Scuola universitaria professionale
della Svizzera italiana
SUPSI

Monitoraggio bus domotico KNX

Studente

Giacomo Rivola

Relatore

Ragazzini Gian Luca

Correlatore

Dozio Gian Carlo

Committente

-

Corso di laurea

Ingegneria elettronica

Modulo

Progetto di diploma

Anno

2020-2021

Data

13 settembre 2021

Monitoraggio bus Domotico KNX/LonWorks Tramite Raspberry

Persone coinvolte

Proponente	Ragazzini Gian Luca
Relatore	Ragazzini Gian Luca
Correlatore	Dozio Gian Carlo

Dati generali

Codice	C10372
Anno accademico	2020/2021
Semestre	Semestre estivo
Corso di laurea	Ingegneria elettronica (Ingegneria elettronica TP)
Opzione	Nessuna opzione
Tipologia del progetto	Diploma
Stato	Proposta
Confidenziale	No
Pubblicabile	Sì

Descrizione

Per il corso di Domotica, si vuole sviluppare un dispositivo che si interfacci con il bus KNX e/o LonWorks permettendo la visualizzazione e il logging dei messaggi.
L'utilizzo del dispositivo sarà inizialmente didattico, ma potrebbe evolvere in un sistema di gestione domotico multistandard.
Preferenziale l'utilizzo di sistemi tipo Raspberry o Arduino per standardizzazione. In alternativa altri sistemi embedded SUPSI possono essere valutati. Da valutare lo sviluppo di una logica programmabile a bordo tipo "Home PLC". In caso di successo è possibile valutare la certificazione del dispositivo per lo standard KNX.

Compiti

- Valutazione della piattaforma HW/SW da utilizzare;
- Selezione dei dispositivi e dell'interfaccia KNX/LonWorks;
- Selezione della modalità di visualizzazione dati (webserver/HMI/PC/...);
- Scrittura del driver KNX/LonWorks e driver di visualizzazione;
- Definizione della logica di decodifica dei messaggi e di logging;
- Scrittura del codice di logging e routing verso HMI;
- Debug.

Obbiettivi

Rendere disponibile il dispositivo in modalità didattica all'interno del corso di Domotica.

Tecnologie

- KNX;
- LonWorks;
- Arduino;
- Raspberry;
- PLC;
- Automazione domestica e di edificio.

Contatto esterno

Nessun contatto esterno presente.

Documenti allegati

Nessun allegato presente.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei Listati

Glossario

CH_4 Metano.

CO_2 Anidride carbonica.

NO_x Ossidi di azoto.

.csv *Comma-separated values*.

Ack *Acknowledgement*.

APCI *Application Layer Protocol Control Information*.

BCI *BatiBUS Club International*.

bps *bit per seconds*.

CSS *Cascading Style Sheets*.

EHSA *European Home Systems Association*.

EIBA *European Installation Bus Association*.

EVB *Evaluation board*.

HMI *Human Machine Interface*.

HTML *HyperText Markup Language*.

HTTP *Hypertext Transfer Protocol*.

LDO *Low-dropout regulator*.

MAC *Media Access Control*.

NTP *Network Time Protocol*.

OS *Operating system*.

ppmv Parti per milione in volume.

RTC *Real Time Clock*.

SELV *Safety Extra Low-Voltage*.

SPI *Serial Peripheral Interface*.

SPIFFS *Serial Peripheral Interface Flash File System*.

TFC *Total final consumption*.

UART *Universal Asynchronous Receiver-Transmitter*.

URL *Uniform Resource Locator*.

Abstract

L'obiettivo del progetto è realizzare un dispositivo di monitoraggio del bus KNX con la possibilità di inviare dei telegrammi. Oltre a ciò una funzionalità richiesta è quella di poter salvare un *file* contente tutti i telegrammi registrati dal dispositivo.

Dopo una prima fase di pianificazione del lavoro da compiere, e conseguentemente la ricerca dei componenti necessari, si è passati allo sviluppo del *software*. Questo rapido passaggio all'implementazione del codice è reso possibile dal fatto che non si è dovuto sviluppare un *hardware* specifico per questo progetto, in quanto si è optato per l'utilizzo di *Evaluation board* (EVB).

In seguito alla fase di sviluppo si è ottenuto un sistema in grado di ricevere i telegrammi KNX ed interpretarli, permettendo la visualizzazione di questi dati attraverso un *webserver*. Oltre a ciò è possibile salvare un *file* di testo in locale con la registrazione del traffico intercettato, e da ultimo ma non meno importate, si è implementato la base per l'invio di telegrammi KNX attraverso il *webserver*. Un interessante ulteriore sviluppo sarebbe quello di realizzare un PCB, e fare in modo che questo dispositivo diventi un vero e proprio nodo KNX.

1. Obiettivo

Il lavoro da svolgere in questo progetto è quello di realizzare un dispositivo in grado di interfacciarsi con il bus KNX, riuscendo dunque a monitorarlo e a mandare messaggi sullo stesso. In aggiunta allo sviluppo del *software* necessario alla decodifica dei messaggi e all'invio di quest'ultimi, si deve eseguire un'attenta valutazione della piattaforma *hardware* e *software* da utilizzare. Oltre a ciò è imperativo trovare una *Human Machine Interface* (HMI) adeguata a questo progetto, in modo da semplificare l'approccio dell'utente al dispositivo.

L'obiettivo ultimo è fondamentalmente quello di rendere disponibile il *device* in modalità didattica all'interno del corso di Domotica.

I vincoli presenti sono essenzialmente due: il primo è rappresentato dal *budget* di CHF 300.00 a disposizione. Il secondo è invece il tempo a disposizione per lo sviluppo dell'apparecchio, ovvero all'incirca 3 mesi. Un altro aspetto stabilito con la collaborazione del Professor Gian Luca Ragazzini è che vista l'esigua quantità di tempo per la realizzazione del progetto, si preferisce creare un primo prototipo con esclusivamente l'utilizzo di EVB.

2. Introduzione

L'impellente bisogno di ridurre i consumi dovuti alle attività umane ha spinto alla creazione di tecnologie per assecondare questa necessità. Gli ambiti coinvolti dall'avvento di queste nuove tecnologie sono molteplici e spaziano dalla mobilità, all'industria sostenibile. Una tematica che negli ultimi anni sta suscitando sempre più interesse e attira su di se l'attenzione è quella della *building automation*. Con questo termine ci si riferisce a tecnologie in grado di rendere un edificio "intelligente" e ridurre i consumi di energia grazie alla gestione dell'illuminazione, riscaldamento, ventilazione, aria condizionata, e così via.

Siccome sia nel settore residenziale che terziario i consumi maggiori sono causati dal riscaldamento e rispettivamente dal raffrescamento, ed in seguito quelli causati da carichi elettrici, si capisce che attraverso una gestione intelligente di quest'ultimi si otterrebbe un risparmio energetico enorme.

Tale gestione intelligente avviene attraverso degli impianti domotici. Per questo motivo all'interno del ciclo di studio di ingegneria elettronica alla SUPSI è presente il corso di Domotica. Il corso appena menzionato, ha quale obiettivo quello di fornire agli studenti la conoscenza delle tecnologie moderne per l'automazione di ambienti residenziali e industriali nell'ottica del risparmio energetico.

Altro punto chiave del corso è acquisire la capacità di valutare i vantaggi e i limiti dell'automazione in svariati contesti. Dato che all'interno del corso vi è anche una parte pratica nella quale si prende dimestichezza con degli impianti basati sulla tecnologia KNX, si vuole sviluppare un sistema di monitoraggio del bus che permetta agli studenti di capire cosa sta succedendo in ogni istante.

3. Metodologia

Il progetto è sostanzialmente diviso in due parti: la prima, soprattutto organizzativa, consiste nel prevedere quali siano le varie fasi del progetto, e per ognuna di esse, identificare delle possibili problematiche e rispettivamente difficoltà, riconoscendo un eventuale dispendio di tempo dovuto a fattori di natura logistica. La seconda parte, prevalentemente pratica, è incentrata sullo sviluppo del *software* necessario al funzionamento del prototipo e sulla costruzione dell'*hardware* richiesto per raggiungere lo scopo del progetto.

L'inizio del lavoro è caratterizzato da un'attenta raccolta di informazioni, necessarie per stabilire quali fossero i componenti più adatti per raggiungere l'obiettivo. Successivamente per padroneggiare le basi teoriche atte allo sviluppo del prototipo, è necessario consultare varie fonti tecniche riguardanti il protocollo KNX. Per quanto riguarda la contestualizzazione del progetto, le informazioni sono state prevalentemente raccolte da libri e articoli tecnici e scientifici. Grazie a queste letture è stato possibile costruire un quadro sommario della situazione energetica e climatica che si sta vivendo.

Per tener traccia dell'andamento del progetto, ed essere sicuri che si stesse rispettando le tempistiche stabilite, ci sono stati incontri settimanali con il professor Gian Luca Ragazzini. Tali colloqui hanno permesso sin da subito di confrontarsi e capire quali fossero le aspettative reciproche. Quest'ultimi sono funti da *checkpoint* per assicurarsi che tutto stesse procedendo come previsto, e per discutere alcune questioni qualora fosse necessario. In figura 3.1 è presente il diagramma di Gantt utilizzato per meglio capire quale fosse lo stato del progetto.

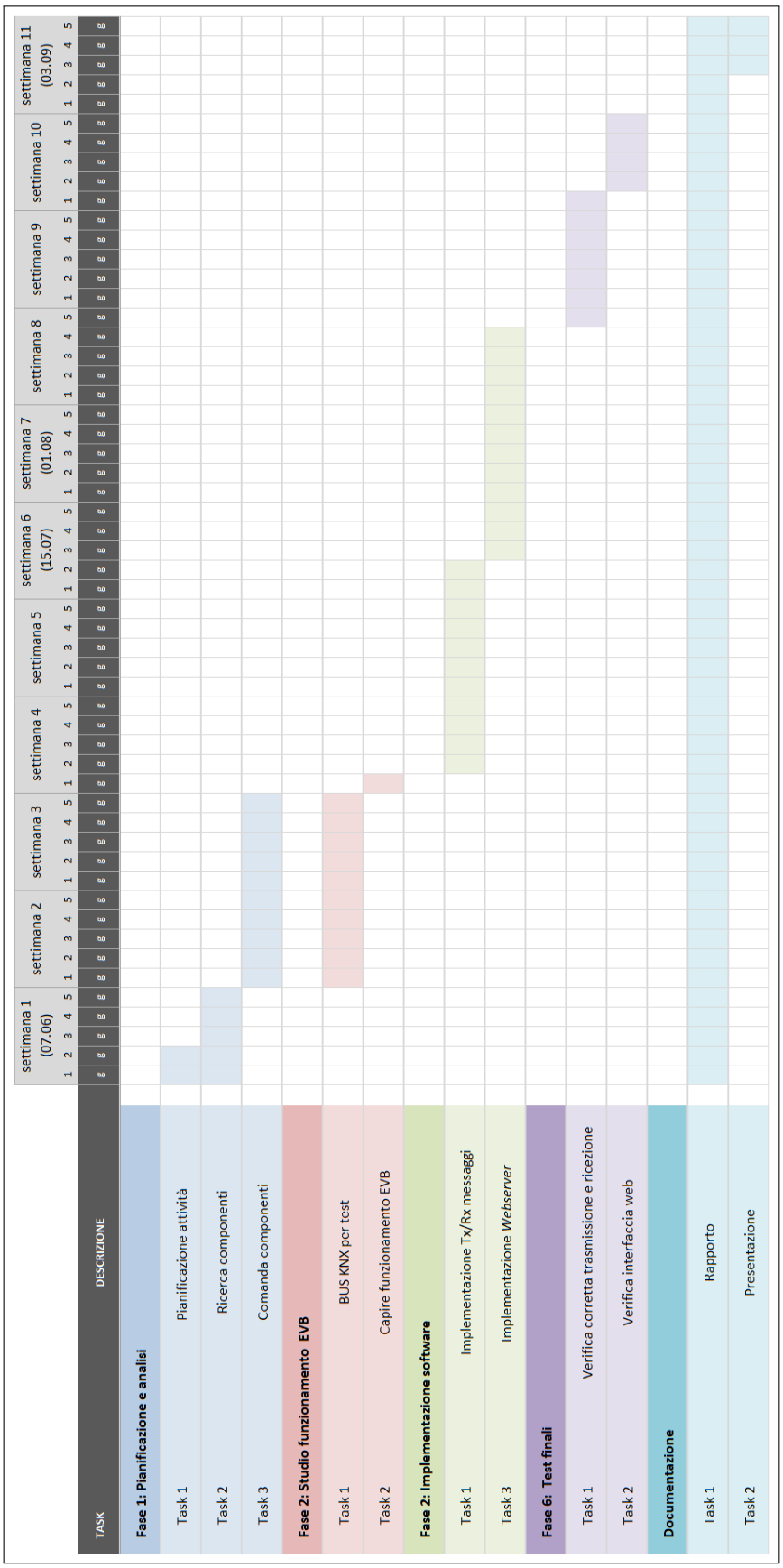

Figura 3.1: Gantt

4. Contestualizzazione

4.1 Situazione climatica attuale

Il cambiamento climatico ha fatto si che negli ultimi anni si iniziasse a dibattere su quali strategie adottare per contrastarlo. Un argomento sul quale si presta sempre molta attenzione, è come diminuire la concentrazione di Anidride carbonica (CO_2) nell'atmosfera. Il gas appena citato è il principale responsabile dell'aumento della temperatura del pianeta, quest'ultimo causa infatti l'effetto serra. Il valore di concentrazione naturale di CO_2 è di 280 Parti per milione in volume (ppmv) e negli ultimi $70/80$ anni in modo particolare, vi è stato un considerevole aumento; tanto che si è stimato che tra aprile e giugno del 2021 la concentrazione di CO_2 nell'aria raggiunga i $417ppm$ [5]. Il che equivale ad un aumento di quasi il 49%.

Nell'atmosfera terrestre il CO_2 e il vapore acqueo sono degli assorbitori selettivi, ovvero consentono alla radiazione solare (compreso le lunghezze d'onda dello spettro visibile) di giungere sulla superficie terrestre, ma allo stesso tempo riflettono parte della radiazione infrarossa nello spazio. La radiazione infrarossa è fondamentale per lo sviluppo di vita su un pianeta, infatti grazie a questi raggi il clima sulla terra può riscaldarsi. Ovviamente è un equilibrio molto sottile, un eccesso o un deficit di questi raggi è dannoso per l'ecosistema. Quanto appena citato è definito effetto serra, conseguentemente i gas che contribuiscono al verificarsi del fenomeno sono definiti gas ad effetto serra. I gas ad effetto serra che sono naturalmente presenti all'interno dell'atmosfera sono il vapore acqueo (H_2O), CO_2, Metano (CH_4) e anche gli Ossidi di azoto (NO_x). Come già accennato in precedenza la presenza di questi gas è essenziale per la vita sulla Terra, in assenza di essi la temperatura presente non permetterebbe lo sviluppo di vita. La ricerca dei gas appena menzionati e lo spessore dell'atmosfera sono degli elementi (assieme a molti altri) che aiutano gli esperti ad escludere o meno pianeti sui quali lo sviluppo di vita potrebbe essere fattibile. Negli ultimi 120 anni la concentrazione dei gas ad effetto serra è però aumentata, ciò ha portato all'innalzamento della temperatura media di circa $0.6K$[6]. Una differenza di temperatura di quest'entità costituisce un grande aumento, basti pensare che una differenza di appena $2K$ è ciò intercorre tra il clima odierno e quello dell'ultima glaciazione.

4.2 Contesto energetico

Come visibile in figura 4.1, il consumo di energia negli ultimi anni è aumentato sempre più. La produzione come il consumo di energia, hanno un impatto diretto sull'inquinamento dell'ambiente e contribuiscono all'aumento dei gas ad effetto serra. Ulteriori problemi si riscontrano con l'inquinamento delle acque dovuto all'estrazione dal sottosuolo di combustibili fossili o per eventuali fuoriuscite di sostanze tossiche, inoltre vi sono pure le scorie nucleari che hanno un'influenza negativa sull'ecosistema. Il motivo per il quale il consumo energetico aumenta è perché attualmente rappresenta uno dei principali propulsori dello sviluppo economico e sociale, basti pensare quanto il riscaldamento di un edificio e la sua illuminazione favoriscano al miglioramento della qualità di vita.

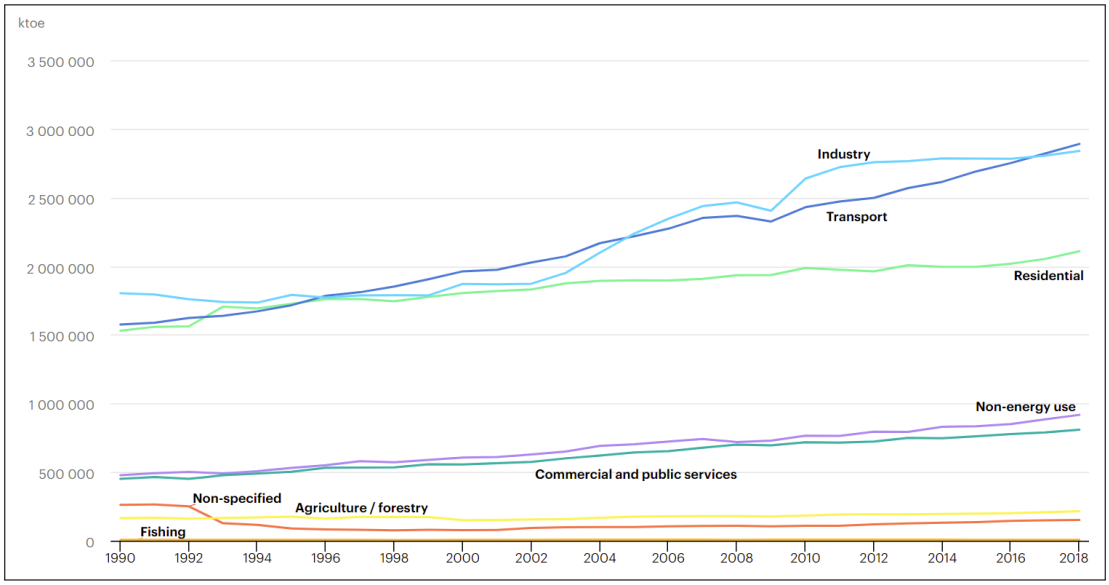

Figura 4.1: TFC *by sector, World* 1990-2018[1]

I consumi finali illustrati in figura 4.1 mostrano che i settori nei quali vi è un maggior consumo di energia sono l'industria, il trasporto e il settore residenziale. L'aumento dei consumi anche nel settore residenziale fa si che meriti delle attenzioni in più rispetto agli anni passati. Appare dunque chiaro che per ridurre le emissioni di gas ad effetto serra tutti i settori principalmente coinvolti nella produzione di quest'ultimi debbano prendere dei provvedimenti. Ecco che a questo punto, oltre all'isolamento termico di un edificio, entrano in gioco impianti domotici e di *building automation*, i quali permettono di ridurre notevolmente il fabbisogno di energia.

4.3 Impianto domotico

Un impianto domotico è un sistema che coinvolge vari impianti presenti in un edificio e permette di metterli in comunicazione. A differenza di un impianto elettrico tradizionale, come visibile in figura 4.2, si hanno tutti i sistemi collegati dallo stesso filo, ovvero il bus.

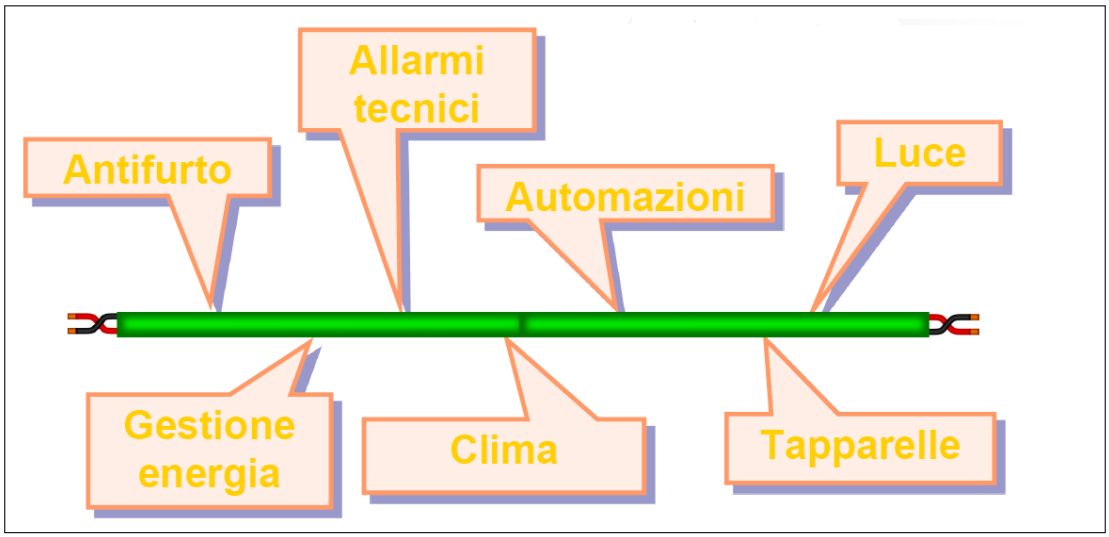

Figura 4.2: Esempio di bus domotico [2]

Una peculiarità di questi sistemi è che ai comandi (per esempio un interruttore) viene

collegato solo il bus, mentre invece l'alimentazione viene portata solo dove necessario. Grazie a dei messaggi inviati sul bus vengono eseguiti vari comandi, per esempio alla pressione di un pulsante ogni carico riceve un messaggio che traduce poi in un'azione. L'automazione di alcuni compiti come l'accensione o lo spegnimento del riscaldamento oppure come l'illuminazione permettono una riduzione dei consumi.

