

La periferica di comunicazione seriale è stata configurata nel seguente modo. È stata impiegata l'UART4 configurata in modalità asincrona, 8 bit di messaggio senza bit di parità, 1 bit di stop e un baud rate di 115200 bit/sec. Dal punto di vista del trasmettitore è stato configurato il pin PC10 per la trasmissione, mentre per quanto riguarda il ricevitore è stato configurato il pin PC11. È stato abilitato il pin PA0 per l'uso del pulsante USER, in modalità interrupt esterno sul fronte di salita, ovvero l'interrupt si scatena alla pressione del pulsante. Sono state abilitate le interruzioni tramite NVIC lasciando i livelli di priorità di default, abilitando la linea EXTI0 per il pulsante USER e la linea EXTI34 per l'UART4. L'interrupt dell'UART4 si scatena all'avvenuta ricezione di un intero messaggio. Per consentire lo scambio di messaggi tra porto di trasmissione e porto di ricezione dell'USART4 è stato usato un jumper, disponibile sulla board di sviluppo, che collega i pin PC11 e PC10.

1.4 Descrizione del programma implementato

Il programma prevede una prima parte di codice autogenerata dovuta alla configurazione hardware del dispositivo attraverso l'apposito tool fornito dall'ide cube di ST.

Dalla Pinout view, si abilita il pin PA0, per l'uso del pulsante USER, cambiando la modalità di funzionamento da GPIO_input in GPIO_EXTI, poi si abilitano le interrupt andando ad attivare la linea EXTI0 dal menù dell'NVIC, infine si configura il pin PA0 dal menù GPIO in *“External Interrupt Mode with Rising Edge trigger detection”*.

Si attiva la USART4 dal menù *“Connectivity”* in modalità asincrona e dal menù NVIC si abilita la linea di interruzione EXTI34.

Sono state usate le seguenti variabili globali:

- N, usata per definire la dimensione in numero di byte del singolo messaggio;
- M, usata per definire il massimo numero di messaggi da ricevere (dopo il quale il sistema “riparte”);
- Counter, variabile che gestisce il conteggio dei messaggi ricevuti ed è usata per l'accensione progressiva dei led;
- Bufftrasmit, matrice di char che su ogni riga conserva le informazioni di un singolo messaggio da trasmettere, definito hard-coded nel codice;
- Buffreceive, matrice di char che su ogni riga conserva le informazioni di un singolo messaggio ricevuto.

Il codice è riportato in figura 1.3.

```
uint8_t N = 5; //numero di byte per messaggio
uint8_t M = 6; //numero di messaggi
uint8_t counter = 0;
char* bufftrasmit[] = {"ciao ", "marco", "plut0", "pipp0", "tutto", "bene?"};
char* buffreceive[] = {"00000", "00000", "00000", "00000", "00000", "00000"};
```

Figura 1.3

Per consentire l'invio del messaggio al rilascio del pulsante USER è stata sovrascritta la funzione *“HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)”* dichiarata come void che ha in ingresso un pin GPIO.

Nella funzione è presente una chiamata alla funzione di trasmissione UART *“HAL_UART_Transmit_IT”*, che ha in ingresso l'indirizzo del descrittore del dispositivo seriale, un puntatore ad unsigned int da 8 bit per l'indirizzo di testa del

messaggio da trasmettere e la dimensione in byte del messaggio; non è prevista la definizione di un intervallo di tempo essendo la funzione non bloccante.

Il codice è riportato in figura 1.4.

```
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin){  
    HAL_StatusTypeDef status = HAL_UART_Transmit_IT(&huart4, (uint8_t*) bufftrasmit + N*counter, N);  
    assert(status == HAL_OK);  
}
```

Figura 1.4

Per la ricezione è stata sovrascritta la funzione “*HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)*” dichiarata come void che prevede in ingresso un puntatore all’UART che ha appena ricevuto un messaggio. La funzione prevede una variabile locale “*last_byte*” che conserva le informazioni dell’ultimo byte del messaggio ricevuto.

