SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione di un microcontrollore per la domotica con comunicazione Bluetooth

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Corso di laurea in Ingegneria Informatica

Arianna Spoletini
Matricola 1755679

Relatore
Prof. Giorgio Grisetti

A.A. 2021-2022

Indice

1. Introduzione

1.1 Microcontrollori

In ogni settore di applicazione dell'elettronica i microcontrollori sono uno dei componenti fondamentali.

Il microcontrollore è utilizzato soprattutto nei sistemi embedded per applicazioni specifiche di controllo digitale. A differenza dei microprocessori progettati per applicazioni generali, il microcontrollore rivolge la massima efficienza verso una particolare applicazione ottimizzando il rapporto costo-prestazione.

I microcontrollori sono circuiti integrati digitali che contengono in un unico chip almeno un microprocessore, una memoria di programma di tipo non volatile, una memoria dati volatile, un'unità di ingresso-uscita (I/O) ed un timer ma in generale il microcontrollore è costituito dai seguenti componenti:

- CPU (Central Processing Unit) a 8, 16 o 32 bit, ovvero l'unità centrale di elaborazione che ha come scopo l'interpretazione e l'esecuzione delle istruzioni del programma.
- Memoria RAM (Random Access Memory), ovvero una memoria nella quale la CPU legge o scrive le variabili utilizzate dal programma e i cui dati vengono persi quando non vi è alimentazione.
- Memoria Flash, è la memoria che contiene il firmware da eseguire, il cui contenuto non viene perso in assenza di alimentazione ma garantisce un numero limitato di scritture.
- Memoria EEPROM (Eletrically Erasable Programmable Read Only Memory), ovvero una memoria nella quale tipicamente vengono salvati i parametri di un dispositivo o i dati che necessitano di essere mantenuti anche dopo l'assenza di alimentazione.

- Porte di I/O per acquisire o inviare dati e informazioni all'esterno.
- Contatori di tempo, o timers.
- Moduli controllori di interrupt.
- Ulteriori componenti aggiuntivi, come per esempio convertitori da Analogico a Digitale, convertitori da Digitale ad Analogico, oppure moduli per la comunicazione seriale.

Affinché il microcontrollore sia utilizzabile deve essere programmato attraverso un insieme di istruzioni che rappresenta il firmware del sistema installato nella memoria del dispositivo.

Le applicazioni dei microcontrollori sono molteplici e trovano grosso impiego nella domotica e nei sistemi di controllo, proprio come mostrato in questo progetto.

1.2 Scopo del progetto

Il progetto consiste nella programmazione di un microcontrollore in grado di eseguire le operazioni basiche utilizzate nella domotica, come la gestione dell'illuminazione o il controllo di sensori ambientali. Per fare ciò, ho implementato le seguenti specifiche:

- controllo di otto interruttori/dimmer su pin digitali dotati di PWM
- lettura di otto ingressi analogici tramite l'utilizzo di un ADC
- lettura di otto ingressi digitali
- memorizzazione della configurazione dei pin sulla EEPROM del microcontrollore

Infine, ho realizzato un'interfaccia con il microcontrollore per inviare comandi e interrogarlo. Questa consiste in un programma per PC in ambiente Linux. I due programmi comunicano tramite Bluetooth.

2. Hardware

L'hardware di base utilizzato per questo progetto è piuttosto semplice e consiste solo in un microcontrollore con microprocessore ATMega2560, un modulo wireless Bluetooth DSD TECH SH-H3, e delle componenti aggiuntive per testare il programma: LED, breadboard, cavi, potenziometro e resistenze.

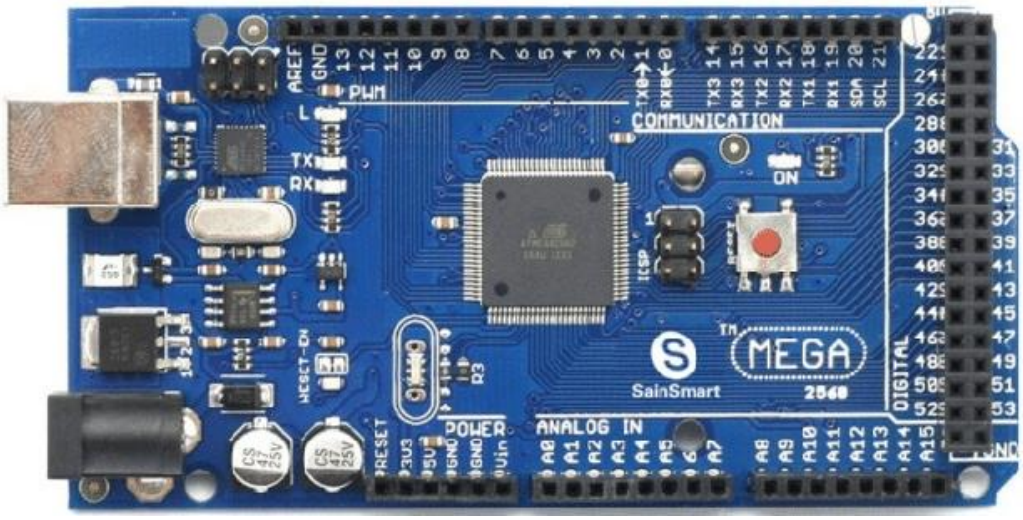


Figura 1 Scheda Mega2560

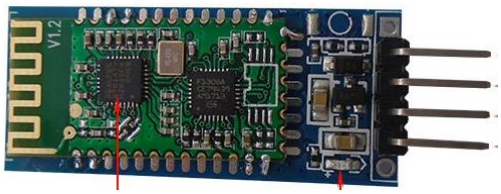


Figura 2 Modulo Bluetooth SH-H3

Figura 3 Componenti aggiuntive

3. Comunicazione Client – Server

3.1 Protocollo di comunicazione

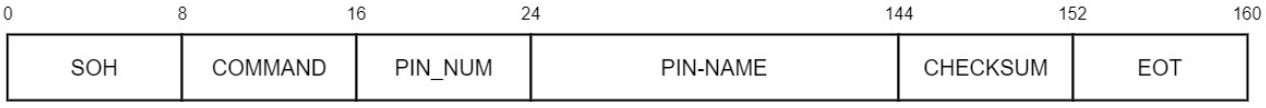
Il protocollo di comunicazione avviene tramite l'invio di pacchetti di dimensione variabile, che sono provvisti, oltre ai dati, di un carattere di inizio (SOH, 0xAA), di un carattere di fine (EOT, 0xBB) e di un carattere di controllo (CHECKSUM). I primi due servono per delimitare il pacchetto, permettendone una corretta lettura. L'ultimo, invece, permette di verificare l'integrità dei dati trasmessi. Viene calcolato utilizzando l'operatore bit a bit OR sul campo DATA dei pacchetti.

I pacchetti sono di tre tipi:

- pacchetto di configurazione → per inviare e ricevere nomi di dispositivi e pin
- pacchetto operativo → per inviare comandi e per ricevere le letture
- pacchetto di controllo → per il controllo della comunicazione e per leggere o ripristinare la configurazione

Inoltre, il protocollo prevede un ACK (per la conferma di ricezione) e un NACK (per chiedere il rinvio del pacchetto).