4.4 Importanza degli impianti domotici

Per meglio comprendere il ruolo degli impianti domotici in uno scenario futuro, è bene farsi un'aspettativa del risparmio energetico da essi scaturito. La norma europea UNI EN 15232 fornisce dei metodi di calcolo del risparmio energetico in un edificio e quattro classi di automazione per l'efficienza energetica:

- Classe D "*non energy efficient*": comprende gli impianti tradizionali. Sono privi di automazione e non efficienti dal punto di vista energetico;
- Classe C "*standard*": è la classe di riferimento e comprende impianti dotati di automazione e controllo senza una supervisione centrale;
- Classe B "*advanced*": impianti provvisti di sistemi di automazione e controllo completi di TBM (*Techincal Building Management*) per un controllo centralizzato;
- Classe A "*high energy perfomance*": come la classe B ma con livelli di precisione e completezza del controllo automatico tali da garantire elevate prestazioni energetiche all'impianto.

La norma definisce poi delle funzioni di automazione, controllo e supervisione per poter poi stabilire a quale classe di efficienza appartiene un certo controllo di un edificio.

A questo punto per poter effettuare il calcolo del risparmio energetico di un edificio vi sono fondamentalmente due strategie: il metodo dettagliato e il metodo dei fattori BAC. Il metodo dettagliato è quello più accurato dei due, tuttavia è anche più complesso. Non è scopo di questo capitolo trattare come avviene il calcolo dettagliato del risparmio energetico, ma piuttosto è quello di fornire un'idea generale di quanto possa ridurre i consumi un impianto domotico. Ecco che dunque il metodo dei fattori BAC è quello al quale ci si appoggia. È fondamentalmente molto semplice: una volta stabilita la classe di efficienza al quale un edificio appartiene (A, B, C o D), che è stabilita a seconda delle tecnologie che sono presenti al suo interno, la tabella 4.1 mostra il consumo che si otterrebbe rispetto ad un edificio standard, ovvero di classe C.

Tabella 4.1: Fattori di efficienza secondo la norma UNI EN 15232

Edificio	**Tipi di edifici**	**fbac, el**				**fbac, hc**			
		D	C	B	A	D	C	B	A
Non residenziale	Uffici	1.10	1	0.93	0.87	1.51	1	0.80	0.70
	Sala di lettura	1,06		0.94	0.89	1.24		0.75	0.50
	Scuole	1.07		0.93	0.86	1.20		0.88	0.80
	Ospedali	1.05		0.98	0.96	1.31		0.91	0.86
	Hotel	1,07		0.95	0.90	1.31		0.85	0.68
	Ristoranti	1.04		0.96	0.92	1.23		0.77	0.68
	Strutture di vendita all'ingrosso e non	1.08		0.95	0.91	1.56		0.73	0.6
Residenziale	Abitazioni singole	1.08		0.93	0.92	1.10		0.88	0.81
	Appartamenti								
	Altre tipologie di edificio								

La tabella 4.1 mostra chiaramente come in alcuni contesti il risparmio raggiungibile è considerevole. Per esempio, nel caso degli uffici, il divario dei consumi tra un edificio di classe A e uno standard sarebbe del 30%. Tale differenza aumenta ancor più rispetto ad un edificio privo di automazione, salendo fino al 81%. Sebbene la norma assuma come riferimento la classe C, in pratica sovente la situazione reale corrisponde alla classe D, ciò significa che il potenziale risparmio energetico applicando questo tipo di sistemi è di una certa rilevanza.

È plausibile pensare che nei prossimi anni i sistemi domotici e di *building automation* assumeranno un ruolo sempre più centrale all'interno degli edifici. Il primo standard di *building automation* è il bus KNX, nella sezione 4.5 vengono esplicitate le principali caratteristiche di questo sistema.

4.5 Il bus KNX

Quando negli anni '80 l'automazione prendeva sempre più piede e vi fu il passaggio da una logica centralizzata ad una distribuita, i costruttori di impianti domotici si trovarono di fronte ad un bivio: creare un protocollo proprietario oppure crearne uno standard. Un decennio più tardi in Europa erano presenti tre aziende in questo ambito, le quali proponevano ognuno la propria tecnologia. I tre concorrenti in questione erano:

- *European Installation Bus Association* (EIBA);
- *BatiBUS Club International* (BCI);
- *European Home Systems Association* (EHSA);

Le aziende appena citate si unirono, creando così la cosiddetta Konnex e il protocollo KNX. I principali vantaggi di questo protocollo sono la sua interpolabilità, l'ampia scelta di dispositivi di diversi produttori, semplicità nel cablaggio e la flessibilità dell'impianto una volta ultimato.

Il cavo del bus KNX lavora con una tensione *Safety Extra Low-Voltage* (SELV) di 29V DC. È schermato ed ha un doppio isolamento, garantito fino ad una tensione di 4 kV. Al suo interno oltre al doppino per la trasmissione dell'alimentazione e dei telegrammi è presente un filo di continuità. Questi tre cavi sono avvolti a loro volta da uno schermo contro i disturbi elettromagnetici.

4.5.1 Architettura

In un sistema KNX si fa distinzione tra aree, linee e dispositivi. In figura 4.3 è presente una rappresentazione grafica di quanto appena detto.

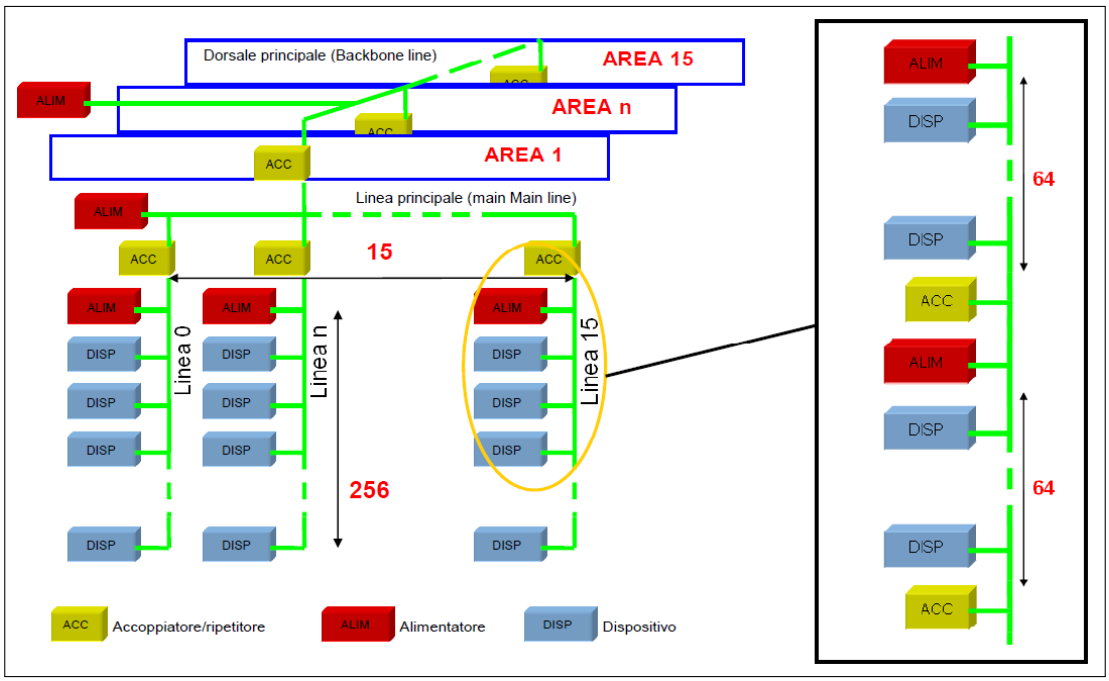

Figura 4.3: Architettura del sistema KNX [2]

Avendo a disposizione 15 aree, 15 linee, e su ogni linea 256 dispositivi si possono collegare ad un bus KNX più di 60'000 dispositivi.

I dispositivi sono inoltre caratterizzati da:

- indirizzo fisico;
- indirizzo di gruppo.

L'indirizzo fisico viene attribuito ad ogni dispositivo in base alla posizione e alla topologia dell'impianto. Per esempio, l'indirizzo 1.5.25 indica che il dispositivo è collocato nell'area 1, quinta linea, ed è il venticinquesimo dispositivo presente sulla linea. Da notare che i numeri sono separati da dei punti.

Per quanto riguarda invece l'indirizzo di gruppo, il suo scopo è quello di collegare a livello logico i vari dispositivi. Per questo tipo di indirizzo invece i numeri sono separati da delle barre oblique. Per esempio nell'indirizzo 0/1/12 il primo numero indica il gruppo principale, il secondo il gruppo centrale e il terzo invece il sottogruppo.

Come visibile in figura 4.3 ogni 64 dispositivi è necessario inserire un'alimentazione e un accoppiatore/ripetitore.

4.5.2 Telegramma

Tutte le informazioni vengono veicolate attraverso il bus con il cosiddetto telegramma. Al verificarsi di un dato evento il telegramma viene spedito sul bus, la trasmissione avviene solo dopo che il bus è stato libero per un certo istante di tempo (t1). Una volta trasmesso si attende un altro periodo di tempo (t2) per verificare che la ricezione sia avvenuta nel modo corretto. Successivamente un *Acknowledgement* (Ack) viene mandato da tutti i dispositivi indirizzati dal telegramma. Quanto appena esposto è riassunto nella figura 4.4.

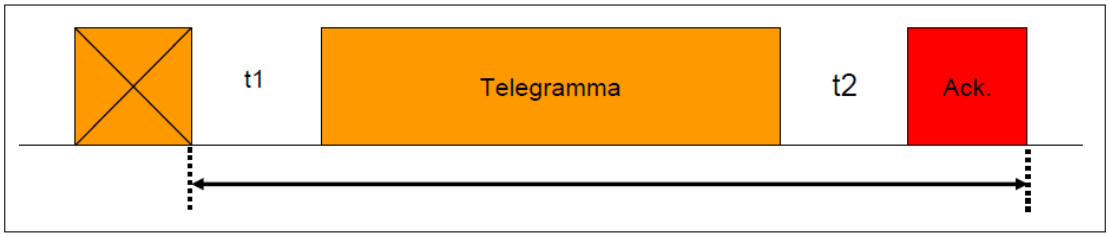

Figura 4.4: Invio di un telegramma [2]

Come visibile in figura 4.5 il telegramma è composto da:

- 8 *bit* di controllo: servono per impostare la priorità di trasmissione e la ripetizione;
- 16 *bit* di indirizzo sorgente: sono sempre l'indirizzo fisico del dispositivo mittente;
- 16+1 *bit* di indirizzo di destinazione: solitamente è l'indirizzo di gruppo al quale viene inviato il telegramma. Siccome potrebbe indicare anche un indirizzo fisico l'ultimo bit specifica se quello ricevuto ricevuto sia un indirizzo fisico o di gruppo (0 = indirizzo fisico);
- 3 *bit* di routing counter;
- 4 *bit* che indicano la lunghezza del telegramma;
- fino a 16 $\times$ 8 bit di dati;
- 8 *bit* di verifica.

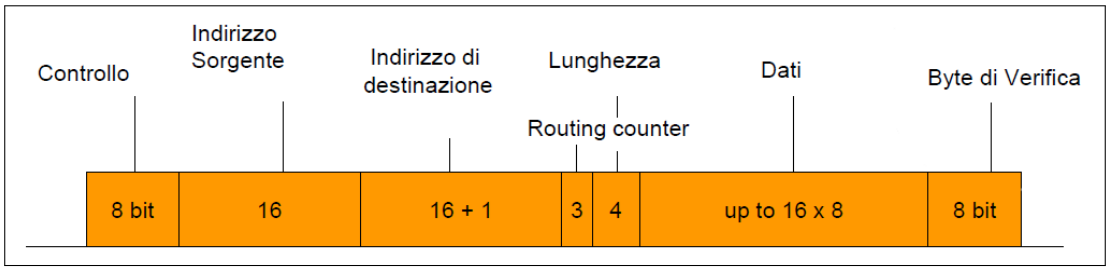

Figura 4.5: Composizione di un telegramma [2]

4.5.2.1 *Byte* di controllo

Il *byte* di controllo è il primo elemento del telegramma KNX.

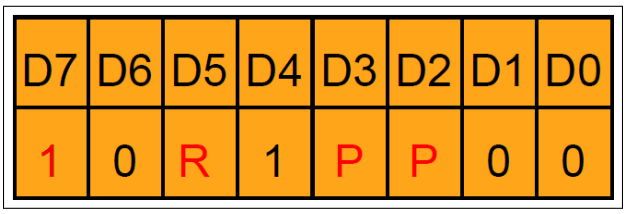

Figura 4.6: *Byte* di controllo [2]

Come visibile in figura 4.6 i *bit* D6, D4, D1 e D0 non cambiano mai.

Il *bit* D7 indica il tipo di telegramma: se a '0' si tratta di un *extended frame*, ovvero di un telegramma la cui lunghezza varia tra 9 e 263 *byte*. Nel caso in cui invece D7 è a '1' il telegramma è uno *standard frame*, quindi con una lunghezza che varia da 8 a 23 *byte*.

Il *bit* D5 indica invece la ripetizione del telegramma: se a '0' il telegramma non si ripete, quando invece a '1' sì.

È importante notare che i *bit* quando ricevuti da un dispositivo sono nel senso opposto, ovvero il primo che si riceve è l'LSB e l'ultimo il MSB.

I *bit* D2 e D3 definiscono la priorità del telegramma:

D3	D2	Priorità di trasmissione
0	0	$Funzioni\ di\ sistema\ (priorita'\ piu'\ alta)$
1	0	$Funzioni\ di\ allarme$
0	1	$Priorita'\ alta\ (normale)$
1	1	$Priorita'\ bassa$

4.5.2.2 Indirizzo sorgente

I due *byte* che seguono quello di controllo rappresentano l'indirizzo sorgente.

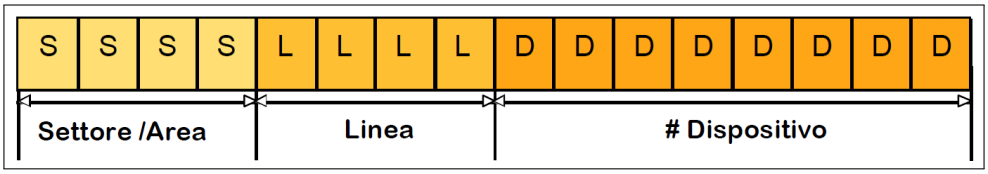

Figura 4.7: Indirizzo sorgente [2]

Come illustrato in figura 4.7 il primo *nibble* indica il settore o l'area in cui si trova il dispositivo. Il secondo *nibble* rappresenta invece la linea. L'ultimo *byte* è l'identificativo del dispositivo.

4.5.2.3 Indirizzo di destinazione

Il campo dell'indirizzo di destinazione è composto da 2 *byte*. Può essere un indirizzo fisico o di gruppo. È fisico sono quando la comunicazione è punto/punto, generalmente è dunque di gruppo.

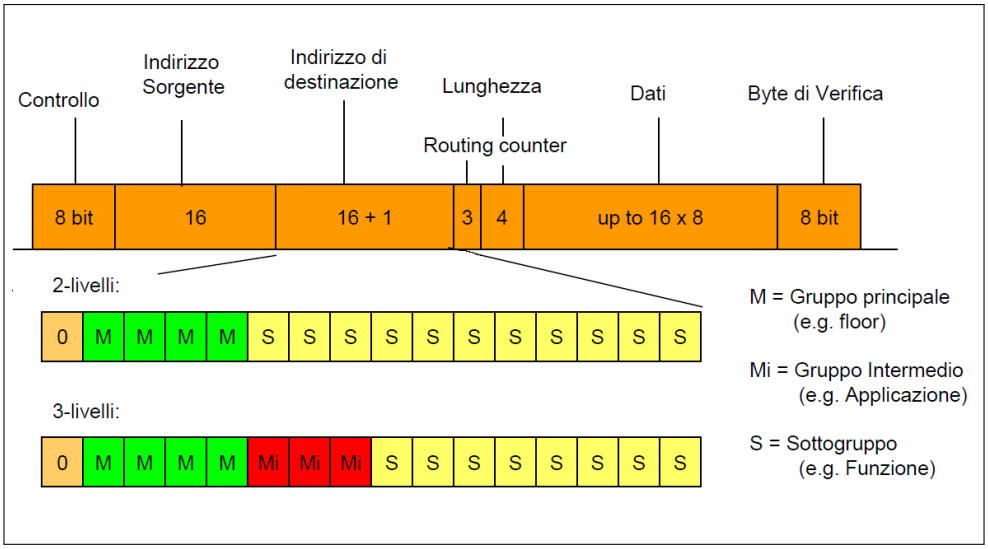

Figura 4.8: Indirizzo di destinazione [2]

Come visibile nella in figura 4.8 un indirizzo di gruppo può essere a sua volta di due o tre livelli. Nella versione a tre livelli si ha un gruppo principale, uno intermedio e un sottogruppo.

4.5.2.4 Tipo di indirizzo, *routing counter* e lunghezza

Riprendendo la figura 4.5 si può osservare che il *byte* che segue l'indirizzo di destinazione fornisce tre informazioni:

- Il primo *bit* indica se l'indirizzo di destinazione trasmesso è di tipo fisico o di gruppo. Nel caso in cui questo *bit* assume il valore '0' l'indirizzo è di tipo fisico, è invece di gruppo se il livello logico è '1';
- I seguenti 3 *bit* compongono il *routing counter*; necessario nel caso in cui nella rete KNX è presente un accoppiatore. Di fatto il *routing counter*, o il conteggio *hop* in italiano, fa in modo che i telegrammi *multicast* non viaggino all'infinito in una rete KNX anche in caso di errori topologici;
- I restanti 4 *bit* indicano il numero di *byte* di dati utili che compongono il *payload*. Nel caso in cui questo *nibble* abbia un valore pari a 0001, la lunghezza totale del *payload*, considerando il *byte* 0 e il *byte* 1 sarebbe di due *byte*.

4.5.2.5 Dati

Il campo "Dati" è quello più elaborato di tutti. Il suo contenuto non è costante, e varia a seconda del comando inviato sul bus KNX. Per questo motivo non si descrivono le generalità di questa parte di telegramma ma si analizza il campo "Dati" di due tipi di comandi molto utilizzati: la funzione di *switch* e di *dimming*.

La funzione di *switch* è utilizzata per commutare lo stato di un attuatore, come per esempio una luce o l'accensione e lo spegnimento di un riscaldamento.

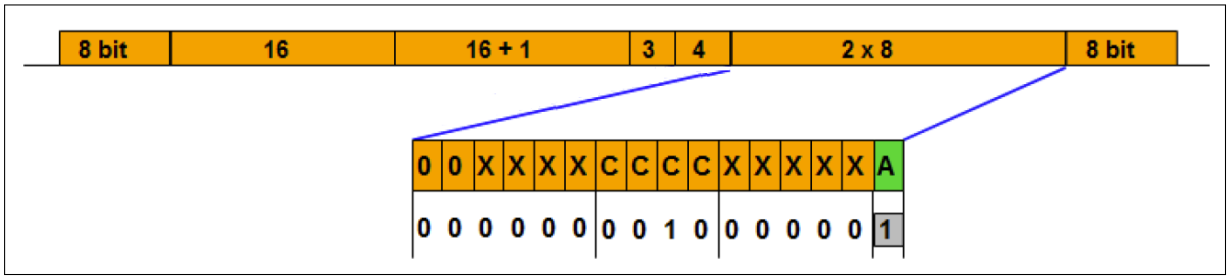

Figura 4.9: *Payload* della funzione *switch*

La figura 4.9 illustra il *payload* della funzione di *switch*, è bene innanzitutto chiarire che cosa indicano le varie lettere:

- A: *group object*;
- C: comando;
- X: non considerato.

