UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sviluppo di un nodo embedded per la connessione a database SDS

Development of an embedded node for connecting to SDS database

Candidato:
Leonardo Pieralisi

Relatore:
Prof. Luca Spalazzi

Correlatore:
Ing. Fernando Negozi

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Sviluppo di un nodo embedded per la connessione a database SDS

Development of an embedded node for connecting to SDS database

Candidato:
Leonardo Pieralisi

Relatore:
Prof. Luca Spalazzi

Correlatore:
Ing. Fernando Negozi

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Al mio caro nonno Gino
Grazie di tutto

Ringraziamenti

Vorrei spendere delle parole per ringraziare tutti coloro che in un modo o nell'altro mi hanno accompagnato durante questo importante percorso. Vorrei ringraziare in primis il mio relatore, il Prof. Luca Spalazzi per avermi aiutato durante la stesura della tesi, per la sua pazienza e professionalità. Ringrazio veramente di cuore il mio correlatore nonchè tutor aziendale l'Ing. Fernando Negozi per essere stato la figura di riferimento durante il tirocinio, al quale devo molto per ciò che mi ha trasmesso e insegnato. Al Prof. Massimiliano Pirani e a tutto lo staff dell'azienda Eletica un ringraziamento sincero, mi avete subito accolto a braccia aperte.

Ringrazio ora tutta la mia famiglia, sono un ragazzo molto fortunato, a loro devo tutto, le parole non bastano mai in questi casi, sappiate solo che non avrei potuto desiderare di meglio, vi voglio bene.

A tutti i miei cari amici, ragazzi siete i migliori, siate sempre così, non ci sono parole per descrivere tutto quello che avete fatto per me, ne abbiamo passate tantissime insieme, grazie per esserci sempre stati, io ci sarò sempre per voi.

Un ringraziamento veramente speciale va alla mia ragazza, è stata sempre al mio fianco sin dal primo giorno, riuscendo a farmi tirare fuori ogni volta il meglio, ho condiviso con lei ogni gioia e ogni dolore di questo percorso. Sei una ragazza d'oro, e mi fai stare bene, grazie infinite.

Per ultimo vorrei ringraziare me stesso, ho fatto cose di cui non credevo fossi capace. Se il me del passato mi vedesse oggi direbbe: "Wow, vorrei essere proprio come lui".

Ancona, Ottobre 2024

Leonardo Pieralisi

Sommario

La seguente tesi si colloca nello scenario di un tirocinio svolto in azienda, dove è stato seguito lo sviluppo del progetto ENOUGH. In particolare uno dei tanti sottoprogetti che riguarda lo sviluppo di un nodo embedded per la connessione tra due infrastrutture. L'obiettivo quindi è stato quello di familiarizzare con le tecnologie utilizzate, e dare il proprio contributo per andare avanti con lo sviluppo del sottoprogetto del quale l'azienda è leader.

Indice

Elenco delle figure

Capitolo 1

Ambito e contesto del progetto

Introduzione

Nell'ultimo decennio la tecnologia sta continuando a fare passi da gigante, infatti sono sempre più numerosi gli ambiti in cui sono necessarie soluzioni intelligenti per migliorare affidabilità, efficienza e prestazioni. Molti dei dispositivi che stanno rendendo tale questa trasformazione sono i cosiddetti sistemi embedded, ed è proprio su quest'ultimi che il progetto è incentrato. Questa tesi racchiude in se l'esperienza di tirocinio che è stata svolta all'interno dell'azienda Eletica, la quale era già al lavoro su tale progetto.

La struttura della tesi è articolata nel modo seguente: una panoramica dell'azienda Eletica e del progetto che stava seguendo, gli obiettivi da raggiungere, le tecnologie utilizzate, l'architettura, e infine lo sviluppo e quindi il raggiungimento degli obiettivi.

In questo capitolo si farà una panoramica della realtà in cui il progetto è inserito, andando a descrivere anche quali sono le motivazioni e l'importanza di quest'ultimo.

1.1 L'azienda Eletica s.r.l.

Figura 1.1: Logo dell'azienda Eletica s.r.l.

Eletica s.r.l. si presenta come una realtà capace di combinare diverse competenze ingegneristiche dell'elettronica all'informazione. E' un'azienda dinamica e in costante crescita, attiva in numerosi mercati con una vasta gamma di servizi e competenze che la rendono un partner strategico per molte realtà industriali. La società è specializzata nella progettazione hardware e software, offrendo soluzioni personalizzate, capaci

di soddisfare le esigenze più specifiche dei clienti. Oltre alla progettazione, Eletica si occupa anche della prototipazione e della produzione, garantendo un supporto completo durante tutte le fasi di sviluppo del prodotto, fino all'analisi dettagliata per il miglioramento delle performance e dell'affidabilità.

Uno dei punti di forza di Eletica è la sua solida esperienza nel settore embedded, dove si è affermata come un'azienda leader nel territorio. Con queste competenze, integrando le conoscenze hardware e software, riesce a ripondere alle esigenze di numerosi settori che mirano al progesso tecnologico. La sua sede operativa è situata a Castelplanio, in provincia di Ancona, l'ambiente e il contesto in cui si lavora è molto piacevole e stimolante. La sede legale è situata a Fabriano, sempre nella provincia di Ancona, conclude le sedi il distaccamento in Albania.

Eletica è attivamente coinvolta in una serie di progetti innovativi, che spaziano dall'industria 4.0 alla sostenibilità ambientale. L'attenzione all'innovazione è un pilastro fondamentale per Eletica, che investe continuamente in ricerca e sviluppo, puntando a offrire così soluzioni smart e tecnologicamente all'avanguardia.

1.2 Motivazioni del lavoro, il progetto Enough

Uno dei lavori di cui l'azienda è partner e sul quale la trattazione incentrata in parte è il Progetto Enough.

1.2.1 Cos'è Enough?

Il progetto ENOUGH[3] nasce con lo scopo di ridurre le emissioni del settore alimentare. I sistemi alimentari sono responsabili, a livello globale, di una percentuale compresa tra il 20 e il 40% delle emissioni totali di gas serra.

Figura 1.2: Logo del progetto Enough.

Lo scopo principale è di supportare la strategia sostenibile dell'UE dall'agricoltura alla tavola, fornendo strumenti e soluzioni tecniche, finanziarie e politiche per ridurre le emissioni di gas a effetto serra (entro il 2030) e raggiungere la neutralità del carbonio (entro il 2050) nell'industria alimentare. Verranno quindi stabilite nuove

pratiche per ogni anello della catena alimentare dalla raccolta al consumo (lavorazione, confezionamento, stoccaggio, trasporto ecc.) volte a limitare queste emissioni di GHG (acronimo di Greenhouse Gases, ossia gas a effetto serra). Per quanto detto fin'ora, Enough è un progetto davvero ambizioso il quale punta al miglioramento di numerose aree nella filiera alimentare, e per questo motivo sono numerosi anche i partner che si occupano della progettazione e sviluppo.

Considerando la grandezza del progetto con tutte le sue specifiche nei vari contesti, è necessario suddividere i campi dello sviluppo anche in base alle tecnologie utilizzate. Infatti una macro categoria è quella relativa ai cosiddetti nodi.

Sono presenti tre tipi di nodi, in base a ciò che utilizzano si dividono in:

- Nodo 1: dove il lavoro viene svolto principalmente da normali computer ma in particolare da dei programmi scritti in Python.
- Nodo 2: dove il lavoro viene svolto principalmente da normali computer ma in particolare da dei programmi scritti in C.
- Nodo 3: dove il lavoro viene svolto principalmente dai sistemi embedded.

Detto ciò un team di sviluppo che vuole inserirsi con un progetto, potrà farlo, tenendo conto delle interfacce consentite dalla tipologia dei nodi. Quindi prima di cominciare il lavoro si dovrà fare riferimento ad un determinato nodo, e in base a quello cominciare lo sviluppo.

1.3 L'infrastruttura

Per come è stato concepito, il progetto è sicuramente di grande valore, che promuove la cooperazione e lo scambio. Infatti l'intera infrastruttura è stata pensata come una grande rete connessa ad un punto comune, il cosiddetto Smart Data System. Quest'ultimo ha come obiettivo l'andare oltre alla classica memorizzazione dei dati, infatti è un sistema intelligente e innovativo in grado di trasformare i dati in informazioni utili e "azionabili".

L'obiettivo è quello di dare l'accesso alla rete secondo criteri specifici, così che poi i cosiddetti "impianti" saranno connessi tra loro e potranno interagire per avvalersi di informazioni utili che il sistema SDS mette loro a disposizione, secondo un protocollo "Publisher-Subscriber". Questi dati poi non saranno disponibili a tutti e per tutti, ma anche in questo caso saranno presenti delle regole e delle politiche di scambio da rispettare.

1.3.1 Nodo di tipo 3

L'azienda Eletica s.r.l. in quanto partner partecipa allo sviluppo di un "nodo di tipo 3", il quale è responsabile del trasferimento dei dati provenienti da un innovativo camion refrigerante, che sfrutta l'anidride carbonica come gas, (un'altro progetto all'interno di Enough) al database SDS presente all'interno dell'Università Politecnica delle Marche, anch'essa partner del progetto.

Figura 1.3: Schematizzazione del nodo 3.

1.4 Obiettivi da raggiungere

Essendo questo un progetto sul quale Eletica stava già lavorando, l'esperienza di tirocinio comincia con il capire cosa era stato fatto fino al quel momento. Quindi il primo obiettivo è stato familiarizzare con le tecnologie in uso, andando a studiare le basi su cui il progetto si fonda. Una volta comprese le basi, e avendo quindi una visione dell'insieme, si è proceduto con dei passi, i quali sono stati:

- Capire come funziona il microcontrollore specifico del progetto
- Comunicazione tramite il protocollo RS485
- Implementazione del file system sul micro
- Implementazione del database sul micro
- Comunicazione con il modulo GSM
- Implementazione del driver per il modulo GSM

L'importante alla fine di questi obiettivi è arrivare alla conoscenza e la padronanza della tecnologia e soprattuto riuscire ad avere un sottoprogetto di esempio funzionante. Tutto questo poi sarà utile nello sviluppo futuro per riuscire così a integrare tutte queste parti nel progetto finale.

Capitolo 2

Tecnologie utilizzate

Nel campo dell'informatica e in particolar modo nella maggioranza dei sistemi informatici come quello in oggetto di questa trattazione, è possibile distinguere una parte fisica chiamata hardware e una parte un pò più astratta detta software. In questo capitolo si andranno a descrivere tutti gli strumenti e le tecnologie hardware e software che sono state utilizzate all'interno del progetto.

2.1 Hardware

Come accennato poco fa per hardware si intende tutto ciò che c'è di fisico in un computer o dispositivo elettronico. Nel nostro caso siamo di fronte ad un micro-controllore.