3.1.1 Pacchetto di configurazione

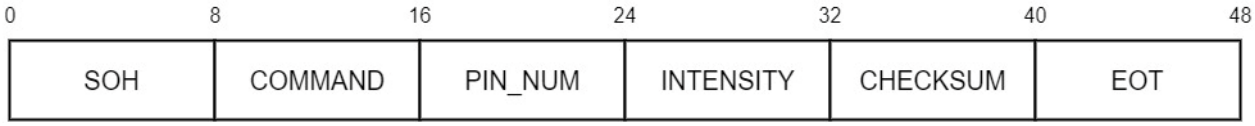


L'utilizzo di questo pacchetto è comune al client e al server. Il client lo utilizza per inviare una nuova configurazione al server che la memorizza sulla memoria del microcontrollore, in modo tale da salvarla anche quando si interrompe la connessione. Il server, invece, lo

utilizza per rispondere ad una richiesta di lettura della configurazione da parte del client. Per semplicità di implementazione il campo PIN_NAME ha una dimensione fissa, pari a 15 byte. Prima di trasmettere il pacchetto, se la dimensione del nome del pin è minore della lunghezza dell'array, la parte restante deve essere inizializzata a zero dal client. Il server invece, all'avvio del client e se non è presente alcuna configurazione precedentemente salvata, inizializza a zero la parte della memoria EEPROM destinata alla configurazione. Quindi il server, prima dell'invio dei pacchetti, non si preoccupa della lunghezza dei nomi salvati.

L'unico comando inserito nel campo COMMAND è **conf** (0x06)

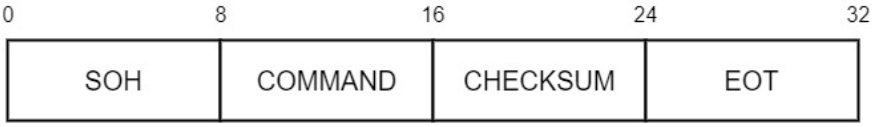
3.1.2 Pacchetto operativo



Questo pacchetto viene utilizzato dal client per richiedere al server le diverse funzionalità. I comandi inseriti nel campo COMMAND sono:

- **ledOn** (0x01) per accendere la luce.
- **ledOff** (0x02) per spegnere la luce.
- **dimmer** (0x03) per regolare l'intensità della luce su una scala da 0 a 100.
- **input** (0x04) per leggere lo stato 0/1 del pin digitale o la tensione del pin analogico da 0 a 5.00 V.
- **status** (0x05) per leggere lo stato acceso/spento del pin digitale e la sua intensità.

3.1.3 Pacchetto di controllo



Questo tipo di pacchetto viene utilizzato per gestire la comunicazione e la configurazione. I comandi inseriti nel campo COMMAND sono:

- **readConfig** (0x07), richiede al server la configurazione salvata sulla EEPROM.
- **stopConfig** (0x08), quando il server riceve questo comando scrive sulla EEPROM che vi è una configurazione salvata, quando il client riceve questo comando significa che ha ricevuto l'intera configurazione.
- **ack** (0x09), serve al client per verificare che il server ha ricevuto correttamente il pacchetto. Nel caso in cui il client si aspetta un pacchetto contenente dei dati, quest'ultimo viene considerato come un ack.
- **nack** (0x0A), serve al client per poter rinviare il pacchetto al server in caso ci sia stato un errore nel primo invio.
- **resetConf** (0x0B), chiede al server di eliminare la configurazione salvata sulla EEPROM.

3.2 Modalità di comunicazione

La comunicazione avviene tramite interfaccia Bluetooth, avendo dotato il microcontrollore del modulo DSD TECH SH-H3 (Bluetooth 2.0).

Il Bluetooth può essere utilizzato per connettere e trasferire dati tra devices in modalità wireless su brevi distanze. È necessario scegliere un device, scegliere un protocollo di comunicazione, stabilire una connessione, per poi poter inviare e ricevere dati. Ogni chip Bluetooth è dotato dal produttore di un indirizzo univoco a 48 bit analogo all'indirizzo

MAC per Ethernet e sono entrambi gestiti dalla stessa organizzazione, la IEEE Registration Authority. Questo indirizzo è utilizzato, a differenza del MAC, in tutti i layers della comunicazione Bluetooth. Per facilitare l'individuazione dei devices può essere assegnato un nome dall'utilizzatore; tuttavia, questo non sarà utilizzato nei protocolli di comunicazione. Il protocollo scelto per comunicare è l'RFCOMM, un protocollo simile al TCP anche se nato per emulare la porta seriale RS-232.

Il numero della porta di comunicazione è assegnato in modo statico.

Per inizializzare il modulo Bluetooth SH-H3 è necessario entrare in AT mode. Ho quindi collegato il modulo ad una porta USB del PC utilizzando il convertitore USB-Seriale "DSD TECH USB to TLL Serial".

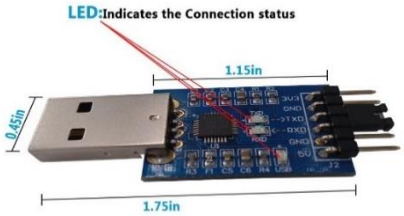


Figura 4 DSD TECH USB to TTL Serial

Ho poi utilizzato un terminale seriale grafico, CuteCom, per inviare i comandi AT necessari ad impostare il nome del device a "Smarthouse" e il Baud rate a 9600.

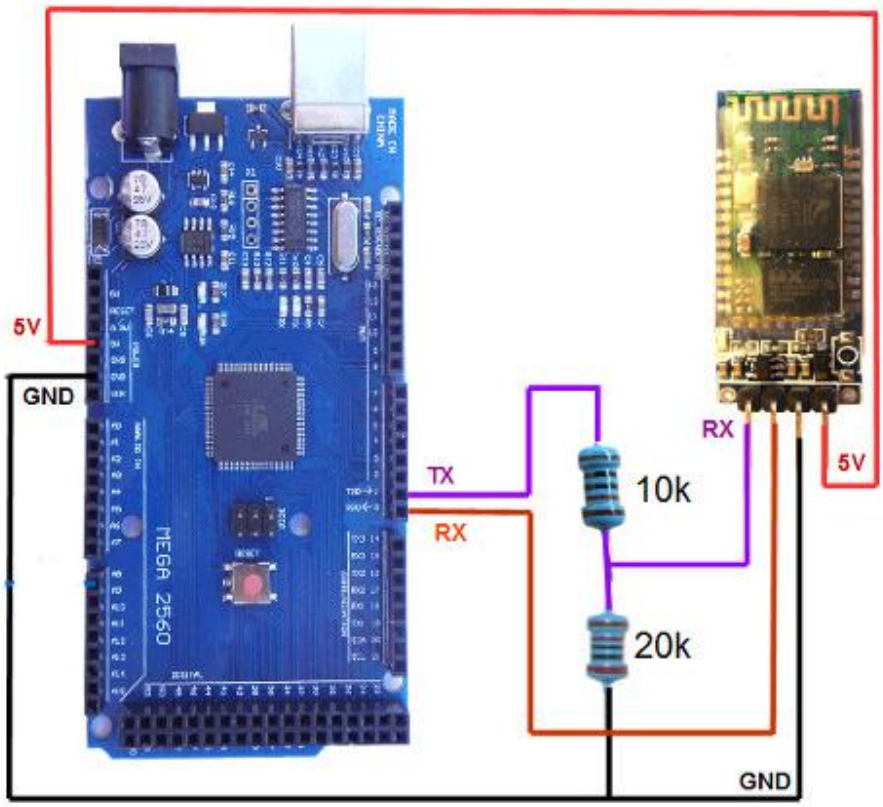


Figura 5 Connessione Bluetooth a scheda Mega

Il modulo Bluetooth ha quattro connessioni, GND, 5V, RX and TX. Il pin TX sul modulo Bluetooth si collega al pin RX sulla scheda Mega e viceversa. Per la comunicazione seriale, una connessione di trasmissione (TX) deve essere ricevuta da una connessione (RX) (tipo di collegamento che è stato necessario anche per il convertitore USB-Seriale). Il modulo Bluetooth funziona a 3,3V. Fornire 5 V al pin RX Bluetooth potrebbe danneggiarlo, per cui è necessario utilizzare un partitore di tensione per fornire un segnale a 3,3 V al pin RX. Ciò si può ottenere con un resistore da 20k e 1k secondo lo schema nella figura sottostante. Il pin TX del modulo Bluetooth non necessita di modifiche e può connettersi direttamente al pin RX della scheda Mega. Questo perché la logica HIGH su 3.3V sarà comunque riconosciuta come HIGH sul circuito logico 5V della scheda Mega.