Sono presenti poi tre if:

- il primo verifica se il messaggio ricevuto è il primo in tal caso spegni tutti i led;
- il secondo controlla se l’ultimo byte del messaggio è diverso da zero, in tal caso abilita il led i-esimo in base al valore del counter;
- il terzo verifica se counter è pari ad M, in tal caso resetta il valore di counter.

È presente una chiamata alla funzione di ricezione “*HAL_UART_Receive_IT*” usata per indicare al trasmettitore che la periferica è pronta a ricevere, che ha in ingresso l’indirizzo del descrittore del dispositivo seriale, un puntatore ad unsigned int da 8 bit per l’indirizzo di testa della locazione libera su cui memorizzare il prossimo messaggio e la dimensione in byte dell’area di memoria, ovvero il numero di byte da ricevere; non è prevista la definizione di un intervallo di tempo essendo la funzione non bloccante.

Il codice è riportato in figura 1.5.

```

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    char last_byte = buffreceive[counter][N-1];

    if (counter == 0)
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_All , GPIO_PIN_RESET);

    if (last_byte != '0')
        HAL_GPIO_WritePin(GPIOE, 1<<(8+(counter%8)), GPIO_PIN_SET);

    counter++;

    if (counter == M)
        counter = 0;

    HAL_StatusTypeDef status = HAL_UART_Receive_IT(&huart4, (uint8_t*) buffreceive + N*counter, N);
    assert(status == HAL_OK);
}

```

Figura 1.5

Infine, nel main è presente una chiamata a “*HAL_UART_Receive_IT*”, per indicare la disponibilità a ricevere della periferica UART4.

2. Seriale ST – Seriale Software Arduino – Seriale Computer

2.1 Specifiche di progetto

Questo progetto riguarda il collegamento tra più nodi. In particolare, si collega la seriale della scheda STM32F3 alla seriale di una scheda Arduino Uno. La scheda Arduino è collegata ad un Personal Computer che funge da monitor, e ad una breadboard su cui sono presenti un bottone ed un LED. Quando una delle due schede invia un byte all'altra scheda mediante la seriale, l'altra fa commutare lo stato di un LED, come diagnostica di una corretta comunicazione. Quando un utente preme il bottone USER sullo schedino dell'ST, viene avviata la trasmissione di un byte sulla seriale; una cosa analoga avviene quando l'utente preme il bottone presente sulla breadboard collegata ad Arduino. Inoltre, la scheda Arduino mantiene due contatori, uno per i messaggi che ha ricevuto dalla scheda ST, ed uno per i messaggi che ha inviato: in questo modo, in fase di invio/ricezione di un messaggio, trasmette mediante la seriale USB connessa al PC il numero di messaggi inviati/ricevuti.

2.2 Architettura del sistema

La seriale hardware dell'Arduino è impegnata nella comunicazione con il Personal Computer, dunque l'unica possibilità per collegare l'Arduino con la scheda dell'ST mediante seriale sta nell'utilizzo di due PIN liberi sui quali implementare il protocollo seriale via software (useremo una libreria per questo).

Dunque, la vista logica è rappresentata in figura 2.1.

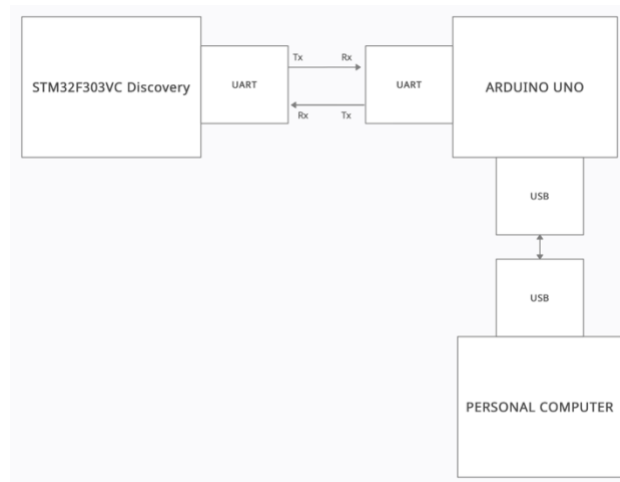


Figura 2.1

La vista fisica è rappresentata in figura 2.2.