2.1.1 Micro-controllore

Un micro-controllore altro non è che una sorta di piccolo computer integrato su un unico componente, utilizzato nello sviluppo di schede elettroniche intelligenti. Quindi per come è definito un computer anche il micro-controllore avrà:

- CPU che è diciamo il cervello, esegue le operazioni aritmetico-logiche
- memoria RAM, un tipo di memoria temporanea che conserva i dati solo quando il microcontrollore è acceso
- memoria flash, un tipo di memoria non volatile, cioè che conserva i dati anche dopo lo spegnimento
- periferiche per l'input e output fondamentali per permettere al microcontrollore di interagire con l'esterno

Figura 2.1: Raffigurazione di un generico microcontrollore.

Il modello del nostro micro è STM32L4R9ZIJ6U montato su una scheda che prende il nome di STEVAL-STWINKT1B utile quando si ha bisogno di bassi consumi come nel caso del progetto .

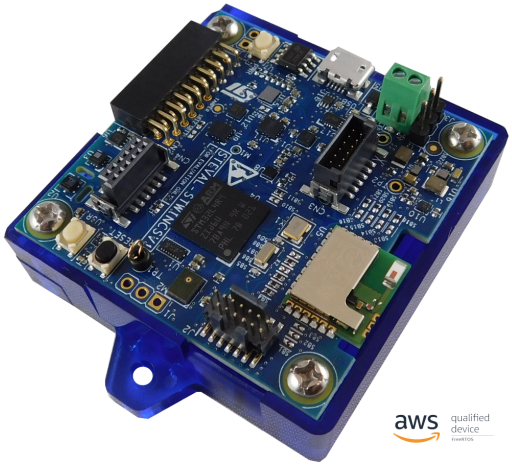

Figura 2.2: Scheda STEVAL-STWINKT1B che monta il micro STM32L4R9ZIJ6U.

Sulla scheda elettronica poi come si accennava, sono presenti numerosissimi componenti come i sensori, moduli di comunicazione, led, elementi per la memoria, ecc.

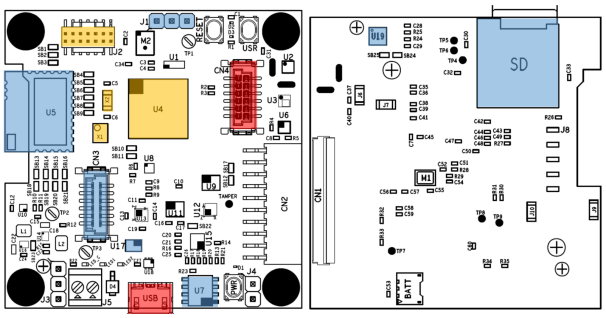

Figura 2.3: Schematizzazione della scheda con MCU e componenti per la connettività in evidenza.

2.1.2 Debugger STLINK-V3MINIE

Un componente indispensabile per lo sviluppo di questo tipo di progetti è il cosiddetto debugger. Viene utilizzato per programmare il microcontrollore, ovvero caricare il codice scritto dallo sviluppatore all'interno della memoria, così che poi possa eseguirlo autonomamente una volta fornita l'alimentazione. Senza questo dispositivo sarebbe impossibile espletare la vera funzione dei microcontrollori, ovvero avere una sorta di computer in miniatura per rendere "intelligenti" dei dispositivi facendo eseguire loro tutte funzioni che altrimenti non potrebbero. Nel progetto è stato utilizzato il modello STLINK-V3MINIE prodotto sempre dalla stessa azienda del microcontrollore, STMicroelectronics.

Figura 2.4: Debugger STLINK-V3MINIE.

2.1.3 Modulo GSM

Un componente fondamentale di tutto il sistema è il cosiddetto modulo GSM. La sigla GSM è l'acronimo di Global System for Mobile Communications che rappresenta uno standard aperto di seconda generazione (2G) per la telefonia mobile. Questo

dispositivo, una volta connesso fisicamente con la scheda elettronica e adeguatamente configurato (verrà spiegato tutto nel dettaglio nel capitolo 5.5), permette di connettersi alla rete cellulare e quindi di sfruttarne tutte le funzionalità. Per esempio mandare messaggi, effettuare chiamate o anche trasferire dati tramite internet. Inoltre è dotato anche di geolocalizzazione.

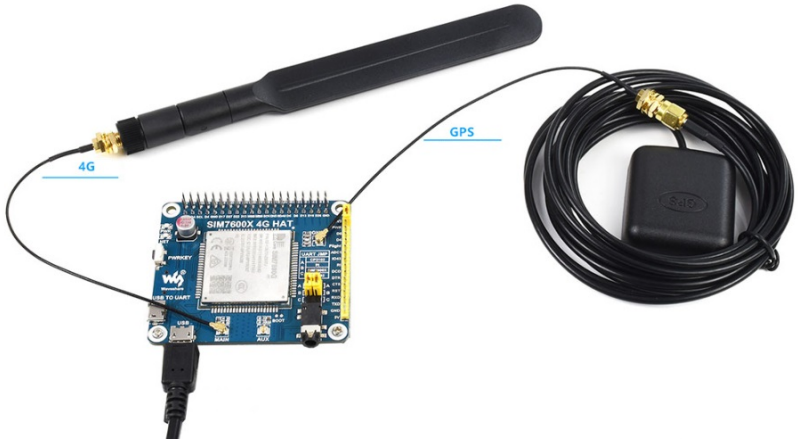

Figura 2.5: Modulo SIM7600X 4G HAT completo delle antenne.

Da come si può leggere nella figura, il modello è il SIM7600X 4G HAT, prodotto dall'azienda Waveshare. La decisione è ricaduta su questo tipo in quanto non è complesso nell'utilizzo e si presta bene nell'ambiente dei microcontrollori.

2.2 Software

Come detto nell'introduzione del capitolo, per software si intendono tutti quei componenti che sono intangibili all'interno di un sistema informatico, ovvero l'insieme delle istruzioni volte allo svolgimento di un determinato compito/operazione. Possiamo avere diversi tipi di software, in particolare in questa sezione si andranno a descrivere i software di sviluppo, che sono utilizzati per crearne di nuovi.

2.2.1 STM32 CUBE IDE

Come software principale di sviluppo è stato utilizzato STM32CubeIDE, sviluppato da STMicroelectronics per la programmazione di microcontrollori della famiglia STM32. Attraverso questo è stato possibile settare tutti i vari componenti della scheda, e successivamente farli interagire attraverso il firmware. Andando più nel dettaglio l'ambiente di sviluppo mette a disposizione:

- un'interfaccia grafica dove vengono visualizzati tutti i vari pin del microcontrollore per poi configurarli
- un editor di codice, nello specifico è stato utilizzato il linguaggio C.

Figura 2.6: STM32CubeIDE.

- un'interfaccia di debug, che insieme al debugger sopra citato permetteva il debugging

Un'utilità importante di questo ambiente di sviluppo sta nel fatto che solleva il programmatore da tanto lavoro di scrittura codice, in quanto una volta settati i vari parametri iniziali nell'interfaccia grafica, si occuperà lui stesso di generare tutto il codice per noi. Ovviamente poi il lavoro dello sviluppatore sarà quello di integrarlo con le funzioni specifiche del progetto, sennò il codice generato di default risulterebbe senza alcuna utilità.

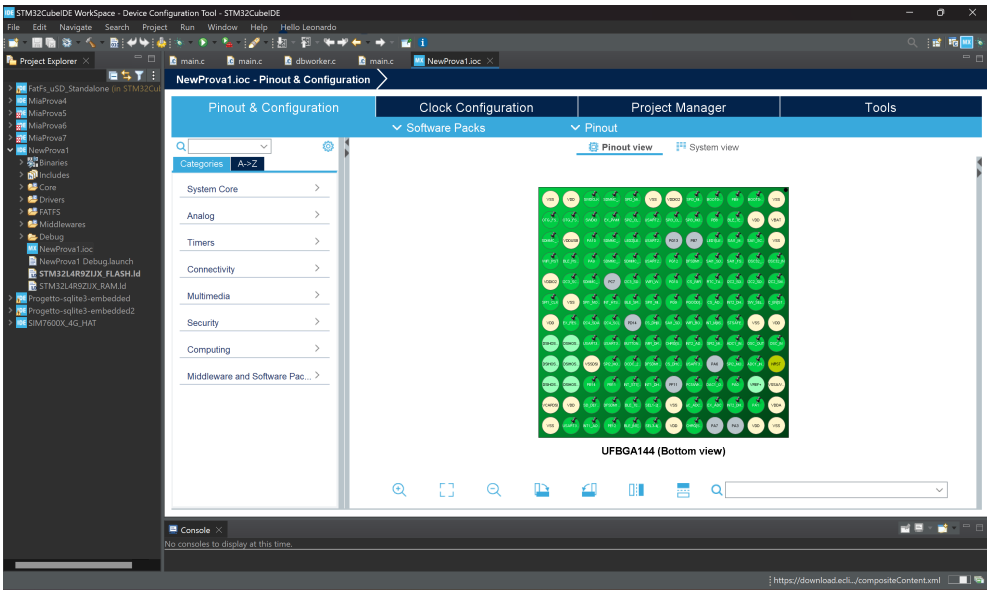

Figura 2.7: Schermata dell'IOC, dove si settano i vari parametri.

2.2.2 Docklight

L'utilizzo del software Docklight è stato necessario durante delle fasi di testing di porte seriali, infatti è nato proprio con questo scopo.

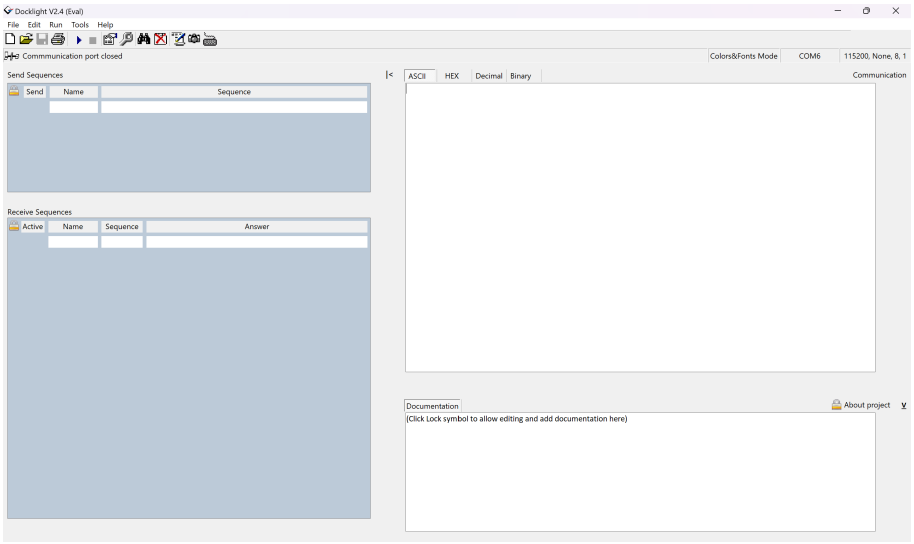

Figura 2.8: Schermata principale del software Docklight.