I *bit* d'interesse sono dunque quelli contrassegnati dalle lettere A e C. Il *nibble* indicato con le lettere C rappresenta l'*Application Layer Protocol Control Information* (APCI). L'APCI è l'identificatore di servizi. Di fatto indica qual'è lo scopo del telegramma inviato. Nella tabella 4.3 sono indicati tutti i nomi delle APCI presenti, in sostanza facendo capo a questa tabella si può capire il compito svolto da qualsiasi telegramma inviato. In questo esempio si nota che l'APCI è 0010, che corrisponde quindi all'azione *GroupValueWrite*. Ciò è sensato in quanto il telegramma commuta lo stato in un attuatore. A questo punto l'ultimo *bit* 'A' indica lo stato che si vuole imporre all'attuatore:

A	Azione
0	$On/Enable/True/Alarm$
1	$Off/Disable/False/no\ Alarm$

Prendendo invece in analisi la funzione di *dimming*, si osserva in figura 4.10 che nel telegramma vi sono delle sostanziali differenze. In comune tra i due vi è ancora l'APCI, infatti lo scopo è sempre quello di impostare l'attuatore ad un valore desiderato. In questo caso per poter impostare correttamente l'attuatore non è più sufficiente considerare solo l'ultimo *bit*, occorre infatti prendere in considerazione gli ultimi quattro.

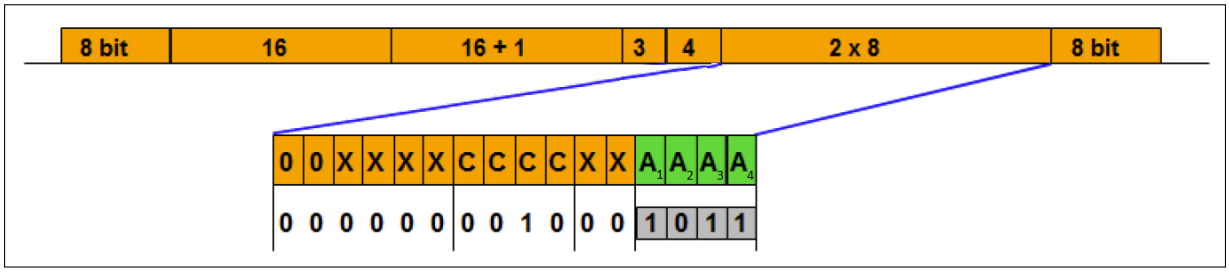

Figura 4.10: *Payload* della funzione *dimming*

Il primo *bit*, ovvero A_1, indica se il *dimming* sta rendendo la luce più forte o più fievole e va interpretato come segue:

A_1	***Dimming direction***
0	*dim darker*
1	*dim brighter*

I restanti 3 *bit* (A_2, A_3, A_4) è il cosiddetto *dimming code*, ed indica percentualmente di quanto deve essere aumentata o diminuita la luminosità di una luce.

Tabella 4.2: Livello di luminosità associato al dimming code

Codice	Livello di luminosità
000	Stop dimming
001	Dimming level 1
010	Dimming level 2
011	Dimming level 4
100	Dimming level 8
101	Dimming level 16
110	Dimming level 32
111	Dimming level 64

L'interpretazione di questo esempio è dunque "aumenta la luminosità del 25%".

Quelli visti in precedenza sono solo due esempi di funzioni disponibili, ce ne sono moltissime e ad esse corrispondono vari telegrammi. Appare dunque evidente che non si può interpretare un telegramma in maniera univoca. Per poterlo decodificare completamente è necessaria una descrizione dei dispositivi che contenga il *device ID*. Quest'informazione non è contenuta nel telegramma, perciò basandosi solo sul traffico dati presente sul bus la decodifica sarà parziale.

Tabella 4.3: Nome delle varie APCI

APCI	**Nome**
0000	*GroupValueRead*
0001	*GroupValueResponse*
0010	*GroupValueWrite*
0011	*IndividualAddrWrite*
0100	*individualAddrRequest*
0101	*IndividualAddrResponse*
0110	*AdcRead*
0111	*AdcResponse*
1000	*MemoryRead*
1001	*MemoryResponse*
1010	*MemoryWrite*
1011	*UserMessage*
1100	*MaskVersionRead*
1101	*MaskVersionResponse*
1110	*Restart*
1111	*Escape*

5. Requisiti

I requisiti richiesti per questo lavoro di tesi sono:

- Visualizzazione del traffico sul bus KNX;
- Uso di Arduino o Raspberry preferito per standardizzazione;
- Uso del dispositivo per scopo didattico;
- Visualizzazione dei dati con *webserver*/HMI/PC;
- Invio di telegrammi KNX (opzionale).

5.1 Analisi SMART dei requisiti

Tabella 5.1: Analisi SMART requisiti

	Specific	**Measurable**	**Achievable**	**Relevant**	**Timely**
Visualizzazione traffico sul bus KNX	si	si	si	si	si
Uso di Arduino o Raspberry	no	no	si	si	si
Uso del dispositivo didattico	si	si	si	si	si
Visualizzazione dati con HMI	no	no	si	si	si
Invio telegrammi	si	si	si	si	si

Come visibile in tabella 5.1 i requisiti "Uso di un Arduino o Raspberry" e "Visualizzazione dati con HMI" non sono SMART, nel capitolo 6 vengono effettuate delle proposte per avere poi delle specifiche più dettagliate.

6. Soluzioni proposte

6.1 Proposta 1

Un possibile modo di affrontare il progetto è quello di utilizzare fondamentalmente questi due componenti:

- microcontrollore: ESP8266;
- *transceiver*: KNX TinySerial Interface 810.

Il microcontrollore ESP8266 può essere programmato utilizzando il *framework* di Arduino, sfruttando quindi i vantaggi offerti dalla comunità di Arduino sul *web*, possiede tuttavia delle prestazioni notevoli e inoltre ha integrato un modulo Wi-Fi e *Bluetooth*, caratteristiche interessanti per pensare anche ad eventuali sviluppi futuri.

Il *KNX TinySerial Interface 810* è una semplice connessione al bus KNX. Questo modulo una volta ricevuto in entrata un telegramma KNX lo traduce e lo invia ad un eventuale microcontrollore via UART. Questo modulo è sviluppato dalla ditta Weinzierl e *online* si trova una documentazione ben redatta.

6.2 Proposta 2

Alternativamente si potrebbero utilizzare anche i seguenti componenti:

- microcontrollore: ESP8266;
- *transceiver*: NCN5130ASGEVB.

L'*evaluation board* NCN5100ASGEVB è compatibile con Arduino e tutti i componenti esterni per far funzionare il *transceiver* sono già presenti sulla *board*. Ci sono tre differenti modelli del *transceiver*: il NCN5110, il NCN5121 e il NCN5130. Preferibilmente si opterebbe per il NCN5121 o il NCN5130 in quanto hanno già implementato il *Media Access Control* (MAC) *layer*. Sul *datasheet* del componente è indicato che è compatibile anche con altri microcontrollori. È importante notare che l'*evaluation board* ha un *form factor* identico a quello di dell'Arduino UNO, è pertanto conveniente trovare una scheda di sviluppo per l'ESP8266 con il suddetto *form facotr*. Per questo motivo si è optato per il microcontrollore indicato nell'elenco puntato all'inizio di questo sottocapitolo. Procedendo in questo modo sarebbe possibile montare il *transceiver* direttamente sopra il microcontrollore, mantenendo quindi l'intero sistema relativamente compatto.

In alternativa a questa scheda di sviluppo vi sarebbe anche il *transceiver* 5WG1117-2AB1 prodotto da Siemens. Il problema rappresentato da questo prodotto è che la sua forma non lo rende appetibile per rendere il sistema compatto una volta terminato.

6.3 Proposta 3

La terza e ultima alternativa per la realizzazione di questo progetto si basa sui seguenti componenti:

- RASPBERRY PI® – KNX INTERFACE;
- RASPBERRY PI® 4.

Di fatto questa proposta è simile alle altre: un HAT permette di interfacciare il RASPBERRY al bus KNX. C'è però anche da considerare che le prestazioni offerte dal RASPBERRY sono sì superiori a quelle di un microcalcolatore, ma potrebbero anche rivelarsi sovradimensionate per questo progetto. Inoltre prestazioni maggiori significa anche costo maggiore, e lo si vuole mantenere, per quanto possibile, basso.

6.4 Interfaccia grafica

Per interfacciarsi con l'utente, per tutte le possibilità sopracitate si opterebbe per la realizzazione di un *webserver*. Nonostante si debba sviluppare un *software* dedicato, è la soluzione che meglio si presta a queste tre opzioni.

6.5 Analisi soluzioni proposte

Tabella 6.1: Metodo scientifico per la comparazione delle proposte

	Importanza	Proposta 1	**Nota (0-6)**	Proposta 2	**Nota (0-6)**	Proposta 3	**Nota (0-6)**
Reperibilità	0.1	1	6	6	6	6	6
Esperienza	0.1	5	6	5	6	0	6
Facilità di utilizzo	0.3	2	4.5	4	6	3	2
Prezzo	0.5	0	6	5	4.5	4	2
	1		1.2		4.8		3.5

Per una scelta il più obiettiva possibile si è optato per un analisi delle proposte applicando il metodo scientifico, e perciò si è redatto la tabella 6.1. Quest'analisi stabilisce che la proposta migliore è la seconda, rappresenta per tanto l'approccio migliore per il progetto da svolgere.

Dopo queste considerazioni si può affermare che si necessita una cifra di CHF 300.00 per sviluppare il progetto.

7. Specifiche

Dopo le considerazioni presenti nella sezione 6.5, si possono definire le seguenti specifiche.

7.1 *Transceiver* per bus KNX

Il *transceiver* scelto è il NCN5130 prodotto dalla ditta *ON Semiconductor*. Si vuole utilizzare un *evalation board* per questo componente, la NCN5130ASGEVB, così da non dover costruire un PCB *ad-hoc* e risparmiare dunque tempo per l'implementazione del software. Come già accennato nella sezione 6.2 il *form factor* di questa scheda è identico a quello dell'Arduino UNO.

7.2 Microcontrollore

Il microcontrollore scelto è un ESP8266 montato su una scheda di sviluppo con un *form factor* identico a quello dell'Arduino UNO, ciò permette di collegare facilmente la NCN5130ASGEVB come *HAT* sul microcontrollore ESP8266. Questo micro ha inoltre anche un modulo Wi-Fi e uno *Bluetooth* integrati.

7.3 HMI

L'interfaccia grafica scelta è un *webserver*. Siccome il microcontrollore utilizzato è un ESP8266, è già dotato di un modulo Wi-Fi, ciò toglie anche la necessità di dover aggiungere un modulo esterno. Inoltre il *webserver* rappresenta la soluzione più flessibile e meno vincolante, permettendo eventuali cambiamenti nel corso del progetto, qualora fosse necessario.

7.4 Funzionamento generale

Il sistema una volta completato avrà le seguenti funzionalità:

- *Sniffing* del traffico sul bus KNX;
- Interpretazione dei telegrammi KNX;
- Visualizzazione del *logging* dei messaggi su schermo;
- Invio di telegrammi KNX tramite una *webserver* (opzionale).

8. Principio di funzionamento

Come visibile in figura 8.1 il *transceiver* legge un telegramma KNX e lo converte in un formato leggibile per un microcontrollore. Si può scegliere se comunicare al micro i dati rilevati via *Universal Asynchronous Receiver-Transmitter* (UART) oppure tramite *Serial Peripheral Interface* (SPI), per quest'applicazione si opta per la prima delle opzioni elencate. Il micro una volta ricevuto il messaggio trasmesso sul bus KNX lo mostra all'utente attraverso il *webserver*. Grazie a quest'ultimo è possibile scegliere se visualizzare il traffico dati presente in quell'istante, inviare telegrammi precedentemente registrati, visualizzare lo storico degli ultimi dieci telegrammi, oppure scaricare un *file* di testo contenente il *log* di tutto il traffico registrato.

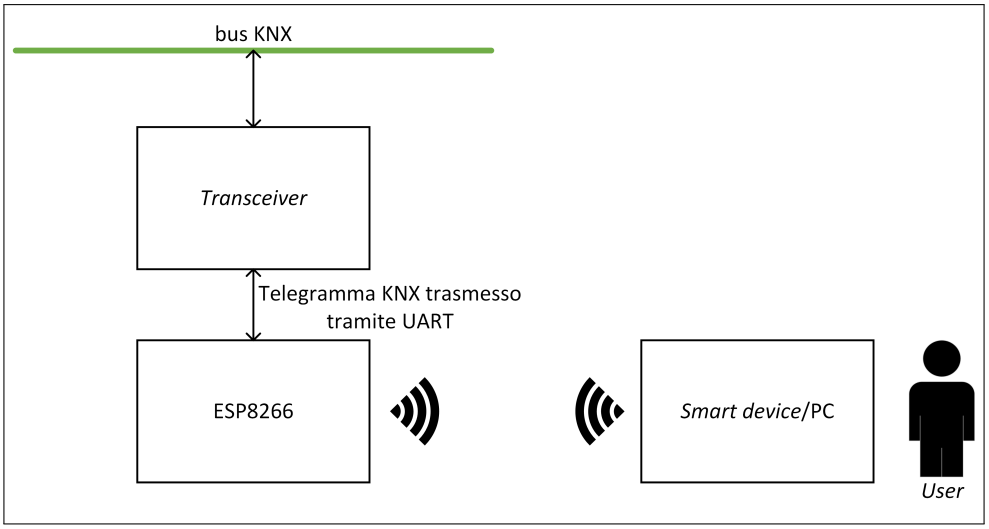

Figura 8.1: Schema a blocchi del dispositivo

9. Sviluppo *software*

In questo capitolo vengono analizzati gli aspetti più importanti della parte *software* di questo progetto. I codici sono modulari, nel senso che sono indipendenti l'uno dall'altro, facilitando così anche eventuali cambiamenti o ulteriori sviluppi.

9.1 SerialCommunication.c

Lo sviluppo *software* inizia con la comunicazione tra bus KNX, *transceiver* e microcontrollore. Questa parte è ovviamente la base del progetto, infatti per poter capire e interpretare i messaggi inviati sul bus si necessita forzatamente del *transceiver* per poter comunicare via UART con il micro. Per raggiungere questo scopo si è implementato il *file* denominato *SerialCommunication.c*. Per gestire il *transceiver* non è presente una libreria specificata dal produttore. *Online* se ne trova una, tuttavia siccome la documentazione ad essa annessa è scarsa si è deciso di implementare l'intero *software* da zero. Essenzialmente il lavoro svolto dal codice in questo modulo è quello di leggere i messaggi KNX e interpretare il telegramma ottenuto in modo da poter capire per esempio l'indirizzo fisico, quello di destinazione ecc... . Una volta che il telegramma è stato interpretato i dati ricavati vengono preparati e memorizzati in variabili all'interno di strutture. Qualora un altro modulo che compone il *software* del sistema necessitasse di questi dati, accederebbe semplicemente alla variabili appena menzionate.

Come visibile in figura 9.1 il codice è diviso in due parti. Una parte è quella che si occupa dell'acquisizione dei dati e del monitoraggio del sistema, l'altra è una componente che rende possibile l'invio dei messaggi sul bus. Nei prossimi due sotto capitoli si analizzano più nel dettaglio le due parti di codice appena menzionate.

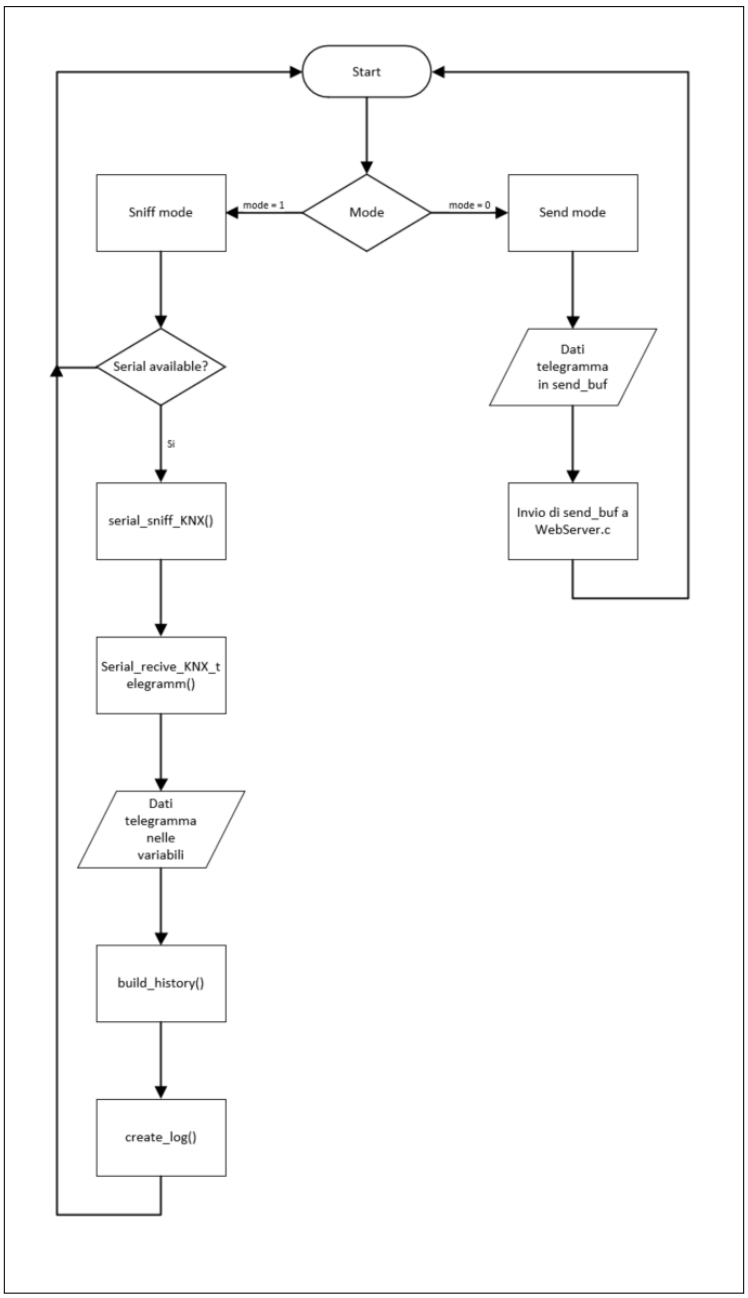

Figura 9.1: Diagramma funzionale del modulo SerialCommunication.c

9.1.1 Sniff mode

In questa modalità il *transceiver* è impostato su *U_Busmon.req*, quindi come spiegato più nel dettaglio nella sezione 10.1.3, il *trasceiver* legge in modo trasparente tutti i messaggi che passano sul bus e li comunica al micro via UART. All'interno di questa modalità vengono utilizzate principalmente, assieme ad altre funzioni, la serial_sniff_KNX() e la serial_recive_KNX_telegramm().

All'interno della funzione serial_main(), quando ci si trova nella modalità di monitoraggio del sistema, vengono chiamate le due funzioni appena citate. In questa modalità lo scopo è suddividere i dati ricevuti dal bus per permettere poi una visualizzazione più chiara all'utente. Di fatto si parte dunque dal telegramma grezzo ricevuto dal bus e lo si suddivide nelle varie parti che lo compongono. Per memorizzare i vari campi del telegramma si sono creati due tipi di dati, come visibile nel listato 9.1 e nel listato 9.2. La creazione di due tipi di dato diversi, è resa necessaria dal fatto che un telegramma KNX è suddiviso su più livelli: si comincia dal telegramma grezzo, il quale viene suddiviso nei suoi campi principali. I dati scaturiti da questa suddivisione vengono memorizzati nella struttura illustrata nel listato 9.1. Considerando che a sua volta ogni elemento della struttura si suddivide in ulteriori sotto campi, occorre un'ulteriore elaborazione dei dati. Prendendo come esempio il *byte* di controllo presente nel listato 9.1, esso deve venir scomposto in modo da poter ottenere la ripetizione del telegramma e la sua priorità. I valori ottenuti da questa scomposizione vengono poi memorizzati nella struttura presente nel listato 9.2. Quest'operazione la si esegue per ogni variabile presente nella struttura KNX_telegram_raw_t. Procedendo in questo modo si ottiene una scomposizione completa del telegramma, perciò qualora si necessitasse di un certo dato contenuto nel telegramma, per accedere a quest'informazione, si deve far capo alla struttura KNX_telegram_t.

```
typedef struct
{
   uint8_t control;
   uint16_t sourceAddress;
   uint16_t destinationAddress;
   uint8_t routing;
   char paylaod[16];
   uint8_t checksum;
} KNX_telegram_raw_t;
```

Listato 9.1: Tipo di dato KNX_telegramm_raw_t

```
typedef struct
{
   uint8_t repetition;
   uint8_t priority;
   uint8_t sector;
   uint8_t line;
   uint8_t device;
   uint8_t mainGroup;
   uint8_t intermediateGroup;
   uint8_t subGroup;
   uint8_t groupAddressbit;
   uint8_t routingCounter;
   uint8_t payloadLength;
   uint8_t command;
   uint8_t object[16];
   uint8_t checksum;
} KNX_telegram_t;
```

Listato 9.2: Tipo di dato KNX_telegramm_t

Le operazioni appena citate sono rese possibili, come già accennato in precedenza, dalle funzioni serial_recive_KNX_telegramm() e serial_sniff_KNX(). La prima è responsabile del prelievo dal *buffer* di lettura del micro, dei dati ricevuti dal bus. Questa funzione resta in ascolto di eventuali messaggi a partire dalla ricezione del primo dato fino allo scadere di un certo tempo. Questo lasso di tempo assicura che tutti i dati del telegramma siano stati ricevuti, tradotti e mandati nel buffer di ricezione UART, evitando quindi di effettuare più letture dello stesso telegramma oppure eseguirne solo una parziale. Una volta che la ricezione di un telegramma è completata interviene la serial_sniff_KNX(), la quale esegue le operazioni di suddivisione del telegramma su più livelli, come spiegato nel paragrafo precedente.