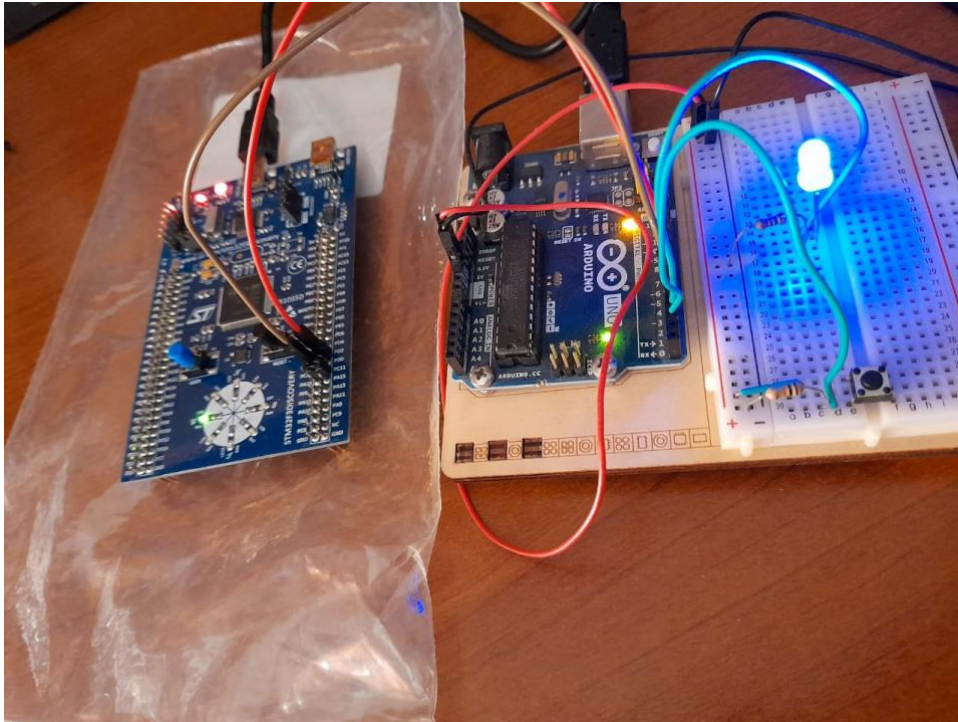


Figura 2.2

In particolare, il pin PC12 è il TX dell'ST ed è collegato all'RX software dell'Arduino, che in questo caso è il pin 4; il pin PD2 è l'RX dell'ST ed è collegato al TX software dell'Arduino, che in questo caso è il pin 5. La seriale hardware dell'Arduino è impegnata dall'interfaccia USB, mentre sul Personal Computer viene implementato via software il protocollo seriale sul collegamento USB.

2.3 Configurazione periferiche e pin

Sulla scheda ST, la configurazione è simile a quella dell'esercizio precedente, solo che è stata usata la UART5 invece dell'UART4, ed è stata fatta una configurazione aggiuntiva: poiché sulla scheda Arduino usiamo una seriale implementata via software, non possiamo avere un baud rate molto elevato, perché non lo supporterebbe e come fenomeno si avrebbe che per ogni messaggio inviato dall'Arduino si avrebbero N ricezioni sull'ST, e al contrario per ogni trasmissione avviata dall'ST si potrebbe perdere il messaggio sull'Arduino perché il campionario non riuscirebbe a sovracampionare la linea. Dunque, per andare "safe" configuriamo un baud rate pari a 9600, da entrambi i capi (nota: il baud rate di default dell'ST è 115200!). Su STM32Cube IDE, tale parametro si può configurare nella pinout configuration, andando in Connectivity -> UART5 -> Parameter Settings -> Basic Parameters -> Baud Rate. Dunque, sulla falsa riga dell'esercizio precedente, configuriamo la scheda ST in modo

che essa possa accettare le interruzioni sulla pressione dell'USER button e sulla ricezione da UART.