2.2.3 SIM7600 serial debugging assistant

La casa madre del prodotto mette a disposizione questo software, molto utile per quanto riguarda il debugging del modulo GSM e per testarne tutte le varie funzionalità. Nella schermata principale è possibile scrivere i comandi che si vogliono testare per poi mandarli in input, ricevendo di seguito l'output con le informazioni del caso.

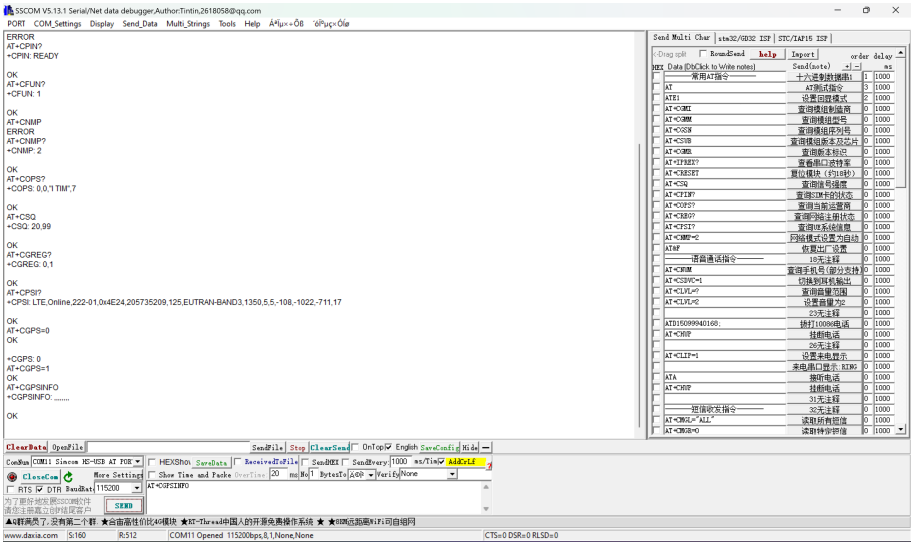

Figura 2.9: Schermata principale del software SIM7600 serial debugging assistant, con degli esempi.

2.3 Sistema embedded

Dopo la descrizione degli hardware e dei software che sono stati utilizzati, si fornisce ora una panoramica generale sul particolare tipo di sistema attorno al quale la trattazione è incentrata, il cosiddetto sistema embedded. La descrizione che segue è doverosa in quanto molto spesso inizialmente il concetto di embedded può rimanere difficile da comprendere.

Quando parliamo del mondo embedded, ci riferiamo a una vasta categoria di sistemi informatici che sono integrati direttamente all'interno di dispositivi hardware specifici. Questi sistemi non sono pensati per essere utilizzati come computer generici, ma piuttosto sono progettati per eseguire una o più funzioni ben definite, in modo efficiente e spesso con requisiti particolari.

Prendendo per esempio un elettrodomestico come una lavatrice, il sistema embedded al suo interno gestisce diverse operazioni, come il controllo del motore per il movimento del cestello, il monitoraggio della temperatura dell'acqua etc. Tutto questo è realizzato tramite un software chiamato firmware, progettato per il particolare hardware su cui è installato. Il firmware è un tipo di software che è profondamente integrato nell'hardware e che si occupa del funzionamento di base di quest'ultimo.

Dal punto di vista strutturale, un sistema embedded è composto da tre elementi principali:

- Hardware: si tratta dei componenti fisici che includono la CPU (di solito un microcontrollore o un microprocessore), la memoria (RAM, ROM), e altri circuiti necessari per interfacciarsi con il resto del dispositivo. L'hardware deve essere progettato per soddisfare i requisiti specifici del sistema, come il consumo energetico, la resistenza alle condizioni ambientali e l'affidabilità.

- Firmware: è il software che gira sull'hardware e che viene scritto specificamente per controllare e far funzionare il dispositivo embedded. A differenza del software di un computer classico, il programmatore deve tenere conto dell'efficienza e delle risorse che occupa, essendo notevolmente limitate rispetto ad un computer generico.

- Periferiche: questi sono i dispositivi o componenti aggiuntivi che sono collegati al sistema embedded e che interagiscono con l'ambiente esterno. Le periferiche includono sensori, attuatori, display, pulsanti, interfacce di rete, etc. Prendendo sempre come esempio una lavatrice, il sensore di temperatura e il motore sono periferiche che vengono controllate dal sistema embedded per eseguire correttamente il ciclo di lavaggio.

Per comprendere meglio questo tipo di dispositivi è bene notare che sono ormai onnipresenti e si trovano in una miriade di oggetti che si usano nella vita di tutti i giorni. Altri esempi di dove possono trovarsi:

- Mezzi di trasporto: in un'automobile moderna, ci sono dozzine di sistemi embedded che si occupano di tutto, dalla gestione del motore e del sistema di frenata, fino alla regolazione del clima e ai sistemi di infotainment.
- Elettrodomestici: lavatrici, forni a microonde, frigoriferi e condizionatori d'aria sono tutti esempi di elettrodomestici che utilizzano sistemi embedded per gestire le operazioni interne e ottimizzare il funzionamento.
- Strumenti medici: apparecchi come i pacemaker, i glucometri e gli strumenti di monitoraggio del paziente sono dispositivi critici che richiedono una gestione precisa e affidabile, ed è qui che i sistemi embedded dimostrano la loro importanza.
- Smartphone:questi tipi di telefoni includono numerosi sistemi embedded al loro interno che gestiscono diverse funzionalità, come il controllo della fotocamera, la gestione del touchscreen e la gestione della connettività per esempio WIFI e Bluetooth.

In sostanza, i sistemi embedded sono, se così si può dire il "cervello e la mente" che danno vita a tanti dispositivi, tenendo sempre conto dei vincoli stringenti che ogni tipo di oggetto può avere, e nonostante ciò essere comunque molto efficienti.

Capitolo 3

Architetture

In questo capitolo verranno illustrate nel dettaglio le architetture dei due sistemi principali,il nodo di tipo 3, e il sistema embedded contenente il microcontrollore.

3.1 Architettura del nodo di tipo 3

In questa sezione si andrà a analizzare più nel dettaglio l'architettura del suddetto nodo di tipo 3 del progetto. Come accennato nel capitolo 1, questo è il nodo responsabile della comuncazione tra l'impianto generico, che può essere un camion nel nostro caso, ma che potrebbe essere benissimo altro, e il sistema SDS. Come avviene effettivamente lo scambio dei dati?

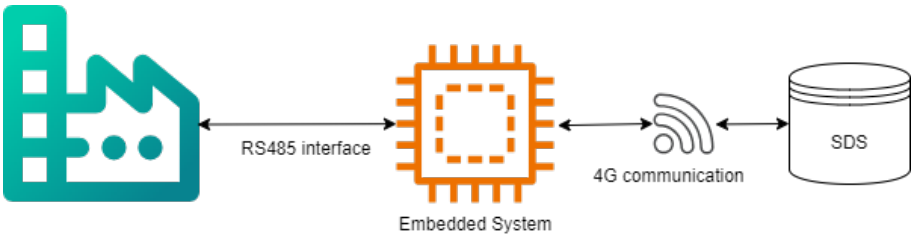

Figura 3.1: Schematizzazione dell'architettura del nodo di tipo 3.

Come si può notare dalla figura 3.1, il fulcro è rappresentato dal sistema embedded. L'impianto in questo caso ha un'interfaccia hardware per comunicare, che fa da tramite tra lui e il microcontrollore, ovvero lo standard RS485 (verrà spiegato nel dettaglio nei successivi capitoli). Una volta acquisiti i dati da parte del microcontrollore dovranno essere mandati al database SDS, attraverso il modulo GSM, stabilendo una connessione di rete, attraverso un tunnel SSH.

3.2 Architettura del sistema embedded

Questo tipo di architettura nasce con lo scopo di mettere in comunicazione il camion frigorifero, quindi tutti i vari dati che esso preleva dal suo interno, con il database dell'SDS. Per fare ciò come accennato nella sezione precedente, si è optato per un microcontrollore che facesse da fulcro, un modulo GSM necessario per la connessione internet e la geolocalizzazione, e un'interfaccia hardware per comunicare con il camion, si è creato a questo punto il sistema embedded. Il sistema è semplice, si hanno 3 dispositivi che devono comunicare, dove il secondo ovvero il microcontrollore fa da tramite tra questi. Quindi il primo comunica con il secondo attraverso l'interfaccia RS485, mentre il secondo con il terzo con un protocollo UART (verrà spiegato nei successivi capitoli).

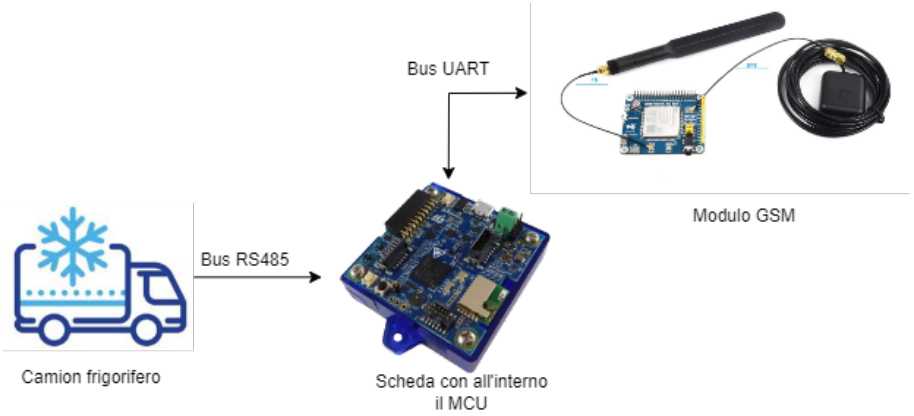

Figura 3.2: Schematizzazione dell'architettura del MCU.

Capitolo 4

Sviluppo

In questo capitolo viene spiegato come sono stati raggiunti tutti gli obiettivi fissati in precedenza nel cap. 1. Si comincerà quindi con la descrizione più approfondita del microcontrollore e della scheda che lo monta, in maniera tale da risultare molto più chiare tutte le successive fasi dello sviluppo che ricordiamo essere:

- Comunicazione tramite protocollo RS485
- Implementazione del file system sul microcontrollore
- Implementazione del database sul microcontrollore
- Comunicazione con il modulo GSM
- Implementazione del driver per il modulo GSM

Ogni singolo obiettivo è stato raggiunto gradualmente, cercando sempre la comprensione generale per poi andarla ad applicare nel caso specifico del progetto. Verranno evidenziati anche i problemi che si sono riscontrati e ovviamente la loro risoluzione che è stata trovata.