Il modulo Bluetooth, oltre all'interfaccia UART di connessione, ha incorporato il Bluetooth stack profiles SPP (Serial Port Profile) e rimane in slave mode in attesa di connessione.

4. Server

4.1 Panoramica generale

Il programma si compone di due parti. L'inizializzazione e l'elaborazione. L'inizializzazione è una parte di codice eseguita solo una volta, quando il microcontrollore viene alimentato e si avvia o quando viene resettato. Questa serve per definire variabili e costanti, per configurare opportunamente le porte richieste e per gestire la cattura degli eventi a fronte dei quali vengono generati segnali appositi, denominati interrupt. La parte di elaborazione consiste invece in un ciclo ripetuto all'infinito in cui vengono eseguite le istruzioni di ricezione di pacchetti, esecuzione del comando e invio di pacchetti. I comandi che può eseguire il microcontrollore sono stati già elencati nella descrizione dei pacchetti.

Il microcontrollore utilizzato appartiene all'AVR, una famiglia di microcontrollori RISC che utilizzano una memoria flash interna per memorizzare il contenuto del programma. Il programma per il microcontrollore è scritto in linguaggio C, senza l'utilizzo dell'ambiente di sviluppo Arduino IDE. È stato compilato nel linguaggio macchina AVR utilizzando AVR-GCC, un cross-compilatore della GNU Compiler Collection, specificando il tipo di processore montato sulla scheda (-mmcu=atmega2560) e la frequenza di clock del processore (-DF_CPU=16000000UL). Per caricare il programma sul microcontrollore, alcune schede sono caricate con uno speciale programma, chiamato bootloader, che si trova su una parte della memoria flash dedicata. Quest'ultimo rimane in ascolto sulla seriale – USB in attesa di una particolare sequenza di attivazione che fa un salto ad una routine che dialogherà con il computer attraverso la seriale, offrendo un protocollo per scrivere sulla memoria flash. La scheda Mega è dotata di un ulteriore microcontrollore, l'ATmega16U2, che si occupa di gestire la conversione USB – seriale e di fungere da programmatore, implementando un'interfaccia bootloader per l'ATmega 2560. Infine, si usa l'Avrdude, una utility sul PC, che è un programma in grado di controllare/dialogare con il bootloader.

4.2 Funzionalità

Sulla scheda Mega sono disponibili 54 pin digitali che vanno da D0 a D53. Tutti questi pin, che possono essere usati sia come input che come output, sono in grado di generare o assorbire 20 mA di corrente, con un massimo di 40mA oltre il quale si rischiano danni permanenti al microcontrollore. Una caratteristica aggiuntiva di questi pin è quella di avere a disposizione un resistore di pull-up interno, di valore compreso tra 20KΩ e 50KΩ.

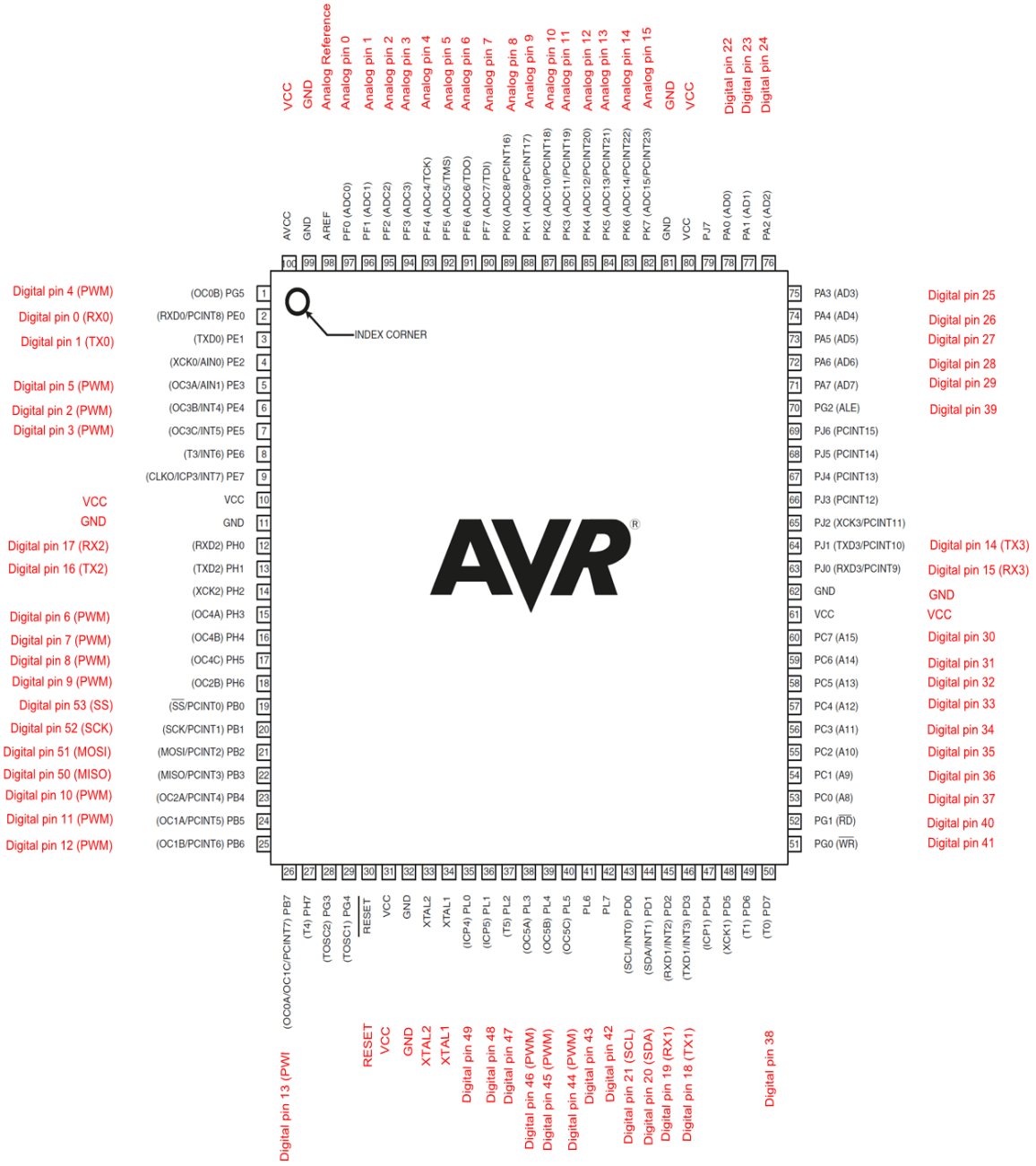


Figura 6 ATmega2560 Pin Mapping

Di questi, i pin 2 – 13 e 44 – 46 sono in grado di produrre segnali PWM. Mentre i pin 0 – 1 e 14 – 19 sono usati per la trasmissione/ricezione seriale. In particolare, i pin 0 e 1 sono collegati ai rispettivi pin del microcontrollore Atmega16U2, responsabile della conversione USB – TTL.

Ci sono anche 16 pin di ingresso analogico, che vanno da A0 ad A15. Tutti i pin di ingresso analogico forniscono una funzione ADC con risoluzione a 10 bit, in grado di misurare da 0 a 5.00V, o ad un valore legato al pin AREF.