Dalla figura 9.1 si nota che vengono chiamate anche altre due funzioni quando ci si trova in *sniff mode*, ovvero la build_history() e la create_log(). La prima delle due funzioni è necessaria per poter mostrare sul *webserver* uno storico dei telegrammi ricevuti. Il compito svolto da questa funzione è unicamente quello di registrare in un *array* i dati degli ultimi dieci telegrammi ricevuti. Per quanto riguarda invece la *create_log()*, il suo scopo è quello di scrivere sotto forma di stringa i dati di tutti i telegrammi ricevuti in un *file* di testo, contenuto nel *Serial Peripheral Interface Flash File System* (SPIFFS) del microcontrollore.

9.2 Webserver.c

In questo capitolo si spiegano le generalità con le quali si è implementato il *webserver* ed in seguito si analizzano più nel dettaglio le parti principali del codice sorgente.

Il compito del *webserver* è quello di permettere la visualizzazione del traffico dati sul bus KNX all'utente. La seconda funzionalità è invece quella di poter mandare dei telegrammi sul bus, ovvero impostare dei valori agli attuatori presenti nel sistema domotico in questione. L'idea di base è quella di realizzare il *webserver* andando a memorizzare i *files HyperText Markup Language* (HTML) e *Cascading Style Sheets* (CSS) nello SPIFFS dell'ESP8266. Procedendo in questo modo si evita di dover mandare un stringa contenente tutto il codice HTML qualora fosse necessario; infatti si carica direttamente il *file* richiesto. Un ulteriore vantaggio di questo approccio è che si ottiene un codice più pulito e ordinato, mantenendo ben distinte le parti di codice C, HTML e CSS. In figura 9.2 è illustrata l'organizzazione dei *files* sorgente dedicati al *webserver*.

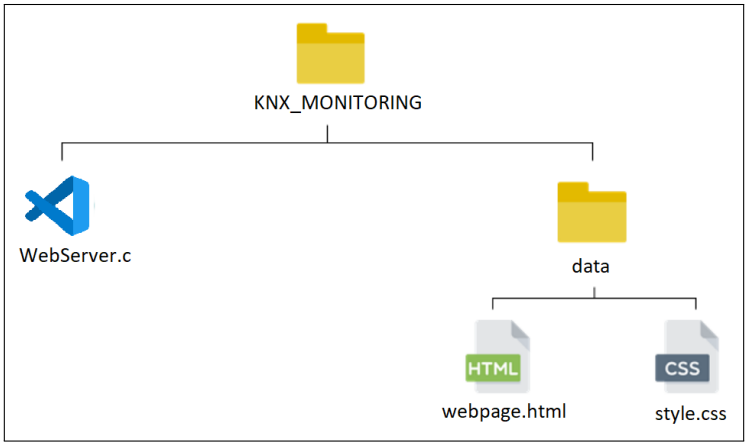

Figura 9.2: Organizzazione *files* sorgente *webserver*

In figura 9.3 è raffigurato un diagramma di come funziona il *webserver*. Nei prossimi sotto capitoli si analizza più nel dettaglio le varie parti e ci si sofferma maggiormente sulle parti di codice più interessanti.

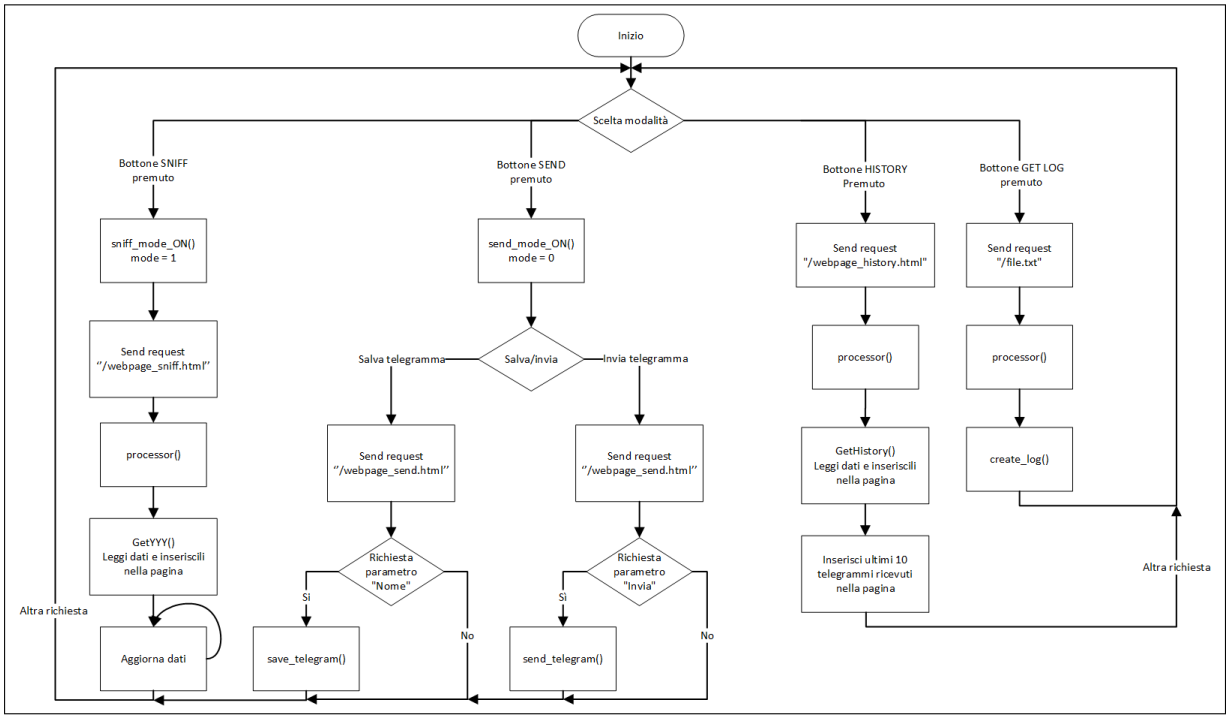

Figura 9.3: Diagramma funzionale del *webserver*

Per accedere al *webserver*, siccome l'ESP8266 funge da *hotspot*, occore cercare tra le reti Wi-Fi del proprio dispositivo (PC, *tablet* o *smartphone*) la rete denominata "KNX-monitoring". Nel caso in cui si stia cercando di fare l'accesso con un dispositivo Android, è sufficiente selezionare la rete appena citata, e grazie al *capitve portal*, la pagina *web* che permette la visualizzazione del traffico sul bus si apre automaticamente. Nel caso invece si volesse accedere alla pagina *web* con un PC, oppure con *tablet*/*smartphone* che operano con altri *Operating system* (OS), una volta connessi alla rete fornita dal microcontrollore, si deve inserire nella barra di ricerca di un qualsiasi *browser* l'indirizzo IP 192.168.4.1.

9.2.1 *SNIFF mode*

Una volta selezionata questa modalità, attraverso la pressione dell'apposito bottone presente nel *webserver*, il microcontrollore invia al *transceiver* il comando per entrare in *monitoring mode*. Siccome il *webserver* è asincrono, è possibile configurare dei percorsi per i quali il *webserver* resta in ascolto e una volta riconosciuti agisce di conseguenza. Nel listato 9.3 si può osservare che qualora si ricevesse una richiesta *Hypertext Transfer Protocol* (HTTP) che coincide con l'*Uniform Resource Locator* (URL) "/mode1" il *webserver* oltre ad eseguire la funzione che permette di entrare in *monitoring mode* invia anche il *file* "/webpage_sniff.html" al *client*. In oltre un altro paramentro importante della funzione *send()* è la funzione *processor()*, necessaria per poter sostituire i *placeholder*. Più avanti si approfondisce il lavoro svolto dalla funzione appena menzionata.

```
webServer.on("/mode1", [](AsyncWebServerRequest *request)
                {
                    sniff_mode_on();
                    request->send(SPIFFS, "/webpage_sniff.html", String(), false, processor);
                });
```

Listato 9.3: Esempio di richiesta HTTP

Una volta che il *file* HTML è ricevuto dal *client*, viene caricato e la pagina web visualizzata è generata grazie al codice HTML contenuto nel *file*. È importante far notare che per ogni bottone presente nella pagina *web* si è definito ed associato a quest'ultimo un URL inviato al momento della pressione. Come visibile nel listato 9.4 al momento dell'istanza di un bottone, all'interno del codice HTML gli viene associato il percorso "/mode1". Ogni bottone presente ha un diverso percorso da inviare qualora venisse premuto.

```
<a href="/mode1"><button class="button">SNIFF</button></a>
```

Listato 9.4: Associazione URL ad un bottone

Come menzionato in precedenza si necessita della funzione processor() per la sostituzione dei *placeholders* presenti nel codice HTML. Come visibile in figura 9.4 i *placeholders* altro non sono che stringhe con al loro inizio e fine il simbolo "%".

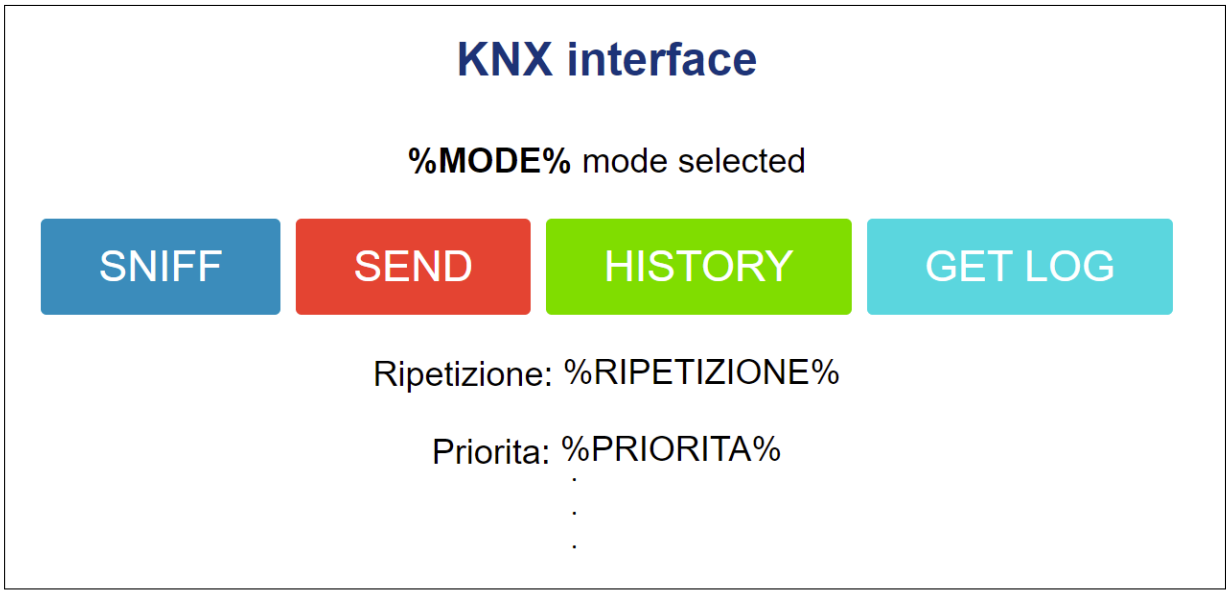

Figura 9.4: *Placeholders* presenti nella pagina *web*

Senza la funzione *processor()* sarebbe impossibile poter realizzare una pagina *web* dinamica, ovvero con dei valori o scritte che variano nel tempo. Per spiegarne il funzionamento ci si appoggia al listato 9.5, come si può vedere una volta chiamata questa funzione, confronta la stringa racchiusa tra simboli "%" presente nel codice HTML con quelle conosciute. Nel momento in cui dovesse trovare una corrispondenza agirebbe di conseguenza. Per esempio nel momento in cui incontra il *placeholder* %MODE%, controlla in quale modalità ci si trova e ritorna una stringa che viene messa al posto di %MODE% all'interno della pagina *web*. In questo caso la sostituzione è avvenuta direttamente, potrebbe anche capitare che si debba effettuare una chiamata ad una funzione per poter stabilire con cosa sostituire un *placeholder*. Effettivamente è proprio ciò che accade nel caso in cui la funzione trovi la stringa %RIPETIZIONE%: dapprima viene chiamata la funzione getRipetizione(), la quale ritorna una stringa che indica se la ripetizione del telegramma avviene oppure no, in seguito si sostituisce all'interno della pagina *web* %RIPETIZIONE% con "sì" oppure "no". Con la stessa logica tutti gli altri *placeholder* contenuti nel codice vengono sostituiti, ottenendo dunque una pagina *web* dinamica, come visibile in figura 9.5.

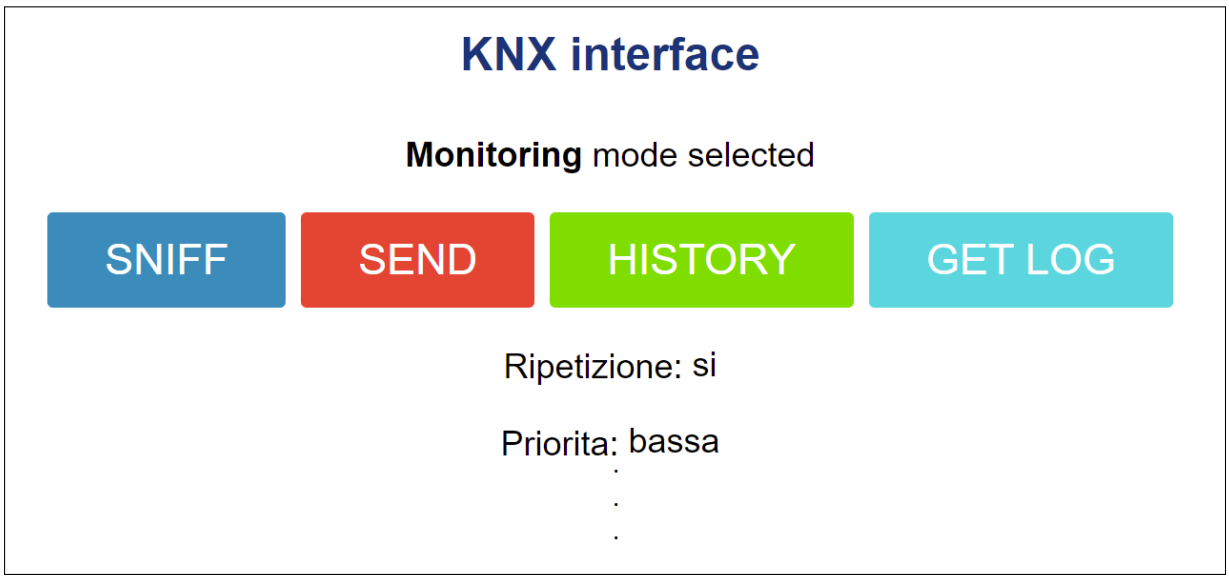

Figura 9.5: *Placeholders* sostituiti all'interno della pagina *web*

```
String processor(const String &var)
{
    if (var == "MODE")
    {
        String selectedMode;
        if (mode == 1)
        {
            selectedMode = "Monitoring";
        }

        else if (mode == 0)
        {
            selectedMode = "Send";
        }
        return selectedMode;
    }
    else if (var == "RIPETIZIONE")
    {
        return getRipetizione();
    }
    .
    .
    .
}
```

Listato 9.5: Estratto di codice dalla funzione *processor()*

Giunti a questo punto si ha una pagina *web* che in base al valore di alcuni dati varia, il problema sta nel fatto che in questo modo per poter sapere se un dato valore è cambiato occorre ricaricare manualmente la pagina *web*. Per ovviare a questa problematica si è aggiunto un pezzo di codice in *JavaScript* nel *file* HTML che permette di sostituire automaticamente i *placeholders* con l'ultimo valore rilevato. Dal listato 9.6 si può osservare che questa funzione viene chiamata ogni 100 ms e fondamentalmente fa una richiesta all'URL "/ripetizione" per avere l'ultimo valore rilevato dall'ultimo telegramma ricevuto. Una volta ricevuto, aggiorna l'elemento HTML con *id* "ripetizione". Lo stesso principio viene applicato a tutti gli altri *placeholders*, ottenendo dunque una pagina *web* in continuo aggiornamento, permettendo la visualizzazione istantanea di ciò che accade sul bus KNX. In questo modo, se il sistema resta impostato sulla modalità di monitoraggio del bus, continua ad aggiornare i dati ricevuti fintanto che l'utente non preme un altro dei pulsanti presenti nel *webserver* per cambiare modalità.

```
setInterval(function ( ) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("ripetizione").innerHTML = this.responseText;
      }
    };
    xhttp.open("GET", "/ripetizione", true);
    xhttp.send();
  }, 100);
```

Listato 9.6: Funzione responsabile dell'aggiornamento del *placeholder* della ripetizione

9.2.2 *Send mode*

Quando viene premuto l'apposito bottone, la pagina *web* caricata è quella illustrata nella figura 9.6.

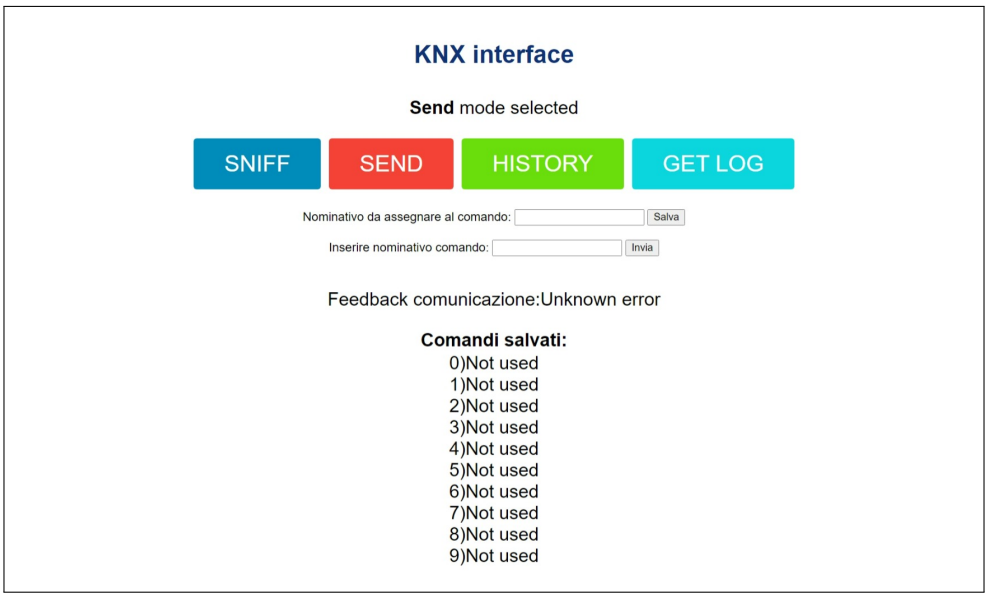

Figura 9.6: Pagina *web* in modalità *send*

Come si può osservare vi sono due campi nei quali si possono inserire *input* da tastiera. Per poter mandare un messaggio sul bus KNX si deve dapprima salvare il telegramma d'interesse, per farlo occorre essere in modalità *sniff* e una volta che sul bus è stato mandato il comando desiderato, per esempio "luce camera accesa", si entra in modalità *send* e lo si può salvare con un nominativo a scelta. È possibile salvare solo l'ultimo telegramma inviato sul bus. Quando si è eseguita quest'operazione e si è premuto salva, il nuovo comando salvato appare nella lista sottostante alla pagina *web*.