4.1 Scheda di sviluppo a STEVAL-STWINKT1B

Nella fase iniziale del tirocinio, il primo obiettivo è stato prendere mano e familiarizzare con questo tipo di microcontrollore. Come accennato nel capitolo 3, il modello è l'STM32L4R9ZIJ6U, e nello specifico è montato sulla scheda STEVAL-STWINKT1B. Il microcontrollore da solo farebbe ben poco se non eseguire calcoli, quindi una scheda con un circuito integrato ad esso è ciò di cui ha bisogno per poter utilizzare i componenti che essa monta.

L'inizio ha richiesto un notevole lavoro soprattutto di studio, in quanto l'azienda Eletica, essendo partner già del progetto Enough, aveva iniziato il proprio lavoro di sviluppo. Quindi c'è stato un focus doppio, sia sulla cosiddetta "teoria" che sulla pratica, andando a vedere cosa era stato fatto e cercando di capirlo di volta in volta con le conoscenze appena apprese studiando e provando. Inevitabilmente quindi si è preso confidenza con l'ambiente di sviluppo STM32CubeIDE, andandosi a creare dei piccoli progettini semplici per imparare le basi dell'IDE e soprattutto le "best

practice" del mondo embedded. L'IDE è un software molto utile e potente, infatti essendo della stessa casa produttrice della scheda, è possibile selezionare nel momento di creazione del progetto il modello del microcontrollore, così da avere già molti parametri settati e preconfigurati. Fatto ciò lo sviluppo prosegue su due binari paralleli, uno sul cosiddetto file .ioc (si rimanda alla figura 2.7), che non è altro che una sorta di configuratore per semplificare il lavoro dello sviluppatore, e uno sui veri e propri file dove si andrà a scrivere il codice nel linguaggio C.

```
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();
MX_DAC1_Init();
MX_DFSDM1_Init();
MX_I2C2_SMBUS_Init();
MX_I2C3_Init();
MX_I2C4_Init();
MX_LTDC_Init();
MX_RTC_Init();
MX_SAI1_Init();
MX_SDMMC1_SD_Init();
MX_SPI1_Init();
MX_SPI2_Init();
MX_SPI3_Init();
MX_TIM2_Init();
MX_TIM5_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();
MX_USB_OTG_FS_PCD_Init();
MX_FATFS_Init();
/* USER CODE BEGIN 2 */



char mess[]="Hello World!";
fr=f_mount(&SDFatFS, (TCHAR const*)SDPath, 0); //""

fr= f_open(&SDFile,"config.jsn",(BYTE)(FA_READ));
//if(fr) return (int)fr;

fr= f_write(&SDFile,mess,sizeof(mess),&bw);
//if(fr) return (int)fr;
```

Figura 4.1: Parte del main del codice sorgente di un progetto

Tuttavia le informazioni che si hanno a questo punto risultano ancora insufficienti per riuscire a fare qualcosa con il microcontrollore, ad esempio non sono noti i pin nel caso in cui si voglia dare un segnale alto per accendere un generico led. Queste informazioni si trovano sulla documentazione ufficiale del modello del microcontrollore, anche chiamato datasheet. Senza quest'ultimo sarebbe impossibile questo tipo di lavoro, in quanto esistono un'infinità di microcontrollori diversi montati su schede diverse. Per questo motivo le case produttrici forniscono una documentazione dettagliata su come è costruito, quindi i vari circuiti elettrici, e come si dovrebbero utilizzare i vari componenti.

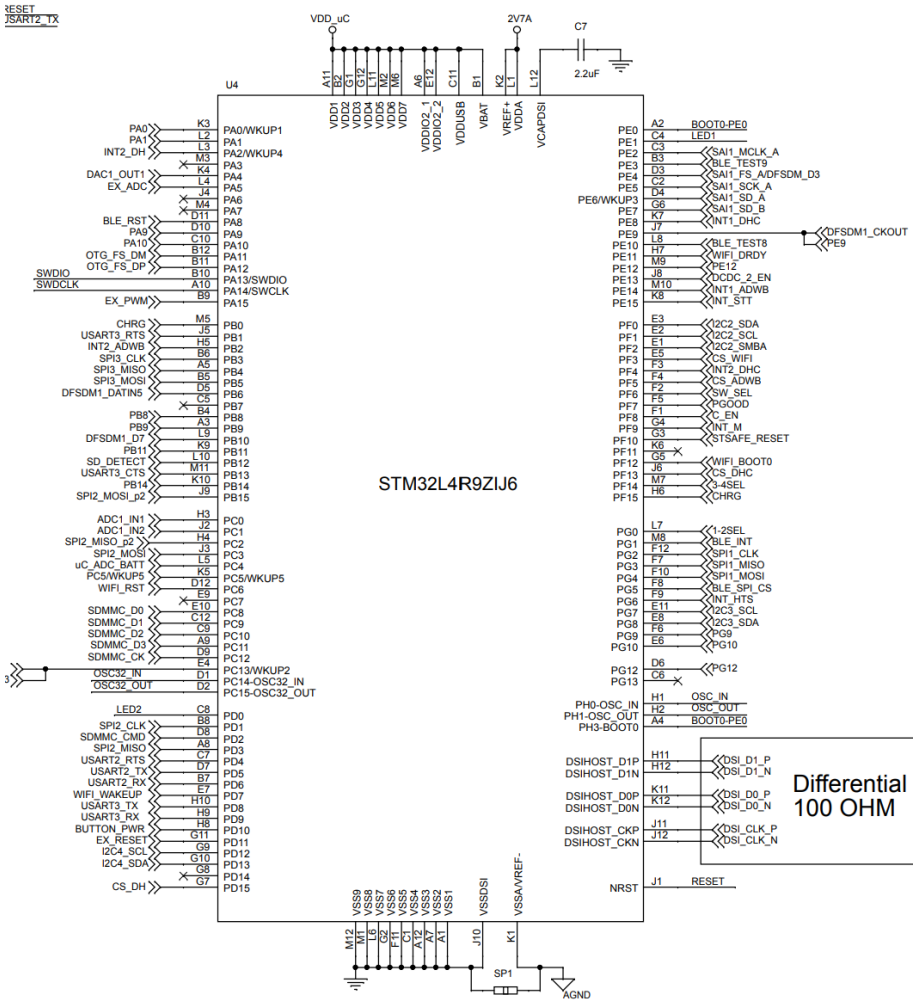

Figura 4.2: Schema del microcontrollore preso dal datasheet[1]

Come si può notare dalla figura 4.2 il microcontrollore ha numerosi pin, ognuno di questi ha un nome identificativo per esempio PD5, che serve per riconoscerlo quando deve essere usato nel programma. Ogni pin è quindi collegato ad una determinata periferica, riconoscibile sempre dalla figura 4.2, quindi andando a cercare PD5 troveremo nel caso della figura USART2TX, che sarebbe trasmettitore della porta di comunicazione seriale.

Un'altra caratteristica tipica che durante queste prime fasi è stata sfruttata, è la versatilità dei microcontrollori. Infatti per capirne il funzionamento sono stati fatti girare ogni volta programmi diversi, e questo è importante perchè ci fa capire che anche solo un microcontrollore può avere un'infinità di utilizzi diversi, tutto dipende dal codice che decidiamo di caricarci.

4.2 Comunicazione tramite protocollo RS485

Come detto nei capitoli precedenti, la comunicazione tra il camion frigorifero e il microcontrollore avviene tramite il protocollo RS485[4]. Prima di cominciare la spiegazione è importante che venga fatta una panoramica su come funziona in generale la comunicazione dei microcontrollori.

Innanzitutto i protocolli si distinguono in base alla trasmissione dei dati, cioè come vengono trasmessi i bit. Il primo tipo viene chiamato comunicazione seriale in quanto i bit vengono mandati in serie uno per volta attraverso un unico canale. Il secondo tipo invece viene chiamato parallelo, perchè appunto come dice il nome, più bit viaggiano contemporaneamente su canali paralleli, per esempio se si vuole mandare un byte (8 bit) sono necessari 8 canali.

Una seconda distinzione viene fatta invece in base alla sincronizzazione. Il primo protocollo viene chiamato sincrono in quanto il trasferimento dei dati tra trasmettitore e ricevitore sono regolati da un clock, che stabilisce, attraverso un altro canale, quando leggere o mandare i dati. Il secondo protocollo è chiamato invece asincrono, perchè non utilizza un clock, ma sfrutta dei bit di start e di stop trasmissione.

La terza distinzione viene fatta invece in base al modo di trasferimento dei dati. Il primo viene chiamato half-duplex, comunicazione bidirezionale alternata, i dati possono essere trasmessi in entrambe le direzioni, quindi da un dispositivo all'altro e viceversa, però mai contemporaneamente. Ciò significa che se in un primo momento il canale è aperto in una direzione, ossia un dispositivo sta trasmettendo, il dispositivo ricevitore deve attendere che il primo finisca di inviare i sui dati prima di poter inviare i propri. I due dispositivi si possono alternare i ruoli di trasmettitore e ricevitore. Il secondo tipo viene chiamato full-duplex, comunicazione bidirezionale simultanea, i dati possono essere trasmessi contemporaneamente in entrambe le direzioni senza problemi. Un esempio di uso comune è il telefono.

Dopo aver introdotto queste classificazioni si introdurranno ora i principali protocolli di comunicazione per quanto riguarda i microcontrollori. Ad esempio I2C e SPI, sono dei protocolli seriali sincroni, che sfruttano l'architettura master-slave. Sempre come comunicazione seriale ma in questo caso asincrona, si ha la cosiddetta UART acronimo di Universal Asynchronous Receiver Transmitter[5].

4.2.1 UART, Universal Asynchronous Receiver Transmitter

E' doveroso spiegare più nel dettaglio questo tipo di comunicazione in quanto è stata utlizzata all'interno del progetto. Come dice il nome si tratta di un protocollo seriale asincrono che può lavorare sia in half-duplex che full-duplex. E' uno dei protocolli più usati per la comunicazione tra dispositivi, soprattuto nel contesto dei microcontrollori. Essendo propriamente un circuito a differenza dell'RS48 (verrà spiegato meglio in seguito), riesce a convertire i dati che arrivano in parallelo in seriale.