Per quanto riguarda la configurazione della scheda Arduino, in questo caso non abbiamo un IDE che genera automaticamente il codice a partire da un file di configurazione; abbiamo usato l'editor in browser di Arduino Cloud e abbiamo scritto il seguente codice per il setup:

```
23 void setup() {  
24     Serial.begin(9600);  
25     pinMode(rx, INPUT);  
26     pinMode(tx, OUTPUT);  
27     pinMode(button, INPUT);  
28     pinMode(led, OUTPUT);  
29     st_serial.attachInterrupt(rcv_from_sw_serial);  
30     st_serial.begin(9600);  
31     while(!Serial){}  
32     Serial.println("Started");  
33 }
```

Figura 2.3

Avendo le seguenti variabili in memoria:

```
4 #include <NeoSWSerial.h>  
5  
6 const uint8_t rx = 4;  
7 const uint8_t tx = 5;  
8 const uint8_t button = 2;  
9 const uint8_t led = 3;  
10 uint8_t data = 7;  
11 uint16_t rcvd_from_st = 0;  
12 uint16_t sent_to_st = 0;  
13 NeoSWSerial st_serial(rx, tx);
```

Figura 2.4

Così facendo abbiamo inizializzato la seriale hardware per il collegamento con il PC, la seriale software sui pin chiamati rx e tx, l'opportuna configurazione dei pin come input o output, abbiamo inizializzato delle variabili ed abbiamo anche configurato la interrupt per la ricezione dalla seriale; per quanto riguarda il bottone presente sulla breadboard, lavoreremo invece in polling all'interno del main. Notare che, alla fine del setup, cominciamo a mandare al monitor seriale un messaggio "Started" di diagnostica.

2.4 Descrizione del programma implementato

A valle della fase di configurazione, l'implementazione del programma è abbastanza straight-forward.

Sulla scheda dell'ST, definiamo un char da inviare *bufftrasmit* ed uno per la ricezione, *buffreceive*. La ISR per l'invio è quella relativa alla pressione del button:

```
71 void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin){
72
73     char* buff = &bufftrasmit;
74     HAL_StatusTypeDef status = HAL_UART_Transmit_IT(&huart5,(uint8_t*) buff , 1);
75     assert(status == HAL_OK);
76     // HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
77
78 }
```

Figura 2.5

Mentre quella per la ricezione, che commuta lo stato di un LED in seguito alla ricezione di un byte dalla seriale:

```
80 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
81
82     char* buff = &buffreceive;
83     HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_15);
84
85     HAL_StatusTypeDef status = HAL_UART_Receive_IT(&huart5,(uint8_t*) buff, 1);
86     assert(status == HAL_OK);
87
88 }
```

Figura 2.6

Come al solito bisogna mettere una Receive_IT nel main prima del loop infinito, per fare sì che avvenga la prima ricezione.

Per quanto riguarda il codice di Arduino, abbiamo già visto il setup e l'inizializzazione delle variabili, ci restano da vedere la ISR di ricezione ed il main loop nel quale si fa polling sul bottone.

Il main loop:

```
35 void loop() {  
36   /* Clear */  
46   if (digitalRead(button) == HIGH){  
47     while (digitalRead(button) == HIGH){  
48       st_serial.write(data);  
49       sent_to_st += 1;  
50       Serial.print("Number of messages sent: ");  
51       Serial.print(sent_to_st, DEC);  
52       Serial.println();  
53     }  
54   }
```

Figura 2.7

Notare che in questo caso il messaggio non viene inviato in seguito alla pressione del bottone, ma in seguito al rilascio.