A questo punto per mandare un qualsiasi comando basta semplicemente inserire nel secondo campo della pagina *web* il nominativo col quale lo si è salvato e premere invia.

Come riportato in figura 9.3 la logica del codice per questa modalità è suddivisa in due. Sia che si voglia salvare o inviare un comando il programma invia il *file* "/webpage_send.html", ciò che cambia nei due casi è se la richiesta HTTP GET contiene il parametro "Nome" o "Invia". Nel caso in cui si voglia salvare un comando, e nella richiesta HTTP GET è contenuto il parametro "Nome", allora viene chiamata la funzione save_telegram(). Similmente quando si

vuole mandare un comando e la richiesta HTTP GET contiene il parametro "Invia", la funzione send_telegram() viene chiamata.

La funzione save_telegram() permette di salvare fino a dieci comandi diversi in un'*array* e nel caso in cui quest'ultimo sia pieno, il comando meno recente viene sovrascritto per permettere il salvataggio dell'ultimo. La send_telegram() si occupa invece di comporre dei pacchetti secondo la descrizione riportata nella sezione 10.1.4. Una volta svolto questo compito, invia semplicemente sulla seriale i dati preparati, il *transceiver* li traduce e li invia poi sul bus KNX.

Il prototipo realizzato, non essendo un nodo KNX, non ha un indirizzo fisico proprio. Ciò significa che mandando i messaggi in questo modo, ovvero replicando il telegramma mandato da un altro dispositivo, potrebbe creare dei conflitti. Quanto appena detto accadrebbe nel momento in cui si tenta di mandare un telegramma precedentemente registrato, e contemporaneamente viene immesso lo stesso telegramma da parte del dispositivo reale. Il metodo attualmente utilizzato è sostanzialmente una violazione del protocollo, si è comunque scelto questa strada per dimostrare che il *transceiver* scelto è in grado di svolgere anche questo compito, e che quindi, qualora si continuasse lo sviluppo di questo progetto, la strada intrapresa rappresenta già una base solida di partenza.

9.2.3 *History mode*

Questa è la modalità che permette la visualizzazione dello storico degli ultimi dieci telegrammi inviati sul bus. Una volta inviata la richiesta HTTP "/history" al *webserver*, viene caricata la pagina HTML "/webpage_history.html". In seguito avviene la chiamata alla funzione *processor()*, che come spiegato in precedenza sostituisce tutti i *placeholders* del caso; per farlo esegue una chiamata alla funzione getHistory(). Il compito svolto da quest'ultima è scrivere tutti i dati relativi ai telegrammi in un'unica stringa che viene poi visualizzata sulla pagina *web*. Per poter ottenere lo storico è però prima necessario costruirlo attraverso la funzione build_history(). La funzione appena citata viene chiamata all'interno della funzione serial_sniff_KNX() (vedi sezione 9.1.1). Ogni volta che viene ricevuto e suddiviso un telegramma, grazie alla funzione build_history(), una copia di quest'ultimo viene salvata in un *array* di tipo KNX_telegram_t. Man mano che si ricevono telegrammi la funzione aggiunge elementi all'*array*, e una volta raggiunto il numero massimo di elementi contenibili, il telegramma che risiede da più tempo nell'*array* viene sovrascritto per dar spazio all'ultimo telegramma ricevuto. La funzione getHistory() fa capo all'*array* appena menzionato per poter mostrare all'utente lo storico. Essenzialmente essa mostra tanti telegrammi quanti ne sono presenti nell'*array* e ritorna una stringa scritta in HTML per permettere la visualizzazione sulla pagina *web*.

```
history += String("<b>Telegramma numero </b> ") + String(i) + String("<br>");
history += String("<b>Ripetizione:</b> ") + String(telegram_history[i].repetition) + String("<br>");`
 .
 .
 .
```

Listato 9.7: Estratto di codice della funzione getHistory()

Nel listato 9.7 è presente un estratto del codice della funzione getHistory(). Come si può vedere si concatena più volte alla stringa *history* tutti i dati relativi al telegramma. Questa operazione viene svolta per tutti i telegrammi presenti nell'*array*, quindi di fatto si aggiungono sempre più dati alla stessa stringa. Il vantaggio di operare in questo modo è che anziché dover ritornare più stringhe se ne può restituire una sola, ed essendo già scritta in HTML una volta che verrà

inserita nella pagina *web* l'intero testo sarà formattato correttamente senza bisogno di ulteriori manipolazioni.

In figura 9.7 è visibile il risultato finale, come si può vedere sono presenti tutti i dati relativi ad un telegramma con una linea di separazione per capire meglio quando finisce e quando ne inizia uno nuovo.

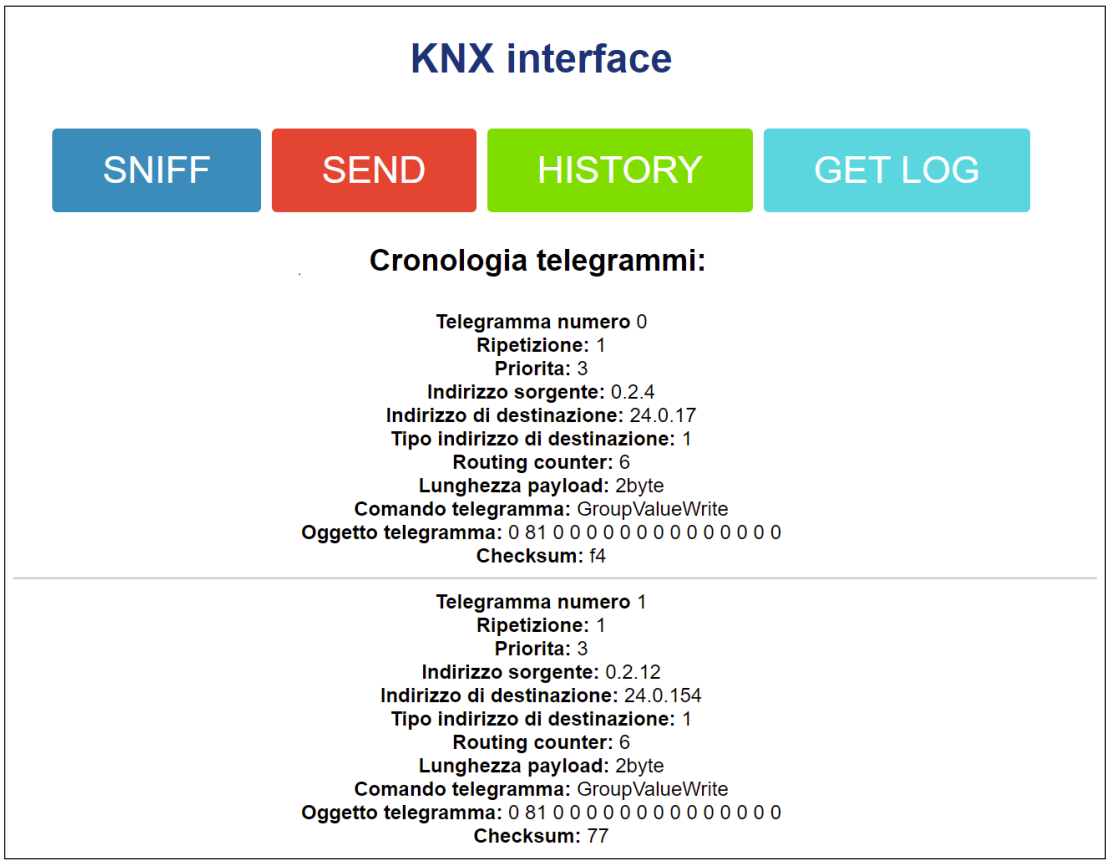

Figura 9.7: Pagina *web* dello storico dei telegrammi

9.2.4 *Get log mode*

Questa è la modalità che permette all'utente di scaricare un *file* di testo contenente tutto il traffico dati registrato dal dispositivo. Inizialmente viene creato e aperto in modalità di scrittura un *file* di testo all'interno dello SPIFFS dell'ESP8266. In seguito si estrapolano i dati caricati dalla funzione serial_sniff_KNX() nelle apposite variabili e vengono scritti nel *file* di testo. Quest'operazione viene eseguita per ogni telegramma ricevuto dal dispositivo, ottenendo dunque un vero e proprio *log file* con rapportati tutti gli eventi passati. Per completezza e per rendere utilizzabile il *log file* il sistema possiede un *Real Time Clock* (RTC), così da poter assegnare ad ogni telegramma registrato la data e l'ora di acquisizione. Per scrivere le informazioni ricevute nel *file* si utilizza la funzione create_log(). Quest'ultima ritorna un'unica stringa composta secondo lo stesso principio trattato nella sezione 9.2.3, listato 9.7. Una volta premuto il pulsante GET LOG, viene mostrata all'utente attraverso la pagina *web* un'anteprima del *file*, come illustrato in figura 9.8. Qualora l'utente volesse poi effettivamente scaricare e salvare in locale il *file* di testo lo può fare semplicemente premendo il tasto destro e "Salva con nome". Una volta salvato si possono effettuare tutte le operazioni possibili come su qualsiasi *file* di testo.

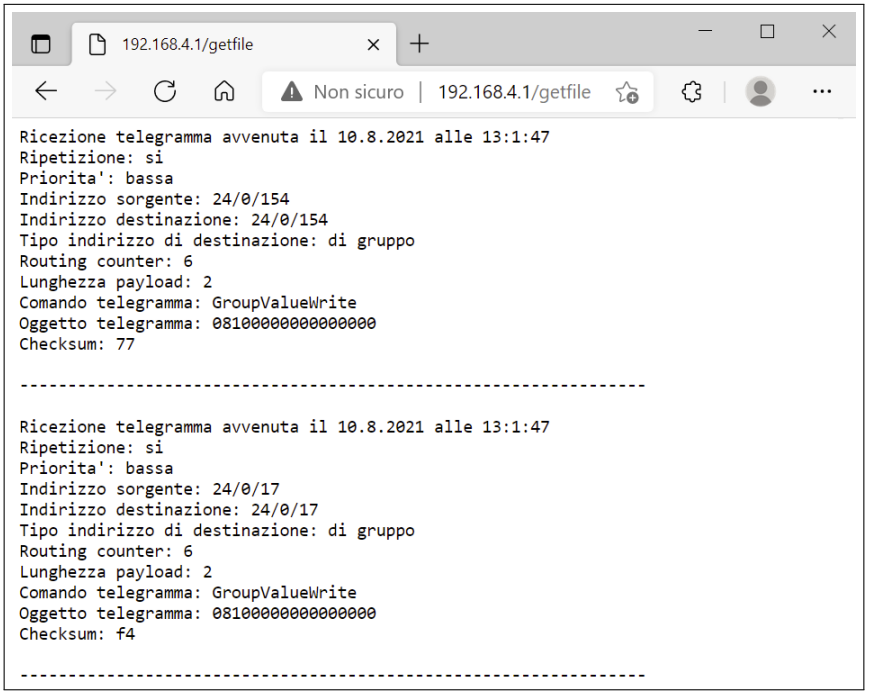

Figura 9.8: Anteprima del *file* di testo scaricabile

Siccome attualmente, al momento dell'accensione del micro si apre il *file* di testo nel quale viene scritto il log, e viene chiuso esclusivamente nel caso di un *reset*, per evitare la perdita dei dati nel caso in cui il *file* non venisse mai scaricato, in uno sviluppo futuro del progetto si potrebbe aggiungere una scheda SD nel quale salvare tutti i telegrammi. Sarebbe possibile per esempio scaricare automaticamente ogni ora un *file* con i telegrammi registrati fino a quel momento. Il micro allo stato attuale del progetto è in grado di memorizzare all'incirca 1500 telegrammi. Se dovesse monitorare un bus per un periodo di tempo molto esteso potrebbe però non essere sufficiente. Con l'aggiunta di una memoria esterna si eviterebbe anche di finire la memoria disponibile, evitando anche di conseguenza un *crash* del sistema.

10. Hardware

10.1 *Transceiver* NCN5130

Affinché ci si possa interfacciare al bus KNX è necessario un *transceiver*. Siccome sul bus l'alimentazione e i dati vengono trasmessi solo su due fili, si necessita un dispositivo che separa queste due componenti. Oltre a ciò si deve trasformare un telegramma KNX in un protocollo leggibile da un microcontrollore.

Questo è proprio il compito del NCN5130, infatti è in grado di ricevere ed interpretare il segnale elettrico ricevuto sul bus e rielaborarlo per poi fornire in *output* sull'interfaccia UART il suo equivalente leggibile dal microcontrollore. Supporta il collegamento di attuatori, sensori, microcontrollori e altri dispositivi per la domotica. Gestisce la trasmissione e la ricezione di dati sul bus ed è in grado di generare dalla tensione del bus non regolata, delle tensioni stabilizzate per la sua stessa alimentazione e per quella di dispositivi esterni. Sul foglio dati di questo componente è garantito che l'accoppiamento e il disaccoppiamento sul bus avviene in modo sicuro, evitando quindi di dover acquistare un *choke* per questo scopo. Grazie poi al monitoraggio del bus viene avvertito il microcontrollore nel caso in cui ci fosse una perdita di potenza così da permettere la memorizzazione dei dati critici.

Si è scelto di utilizzare questo componente già montato su un *evaluation board*, così si evita di dover progettare e sviluppare un PCB, risparmiando dunque tempo. Come visibile in figura 10.1 la piastra di sviluppo acquistata ha un *form factor* uguale a quello dell'Arduino UNO, rendendo il collegamento tra il *transceiver* e il micro molto semplice.

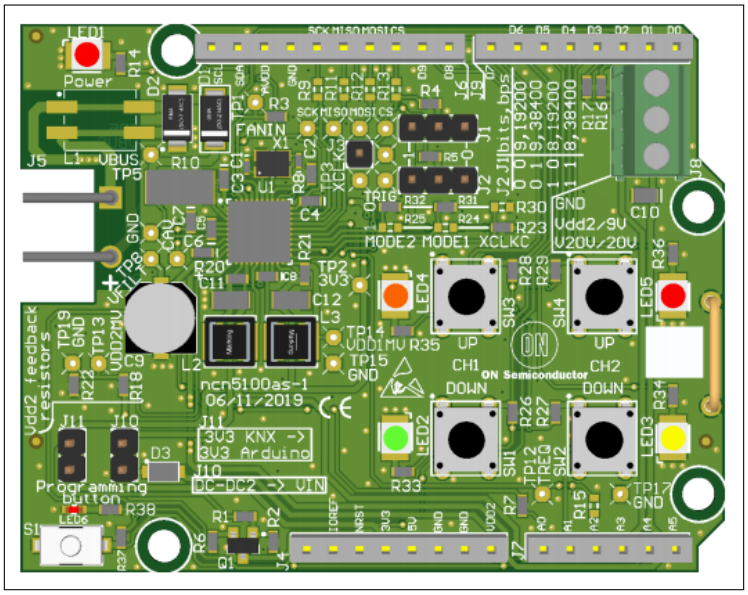

Figura 10.1: *NCN5130* montato sull'*evaluation board* [3]

Il *transceiver* permette la programmazione di dispositivi KNX per eseguire una determinata funzione. In questo progetto però, l'intenzione e il motivo principale per il quale si è acquistato questa piastra non è quello per il quale la *board* è stata progettata. L'intento è quello di effettuare uno *sniffing* del bus KNX, e questa scheda grazie alle caratteristiche esposte in precedenza si presta molto bene a tale obiettivo. Un altro motivo che rende questa piastra

molto performante è che vi è la possibilità di inviare dei telegrammi KNX in modo da poter eventualmente comandare da remoto dei componenti. Altra peculiarità del NCN5130 è che ha il *MAC layer* già implementato, è quindi compito dell'integrato gestire i *timings*, riducendo di conseguenza anche il carico di lavoro da svolgere nella fase di programmazione. In figura 10.2 si può vedere dove si piazza il *transceiver* all'interno degli *OSI layers*.

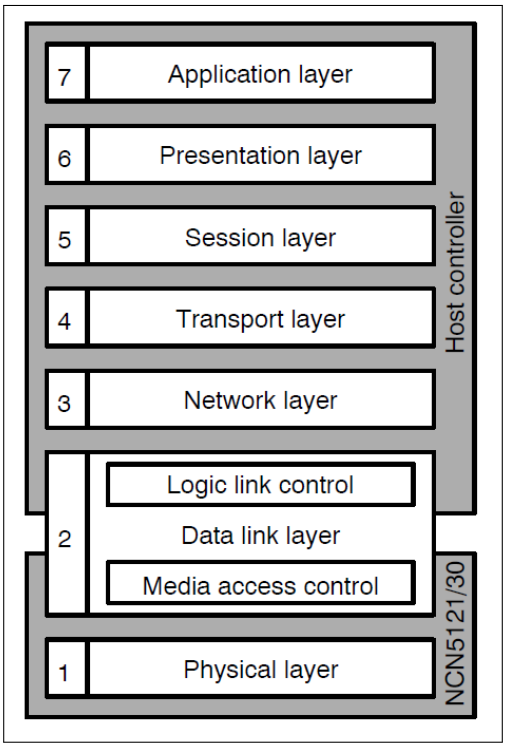

Figura 10.2: *OSI layers* [3]

L'host è dunque chiamato a gestire:

- *Checksum*;
- *Parity*;
- *Adressing*;
- *Length*.

L'NCN5130 gestisce invece:

- *Checksum*;
- *Parity*;
- *Acknowledgment*;
- *Repetition*;
- *Timing*.

Dopo questa prima parte introduttiva dove si sono descritte le generalità del componente si passa ad analizzare alcuni aspetti utili per la messa in servizio.

10.1.1 Alimentazione

La scheda è alimentata direttamente dal bus KNX, ed è accoppiata ad esso con una classica morsettiera KNX come quella rappresentata in figura 10.3. Alimentarla attraverso il bus KNX rappresenta la soluzione più sicura in quanto genera la tensione corretta ed ha una protezione incorporata.

Figura 10.3: *Morsetti KNX*

In alternativa si può anche alimentarla tramite un alimentatore esterno, bisogna però assicurare un'uscita da quest'ultimo stabile e del giusto valore per evitare danni. Inoltre, utilizzando questo metodo, senza aggiungere un *choke* tra l'alimentazione e la scheda non è possibile inviare messaggi sul bus.

La scheda di sviluppo del microcontrollore scelto si può alimentare attraverso il pin 3.3V presente sullo *shield* della scheda di sviluppo del *transceiver*, eliminando quindi la necessità di aggiungere degli *Low-dropout regulator* (LDO). Il compito degli LDO è essenzialmente quello di regolare la tensione in uscita al valore corretto quando la tensione di alimentazione è molto vicina a quest'ultima.

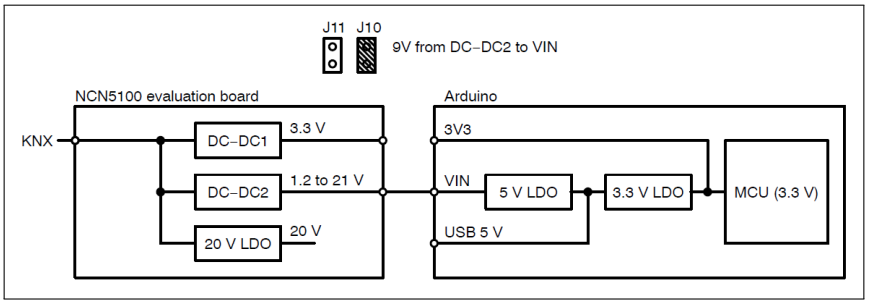

Figura 10.4: *Jumper* J11 da cortocircuitare [3]

In questo progetto si alimenta però il microcontrollore attraverso il pin VIN. Si opta per quest'opzione perché quando quest'ultimo deve gestire la ricezione dei dati e il *webserver*, richiede una quantità di corrente che non può essere fornita dall'alimentazione attraverso il pin 3.3V. Per evitare dunque un *crash* del sistema si cortocircuita il *jumper* J10, come mostrato in figura 10.4, portando una tensione di 9V sul pin VIN.

10.1.2 Comunicazione seriale

Come già accennato in precedenza nella sezione 10.1 il *transceiver NCN5130* implementa già il MAC *layer*. È dunque compito dell'integrato di gestire la codifica e decodifica dei messaggi, mandare *acknowledges* ecc... Pure i *timing* di basso livello sono gestiti dal ricetrasmettitore ed assicura anche che la prevenzione delle collisioni sia gestita in modo opportuno senza che l'*host* debba intervenire.

Per comunicare con il MAC *layer* si può optare per l'interfaccia UART o SPI. In questo progetto si sceglie di utilizzare la comunicazione UART in quanto a livello di cablaggio è più semplice rispetto al SPI. La selezione della seriale da utilizzare avviene tramite il pin MODE2, quando connesso a GND usando la resistenza R32, la comunicazione avviene tramite UART.