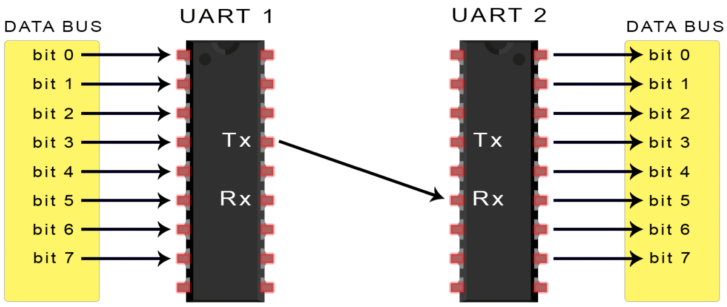

Figura 4.3: Schema UART.

La comunicazione è semplice, i due dispositivi hanno ripettivamente tre pin: TX (trasmettitore), RX (ricevitore), GND (ground). Il trasmettitore del primo dispositivo viene collegato con il ricevitore del secondo, e il ricevitore del primo viene collegato con il trasmettitore del secondo, infine ground con ground.

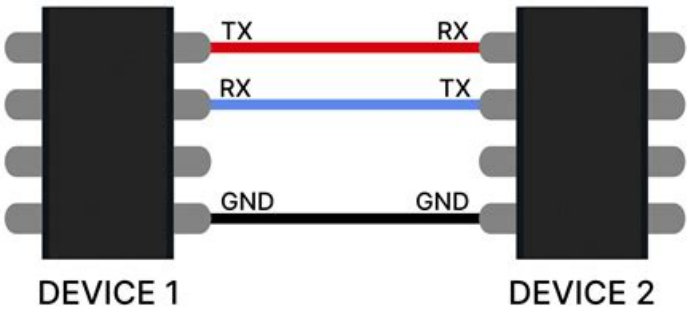

Figura 4.4: Schema del collegamento tra due dispositivi UART.

Esiste anche la versione sincrona chiamata USART. Dopo questa introduzione, è il momento di spiegare cos'è l'RS485.

4.2.2 Cos'è l'RS485?

L'RS485 è uno standard di comunicazione che trasmette i dati in seriale e può essere sia half-duplex che full-duplex. Nasce come versione migliorata del suo predecessore, l'RS232. Questo standard ha diverse caratteristiche che lo rendono molto affidabile, tanto da essere usato anche in contesti industriali. Un primo motivo è la possibilità di poter collegare più dispositivi sulla stessa linea. Oppure permette anche collegamenti a lunghe distanze (1200m) e a seconda di quest'ultima sarà proporzionale la velocità di trasmissione, che nelle distanze più brevi può raggiungere anche i 10Mb/s. La caratteristica più importante è sicuramente il tipo di trasmissione che sfrutta, ovvero la trasmissione differenziale. L'importanza di quest'ultima sta nel ridurre significativamente i disturbi, in quanto il livello logico che viene ricevuto non è dato dalla tensione rispetto a terra, ma bensì dalla differenza di tensione presente tra i due fili. Quindi una polarità indicherà il livello logico 1, mentre l'inversa lo 0 logico. (delle figure sentire a fernando)

4.2.3 Come si usa l'RS485?

Per poter sfruttare questo tipo di standard quando si parla di microcontrollori, è necessario avere il cosiddetto transceiver. I microcontrollori tipicamente comunicano tramite il protocollo UART/USART, che come spiegato nella sottosezione precedente, è un protocollo di trasmissione seriale sincrono e asincrono. Lo standard RS485 però sfrutta una trasmissione differenziale andando in conflitto quindi con l'UART/USART. A questo punto entra in gioco il transceiver, che va a tradurre i segnali differenziali nei segnali logici per il microcontrollore e viceversa. Nel caso del progetto, la scheda STEVAL-STWINKT1B monta il transceiver STR485LV.

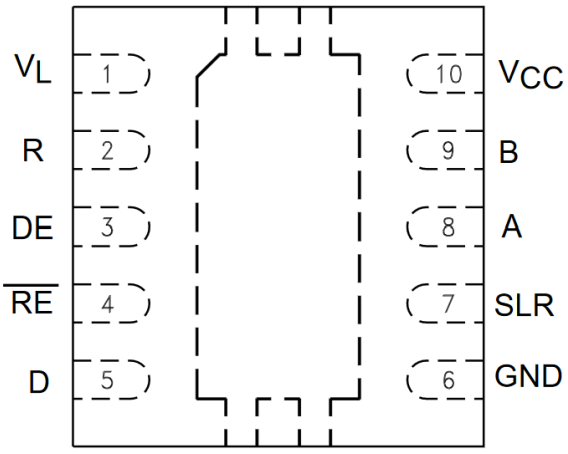

Figura 4.5: Schema dei pin del transceiver STR485LV.

Sul pin 2 che è espresso con la lettera R è connesso il ricevitore del primo dispositivo che utilizza come protocollo l'USART, sul pin 5 invece è connesso il trasmettitore

indicato dalla lettera D. I pin 3 e 4 servono per selezionare rispettivamente la trasmissione o la ricezione, identificati con DE e con RE. Sui pin 9 e 8, B ed A, sono presenti invece i fili con cui poi si andrà a comunicare in RS485.

4.2.4 Implementazione

Come detto nella sottosezione precedente, la scheda utilizzata per il progetto monta non a caso il transceiver STR485LV, ciò ha reso lo sviluppo più agile.

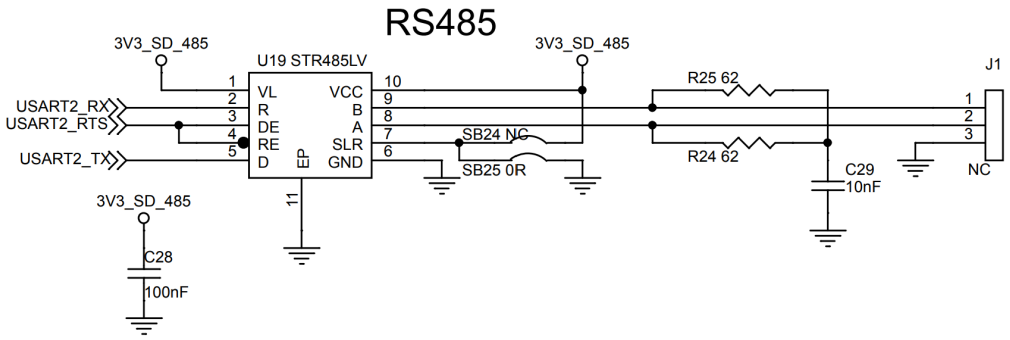

Figura 4.6: Schema circuitale dell'interfaccia RS485 e del transceiver STR485LV.

Come si può notare dalla figura 4.4, il circuito ha sulla sinistra gli ingressi dell'USART2 nel transceiver. Da quest'ultimo sui pin 9 e pin 8 escono i rispettivi fili B ed A. Sulla destra invece è presente il terminale J1, dove avviene fisicamente il collegamento sulla scheda. Sui pin 1 e 2 del J1 verranno quindi collegati i fili B ed A provenienti dall'esterno del microcontrollore.

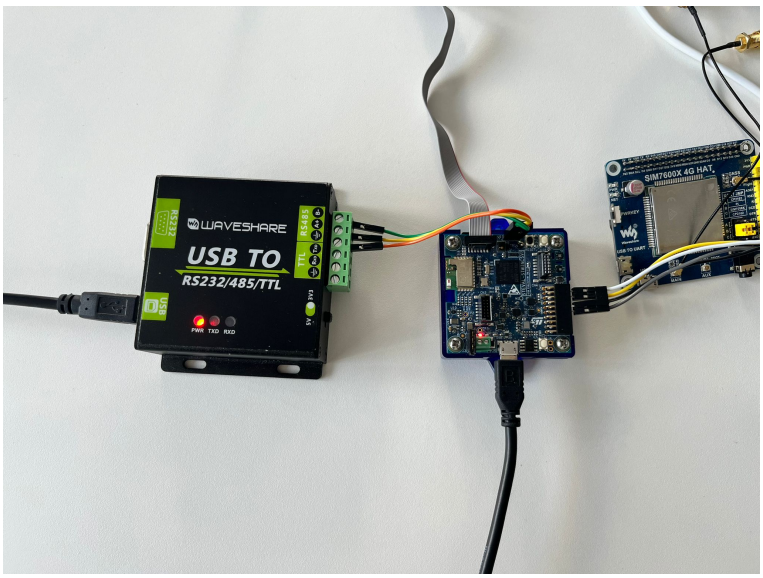

Figura 4.7: Collegamento fisico dell'interfaccia RS485 alla scheda.

A questo punto, una volta effettuate tutte le opportune connessioni, si è arrivati al momento dello sviluppo del firmware. In questo caso l'obiettivo principale era il funzionamento in se, quindi la comunicazione attraverso l'interfaccia. Per una questione di comodità i messaggi scambiati sono banali, anche perchè nel momento dello sviluppo non si sanno ancora quali e come sono i dati che dovranno arrivare dal camion refrigerante tramite l'RS485. Per quanto riguarda la visualizzazione dello scambio dei messaggi è stato utilizzato il software Docklight. Il software è stato utile anche per simulare il dispositivo che manderà i dati.

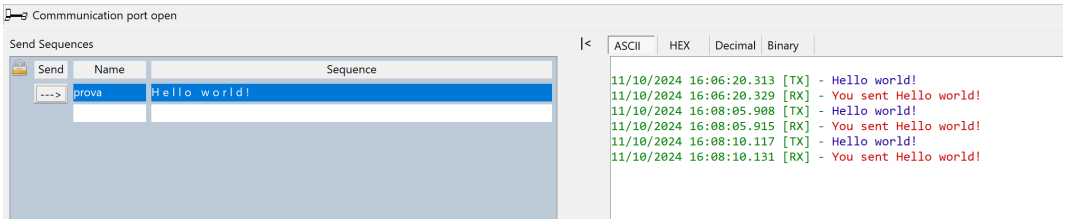

Figura 4.8: Esempio dello scambio dei messaggi tramite l'interfaccia.

4.2.5 Codice di esempio

```
/* USER CODE BEGIN 0 */

uint8_t txData[]="You sent Hello world!";   //stringa di caratteri d'esempio da trasmettere
uint8_t rxData[13];                //buffer contenitore per i caratteri da ricevere

void send_uart_RS485() {

    HAL_UART_Transmit(&huart2, txData, sizeof(txData)-1, HAL_MAX_DELAY);
}

void receive_uart_RS485(){

  HAL_UART_Receive(&huart2, rxData, sizeof(rxData)-1,HAL_MAX_DELAY);
}
/* USER CODE END 0 */
```

Listing 4.1: Porzione del codice main

```
1   while (1)
2   {
3
4     receive_uart_RS485();
5     send_uart_RS485();
6     HAL_Delay(500);
7
8
9     /* USER CODE END WHILE */
10
11    /* USER CODE BEGIN 3 */
12  }
```

Listing 4.2: Porzione del codice all'interno del while()

4.2.6 Spiegazione del codice

Nel listing 4.1 vengono creati i buffer sia per la trasmissione che per la ricezione, rispettivamente "txData[]" e "rxData[]". Successivamente vengono create due funzioni per trasmettere i e ricevere i dati, dove al loro interno hanno la funzione della libreria HAL per l'utilizzo dell'UART corrispondente.