4.2.1 Gestione dei LED (PWM)

Il Pulse Width Modulation, PWM, è utilizzato ad esempio per regolare la velocità di un motore elettrico in corrente continua o per controllare la luminosità dei LED.
Il PWM è una tecnica utilizzata per ottenere risultati analogici con mezzi digitali. Il controllo digitale viene utilizzato per creare un'onda rettangolare, un segnale commutato tra acceso e spento. Questo modello on-off può simulare tensioni comprese tra il valore VCC della scheda e il GND, modificando l'intervallo di tempo trascorso su on rispetto a quello trascorso su off. La durata di "on time" è chiamata ampiezza dell'impulso. Il rapporto tra la durata dell'impulso positivo e dell'intero periodo è chiamata duty cycle e spesso si esprime in percentuale. Abbiamo quindi la possibilità di variare la tensione e quindi la potenza fornita, da zero, per un duty cycle nullo, alla potenza massima disponibile che avremmo in assenza del circuito di modulazione, per un duty cycle pari al 100%.

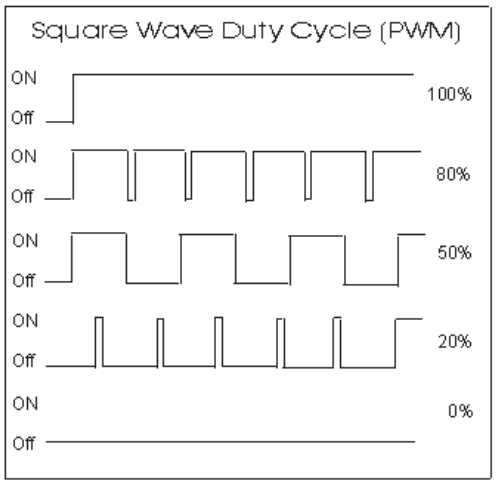


Figura 7 Duty Cycle

Il ciclo viene realizzato utilizzando un Timer/Contatore, il suo incremento può essere regolato o dal clock interno o da uno esterno, quando il suo valore uguaglia il valore espresso dal duty cycle, scritto in un registro, il valore della tensione commuta e quando raggiunge il suo valore massimo, chiamato TOP value, la tensione commuta nuovamente e il contatore viene azzerato per ricominciare il ciclo.

I microcontrollori ATmega2560 hanno 5 modalità di PWM, ognuno con funzionalità diverse. Delle 5 ho utilizzato il fast PWM per controllare la luminosità dei LED, che permette una frequenza doppia rispetto alle altre modalità ed è quello che impegna di meno il tempo della CPU.

La frequenza di output si può ottenere dividendo la frequenza di clock per il prescaler, ossia un divisore di frequenza, e per il range di valori del contatore. Nel nostro caso, avendo scelto di utilizzare il clock interno, abbiamo una frequenza di clock pari a 16 MHz. Ho scelto un valore di prescaler uguale a 1 e un fast PWM a 8-bit che ha un TOP value di 255 e quindi un range di 256. La frequenza di output sarà quindi uguale a 62.500 Hz, un valore che permette di non avere fenomeni di flickering o altri fenomeni di affaticamento della vista. La scelta del fast PWM a 8-bit è dettata dalla necessità di avere una risoluzione di almeno 100 livelli di luminosità.

Ho scelto i pin 2 – 3 – 5, che sono collegati al timer 3, i pin 6 – 7 – 8, collegati al timer 4, e i pin 11 – 12, collegati al timer 1.

Per impostare il funzionamento del PWM si devono settare diversi registri.

Innanzitutto, gli 8 pin scelti si configurano come output, modificando il Data Direction Register (DDRx), ovvero mettendo ad 1 il bit corrispondente al pin che sta sulla porta (x).

DDRx - Port x Data Direction Register

Bit	7	6	5	4	3	2	1	0
	DDRx7	DDRx6	DDRx5	DDRx4	DDRx3	DDRx2	DDRx1	DDRx0

Ad esempio, il pin 2 corrisponde al bit 4 della porta E, come si può vedere nel Pin Mapping di figura 5, quindi sul registro DDRE va posto uguale a 1 il bit DDRE4.

La scelta della modalità fast PWM a 8 bit si effettua impostando i registri (n):

TCCRnA - Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0
	COMnA1	COMnA0	COMnB1	COMnB0	COMnC1	COMnC0	WGMn1	WGMn0
	0	0	0	0	1	0	0	1

TCCRnB - Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0
	FOCnA	FOCnB	-	-	WGMn2	CSn[2:0]		
	1	1	1	1	1	1	0	1

I bit COMnz1:0 del registro TCCRnA controllano il comportamento dei rispettivi pin di confronto (OCnz). Infatti, se uno o entrambi i bit COMnA:B:C vengono scritti su 1, l'uscita OCnA:B:C sovrascrive la normale funzionalità della porta del pin I/O a cui è collegata.
Impostando ad 1 entrambi i bit si è scelta la modalità invertita, ovvero il segnale in uscita è alto quando si ha il timer a 0 e basso quando il timer counter raggiunge il valore memorizzato nel registro OCRnz.

I tre bit CSn2:0 del registro TCCRnB servono a selezionare il clock di funzionamento del timer, ovvero il prescaler e CSn0 impostato ad 1 indica la preferenza di non utilizzare nessun fattore di prescaler ma di considerare come sorgente di clock per il timer i 16MHz di sistema.

Il valore espresso del duty cycle, ossia della luminosità richiesta, viene memorizzato nell'Output Compare Register (OCRnz) e il segnale cambia stato quando il valore del contatore (TCNTn) eguaglia quello del registro. Ad esempio, per il pin 2 il corrispondente registro è lo OCR3B deducibile sempre dal pin mapping di fig. 5. Si inizializzano a zero i bit alti dell'output compare register OCRnz dei tre timer in quanto vengono usati solo i primi 8 bit.

led_DIMMER

Si abilita l'OCRn inizializzando la maschera del registro TTCRnA con i bit COMn1:0 ad uno e si inserisce nell'OCRnL l'intensità desiderata.

PORTx - Port x Data Register

Bit	7	6	5	4	3	2	1	0
	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0

led_ON

Si disabilita l'OCRn inizializzando la maschera del registro TTCRnA con i bit COMn1:0 a zero. Si modifica il registro Pin Output Register, PORTx, mettendo ad 1 il bit sulla porta corrispondente al pin, che porterà a 5V tale pin.

led_OFF

Si disabilita l'OCRn inizializzando la maschera del registro TTCRnA con i bit COMn1:0 a zero. Si modifica il registro Pin Output Register, PORTx, mettendo a 0 il bit sulla porta corrispondente al pin, che porterà a 0V tale pin.

4.2.2 Ingresso Digitale

Se si configurano dei pin digitali come input c'è il rischio del verificarsi di un fenomeno chiamato floating. Ovvero, quando non c'è niente collegato al pin e il programma legge lo stato del pin questo non sarà in uno stato alto (VCC) o basso (0) ma potrà presentare uno stato intermedio casuale. Per prevenire questo stato imprevedibile, si utilizza un resistore di pull-up o di pull-down che assicura che il pin sia in uno stato alto o basso, utilizzando anche una bassa quantità di corrente.

Con un resistore di pull-up, il pin di input leggerà uno stato alto quando non c'è niente a lui collegato. Infatti, il pin di input essendo collegato, tramite il resistore, solamente al pin VCC e non alla terra, sarà anche lui a VCC. Quando lo switch si chiude, collega il pin di

input direttamente a terra e il suo valore di tensione andrà a zero. La corrente che scorrerà verso terra sarà comunque molto bassa perché si usano resistori ad alta impedenza.

Nel microcontrollore vi sono delle resistenze di pull-up integrate. Se in PORTxn è scritto un uno logico e il pin è configurato come pin di input, viene attivata la resistenza di pull-up.