L'UART permette una comunicazione *full duplex* asincrona. Come visibile in figura 10.5 è possibile selezionare due modalità di trasmissione: a 8 o 9 *bit*. La differenza è che la seconda modalità trasmette anche un *bit* di parità addizionale (parità pari). Il nono *bit* viene anche utilizzato per indicare qualora ci fosse un errore della durata dell'impulso KNX. La selezione della modalità avviene tramite il pin UC2 ('0' = 9 *bit*, '1' = 8 *bit*).

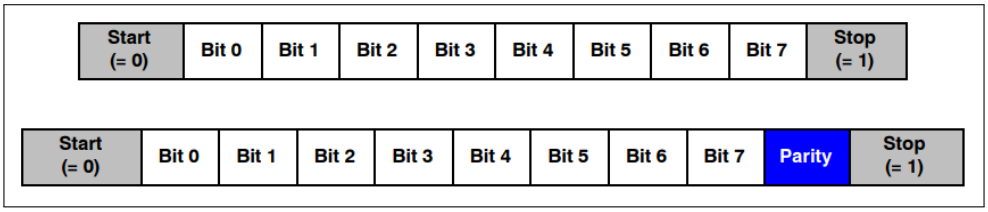

Figura 10.5: *Frame* delle due modalità UART [4]

Le possibili configurazioni dell'UART offerte dalla scheda sono le seguenti:

J2	J1	bits	bps
0	0	9	19200
0	1	9	38400
1	0	8	19200
1	1	8	38400

In questo caso tramite i *jumper* J1 e J2 si è scelto il livello logico '0' e dunque l'UART contiene il *bit* di parità e comunica ad una velocità di 19200 *bit per seconds* (bps).

10.1.3 *Bus monitor service*

Una volta impostato il *transceiver* in questa modalità, tutti i dati ricevuti dal KNX sono convogliati al *controller*. Non vi è quindi nessun indirizzamento da parte del *datalink*, che normalmente manda i dati solo al dispositivo specificato nel pacchetto. In questo modo si è in grado di effettuare uno *sniffing* del bus e leggere tutti i telegrammi indipendentemente dalla loro destinazione. Per questo motivo si dice che il dispositivo in in ascolto è "trasparente". L'unico modo per uscire da questa modalità è attraverso il servizio *reset*. In figura 10.6 si può vedere uno schema delle richieste fatte dall'*host* al NCN5130 e le relative risposte.

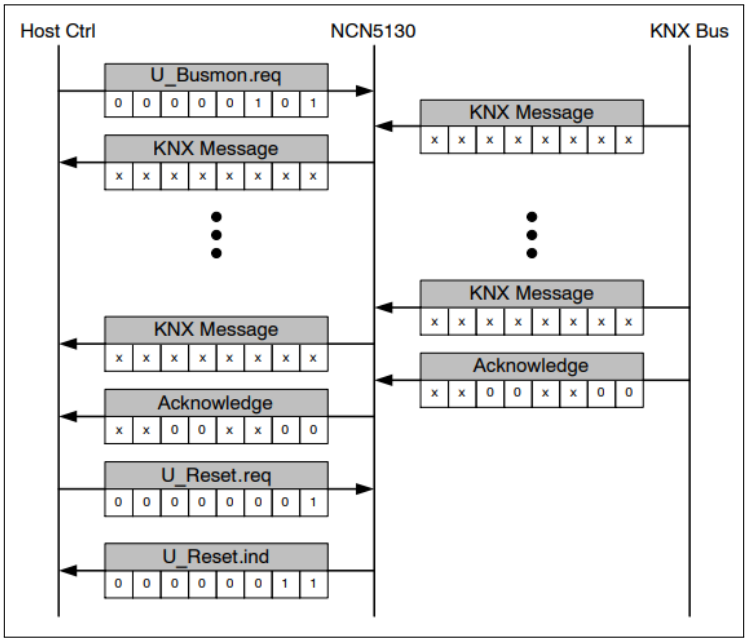

Figura 10.6: *Bus Monitor Service* [4]

Dal momento che il *transceiver* riceve la richesta *U_Busmon.req* inizia ad inoltrare via seriale al micro tutti i telegrammi da esso ricevuti. Ogni telegramma inviato all'*host* è in seguito da un Ack. Come accennato in precedenza una volta che si vuole terminare quest'attività occorre inoltrare una richiesta di reset (*U_Reset.req*). Se come risposta dal *transceiver* si riceve *U_Reset.ind* singifica che esso è uscito dalla modalità di *monitoring* ed è tornato nello stato iniziale dove attende compiti da eseguire.

10.1.4 *Send Frame Service*

Questo servizio è utilizzato per mandare dei telegrammi sul bus KNX. Permette dunque per esempio di impostare un certo valore ad un attuatore.

Per mandare un telegramma sul bus occorre procedere come segue: innanzitutto si invia al *transceiver* il comando *U_L_DataStart.req*, il quale inizializza la trasmissione di un nuovo telegramma sul bus. Dopodiché segue il *control byte* del telegramma. Da questo momento in poi ogni *byte* che si vuole inviare deve essere preceduto dal comando *U_L_DataCont.req*. Quest'ultimo permette di concatenare tutti i *byte* in modo da formare il telegramma KNX. Al suo interno è specificata la posizione occupata dal successivo *byte* contiguo all'interno del telegramma. Si procede in questo modo fino ad arrivare all'ultimo *byte* del *payload*. A questo punto si invia il comando *U_L_DataEnd.req* e subito dopo la *checksum* del telegramma. Ciò indica al *transceriver* che il telegramma da inviare è stato ultimato. Se la *checksum* inviata corrisponde con quella calcolata dall'integrato, il telegramma viene inviato sul bus KNX.

Siccome i comandi *U_L_DataStart/DataCont/DataEnd* forniscono indici di soli 6 *bit*, e un *frame* esteso può avere fino a 263 *byte* si necessitano dei comandi con dei *bit* aggiuntivi, in modo da poter per esempio esprimere la centesima posizione all'interno del telegramma. Tre *bit* aggiuntivi sono forniti dal comando *U_DataOffset.req*. Così facendo si è in grado di indicare la posizione di qualsiasi *byte* all'interno del telegramma e non si è limitati alla 64esima posizione come del caso dei comandi a 6 *bit*.

In figura 10.7 è rappresentato quanto descritto in precedenza, da notare che una volta ultimata

la trasmissione del telegramma viene inviato all'*host* il *byte U_FrameState.ind*. Esso contiene dei *flag* per fornire un *feedback* qualora la trasmissione non avvenisse nella maniera corretta. I *flag* contenuti sono:

- re (*recive error*): a '1' se il *frame* appena ricevuto contiene *byte* corrotti (parità sbagliata, *bit* di stop sbagliato o tempi di *bit* errati);
- ce (*checksum or length error*): a '1' se il *frame* appena ricevuto contiene una *checksum* sbagliata o una lunghezza che non corrisponde al numero di *byte* ricevuti;
- te (*timing error*): a '1' se il nuovo *frame* ricevuto contiene *byte* la cui temporizzazione non è conforme allo standard KNX;
- res (*reserved*): Riservato per uso futuro (sarà '0').

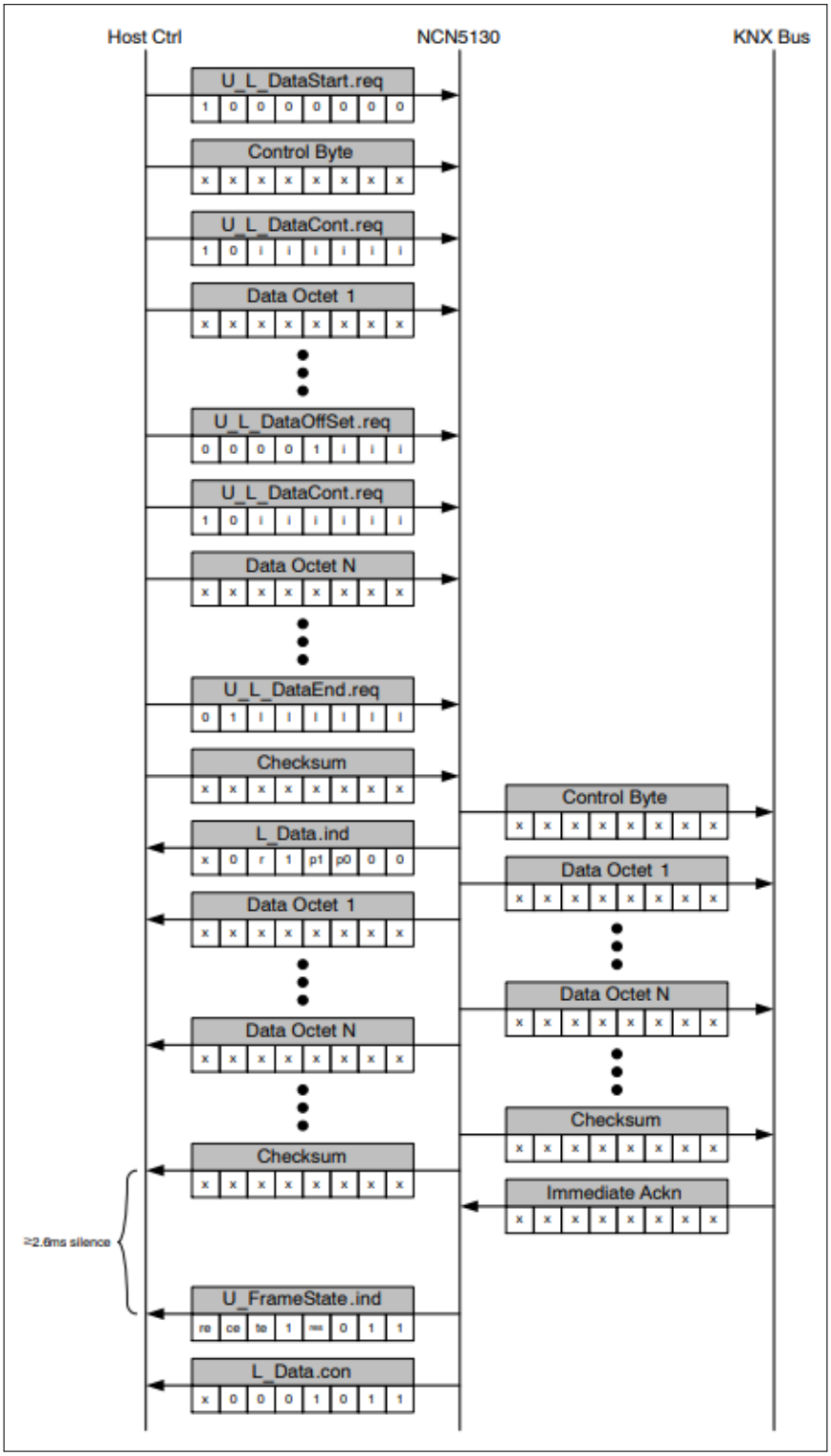

Figura 10.7: *Send Frame*, UART a 8 *bit*, *frame* terminato con *Silence*, no CRC o CCITT [4]

10.2 Microcontrollore ESP8266

Come già precisato nelle specifiche il microcontrollore utilizzato in questo progetto è un ESP8266. Quest'ultimo è montato su uno *shield* identico a quello dell'Arduino UNO e perciò ha il vantaggio di poterlo connettere direttamente al *transceiver*. Ha anche però delle caratteristiche migliori rispetto ad un semplice Arduino UNO, si hanno quindi i vantaggi a livello di *form factor* di uno e in termini di prestazioni dell'altro.

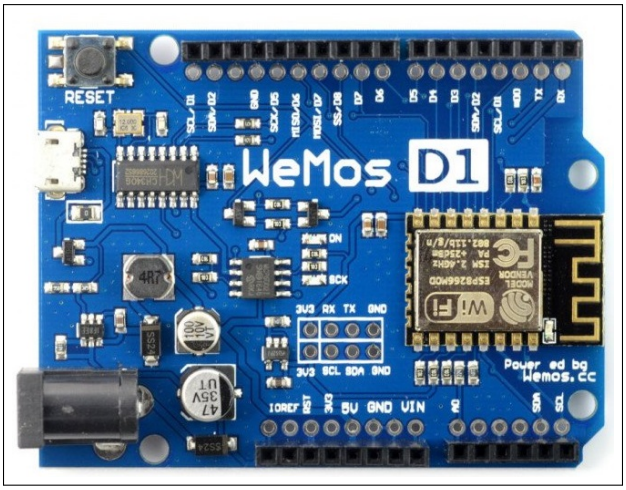

Figura 10.8: ESP8266 montato sullo *shield* dell'Arduino UNO

Le caratteristiche principali sono riportare nella tabella 10.1

Tabella 10.1: Caratteristiche principali ESP8266

Microcontroller	ESP8266 Tensilica 32-*bit*
Nominal voltage	3.3V
Input voltage	7-12V
Digital I/O Pins	11
DC Current per I/O Pin	12 mA MAX
Hardware Serial Ports	1
Flash Memory	4 MB
Instruction RAM	64 kB
Data RAM	96 kB

10.2.1 Problemi riscontrati

Nella primissima fase di *test*, con il *transceiver* montato sulla *board* del micro, si è subito riscontrato un problema: una volta comunicato il comando *U_Busmon.req* anziché ottenere come risposta il traffico presente sul bus KNX si ricevevano dei valori casuali privi di alcun significato.

Il problema sta nel fatto che il microcontrollore ha in parallelo alla seriale usata per comunicare col *transceiver* una linea utilizzata per comunicare con il monitor seriale del pc e per caricare i programmi in memoria. Su questa linea secondaria sono presenti delle resistenze che in

questo caso si sono rilevate troppe piccole. Essendo di un valore troppo basso e la corrente di *sink* fornita dal *transceiver* relativamente bassa, la tensione sulla linea resta sempre troppo alta, non rilevando di fatto mai lo stato '0'. In un primo momento si è sostituito la resistenza con una di un valore maggiore. In questo caso la ricezione dal bus KNX avveniva correttamente ma non era più possibile caricare nella memoria del micro il codice dal PC. Per ovviare a questa problematica si è interrotto la linea atta alla comunicazione con il PC con degli *switch*. Collegando la linea solo al momento del caricamento del *software* e scollegandola quando il sistema è collegato al bus KNX si risolvono entrambi i problemi inizialmente menzionati.

Un ulteriore problema lo si riscontra quando il *pin* di *reset* del micro e del *transceiver* sono collegati. Questo collegamento impedisce al micro di funzionare correttamente, per risolvere il problema si è pertanto evitato di mettere in contatto i due *pin*.

10.3 RTC

Sebbene inizialmente non previsto, l'aggiunta di un RTC si è rivelata necessaria nel corso del progetto. Siccome un obiettivo del progetto è generare un *log file* con tutti i telegrammi registrati, quest'ultimo deve poter fornire un indicazione temporale assoluta della ricezione. Si sono considerati vari metodi, compreso quello di connettersi ad un *server Network Time Protocol* (NTP). Purtroppo questa soluzione non è attuabile dal momento che il dispositivo non è connesso ad alcuna rete, ma funge esso stesso da *hotspot*. Per questo motivo si è optato per l'aggiunta di un componente *hardware* al prototipo. Si è integrato nel sistema un modulo DS3231 RTC, così da poter avere un indicazione sul tempo al quale un dato evento è avvenuto, rendendo di conseguenza il *log file* utilizzabile in un caso reale. Vista la situazione con delle tempistiche relativamente ristrette, quella appena esposta rappresenta la soluzione più rapida e concretamente applicabile.

10.4 *Case* prototipo

Nelle fasi finali del progetto si è sviluppato un *case* in PLC per contenere il prototipo. Con l'ausilio del *tool online* Tinkercad si è dapprima fatto un disegno 3D, inseguito lo si è stampato con una stampante 3D.

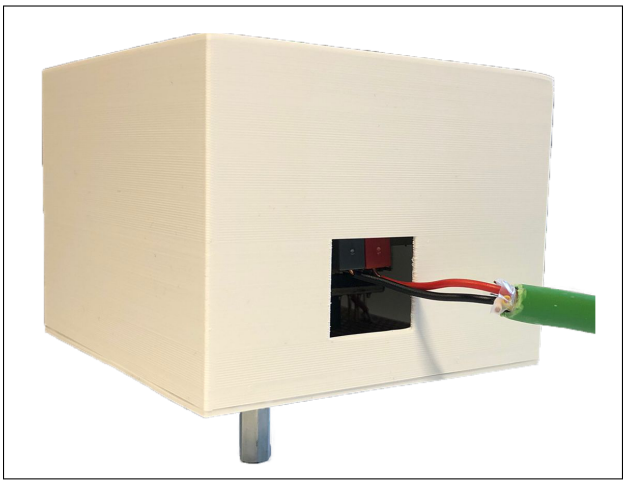

Figura 10.9: *Case* montato sul prototipo

In figura 10.9 è possibile vedere il prototipo con il *case* montato. Per poter collegare il dispositivo al bus KNX è stato creato un passaggio per il connettore.

11. *Test* del prototipo

Dopo aver dato una panoramica sul funzionamento del *software* e qualche dettaglio sulla struttura dell'*hardware*, è indispensabile testare e capire come si comporta il prototipo se confrontato con la diagnostica di un *software* dedicato alla programmazione di dispositivi KNX. Tale confronto mette in evidenza eventuali discrepanze tra i risultati, permettendo dunque di capire quanto è affidabile il dispositivo realizzato.

La comparazione in questione viene fatta con lo storico/log generato dal dispositivo realizzato in questo progetto e l'*easy controller software*. Questo programma è pensato per la programmazione di dispositivi GEWISS Chorus e al suo interno è possibile trovare le informazioni necessarie per capire se i dati forniti dal dispositivo sono concordi. Il paragone avviene per 4 casi diversi:

- funzione di *switch*;
- funzione di *dimming*;
- abbassamento delle tapparelle;
- riprogrammazione di un sensore per vedere i cambiamenti nel telegramma.

11.1 Funzione di *switch*

In questo primo caso si analizza come reagiscono i due sistemi all'accensione di una luce, ovvero ad una funzione di *switch*. Come si può vedere in figura 11.1 la pulsantiera a 4 canali con indirizzo fisico 0.2.2 invia un oggetto di commutazione con valore 1 all'indirizzo di gruppo 24/0/0. Come risposta l'attuatore con indirizzo fisico 0.2.3 manda una notifica di stato all'indirizzo di gruppo 24/0/9.

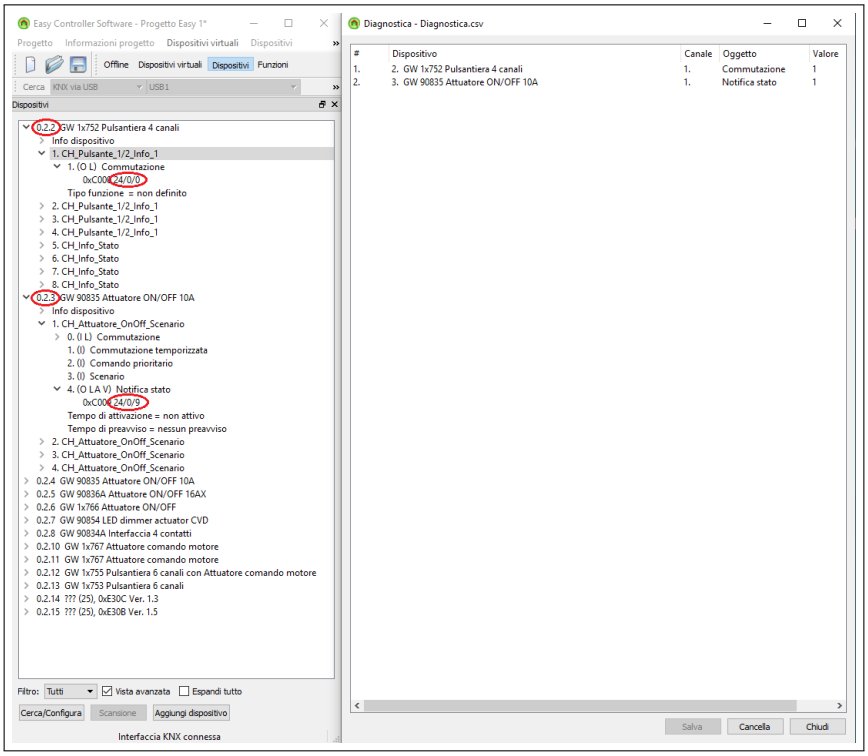

Figura 11.1: Diagnostica *easy controller software* - funzione di *switch*

In figura 11.2 è riportato quanto registrato invece dal prototipo nella stessa identica situazione. Si osserva che il primo telegramma ricevuto anche in questo caso proviene dalla pulsantiera a 4 canali con indirizzo fisico 0.2.2 che manda all'indirizzo di gruppo 24/0/0 un *payload* di 0081 hex. In un secondo istante si registra poi il telegramma proveniente dall'attuatore 0.2.3 che notifica lo stato all'indirizzo di gruppo 24/0/9. Anche in questo caso il *payload* è 0081 hex.