Nel listing 4.2 invece si è all'interno del "while (true)", ovvero la parte di codice che il microcontrollore esegue continuamente, infatti al suo interno sono state inserite le funzioni create in precedenza. Nell'esempio specifico il microcontrollore è in attesa di ricevere qualcosa, una volta ricevuto i dati, trasmette il messaggio presente in "txData". Tutto questo è stato possibile attraverso l'uso del software Docklight, indispensabile come interfaccia per far arrivare e trasmettere i dati al microcontrollore.

4.3 Implementazione del file system sul microcontrollore

4.3.1 Introduzione

Andando avanti con il progetto è nata la necessità di salvare i dati in maniera persistente. La scheda STEVAL-STWINKT1B non a caso, posside uno slot dove è possibile inserire una scheda SD per salvare i dati. Nonostante ciò, per poter riuscire a salvare un file, dato che ci si trova in un ambiente embedded, sono necessari diversi passi. A differenza di un normale computer dove il salvataggio è reso immediato dal sistema operativo, nei microcontrollori non essendone presente alcuno, spetta allo sviluppatore farsi carico di questo lavoro. Infatti il compito del sistema operativo è quello di gestire tutto ciò che il file system gli offre. Prima di cominciare con la spiegazione è opportuno quindi definire cos'è un file system.

4.3.2 Cosa si intende per file system?

Un file system si definisce tale per il fatto di essere un sistema di gestione e accesso ai file. Questo sistema fornisce una struttura per i dati e dei metodi per controllarli. Anche il fatto di definire il concetto di file aiuta la comprensione, infatti lo si definisce come un'insieme logico di dati strutturati contigui, con degli attributi e sui quali è possibile eseguire delle operazioni. La loro rappresentazione semplifica la gestione delle memorie di massa. Un tipo di file che ne contiene altri al suo interno è detto directory, sul quale possono essere eseguite diverse operazioni di gestione.

Per dare un esempio più concreto si introduce ora il file system FAT. Questo file system è stato sviluppato inizialmente da IBM, per poi passare anche da Microsoft. Si può dire che è stato uno dei primi, ma anche esistendone dei migliori più moderni, continua anche oggi ad essere utilizzato per la sua semplicità. Infatti essendo anche open source è molto fruibile per ogni tipo di esigenza.

4.3.3 Il file system FatFs

Come file system per il microcontrollore è stato utilizzato il FatFs[6], che si basa sui modelli FAT/exFAT ed è pensato appositamente per i sistemi embedded. Il fatto di essere open source favorisce la sua fruizione in diversi contesti, infatti il suo funzionamento è indipendente dalla piattaforma in cui verrà fatto girare e dal sottostante dispositivo di memoria.

4.3.4 Implementazione

Il fatto che il file system sia uno strato separato dal sottostante modulo di archiviazione rende necessari diversi passaggi in più per la configurazione, perchè ricordiamo che l'obiettivo è poter salvare i file sulla scheda SD.

Lo sviluppo è cominciato innanzitutto dall'analisi dell'esigenza, infatti per poter salvare un file è necessario avere un file system, e dopo varie considerazioni si è optato per utilizzare il FatFs, che come detto nella sottosezione precedente è spesso utilizzato nel mondo embedded. Una volta deciso questo, si è deciso dove salvare i file, quindi sulla scheda SD.

Come spiegato nei capitoli precedenti il file .ioc del progetto (vedi capitolo x) aiuta lo sviluppo, infatti si può configurare direttamente sulla voce "Middleware and Software Packs", l'utilizzo di tale file system. Successivamente bisogna selezionare la voce "SD Card" in quanto lo si vuole montare su tale dispositivo di archiviazione. Infine sempre sulla stessa pagina alla voce "Platform Settings" si dovrà selezionare il pin PB12. Quindi una volta che il progetto verrà buildato nuovamente avrà già importate tutte le librerie necessarie per l'implementazione e tutti i compinenti inizializzati.

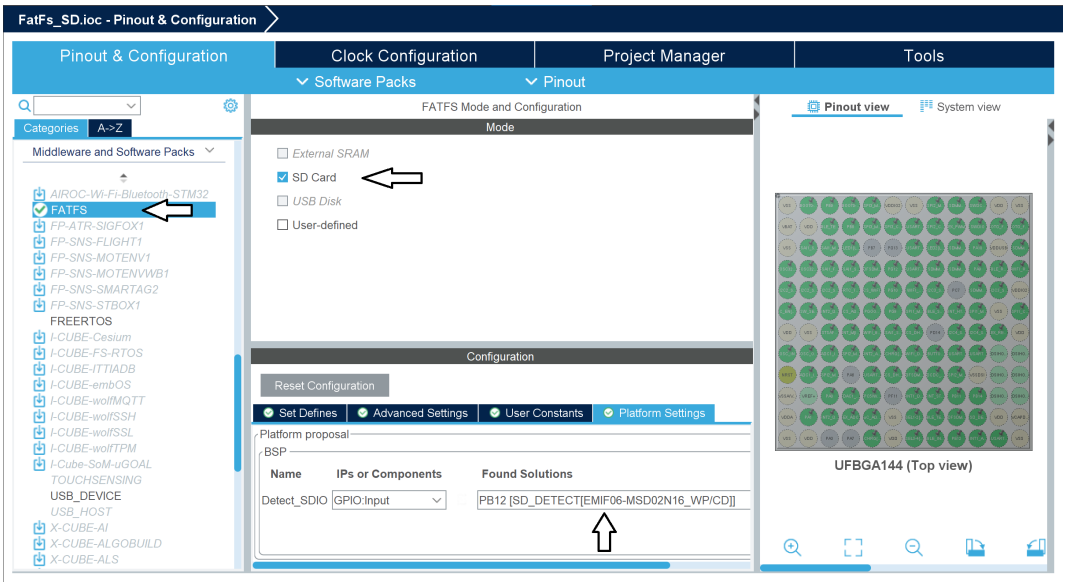

Figura 4.9: Configurazione nel file .ioc.

A questo punto poi, una volta che libreria con tutte le varie funzioni per usare il file system era pronta, si è passato allo sviluppo della parte di codice. All'inizio il compilatore dava diversi errori ma grazie all'interfaccia di debug, fondamentale in questi casi quando bisogna capire cosa non va e provando di volta in volta nuove possibili soluzioni, si è giunti ad una versione del codice funzionante.

4.3.5 Codice di esempio

```
  /* USER CODE BEGIN 2 */


    FRESULT res;
    uint32_t bw;
    uint8_t wtext[]="Hello world!" ;

    res=f_mount(&SDFatFS, (TCHAR const*)SDPath, 0);

    res=f_open(&SDFile,"hello.txt",FA_CREATE_ALWAYS | FA_WRITE);

    res=f_write(&SDFile, wtext, sizeof(wtext), (void *)&bw);

          if((bw>0)&& (res==FR_OK)){
              f_close(&SDFile);
            }


  /* USER CODE END 2 */
```

Listing 4.3: Porzione del codice main

```
uint8_t BSP_PlatformIsDetected(void) {
    uint8_t status = SD_PRESENT;
    /* Check SD card detect pin */
    if(HAL_GPIO_ReadPin(SD_DETECT_GPIO_PORT, SD_DETECT_PIN) !=
    GPIO_PIN_RESET)
    {
        status = SD_NOT_PRESENT;
    }
    /* USER CODE BEGIN 1 */
    status = SD_PRESENT;
    if(HAL_GPIO_ReadPin(SD_DETECT_GPIO_PORT, SD_DETECT_PIN) ==
    GPIO_PIN_RESET)
            {
                status = SD_NOT_PRESENT;
            }
    /* user code can be inserted here */
    /* USER CODE END 1 */
    return status;
}
```

Listing 4.4: Modifica necessaria al codice di default nel file fatfsplatform.c

4.3.6 Spiegazionde del codice

In questa sottosezione verranno spiegate soltanto le porzioni del codice riportate nei Listing 4.1 e 4.2, non perchè il resto non sia della stessa importanza, ma risulterebbe prolisso e fuori dal contesto del progetto. Cominciando dalla riga 4 la prima variabile che viene dichiarata è del tipo FRESULT. L'importanza di questa variabile è elevata in quanto grazie ad essa il debug è molto più agevole. Le funzioni utilizzate ritornano esattamente il tipo FRESULT, altro non è che una stringa con un valore ben preciso del tipo: FR-OK, FR-NOT-READY, FR-INVALID-OBJECT ecc... Questi valori di ritorno servono allo sviluppatore per capire se sta andando tutto bene, oppure per vedere di che tipo è l'errore e quindi attraverso quest ultimo trovare una soluzione. Quindi se il valore ritornato è FR-OK, sta andando tutto bene, altrimenti bisogna vedere cosa ritorna la variabile "res" e da lì cercare il problema. Successivamente viene montato il file system sull'SD, fatto ciò viene creato e aperto il file (i flags sono simili a quelli di fopen()), si scrive il testo di esempio e poi si chiude.

Il Listing 4.2 invece, fa parte della soluzione che è stata trovata ad un errore il quale faceva ritornare la variabile "res" come "FR-NOT-READY". E' stato un lavoro che ha richiesto molto tempo in debugging, perchè non si capiva qual'era il problema all'inizio. Poi però si è arrivati al file che gestisce la presenza della scheda SD, e a quanto pare il codice generato di default faceva si che la scheda non venisse letta. Quindi è stato necessario aggiungere una porzione di codice che la leggesse correttamente.

4.4 Implementazione del database sul microcontrollore

Come già ribadito più volte nella trattazione, un obiettivo fondamentale all'interno del progetto è la comunicazione tra camion frigorifero e il server SDS. Attraverso questa comunicazione avviene lo scambio dei dati provenienti dal camion. Al momento della scrittura non è noto il modello dei messaggi ricevuto, ne tantomeno la frequenza con la quale devono essere mandati. Tuttavia anche non conoscendo questi paramentri è stato possibile cominciare lo sviluppo. L'infrastruttura denotata dal nodo di tipo 3, richiede innanzitutto l'implementazione di un database sul microcontrollore. Questo perchè i dati provenienti dal camion frigorifero hanno bisogno di un'organizzazione, ma soprattutto per il fatto di poter utilizzare le query messe a disposizione dal linguaggio del DB. Infatti è attraverso quest'ultime che i dati vengono estrapolati dal database e mandati al server.

4.4.1 SQL, SQLite

Prima di cominciare con la spiegazione dell'implementazione, si forniscono delle nozioni generali su SQL e SQLite che favoriranno la comprensione successiva.