DDRx - Port x Data Direction Register

Bit	7	6	5	4	3	2	1	0
	DDRx7	DDRx6	DDRx5	DDRx4	DDRx3	DDRx2	DDRx1	DDRx0

Si configurano gli 8 pin come input modificando il DDRx, ovvero mettendo ad 0 il bit corrispondente al pin.

PINx - Port x Input Pins Address

Bit	7	6	5	4	3	2	1	0
	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0

Il valore di un pin è il valore opposto che si legge sul bit PINxn, del registro Port x Input Pins Address, per via delle resistenze di pull-up.

4.2.3 Ingresso Analogico (ADC)

La lettura di un canale analogico, come per esempio un sensore di temperatura, varia in modo continuo e per essere trattata, elaborata, da un dispositivo, come un microcontrollore, è necessario che essa sia convertita in forma digitale. Il processo di conversione di un segnale analogico in forma digitale richiede l'utilizzo di un dispositivo ADC (Analog to Digital Converter).

L'AVR Atmega2560 contiene un singolo convertitore da analogico a digitale a 10 bit con una frequenza di campionamento massima di 15kS/s a piena risoluzione. Utilizza un tipo di ADC ad approssimazioni successive composto da un circuito d'ingresso, chiamato sample and hold, che mantiene costante il valore di tensione durante la conversione, un comparatore e un convertitore digitale-analogico. Ad ogni passaggio l'ADC prova a

impostare un bit, partendo dal bit più significativo, e usando il DAC confronta una frazione della tensione di riferimento con la tensione di ingresso in feedback. Questo convertitore individua un bit ad ogni iterazione e la risoluzione è limitata solo dalla frequenza di campionamento e dal rumore in ingresso.

Questa tipologia di convertitore A/D è la più usata in assoluto perché ha un basso costo, permette risoluzioni elevate ed è abbastanza veloce.

L'AVR Atmega 2560 fornisce 16 pin di ingresso analogico, posti sulla porta F e sulla porta K, che vengono selezionati utilizzando un multiplexer MUX che si trova prima dell'ADC. In questo progetto sono stati utilizzati solo i primi 8 pin legati alla porta F che sono poi stati testati leggendo i valori di un potenziometro.

Prima di utilizzare l'ADC è necessario selezionare la sorgente della tensione di riferimento e la frequenza di clock dell'ADC.

Tensione di Riferimento

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

L'ADC ha bisogno di una tensione di riferimento su cui lavorare che determina il massimale di ciò che l'ADC può convertire. Ci sono diverse scelte per questa tensione: fornire una tensione di riferimento attraverso il pin AREF, scegliere la tensione interna a 1.1V o a 2.56V, oppure la tensione AVCC, ossia la tensione di alimentazione dell'ADC a 5V. Per le ultime due opzioni si può collegare un condensatore attraverso il pin AREF e metterlo a terra per evitare disturbi, oppure si può scegliere di lasciarlo scollegato. In questo caso ho scelto AVCC, impostando a 0 e 1 i bit REFS1 e REFS2 del registro ADMUX.

Frequenza di Clock

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

I bit ADPS2:0 del registro ADCSRA servono per ridurre la velocità di clock dell'ADC di un fattore specificato. Impostando tutti e tre i bit ad 1 viene fornito un fattore di prescaler di 128. In questo caso, lavorando il microprocessore a 16 MHz e con un fattore di prescaler di 128, il clock dell'ADC è impostato su 12 kHz.

Abilitazione e scelta del Pin

Si deve poi abilitare l'ADC mettendo a 1 il bit ADEN del registro ADCSRA, mentre mettendolo a zero, l'ADC si spegne.

Si deve quindi scegliere quale ingresso analogico collegare all'ADC, scrivendo il numero del pin ADC0:7 da collegare nei bit MUX4:0 del registro ADMUX.

Conversione

La conversione ha inizio quando viene messo ad 1 il bit ADSC (ADC Start Conversion bit) del registro ADCSRA. Una volta terminata la conversione il bit torna a zero. Normalmente sono necessari 13 cicli di clock dell'ADC per la conversione, ma quando avviene per la prima volta ne impiega 25 poiché esegue anche l'inizializzazione del circuito analogico di ingresso.

ADCL and ADCH – The ADC Data Register

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL

ADLAR = 1

Il risultato della conversione ADC viene memorizzato nel registro ADC. Poiché ha una risoluzione di 10 bit, sono necessari 10 bit per memorizzare il risultato. Quindi un unico

registro a 8 bit non è sufficiente. Abbiamo bisogno di due registri: ADCL e ADCH (ADC Low byte e ADC High byte). I due possono essere chiamati insieme come ADC. Nel mio caso, avendo scelto una risoluzione di 8 bit, ho impostato il bit ADLAR del registro ADMUX a 1 per avere un aggiustamento sinistro del risultato, ovvero il risultato della conversione si trova sul registro ADCH.

4.2.4 Memorizzazione della Configurazione (EEPROM)

Per la memorizzazione della configurazione del sistema SmartHouse ho utilizzato la memoria EEPROM**,** acronimo di Electrically Erasable Programmable Read-Only Memory. Le EEPROM sono dei particolari microchip utilizzati per poter immagazzinare dei dati anche dopo che il dispositivo su cui sono montati sia stato spento. Alla successiva accensione, il dispositivo avrà quindi a disposizione dei dati da poter caricare.

L'ATmega2560 è dotata di questo genere di memoria, la cui dimensione è di 4096 bytes e con una durata limitata. In particolare, è garantita per gestire 100.000 cicli di scrittura/cancellazione per ogni cella. Le letture, invece, sono illimitate. Quando la EEPROM viene letta, la CPU viene arrestata per quattro cicli di clock prima che venga eseguita l'istruzione successiva. Quando la EEPROM viene scritta, la CPU viene fermata per due cicli di clock prima che venga eseguita l'istruzione successiva.

Prima di iniziare una lettura o una scrittura sulla EEPROM è necessario specificare l'indirizzo di memoria su cui si andrà a leggere/scrivere. Nel caso di scrittura sarà inoltre necessario inserire i dati da scrivere in un apposito registro. Vanno poi disabilitati tutti gli interrupt durante i cicli di scrittura per poi essere riabilitati.

EEARH and EEARL – The EEPROM Address Register

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	EEAR11	EEAR10	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL

I registri degli indirizzi EEPROM: EEARH e EEARL specificano l'indirizzo nella EEPROM da 4 Kbyte di spazio. I byte di dati EEPROM vengono indirizzati linearmente tra 0 e 4096.

EEDR – The EEPROM Data Register

Bit	7	6	5	4	3	2	1	0
	MSB							LSB

Per l'operazione di scrittura EEPROM, il Registro EEDR contiene i dati da scrivere nella EEPROM all'indirizzo fornito dal registro EEAR.

Per l'operazione di lettura EEPROM, l'EEDR contiene i dati letti dalla EEPROM all'indirizzo fornito da EEAR.

EECR – The EEPROM Control Register

Bit	7	6	5	4	3	2	1	0
	-	-	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE

Il registro EECR è quello che effettivamente controlla la lettura/scrittura sulla EEPROM. Prima di iniziare una qualunque delle due operazioni bisogna attendere che il bit EEPE sia tornato a zero, poiché quando la memoria è impegnata in una operazione di scrittura non si può né leggere né scrivere.

Scrittura

Per poter scrivere sulla EEPROM è quindi necessario:

1. Attendere che il bit EEPE sia tornato a zero.
2. Disabilitare gli interrupt.
3. Scrivere il nuovo indirizzo in EEAR.
4. Scrivere i dati su EEDR.
5. Impostare il bit EEMPE ad uno, dopo 4 cicli di clock l'hardware lo azzera.
6. Entro quattro cicli di clock impostare ad uno il bit EEPE che determina la scrittura della EEPROM. Trascorso il tempo di accesso in scrittura, in cui la CPU si ferma per due cicli di clock, il bit EEPE viene cancellato dall'hardware.
7. Riabilitare gli interrupt.