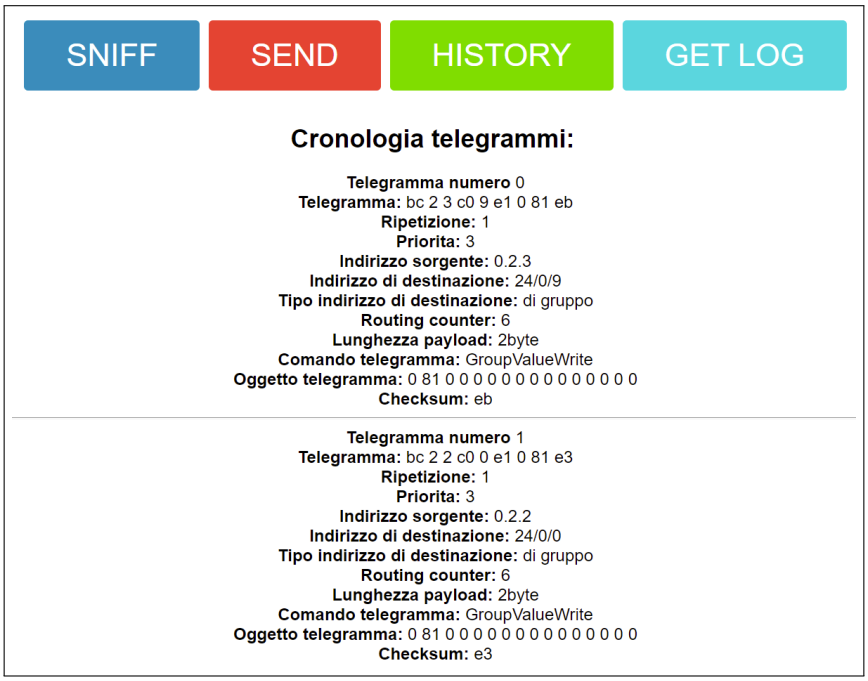

Figura 11.2: Storico prodotto dal dispositivo realizzato - funzione di *switch*

Da questo confronto si può dunque dedurre che i risultati forniti dai due sistemi sono equivalenti. Come ulteriore conferma nel risultato del prototipo è bene far notare che il *payload* 0081 hex è in linea con quanto riportato nella sezione 4.5.2.5 in figura 4.9: il comando è un *GroupValueWrite* e l'ultimo *bit* a '1' indica la commutazione allo stato 'on'.

11.2 Funzione di *dimming*

Il confronto tra i dispositivi è ora con una funzione di *dimming*. In figura 11.3 si osserva che dalla pulsantiera a 4 canali con indirizzo fisico 0.2.2 viene inviato all'indirizzo di gruppo 24/0/2 un oggetto di commutazione 'on'. Dopodiché l'attuatore *dimmer* con indirizzo fisico 0.2.7 invia una notifica di stato all'indirizzo di gruppo 24/0/35. Da ultimo lo stesso attuatore invia dei comandi per la regolazione della luminosità all'indirizzo di gruppo 24/0/68, il valore in questi tre comandi è di rispettivamente 01 hex, 00 hex, 09 hex, 08 hex.

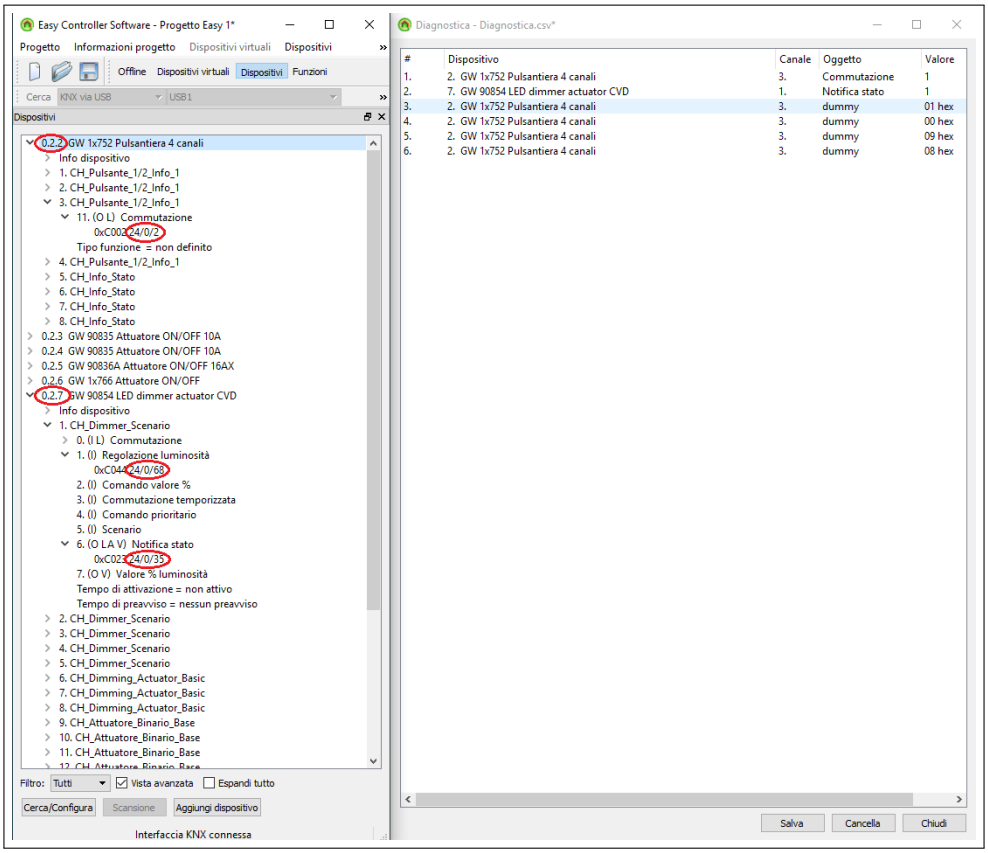

Figura 11.3: Diagnostica *easy controller software* - funzione di *dimming*

Analogamente, dai risultati forniti dal prototipo, si nota che in figura 11.6 il primo telegramma ricevuto (telegramma numero 5) ha un indirizzo sorgente pari a 0.2.2 e manda una *payload* di commutazione all'indirizzo di gruppo 24/0/2. Segue la notifica di stato dell'attuatore *dimmer* all'indirizzo di gruppo 24/0/35. I restanti 4 telegrammi presenti in figura 11.4 e figura 11.5, provenienti dall'indirizzo fisico 0.2.2, inviano all'indirizzo di gruppo 24/0/68 la regolazione della luminosità. I telegrammi in questione hanno dei *payload* pari a 0081 hex, 0080 hex, 0089 hex, 0088 hex.

Anche in questo caso nei risultati non vi sono discrepanze, inoltre si è sempre in accordo con quanto riportato nella sezione 4.5.2.5 in figura 4.10: si sono ricevuti vari telegrammi con *payload* diversi per poter impostare il livello di luminosità della lampada.

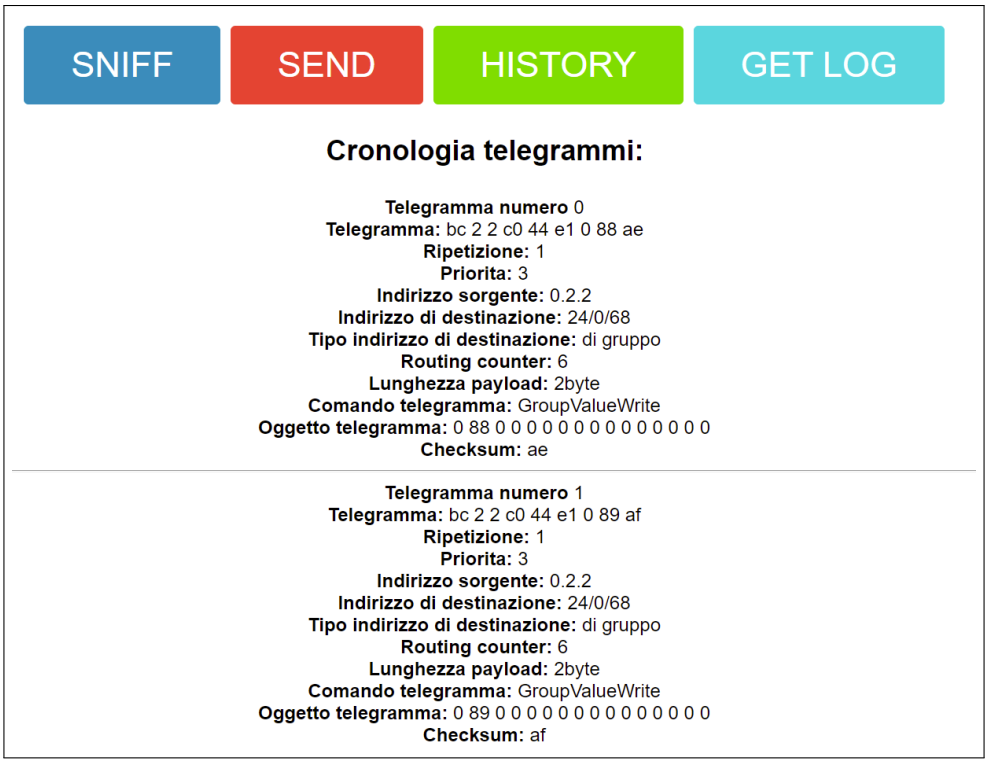

Figura 11.4: Storico prodotto dal dispositivo realizzato (1) - funzione di *dimming*

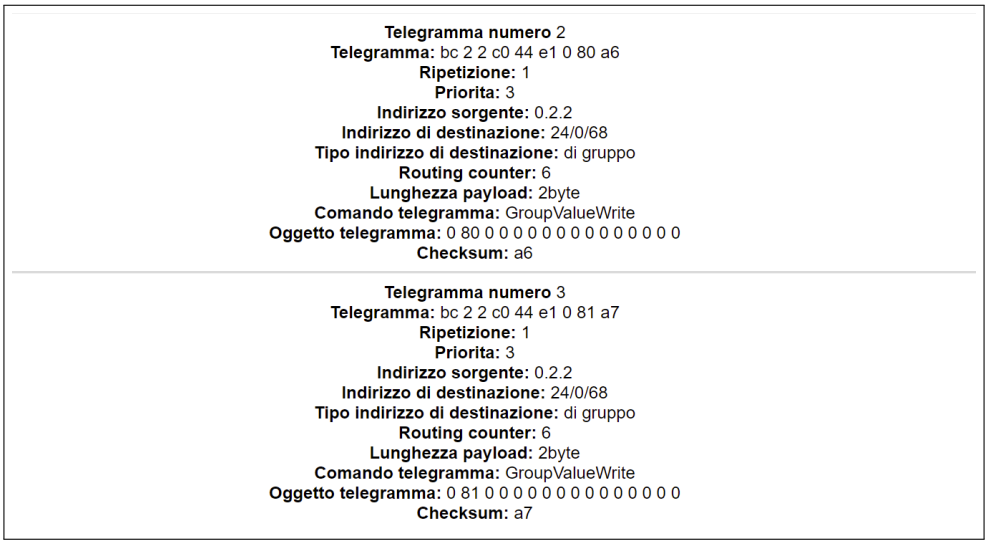

Figura 11.5: Storico prodotto dal dispositivo realizzato (2) - funzione di *dimming*

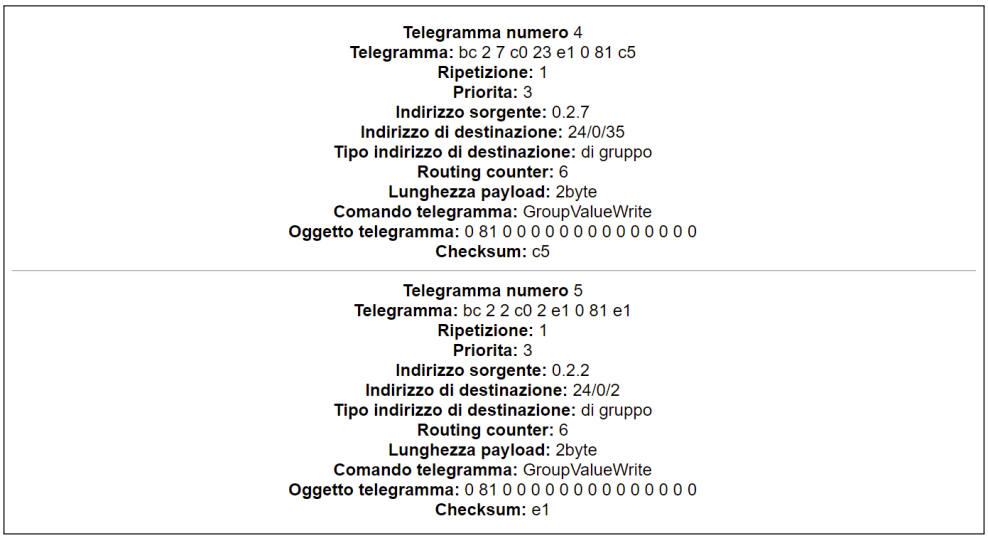

Figura 11.6: Storico prodotto dal dispositivo realizzato (3) - funzione di *dimming*

11.3 Abbassamento tapparelle

Nonostante l'abbassamento delle tapparelle utilizzi la stessa funzione di *switch* già analizzata, per completezza viene considerata nel confronto tra i due dispositivi. In figura 11.7 si vede che l'attuatore comando motore con indirizzo fisico 0.2.10 invia una notifica di movimento all'indirizzo di gruppo 24/0/93 al momento opportuno. Il valore trasmesso è '1', il che significa che le tapparelle sono abbassate.

La stessa cosa è riportata in figura 11.8: dall'indirizzo fisico 0.2.10 proviene un comando che come destinazione ha l'indirizzo di gruppo 24/0/93, il quale ha come *payload* 0081 hex, ovvero tapparelle abbassate.

Ancora una volta i risultati ottenuti sono concordi, da notare però che, a differenza dei casi precedenti, non essendo impostata nessuna notifica di stato, vi è un solo telegramma sul bus.

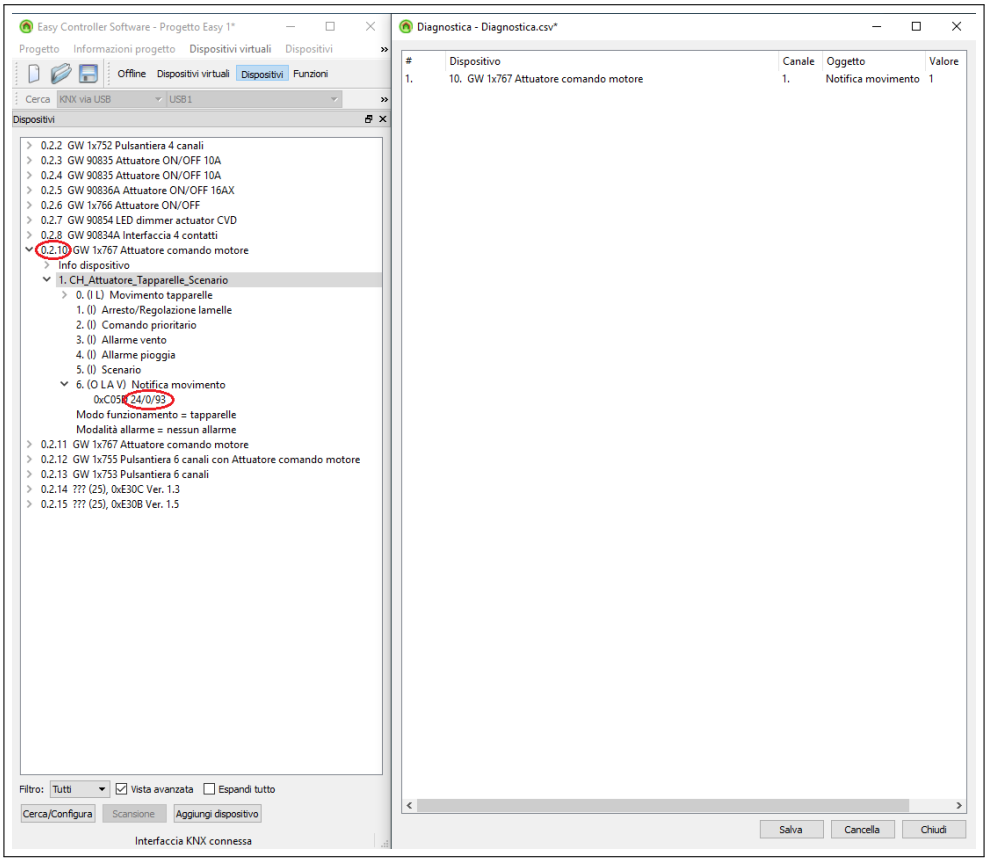

Figura 11.7: Diagnostica *easy controller software* - abbassamento tapparelle

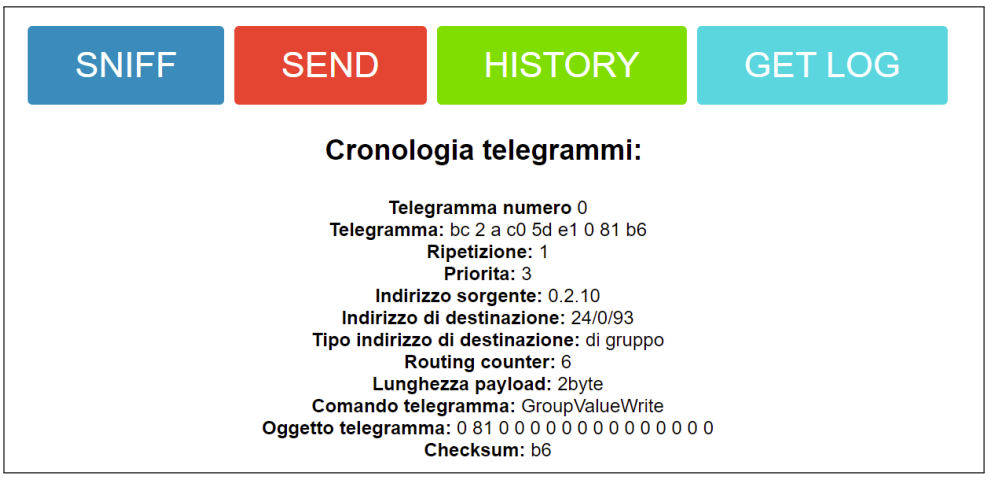

Figura 11.8: Storico prodotto dal dispositivo realizzato - abbassamento tapparelle

11.4 Cambiamenti in un telegramma dovuti alla riprogrammazione di un sensore

In questo ultimo sottocapitolo della fase di *test*, si mostra quello che succede quando un dispositivo viene riprogrammato. In figura 11.9 e in figura 11.10 è illustrato fondamentalmente quanto già visto nella sezione 11.1: al momento dell'invio del comando di accensione ad un attuatore, in questo caso si simula che sia collegato con il riscaldamento, sul bus viene mandato il telegramma contente il comando ed in seguito quello contenente la notifica di stato. I risultati forniti dalla diagnostica e dal prototipo sono gli stessi. Da notare che l'indirizzo fisico dell'attuatore è 0.2.5 e manda una notifica di stato all'indirizzo di gruppo 24/0/31. La pulsantiera ha invece l'indirizzo fisico 0.2.12 e manda la il comando di commutazione all'indirizzo 24/0/157.