Cos'è innanzitutto SQL? SQL sta per Structured Query Language ed è un linguaggio di programmazione standard che si utilizza per la gestione dei database relazionali. Semplificando il concetto, quest'ultimi si basano su un insieme di tabelle strutturate , le quali possono essere messe in relazione, e quindi collegate tra loro attraverso delle chiavi. I dati ovviamente sono contenuti all'interno di queste tabelle, e per gestirli SQL mette a disposizione un'insieme di operazioni fondamentali, che verranno poi utilizzate nella sintassi corretta in delle query.

Per quanto riguarda invece SQLite[7] bisogna andare più nello specifico nella parte di programmazione vera e propria. Il contesto del progetto fa si che lo sviluppo sia a basso livello, ciò vuol dire non poter sfruttare molte interfacce più semplici rese disponibili ai livelli più alti. Per questo motivo, la specifica sull'utilizzo di SQL è stata rispettata utilizzando la libreria SQLite.

Figura 4.10: SQLite.

L'uso di questa libreria è stato fin da subito scontato in quanto è stata sviluppata appositamente per le piattaforme embedded. Tra le varie carattaristiche che la rendono perfetta per l'utilizzo all'interno del progetto vi sono:

- è stata concepita per essere integrata direttamente all'interno delle applicazioni
- SQLite lavora direttamente su un file del disco a differenza di altre piattaforme basate su SQL, quindi il database risulterà essere un unico file
- è una libreria compatta, che non richiede molta memoria, fondamentale nel caso di applicazioni embedded
- è disponibile ormai da diversi anni ed è affidabile oltre che testata ormai da tempo

Dopo questa panoramica, si passa ora alla spiegazione dettagliata dell'implementazione del database sul microcontrollore.

4.4.2 Implementazione

Una volta stabilita la volontà di utilizzare SQLite, si è passati all'implementazione vera e propria all'interno del progetto. Dopo aver importato la libreria, e una volta capito quali fossere le funzioni che si sarebbero dovute utilizzare, si è deciso di creare innanzitutto un file a parte denominato "dbworker.c". Questo per rispettare il principio di separazione, in quanto all'interno di questo file sono contenute tutte le implementazioni delle funzioni che poi verranno utilizzate all'interno del "main.c".

Per poter funzionare, SQLite ha bisogno di un file system, nel caso specifico utilizza un virtual file system (vfs), cioè un file system virtuale che sfrutta per la creazione dei file dove poi salva e gestisce i dati. L'idea di partenza era quella di sfruttare il file system già implementato sulla scheda SD per poter salvare direttamente lì i dati; quindi sfruttare tutte le funzioni di FatFs come vfs.

Problemi riscontrati

Come accennato in precedenza essendo una programmazione a basso livello, capita di trovare delle interfacce incompatibili, cosa che è successa durante lo sviluppo. Infatti i due file system FatFs e il vfs di sqlite, trattavano in maniera diversa la gestione di un oggetto file, il che rende molto complesso per il momento l'utilizzo del FatFs come virtual file system per il progetto. Lo sviluppo futuro dovrà quindi concentrarsi sul creare una sorta di "adattatore" così da poter sfruttare il FatFs come vfs. Nonostante questo problema che è stato incontrato, la progettazione e lo sviluppo sono proseguiti ed è stata trovata una soluzione.

Soluzione

Nonostante questo problema che è stato riscontrato, la progettazione e lo sviluppo sono proseguiti ed è stata trovata una soluzione. Questa sfrutta il cosiddetto "In-Memory Database", ovvero salvare il file nella memoria RAM. Ciò significa che il database rimarrà soltanto finchè al microcontrollore viene fornita l'alimentazione, una volta che si spegne si è costretti a crearne uno nuovo e riempirlo con nuovi dati.

Questa soluzione è molto efficace, perchè per come viene utilizzato, il microcontrollore quando il camion è in uso avrà sempre l'alimentazione disponibile, salvo qualche caso eccezzionale inaspettato. Per quanto riguarda sempre la persistenza dei dati, si è trovato un modo per renderli tali, anche se non all'interno del microcontrollore. Infatti grazie ai cosiddetti "trigger", è possibile inviare direttamente i nuovi dati al server SDS.

Andando più nello specifico, i trigger sono dei meccanismi che il linguaggio SQL mette a disposizione per eseguire direttamente delle istruzioni quando si verifica un determinato evento, in una determinata tabella del DB. Ad esempio:

```
1
2  rc = sqlite3_exec(dbfile_handler,"CREATE TRIGGER new_switch_trigger
     AFTER INSERT ON v_switch BEGIN SELECT new_switch_sender();END;",
     NULL,0,errmsg);
```

Listing 4.5: esempio dell'utilizzo del trigger nel progetto

Come si vede dalla porzione di codice 5.5, il trigger chiama la funzione new-switch-sender() appena dopo un inserimento nella tabella v-switch. Quindi in sostanza una volta che il camion manda un nuovo dato, si attiva il trigger, e verrà mandato al server, così facendo anche se il microcontrollore dovesse spegnersi accidentalmente, i dati sarebbero già al sicuro e salvati all'interno dell'SDS. Di seguito si illustrano le porzioni del codice dedicate all'effettiva creazione del database.

4.4.3 Codice di esempio

```
int initdb()
{
  int rc;
  rc = fs_register();
  if( rc == SQLITE_OK){
    rc = sqlite3_config(SQLITE_CONFIG_GETPCACHE2,pcache_struct_ptr);
    //Force set default methods for pcache2

    rc = sqlite3_config(SQLITE_CONFIG_HEAP, MEMORY_POOL_HEAP_ADDR,
    MEMORY_POOL_HEAP_SIZE, tlsf_block_size_min());

    if(rc == SQLITE_OK){

      rc = sqlite3_config(SQLITE_CONFIG_MALLOC,&mem_methods);
      if(rc == SQLITE_OK){
        rc = sqlite3_config(SQLITE_CONFIG_PAGECACHE,(void*)ROUND8((int
    )MEMORY_POOL_DB_PAGECACHE_ADDR),MEMORY_POOL_DB_PAGECACHE_SIZE,2);
        if(rc == SQLITE_OK){

          MemPoolDB = tlsf_create_with_pool(MEMORY_POOL_DB_ADDR,
    MEMORY_POOL_DB_SIZE);
          if (MemPoolDB){

            rc = sqlite3_open_v2(":memory:", &dbfile_handler,
    SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE,FS_VFS_NAME);
              return SQLITE_OK;


          }
        }
      }
    }
  }
  return -1;
}
```

Listing 4.6: inizializzazione del DB

```
1   rc = sqlite3_exec(dbfile_handler,"CREATE TABLE IF NOT EXISTS Data (
      col0 INTEGER PRIMARY KEY, col1 INTEGER, col2 TEXT NOT NULL, time
      CURRENT_TIMESTAMP);",NULL,0,errmsg);
2   if(rc == SQLITE_OK){
3     sqlite3_free(errmsg);
4   }
```

Listing 4.7: creazione della tabella "Data"

```
1 int populateDB(){
2
3   int rc;
4   rc=sqlite3_exec(dbfile_handler,"INSERT INTO Data (col2) VALUES ('
      CIAO0');"
5                   "INSERT INTO Data (col2) VALUES ('CIAO1');"
6                   "INSERT INTO Data (col2) VALUES ('CIAO2');",0,0,
      errmsg);
7   if(rc!=SQLITE_OK){
8     return SQLITE_ERROR;
9   }else return rc;
10
11
12 }
```

Listing 4.8: Inserimento dei valori sulla prima colonna di "Data"

```
1 void stampDB(){
2
3   sqlite3_prepare_v2(dbfile_handler, "SELECT col2 FROM Data", -1, &
      res1, 0);
4   char *output;
5   while((watch1=sqlite3_step(res1))!=SQLITE_DONE){
6
7     output=sqlite3_column_text(res1,0);
8     strcat(MIO_TX_BUFFER,output);
9     strcat(MIO_TX_BUFFER," ");
10  }
11
12 }
```

Listing 4.9: Selezione dei dati presenti sulla prima colonna dalla tabella "Data" per poi essere messi in un buffer e letti

4.4.4 Spiegazione del codice

Nel listing 4.6 viene inizializzato il database, anche in questo caso viene creata una variabile che serve per verificare che stia andando tutto bene ovvero "rc". Si registra il file system attuale, e successivamente dopo aver assegnato parte della memoria RAM al database, lo si apre con la funzione alla riga 20. Da notare il primo membro ":memory:" necessario creare appunto il file sulla RAM.

Nel listing 4.7 avviene la creazione di una semplice tabella chiamata "Data", dove sono presenti 4 colonne con ognuna il tipo di valore che possono accettare. Tutto sempre verificato dalla variabile "rc".

Nel listing 4.8 invece vengono inseriti delle semplici parole nella terza colonna chiamata "col2".

Nel listing 4.9 invece vi è la parte di estrazione dei dati che sono stati inseriti nella colonna "col2". Una volta selezionati li si inserisce in un buffer per poi essere visualizzati.

Detto ciò l'obiettivo dell'implementazione del DB sul microcontrollore è stato raggiunto, la soluzione adottata è sicuramente molto valida, tuttavia per gli sviluppi futuri ci sarà sicuramente un focus per riuscire a portare il salvataggio del DB su un file all'interno della scheda SD del microcontrollore. Questo conclude la parte sull'implementazione del database, ovviamente anche in questo caso gli esempi che sono stati illustrati sono semplici, si ribadisce però l'importanza dell'effettivo funzionamento rispetto alla complessità.

4.5 Implementazione del driver per la comunicazione con il modulo GSM

Come già detto più volte nei capitoli precedenti, la comunicazione tra il microcontrollore e il server SDS avviene tramite rete wireless, utilizzando il modulo GSM. In questa sezione si illustrerà tutto ciò che è stato fatto per rendere possibile la comunicazione Si partirà dall'hardware, quindi come le due schede sono collegate fisicamente, per poi passare alla parte software e quindi l'implementazione all'interno di un progetto.

4.5.1 Connessioni hardware

Innanzitutto è necessario connettere nella maniera corretta le due antenne fornite con il modulo. L'antenna del 4G verrà collegata al pin denominato "MAIN" mentre l'antenna GPS nel pin con la dicitura GNSS, così da ottenere un risultato come quello nella Figura 3.5. Detto ciò per testare il funzionamento del modulo è stata fatta una prova collegandolo direttamente al computer e usando il programma SIM7600 serial debugging assistant. Testato il corretto funzionamento si è passati al collegamento tra le due schede.

In questi casi la prima cosa da fare è capire come comunicheranno le due schede, quindi che tipo di protocollo usare. In questo caso si è stabilito di usare l'UART. Fatto questo è opportuno controllare i datasheet delle rispettive schede per vedere dove devono essere effettuati i collegamenti per utilizzare tale metodo di comunicazione.