Se si superano 4 cicli di clock per impostare a uno il bit EEPE non verrà scritto niente sulla EEPROM. Per essere sicura di rispettare il limite ho definito due semplici macro, scritte in linguaggio Assembly, che permettono la modifica dei bit EEMPE ed EEPE in un tempo ridotto.

Lettura

Il segnale di abilitazione lettura avviene tramite il bit EERE. Quando l'indirizzo corretto è impostato nel registro EEAR, il bit EERE deve essere scritto in uno logico per attivare la lettura della EEPROM. La lettura della EEPROM richiede una sola istruzione e i dati richiesti sono immediatamente disponibili. Trascorso il tempo di accesso in lettura, in cui la CPU si ferma per quattro cicli di clock, il bit EERE viene cancellato dall'hardware. Anche in questo caso devono essere disabilitati gli interrupt prima della lettura e poi riabilitati.

4.2.5 Comunicazione Seriale (UART)

La comunicazione tra il modulo Bluetooth e il microcontrollore avviene tramite UART, Universal Asynchronous Receiver and Transmitter. L'UART è un semplice protocollo di comunicazione seriale che consente all'host di comunicare con i dispositivi seriali, trasmettendo un byte, scritto in un registro, un bit alla volta, o ricevendo un byte, scrivendolo in un registro, un bit alla volta. La comunicazione UART è un mezzo di comunicazione seriale molto semplice ed economico nei progetti embedded. UART supporta la trasmissione di dati bidirezionale, asincrona e seriale.

Utilizza due linee dati per comunicare tra loro che sono:
TX – Usato per trasmettere
RX – Usato per la ricezione

L'ATmega2560 ha quattro USART (Universal Synchronous and Asynchronous Receiver and Transmitter), USART0, USART1, USART2 e USART3, che verranno utilizzate in modalità asincrona. Inoltre, ha un convertitore da USB a seriale integrato, collegato ai pin

0 e 1, ovvero alla USART0. Per questo ho scelto la USART1 per collegare il modulo Bluetooth alla scheda.

Le tre parti principali dell'USART sono: Clock Generator, Transmitter e Receiver.
L'USART supporta quattro modalità di funzionamento del clock: asincrono normale, asincrono a doppia velocità, sincrono master e sincrono slave. In questo caso ho usato la modalità asincrona normale dove i dati vengono trasmessi/ricevuti in modo asincrono, cioè non abbiamo bisogno degli impulsi di clock e i dati vengono trasferiti a una velocità impostata in un registro.

Il frame di trasmissione ha un formato composto da un bit di inizio seguito da 5,6,7,8 o 9 bit di dati, iniziando dal bit meno significativo, se abilitato, viene inserito il bit di parità, per finire con un bit di stop.

Inizializzazione

Prima di poter iniziare una trasmissione seriale si devono quindi inizializzare i registri che definiscono la modalità e velocità di trasmissione, il formato del frame e l'abilitazione del Transmitter e/o del Receiver.

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n

UCSRnC – USART Control and Status Register n C

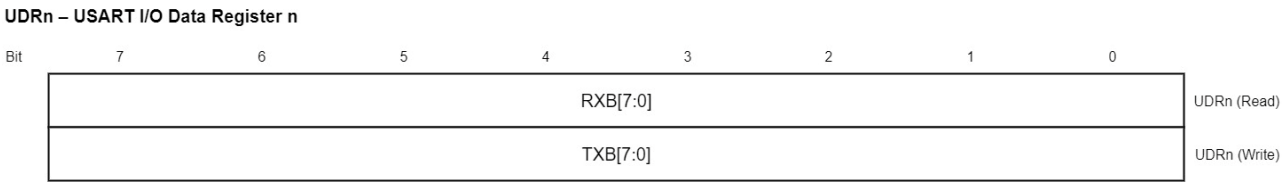
Bit	7	6	5	4	3	2	1	0
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn

- La modalità di trasmissione asincrona normale si ottiene impostando a 0 i primi 2 bit del registro UCSRnC (Control and Status Register).
- La Velocità di trasmissione, BAUD, nel mio caso pari a 9600 bps, si ottiene impostando il registro UBRRn (Baud Rate Register) a $\frac{F_{CPU}}{16 \cdot BAUD} - 1$.

- Il formato del frame, si ottiene impostando a 0 il bit UCSZn2 del registro UCSRnB ad 1 i bit UCSZn1:0 del registro UCSRnC, avendo scelto 8 bit per il campo dati, e a 0 il bit USBSn del registro UCSRnC, avendo scelto un solo bit di stop. Trasmettitore e ricevitore utilizzano la stessa impostazione. Quindi una modifica dell'impostazione di uno qualsiasi di questi bit farà corrompere tutte le comunicazioni in corso sia per il ricevitore che per il trasmettitore.

- Per abilitare il Transmitter e il Receiver si devono impostare a 1 i bit TXENn e RXENn del registro UCSRnB. Quando il bit RXENn è ad 1 il ricevitore sostituirà il normale funzionamento della porta per il pin RXDn. Quando il bit TXENn è ad 1 il trasmettitore annullerà il normale funzionamento della porta per il pin TXDn.

Comunicazione



I dati che devono essere trasmessi o che sono ricevuti sono generalmente contenuti, nei microcontrollori, in un solo buffer hardware, il registro UDRn. Composto da due byte: il Transmit Data Buffer, TXB, e il Receive Data Buffer, RXB, che condividono quindi lo stesso indirizzo. Quando si scrive sul registro UDRn il TXB sarà il destinatario e quando si legge dal registro UDRn si riceverà il contenuto del RXB.

Per gestire i dati scambiati si creano generalmente dei buffer software serviti da interrupt attivati alla modifica dei dati che si trovano nel buffer hardware. Per gestire questi interrupt bisogna scrivere delle Interrupt Service Routine (ISR). Ho definito quindi due routine in corrispondenza di due posizioni dell'Interrupt Vector dell'ambiente AVR-GCC, USART1_RX_vect per la ricezione, USART1_UDRE_vect per la trasmissione.
Gli interrupt possono essere abilitati e disabilitati.

Trasmissione

UCSRnA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn

Per poter trasmettere i dati contenuti nel buffer software dobbiamo abilitare l'interrupt sul bit UDREn del registro UCSRnA, settando a 1 il bit UDRIEn del registro UCSRnB. L'hardware ogni volta che il registro UDR si svuota mette a 1 il bit UDREn generando, se l'interrupt è abilitato, l'interrupt Data Register Empty corrispondente a USART1_UDRE_vect. Viene eseguita l'interrupt service routine che preleva un byte dal buffer software per scriverlo sul registro UDR nel byte TXB. Il trasmettitore a questo punto, se abilitato, caricherà i dati nel Transmit Shift Register e l'hardware riporta a 0 il bit UDREn. Quindi i dati verranno trasmessi in serie sul pin TXDn. L'interrupt service routine quando il buffer software è vuoto disabilita l'interrupt mettendo a 0 il bit UDRIEn del registro UCSRnB.

I dati scritti in UDRn quando il flag UDREn non è impostato, verranno ignorato dal trasmettitore USART.

Ricezione

Per potere caricare i dati nel buffer software dobbiamo abilitare l'interrupt sul bit RXCn del registro UCSRnA, settando a 1 il bit RXCIEn del registro UCSRnB. L'interrupt rimarrà sempre abilitato.

L'hardware ogni volta che completa la ricezione di un byte mette a 1 il bit RXCn generando, se l'interrupt è abilitato, l'interrupt Receive Complete corrispondente a USART1_RX_vect e viene eseguita l'interrupt service routine che preleva il byte RXB del registro UDR per scriverlo nel buffer software. L'hardware, quindi, mette a 0 il bit RXCn.