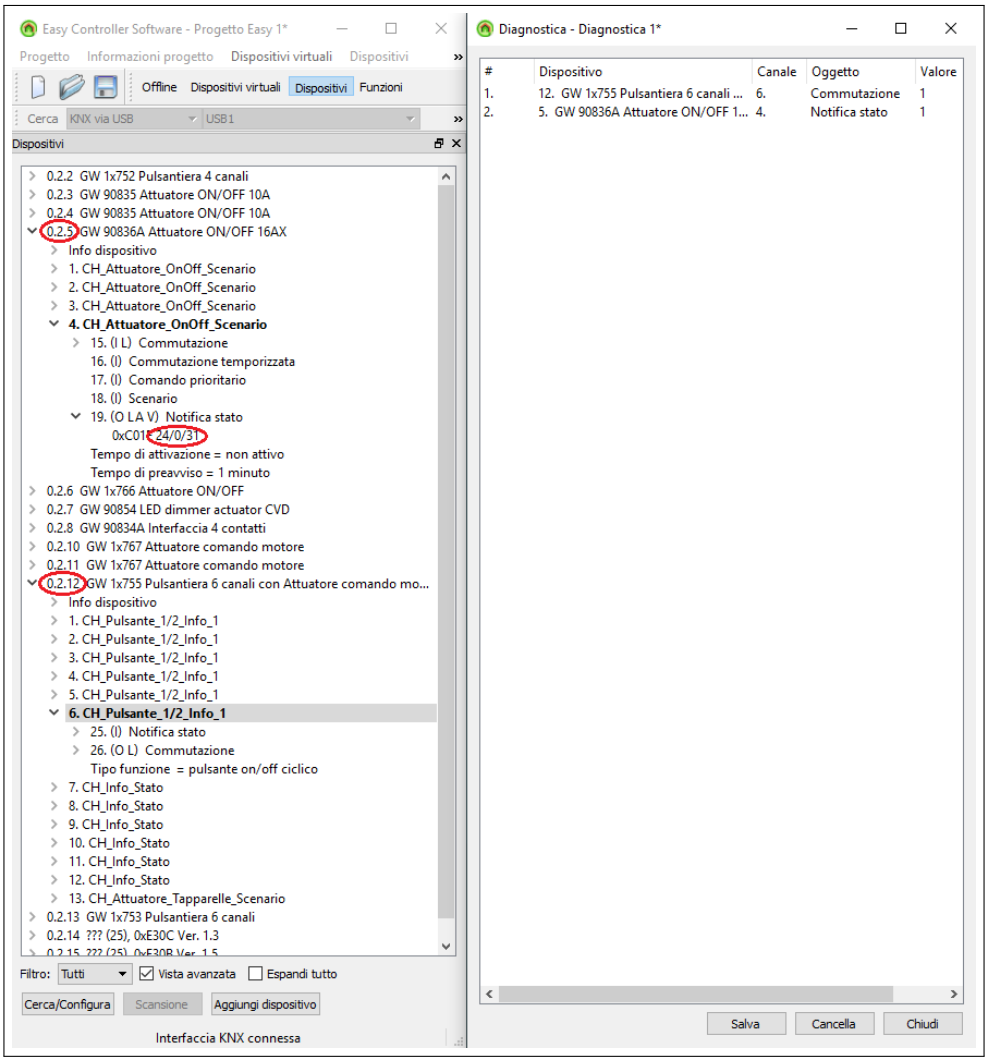

Figura 11.9: Diagnostica *easy controller software* - accensione riscaldamento

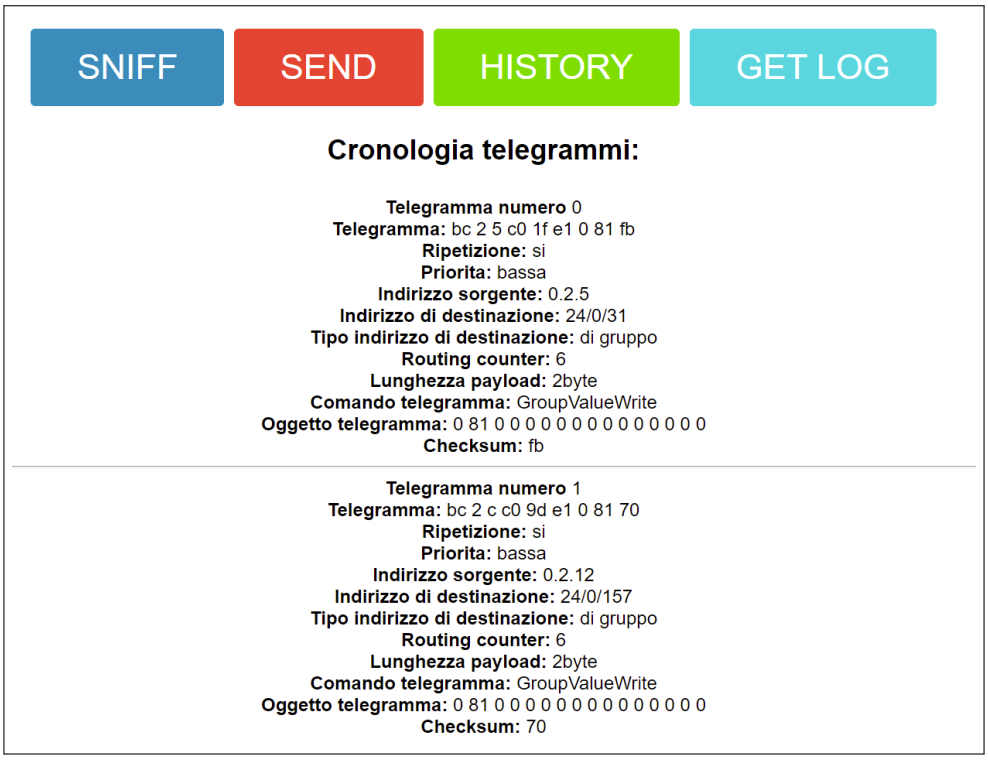

Figura 11.10: Storico prodotto dal dispositivo realizzato - accensione riscaldamento

A questo punto con l'*easy controller software* si riprogramma l'impianto domotico, e si fa in modo che, con la pressione dello stesso pulsante utilizzato in precedenza, anziché accendere il riscaldamento si accende l'aria condizionata.

```
Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:28
Telegramma: bc 11 f 2 5 65 3 e7 f 2 c0 9d 8b
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 1.1.15
Indirizzo destinazione: 0/2/5
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e7f2c09d0000000000
Checksum: 8b

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:29
Telegramma: bc 2 5 11 f 65 3 e6 f 11 c0 1e 1a
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 0.2.5
Indirizzo destinazione: 2/1/15
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e6f11c01e0000000000
Checksum: 1a

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:29
Telegramma: bc 11 f 2 5 65 3 e7 a 0 c0 9d 8c
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 1.1.15
Indirizzo destinazione: 0/2/5
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e7a0c09d0000000000
Checksum: 8c

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:30
Telegramma: bc 2 5 11 f 67 3 e6 a 11
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 0.2.5
Indirizzo destinazione: 2/1/15
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 8
Comando telegramma: Escape
Oggetto telegramma: 3e6a11c000000000000
Checksum: 0

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:30
Telegramma: bc 11 f 2 c 65 3 e7 19 2 c0 1f 16
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 1.1.15
Indirizzo destinazione: 0/2/12
Tipo indirizzo di destinazione: fisico
Routing counter: 6
```

```
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e7192c01f0000000000
Checksum: 16

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:31
Telegramma: bc 2 c 11 f 63 3 e6 19 1 cd
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 0.2.12
Indirizzo destinazione: 2/1/15
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 4
Comando telegramma: Escape
Oggetto telegramma: 3e6191c01f0000000000
Checksum: cd

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:31
Telegramma: bc 11 f 2 c 65 3 e7 19 0 c0 1d 16
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 1.1.15
Indirizzo destinazione: 0/2/12
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e7190c01d0000000000
Checksum: 16

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

Ricezione telegramma avvenuta il 16.8.2021 alle 15:25:32
Telegramma: bc 2 c 11 f 65 3 e6 19 11 c0 1d 6
Ripetizione: si
Priorita': bassa
Indirizzo sorgente: 0.2.12
Indirizzo destinazione: 2/1/15
Tipo indirizzo di destinazione: fisico
Routing counter: 6
Lunghezza payload: 6
Comando telegramma: Escape
Oggetto telegramma: 3e61911c01d0000000000
Checksum: 6

-----------------------------------------------------------------
```

Listato 11.1: Storico prodotto dal dispositivo realizzato durante la riprogrammazione di un dispositivo

Il listato 11.1 mostra i telegrammi presenti sul bus KNX registrati dal prototipo in modalità *sniffing* nel momento in cui avviene la riprogrammazione di parte dell'impianto domotico. Purtroppo in questo caso non si ha mezzo di capire se quanto registrato è corretto. Ovviamente nel momento in cui il *software* sta riprogrammando un dispositivo la diagnostica non è in funzione. Ciò che si può dire è però che in quattro dei telegrammi ricevuti, l'indirizzo sorgente è di uno dei due dispositivi che effettivamente vengono utilizzati, ovvero 2.1.15 e 0.2.12. La ricezione dei telegrammi con un indirizzo fisico noto è intercorsa da un telegramma il quale indirizzo fisico è sempre 1.1.15. In oltre l'indirizzo di destinazione di quest'ultimo è di tipo fisico e combacia con l'indirizzo fisico 2.1.15 in un caso e 0.2.12 nell'altro. Appare dunque chiaro che si è effettivamente registrato un'interazione tra i dispositivi che si desidera utilizzare e il *software* di programmazione, resta ancora da capire esattamente i dati e le informazioni che

sono state scambiate. In un possibile sviluppo futuro sarebbe interessante approfondire la questione per capire se quanto registrato sia effettivamente giusto.

Terminata la fase di riprogrammazione si è controllato nuovamente la diagnostica e lo storico generato dal prototipo. Come illustrato in figura 11.11 e in figura 11.12 avendo cambiato il dispositivo che si va a comandare con la pressione dello stesso pulsante, l'indirizzo fisico di quest'ultimo (0.2.12) è rimasto invariato, cambia invece quello di gruppo dell'attuatore. In questo caso è diventato 24/0/29 e quello fisico, trattandosi sempre dello stesso attuatore, è rimasto invariato (0.2.5).

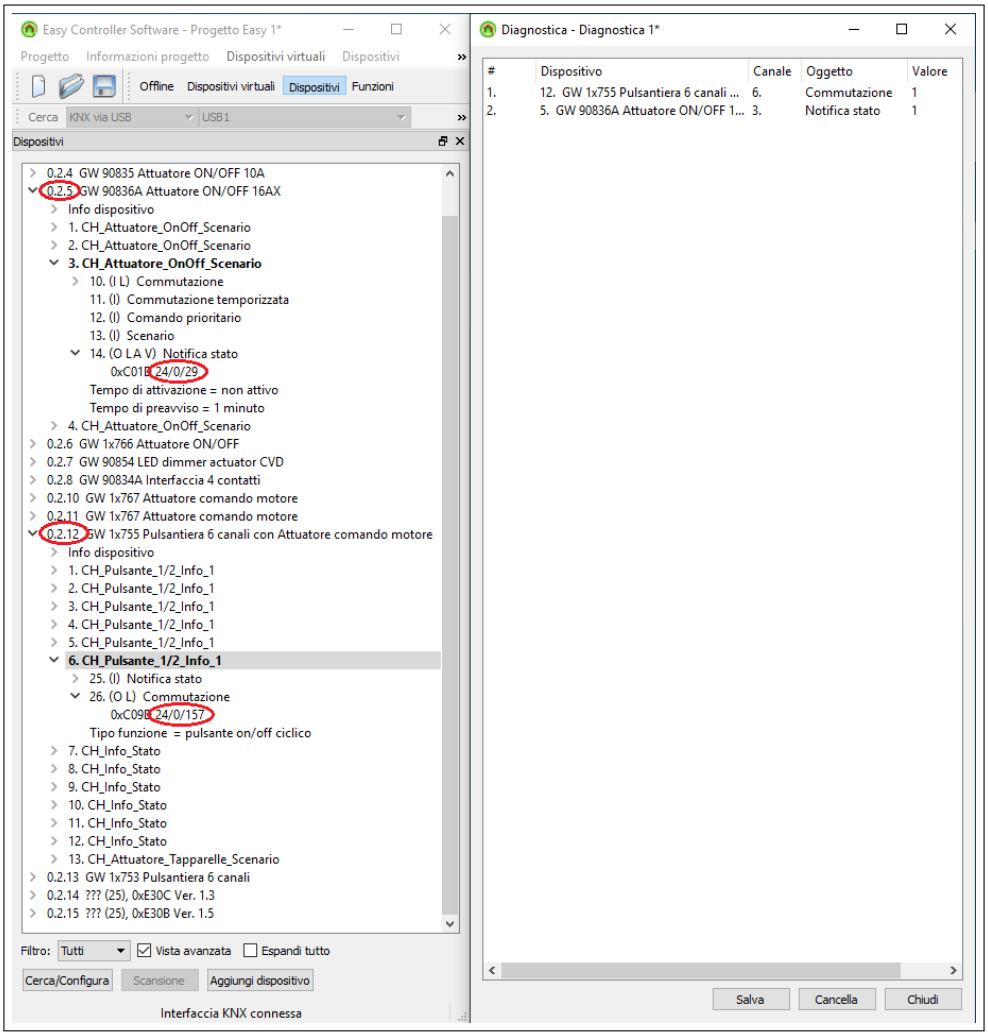

Figura 11.11: Diagnostica *easy controller software* - accensione aria condizionata

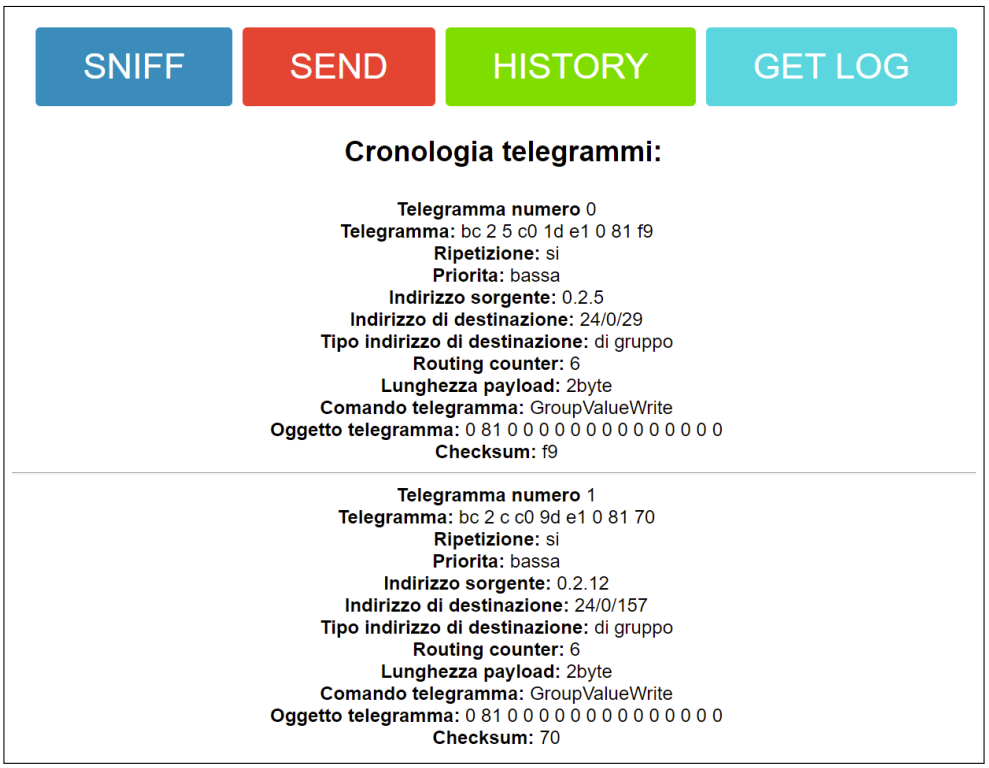

Figura 11.12: Storico prodotto dal dispositivo realizzato - accensione aria condizionata

12. Limiti del progetto

Dopo aver esposto i risultati della fase di *test* del progetto, è bene specificare quali limiti sussistono. Come si è potuto stabilire nel precedente capitolo, il prototipo si è comportato come auspicabile nei quattro casi. Nonostante ciò, non si è in possesso di una casistica sufficientemente ampia per poter stabilire l'affidabilità del dispositivo. Ciò che è certo, è che il prototipo non è in grado di decodificare completamente il significato dei telegrammi. Quanto appena detto, è dato dal fatto che all'interno del telegramma non è contenuta nessun informazione circa il tipo di funzione che il *payload* rappresenta. Ovvero, non si è in grado di stabilire dal telegramma, se per esempio il *payload* va interpretato come quello di una funzione di *switch*, e che quindi solo l'ultimo *bit* è d'interesse; oppure se è una funzione di *dimming*, e quindi i *bit* da considerare sono gli ultimi quattro. Per effettuare questo *step* successivo, serve una descrizione completa dei dispositivi. Attualmente è possibile decodificare completamente tutte le informazioni di un telegramma salvo il *payload*, quest'ultimo viene comunque registrato e salvato sotto forma di dati "grezzi".

Un altro fattore limitante del prototipo riguarda l'invio dei messaggi. Attualmente si è in grado di mandare unicamente messaggi il cui *payload* è di 2 *byte*. Si possono quindi mandare solo telegrammi semplici e non quelli più complessi. Sicuramente uno spunto futuro del progetto è quello di effettuare un *degbug* maggiormente completo dell'invio dei messaggi. Allo stato attuale è infatti ancora presente un *bug* che saltuariamente impedisce il salvataggio corretto del messaggio. Altro limite legato alla parte di invio di telegrammi, è il fatto che il prototipo non ha un proprio indirizzo fisico. Come già spiegato in precedenza ciò potrebbe causare dei conflitti sulla linea, impedendo l'invio del telegramma.

Da ultimo, attualmente non si è in grado di capire se quanto registrato in fase di riprogrammazione di un dispositivo sia effettivamente corretto. In un'implementazione futura sarebbe interessante poter verificare la questione.

Inoltre anche la parte relativa alla contestualizzazione potrebbe essere affinata ulteriormente, apportando un maggior grado di approfondimento. Quanto esposto è infatti unicamente la punta dell'*iceberg* ed è una tematica in continuo mutamento.

13. Conclusioni

La possibilità di monitorare un bus KNX, l'invio di messaggi sullo stesso, e la possibilità di scaricare un *file* con tutti i telegrammi rilevati sono tra le finalità principali del progetto.

Complessivamente si può dire che gli obiettivi concordati con il relatore Ragazzini sono stati raggiunti e che anche le tempistiche sono state rispettate. Il prototipo realizzato è in grado di monitorare il traffico dati sul bus KNX, generare uno storico con gli ultimi dieci telegrammi registrati, e inoltre è in grado di generare un *file* di testo scaricabile sul proprio PC; quest'ultimo contiene tutti i telegrammi registrati. Oltre agli obiettivi raggiunti appena menzionati, vi è anche la possibilità di salvare l'ultimo telegramma ricevuto con un nominativo a scelta; per poi successivamente inoltrarlo sul bus facendo riferimento al nominativo, che permette una classificazione dei comandi salvati semplificata. Inoltre i costi scaturiti dal progetto restano nei limiti, infatti il totale delle spese ammonta a circa CHF 155.00 e non si è superato il *budget* di CHF 300.00 a disposizione.

Naturalmente trattandosi di un primo prototipo c'è ancora margine di miglioramento. In questo capitolo tra i tanti sviluppi futuri possibili, vengono elencati solo i principali. In primis, vista l'applicazione finale che si è pensato, è plausibile progettare un PCB. Così facendo si potrebbe effettivamente collegare a lungo termine questo dispositivo ad un bus KNX. In secondo luogo, un altro aspetto da migliorare è quello di rendere il prototipo un nodo KNX, ovvero assegnargli un indirizzo fisico. Ciò escluderebbe il rischio di eventuali conflitti sul bus nel momento in cui si manda un telegramma che in precedenza si è salvato. Un'altra possibile miglioria da apportare riguarda il *log file* generato: invece di generare un *file* .txt, si potrebbe generare un *Comma-separated values* (.csv). Così facendo si permetterebbe all'utente di caricare direttamente il *file* ottenuto dal dispositivo in un programma (per esempio Excel), consentendo dunque anche un'analisi dei dati ottenuti più efficace. Da ultimo, si potrebbe aggiungere una scheda SD dove poter salvare con un intervallo di tempo prestabilito i vari log parziali. Ciò eviterebbe la perdita dei dati in caso di guasto e di utilizzare tutta la memoria disponibile all'interno dello SPIFFS.

Come verifica del funzionamento del dispositivo, si è effettuato un confronto tra i risultati forniti da quest'ultimo e dalla diagnostica dell'*easy controller software*. Ciò che si è ottenuto è soddisfacente in quanto in tutti i *test* svolti hanno dato un esito concorde tra entrambi i dispositivi.

Bibliografia

[1] IEA, "*Data and statistics - Energy consumption*," 2020.

[2] G. L. Ragazzini, "Corso di efficienza energetica, SUPSI," 10.06.2021.

[3] O. Semiconductor, "*Datasheet NCN5100 Arduino Shield Evaluation Board*," Ottobre 2020 - Rev. 2.

[4] O. Semiconductor, "*Datasheet* ncn5130 - *Transceiver for KNX Twisted Pair Networks*," Agosto 2019 - Rev. 4.

[5] C. Guzzonatto, "Ecologia co2: nel 2021 raggiungerà livelli (quasi) mai visti," 22.01.2021.

[6] A. M. Angelo Baggini, "Efficacia energetica negli edifici," Febbraio 2010.