L'ISR di ricezione:

```
15 static void rcv_from_sw_serial(uint8_t c){  
16   rcvd_from_st += 1;  
17   digitalWrite(led, !digitalRead(led));  
18   Serial.print("Number of messages received: ");  
19   Serial.print(rcvd_from_st, DEC);  
20   Serial.println();  
21 }
```

Figura 2.8

Notare che la funzione che abbiamo impropriamente chiamato ISR prende in ingresso un carattere ricevuto dalla seriale, quindi è più corretto dire che tale funzione viene chiamata dalla ISR; in effetti c'è una `read()` implicita, che non viene chiamata da noi ma è a carico della libreria, che fornisce in input alla nostra funzione il carattere letto. Per completezza, riportiamo anche uno snippet di codice che mostra in che modo si poteva implementare su Arduino la ricezione da seriale lavorando in polling:

```
37 if (st_serial.available()) {  
38   st_serial.read();  
39   rcvd_from_st += 1;  
40   digitalWrite(led, !digitalRead(led));  
41   Serial.print("Number of messages received: ");  
42   Serial.print(rcvd_from_st, DEC);  
43   Serial.println();  
44 }
```

Figura 2.9

La funzione `available()` ritorna il numero di caratteri presenti nella coda di input. Riportiamo infine due screenshot del monitor seriale, relativi a delle interazioni utente con il sistema per mezzo dei bottoni.

Monitor seriale in browser con l'editor di Arduino Cloud:

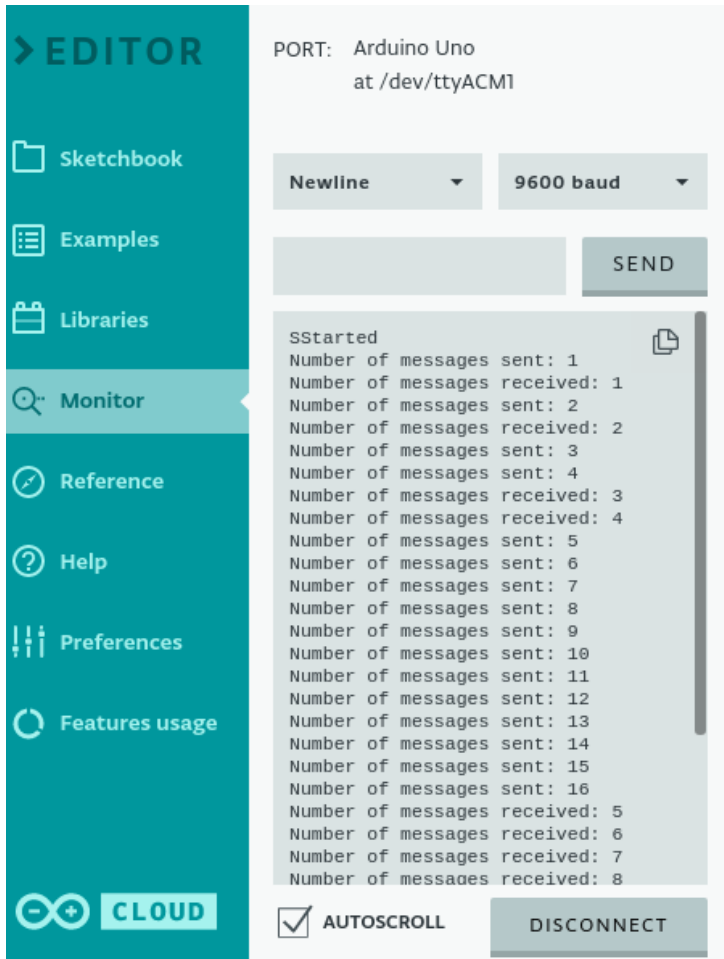
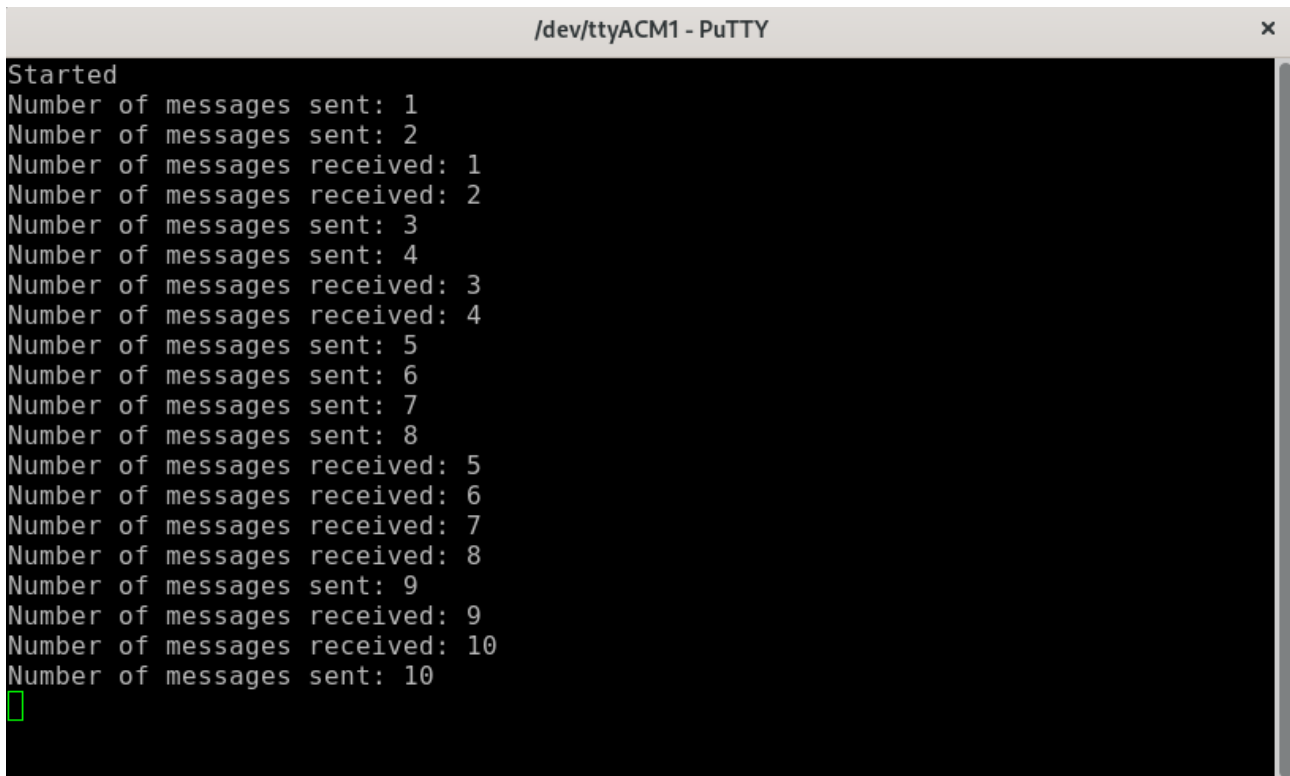


Figura 2.10

Monitor seriale più “vintage” con il software PuTTY:



The image shows a PuTTY terminal window titled "/dev/ttyACM1 - PuTTY". The terminal displays a series of status messages for a serial communication. The messages are as follows:

```
Started
Number of messages sent: 1
Number of messages sent: 2
Number of messages received: 1
Number of messages received: 2
Number of messages sent: 3
Number of messages sent: 4
Number of messages received: 3
Number of messages received: 4
Number of messages sent: 5
Number of messages sent: 6
Number of messages sent: 7
Number of messages sent: 8
Number of messages received: 5
Number of messages received: 6
Number of messages received: 7
Number of messages received: 8
Number of messages sent: 9
Number of messages received: 9
Number of messages received: 10
Number of messages sent: 10
```

A green cursor is visible at the end of the last line.

Figura 2.11