5. Client

5.1 Panoramica generale

Il client è un programma scritto in linguaggio C, in ambiente Linux, che permette di comunicare con il microcontrollore da PC tramite un'interfaccia implementata come una shell a linea di comando. All'utente viene fornita una lista di comandi che sono in grado di gestire il microcontrollore AVR. Oltre a quelli già elencati nella descrizione dei pacchetti, vi è un ulteriore comando "help", che stampa la lista dei comandi con una breve descrizione.

5.2 Funzionamento

All'avvio il client come prima cosa deve stabilire una connessione con il server che avviene tramite Bluetooth.

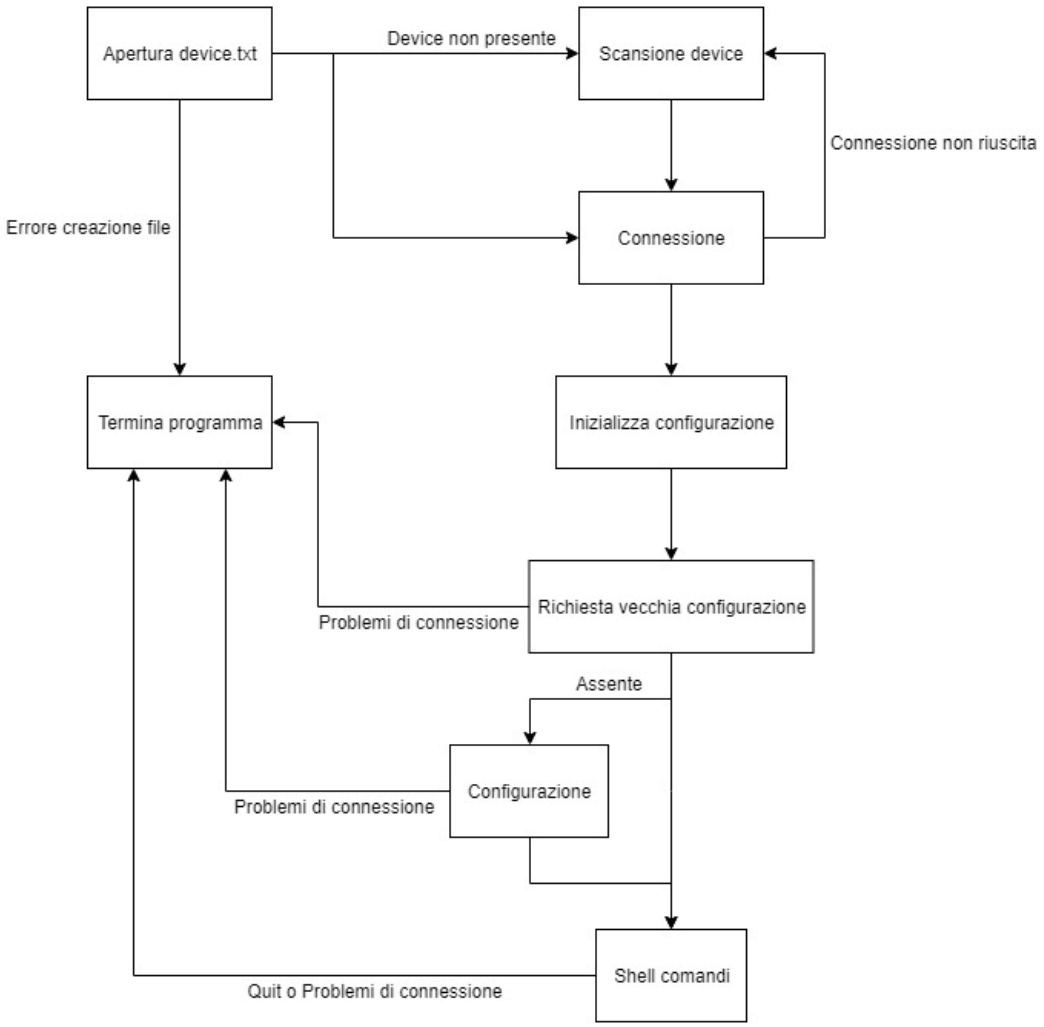


Figura 8 Esecuzione Client

Per la comunicazione con l'AVR, data la similitudine tra la programmazione Bluetooth e la programmazione del networking ho scelto di utilizzare il socket che è una struttura informatica semplice e collaudata.
Programmando in ambiente Linux ho fatto ricorso alla libreria BlueZ.
Con la funzione hci_get_route() acquisisce il resource number del Bluetooth Adapter locale, per poi aprire un socket verso di lui con la funzione hci_open_dev(), effettua, quindi, la scansione dei device vicini con la funzione hci_inquiry(), che ritorna un elenco dei device trovati e pone le informazioni nella struttura inquiry_info. Successivamente con la funzione hci_read_remote_name() legge il nome dei device e crea un elenco con il loro indirizzo e nome per permetterne la scelta. Dopo aver allocato un socket con la funzione socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM), dove il primo parametro specifica che è un socket Bluetooth, il secondo che è un flusso di dati bidirezionali implementato sul protocollo TCP/IP e il terzo che si utilizza il protocollo RFCOMM, tenta la connessione con l'indirizzo del device scelto e sulla porta 1. Se la connessione va a buon fine l'indirizzo viene salvato su file per essere riutilizzato al prossimo avvio del client.
Stabilita la connessione per l'invio e la ricezione dei dati si utilizzano le funzioni send() e recv().

Dopo ogni trasmissione/ricezione di un pacchetto viene effettuato un controllo sul successo della relativa operazione e in caso non sia andata a buon fine il programma termina stampando il seguente messaggio "Bluetooth non funzionante o connessione caduta". In questo caso è necessario riavviare il programma e tentare una nuova connessione.

Successivamente, secondo lo schema in figura 8, si procede con l'interrogazione dell'AVR sulla possibile configurazione salvata sulla EEPROM. In caso non ve ne sia una, si passa ad una nuova configurazione. Durante la fase guidata di configurazione sono effettuati

controlli sulla lunghezza dei nomi, sia del device che dei pin, sulla univocità dei nomi utilizzati e sulla correttezza dei numeri dei pin scelti.

Acquisita la configurazione si avvia la shell comandi.

Dalla shell è possibile inviare i comandi all'AVR. Durante l'inserimento sono effettuati tutti i possibili controlli sulla correttezza dei comandi.

Dopo ogni invio di un pacchetto con il comando inserito dall'utente si attende la ricezione di un pacchetto dall'AVR indicante la corretta ricezione del comando. Se il comando richiesto non necessita di una risposta, il programma attende la ricezione di un ACK, altrimenti il pacchetto contenente i dati richiesti viene interpretato come una corretta ricezione del precedente pacchetto. In caso di NACK si tenta di rinviare il pacchetto, a meno che non ci sia un problema di connessione, per il quale si termina il programma.

6 Caso d'uso

6.1 Compilazione

Il progetto risiede su una repository di github. Per testarlo è necessario aprire il terminale ed eseguire il comando *git clone https://github.com/arispole/SmartHouse.* Spostandosi nella nuova directory dal terminale ci sono tre directory, due delle quali contengono i codici per poter compilare il programma per il server e quello per il client.

6.1.2 Server

Nella directory "arduino", il programma può essere compilato e caricato sul microprocessore, collegato alla porta USB del PC, utilizzando l'utility GNU Make. Prima di procedere con la compilazione è necessario installare il toolchain avr-gcc, la libreria avr-libc e l'utility avrdude. Quindi si apre il terminale nella directory "arduino", si esegue il comando *make clean* e successivamente *make smarthouse.hex*.