Una volta quindi consultato il datasheet della prima scheda sono stati collegati trasmettitore (giallo), ricevitore (bianco), GND (nero) e il power supply da 5V (grigio) come si può notare dalla Figura 4.11. Questi cavi poi dovranno essere collegati con i rispettivi pin della scheda del microcontrollore.

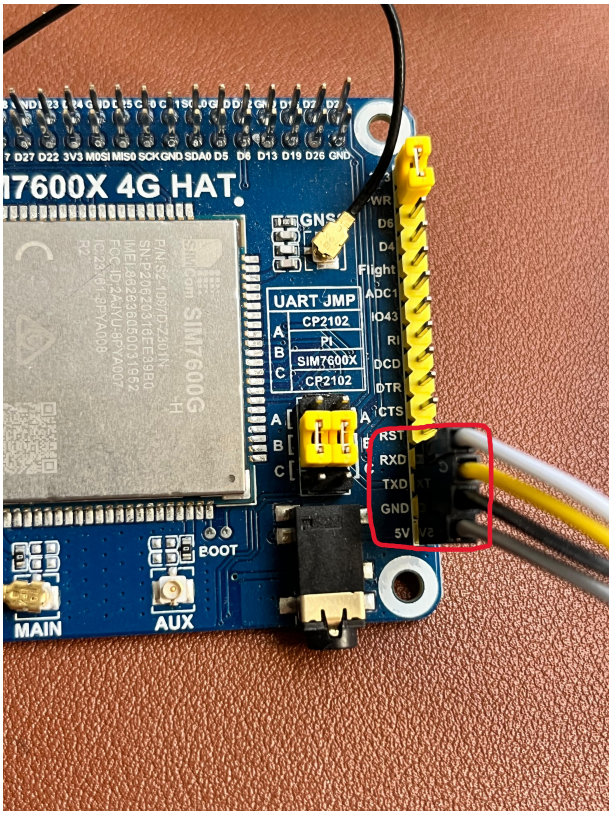

Figura 4.11: Collegamento dei pin necessari per la trasmissione UART sulla scheda SIM7600 4G HAT[2].

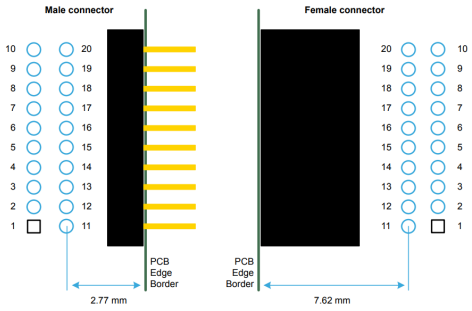

Table 2. STMod+ connector pin assignments and descriptions

STMod+ Pin number	Function(1) of the primary host mapped	Description
1	SPIx_NSS(2) / UARTy_CTS	Output / Input
2	SPIx_MOSIp(3) / UARTy_TX Output / Output	Output / Output
3	SPIx_MISOp(4) / UARTy_RX Input / Input	Input / Input
4	SPIx_SCK / UARTy_RTS Output / Output	Output / Output
5	GND Ground Reference	Ground reference
6	+5 V Power Supply(5)	Power supply

Figura 4.12: Schema dei pin della scheda STEVAL-STWINKT1B.

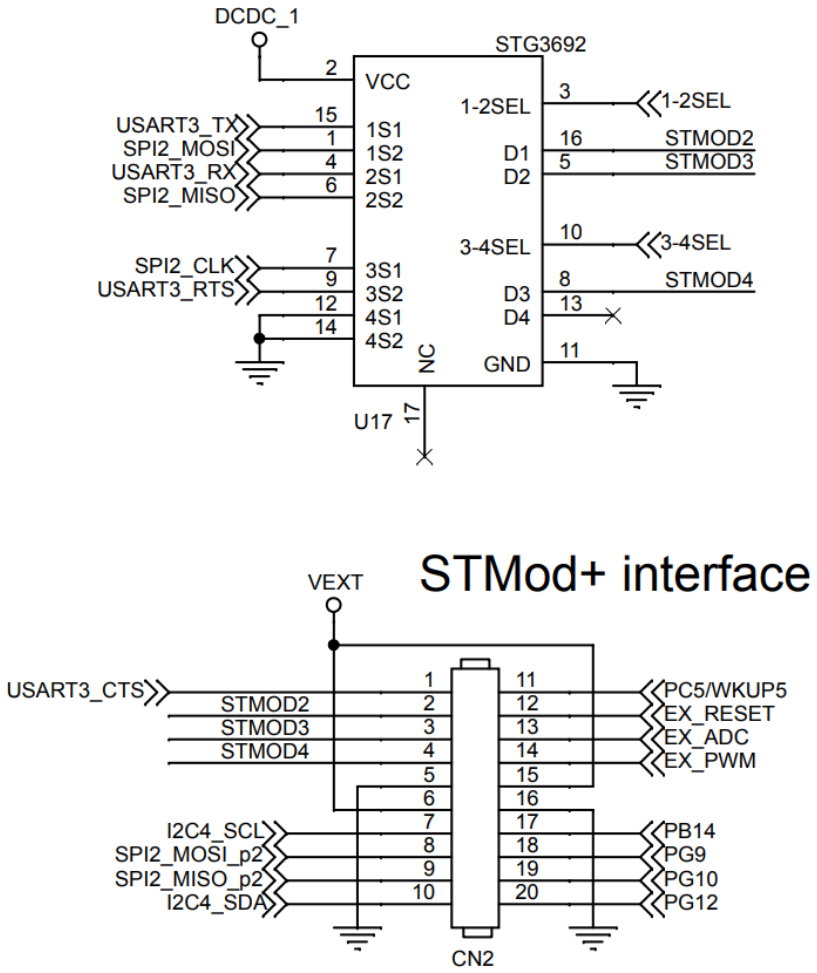

Figura 4.13: Schema circuitale del dispositivo fisico per la connessione.

Per capire quali fossero i pin per effettuare la connessione, ci si è serviti del datasheet[1] della scheda Figure 4.12 e 4.13. Grazie a questi due schemi oltre che ai pin fisici per connettere i cavi, è stata trovata la porta USART connessa al microcontrollore ovvero l'USART3. A questo punto avendo tutte le informazioni note si è proceduto con la connessione fisica, e successivamente con l'implementazione del firmware.

AT Commands

Per controllare le varie funzionalità di un modem attraverso la seriale, molto spesso vengono usati i cosiddetti AT Command. Sono un set di istruzioni standardizzate infatti diversi produttori le utilizzano. Consistono in una serie di piccole frasi che cominciano tutte per "AT..." ad esempio: "AT+CGPSINFO", "AT+CREG?", "AT+COPS?" e così via.

4.5.2 Implementazione

Per l'implementazione in questo caso è stato necessario implementare la trasmissione e la ricezione tramite UART, perchè appunto come detto in precedenza il modulo viene controllato tramite la seriale. Quindi sono stati mandate delle stringhe contenenti i comandi sopra elencati, mentre in ricezione ci si è aspettati la stringa di ritorno, solitamente "AT OK".

4.5.3 Codice di esempio

```
1  /* USER CODE BEGIN 2 */
2
3
4  uint8_t buffer[4];
5  buffer[0]='A';
6  buffer[1]='T';
7  buffer[2]='\r';
8  buffer[3]='\n';
9
10 uint8_t buffer2[10];
11
12             HAL_UART_Transmit(&huart3, (uint8_t*)buffer, 4,
     HAL_MAX_DELAY);
13
14         HAL_UART_Receive(&huart3,(uint8_t*)buffer2,10 , 2000);
15
16 /* USER CODE END 2 */
```

Listing 4.10: Porzione del codice main dove vengono utilizzati i comandi "AT"

4.5.4 Spiegazione del codice

In questo caso è un codice semplice, vengono trasmesse e ricevute delle stringhe attraverso la seriale UART3 che è stata selezionata in precedenza. In questo caso quindi una volta trasmessa la stringa "AT", verrà ricevuta e salvata nel "buffer2" la stringa "AT OK". Una nota importante è da fare per quanto riguarda il Carriage Return e il Line Feed. Infatti come si vede nella riga 7 e 8 il buffer che viene trasmesso ha al suo interno questi due caratteri terminali, e sono fondamentali, devono essere presenti alla fine di ogni stringa che viene trasmessa. Senza di loro il modulo non riconosce i comandi e quindi purtroppo da errore.

Capitolo 5

Conclusione

A questo punto dopo quanto detto, le soluzioni che sono state trovate permettono la trasmissione dei dati provenienti dal camion al microcontrollore, dove vengono memorizzati e gestiti all'interno dell'in-memory-database. In questo caso è importante proseguire lo sviluppo per quanto riguarda l'effettiva implementazione del database sulla scheda SD, per riuscire a salvare i dati su un file senza che si possano perdere con lo spegnimento del microcontrollore e avere così un sistema ancora più affidabile. Al tempo stesso sono state gettate le basi per la comunicazione con il modulo GSM, attraverso il quale nei successivi sviluppi avverrà la connessione ad una rete e quindi l'effettiva comunicazione con il sistema SDS. Saranno poi indispensabili degli esperimenti direttamente sul camion in azione per riuscire a vedere se quanto progettato sia effettivamente corretto.

Terminando la trattazione si può concludere dicendo che tutte le varie parti che sono state implementate saranno fondamentali per gli sviluppi futuri, i quali continueranno all'interno dell'azienda Eletica. I risultati ottenuti sono in linea con quanto era stato progettato, il raggiungimento degli obiettivi ha fatto si che il progetto proseguisse nella maniera corretta, evidenziando anche delle aree di miglioramento. In conclusione l'esperienza di tirocinio è stata di enorme valore perchè ha permesso di poter combinare gli aspetti teorici e pratici, e soprattutto di apprendere tantissime nozioni che si riveleranno fondamentali per il futuro.

Bibliografia

[1] STMicroelectronics. How to use the steval-stwinkt1b sensortile wireless industrial node for condition monitoring and predictive maintenance applications. *User Manual 2777*, 2024.

[2] Modulo gsm. `https://www.waveshare.com/wiki/SIM7600X_4G_%26_LTE_Cat-1_HAT`. Ultimo accesso Ottobre 2024.

[3] Enough. `https://enough-emissions.eu/`. Ultimo accesso Ottobre 2024.

[4] Rs485. `https://en.wikipedia.org/wiki/RS-485`. Ultimo accesso Ottobre 2024.

[5] Uart. `https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter`. Ultimo accesso Ottobre 2024.

[6] Fatfs. `https://en.wikipedia.org/wiki/FatFs`. Ultimo accesso Ottobre 2024.

[7] Sqlite. `https://www.sqlite.org/index.html`. Ultimo accesso Ottobre 2024.