6.1.3 Client

Nella directory "client", il programma può essere compilato utilizzando l'utility GNU Make. Per fare ciò è necessario installare la libreria bluez, in particolare il pacchetto libbluetooth-dev. Quindi si apre il terminale nella directory "client" e si esegue il comando *make*.

6.2 Esecuzione

Per eseguire la shell del client, si apre il terminale nella directory "client" e si esegue il comando *./smarthouse_client*. L'utilizzo di quest'ultimo è guidato e assistito fin dall'inizio della connessione. Un esempio di una possibile esecuzione del programma client è illustrato di seguito e prevede l'utilizzo di tre LED, connessi ai pin PWM 11 – 7 – 3, di un potenziometro, connesso al pin analogico A0, e di due cavi, connessi ai pin digitali 46 – 51. Nella figura 12 si mostra lo schema di collegamento dei componenti e lo stato dei LED in seguito ai comandi dati dalla shell del client.

```
scansione device in esecuzione

  0) DC:0D:30:9F:19:47 Smarthouse
  1) 8C:7A:3D:D1:2A:45 Xiaomi 11T Pro

scegliere device [n] ([-1] per uscire): 0

tentativo di connessione in corso



INIZIO CONFIGURAZIONE

Inserisci il nome del device:
SmartHouse

-----------------------------------------------------------------

CONFIGURAZIONE PIN

 Pin disponibili:   PIN ANALOGICI
                    0 - 1 - 2 - 3 - 4 - 5 - 6 - 7

Inserisci il numero del pin e il nome, [quit] per terminare
0 potenziometro
Inserisci il numero del pin e il nome, [quit] per terminare
quit

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

 Pin disponibili:   PIN DIGITALI/SWITCH/DIMMER
                    2 - 3 - 5 - 6 - 7 - 8 - 11 - 12

Inserisci il numero del pin e il nome, [quit] per terminare
11 led1
Inserisci il numero del pin e il nome, [quit] per terminare
7 led2
Inserisci il numero del pin e il nome, [quit] per terminare
3 led3
Inserisci il numero del pin e il nome, [quit] per terminare
quit

°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°°

 Pin disponibili:   PIN DIGITALI INPUT
                    46 - 47 - 48 - 49 - 50 - 51 - 52 - 53

Inserisci il numero del pin e il nome, [quit] per terminare
51 interruttore1
Inserisci il numero del pin e il nome, [quit] per terminare
47 interruttore2
Inserisci il numero del pin e il nome, [quit] per terminare
46 interruttore3
Inserisci il numero del pin e il nome, [quit] per terminare
quit
```

Figura 9 Shell Client

```
CONFIGURAZIONE TERMINATA


NOME DEVICE: SmartHouse

  PIN ANALOGICI                        PIN DIGITALI/SWITCH/DIMMER            INPUT

  PIN   0 -->    potenziometro         PIN   2 -->                           PIN  46 -->    interruttore3
  PIN   1 -->                          PIN   3 -->            led3           PIN  47 -->    interruttore2
  PIN   2 -->                          PIN   5 -->                           PIN  48 -->
  PIN   3 -->                          PIN   6 -->                           PIN  49 -->
  PIN   4 -->                          PIN   7 -->            led2           PIN  50 -->
  PIN   5 -->                          PIN   8 -->                           PIN  51 -->    interruttore1
  PIN   6 -->                          PIN  11 -->            led1           PIN  52 -->
  PIN   7 -->                          PIN  12 -->                           PIN  53 -->


LISTA COMANDI:

  accendi         [accendi] [nome pin]
  spegni          [spegni] [nome pin]
  intensità       [intensità] [valore: 0-100] [nome pin] (regola l'intensità su una scala da 0 a 100)
  leggi           [leggi] [nome pin]                     (legge lo stato 0/1 del pin digitale input o la tensione del pin analogico da 0 a 5.00 v)
  stato           [stato] [nome pin]                     (legge lo stato acceso/spento del pin digitale e la sua intensità)
  leggiconfig     [leggiconfig]                          (stampa l'elenco dei pin con i rispettivi nomi)
  resettaconfig   [resettaconfig]                        (annulla nome device e nomi pin)
  config          [config]                               (entra in modifica configurazione permettendo il cambiamento/aggiunta dei nomi)



Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

accendi led1

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

intensità 50 led2
```

Figura 10 Shell Client

```
Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

stato led1

Stato led1
Luce accesa
Intensità 100

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

stato led2

Stato led2
Luce accesa
Intensità 50

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

stato led3

Stato led3
Luce spenta
Intensità 0

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

leggi interruttore3

Lettura interruttore3
Valore: 1

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

leggi interruttore2

Lettura interruttore2
Valore: 0

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

leggi interruttore3

Lettura interruttore3
Valore: 1

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

leggi potenziometro

Lettura potenziometro
Valore: 3.27

Inserisci un comando: ([quit] per uscire, [help] per lista comandi)

quit
```

Figura 11 Shell Client

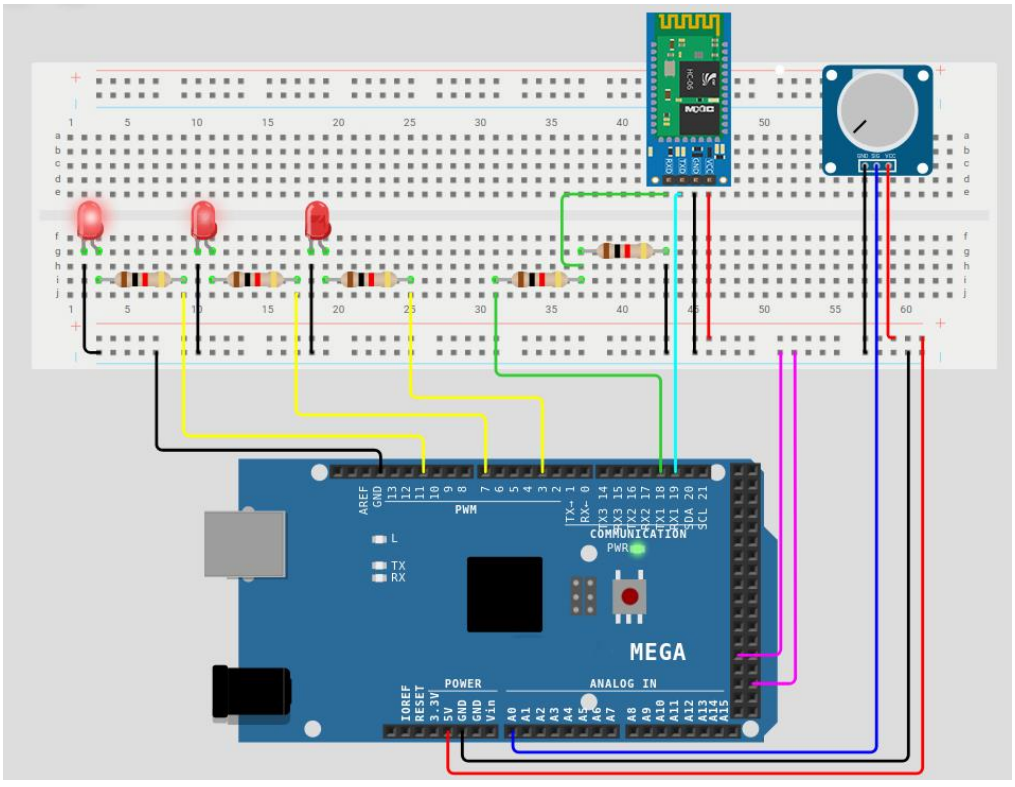


Figura 12 SmartHouse

Bibliografia

- *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet, 2014*, Atmel.
- *Dispense del corso "Sistemi Operativi"*, Giorgio Grisetti.
- *Bluetooth for Programmers*, Albert Huang, Larry Rudolph.