UNIVERSITA' DEGLI STUDI DI PADOVA

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Dipartimento di Ingegneria Industriale DII

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA DELL'ENERGIA ELETTRICA

TESI DI LAUREA MAGISTRALE IN INGEGNERIA DELL'ENERGIA ELETTRICA

Studio e implementazione di un sistema di comunicazione wireless

Relatore: Prof. Manuele Bertoluzzo

Laureando: MARCO GIORIO

Matricola: 1151060

Anno Accademico 2018 - 2019

Ai miei genitori,

con immensa gratitudine

Indice

Sommario

Il protocollo di comunicazione Wi-Fi supera i limiti dei dispositivi connessi alla rete cablata incrementando notevolmente le loro capacità di comunicazione, fornisce infatti un metodo per scambiare informazioni attraverso l'etere, e permette ai dispositivi appartenenti ad una rete wireless locale di accedere alla rete di Internet, stabilendo quindi la comunicazione con un dispositivo remoto.

In questa tesi ho applicato le funzioni su cui si fonda la suite di protocolli Internet per realizzare un sistema di comunicazione wireless tra due microcontrollori. Il progetto realizzato durante l'attività di tesi si svolge in due fasi: la prima fase è stata esplorativa perché ho studiato come la libreria C, realizzata da Atmel per gestire la comunicazione Wi-Fi tra i suoi dispositivi e il modulo Wi-Fi ATWINC1500, interagisce con l'hardware, invece nella seconda fase ho eseguito il *porting* di alcuni file sorgente per conferire ad un microcontrollore della Cypress, il PSoC CY8CKIT-059 5LP, la possibilità di scambiare informazioni con il modulo Wi-Fi.

Questo progetto è stato realizzato per poter sostituire in futuro il sistema di comunicazione utilizzato dal caricabatterie wireless presente in laboratorio. Attualmente le schede di controllo del sistema utilizzano un chip ricetrasmettitore soggetto a periodi di latenza durante la trasmissione dei dati. Il protocollo Wi-Fi implementato nei moduli ATWINC1500 invece assicura tempi di attesa minori quindi dovrebbe garantire una migliore efficienza in comunicazione.

Abstract

The Wi-Fi communication protocol overcomes the limits of the devices connected to wired network, increasing considerably their communication capabilities, it gives a method to exchange informations through aether, and it allows devices belonging to a local wireless network to get access the Internet capabilities, establishing the communication with a remote device.

In this thesis I applied the Internet protocol suite functions to realize a wireless communication system between two microcontrollers. The project, developed during the thesis activity, takes place in two phases: the first one is based on the analysis of a C library, coded by Atmel, which handles the Wi-Fi communication between its devices and the Wi-Fi module ATWINC1500 and its interaction with the hardware. The second one is based on the porting of source files to confer to a Cypress microcontroller, the PSoC CY8CKIT-059 5LP, the capability of exchanging informations with the Wi-Fi module.

This project is realized in order to replace the communication system utilized by a prototipal wireless charger available in the laboratory. Actually, the control boards use a transceiver chip influenced by latency periods during data transfer. The Wi-Fi protocol implemented in ATWINC1500 modules ensures shorter waiting time and so a better communication efficency.

Introduzione

Il termine Wi-Fi fa riferimento alle tecnologie utilizzate dai dispositivi connessi alla stessa rete wireless locale per scambiare informazioni attraverso l'etere usando il protocollo 802.11. Le tecniche trasmissive hanno subito notevoli miglioramenti dalla nascita dello standard, la comunicazione è stata resa più affidabile ma soprattutto più veloce. Il data rate è stato sensibilmente incrementato passando gradualmente da 1 Mbps ad oltre 500 Mbps, anche la copertura radio è stata estesa e con l'implementazione delle tecniche multi-antenna è stato possibile raggiungere distanze superiori ai 100 [m] in ambienti privi di ostacoli.

Negli ultimi anni sono apparsi sul mercato dispositivi embedded che implementano la famiglia di standard IEEE 802.11. I moduli Wi-Fi possono essere collegati ai microcontrollori estendendo le loro funzionalità, infatti possono essere programmati in modo tale da accedere alla rete Internet e scambiare informazioni con dispositivi remoti.

In questa tesi sono state studiate le versioni b, g, n di IEEE 802.11, e la suite di protocolli di Internet che hanno permesso la realizzazione del sistema di comunicazione wireless tra due microcontrollori ATSAMD21 prodotti da Atmel. Ciascun microcontrollore è stato dotato dell'extension header ATWINC1500, un modulo Wi-Fi anch'esso prodotto da Atmel. Le conoscenze acquisite sono state poi utilizzate per implementare il protocollo IEEE 802.11 anche in altri microcontrollori, in particolare nel progetto è stato utilizzato un CY8CKIT-059 PSoC 5LP.

Lo studio del protocollo Wi-Fi e la sua successiva applicazione nel progetto di tesi, nasce dalla volontà di sostituire in futuro il sistema di comunicazione del caricabatteria wireless del laboratorio. Il chip attualmente utilizzato dalle schede di controllo per scambiare dati è un modulo che implementa un protocollo di trasmissione wireless soggetto a problemi di latenza nella gestione della comunicazione. Il dispositivo impiega 130 [μs] per cambiare stato di funzionamento da ricevitore a trasmettitore e viceversa, inoltre il data rates massimo supportato è pari a 2 Mbps. Sostituendo i chip della scheda di controllo con i moduli ATWINC1500, il sistema di comunicazione aumenterebbe notevolmente la sua efficienza, ATWINC1500 quando è connesso ad una rete in cui è implementato il protocollo IEEE 802.11n presenta data rates di 72 Mbps e latenza intrinseca pari a 10 [μs].

Di seguito è fornito un breve riassunto degli argomenti affrontati nei capitoli.

I primi quattro capitoli definiscono i concetti cardine sul funzionamento del protocollo di comunicazione utilizzato per sviluppare il progetto, in particolare saranno analizzate le differenze salienti tra gli standard della famiglia 802.11. Un'analisi preliminare riguarderà la definizione degli elementi base necessari per costituire una rete wireless di tipo locale, successivamente saranno indagate le tecniche di modulazione che contraddistinguono gli standard della famiglia 802.11.

Gli ultimi due capitoli affrontano lo sviluppo del progetto di tesi, che si suddivide in due parti: la prima parte riguarda l'implementazione del sistema di comunicazione con dispositivi ATMEL, microcontrollore e modulo Wi-Fi della stessa marca, mentre nella seconda parte si è deciso di lavorare con un microcontrollore della Cypress Semiconductor, perché tramite l'interfaccia grafica del suo ambiente di sviluppo è possibile configurare le impostazioni del sistema per la realizzazione di qualsiasi applicazione, come ad esempio il *porting* del firmware. Questa caratteristica rende l'ecosistema Cypress estremamente flessibile e di semplice utilizzo.

Lo studio di entrambi i sistemi si articola in due fasi: una descrittiva delle caratteristiche salienti del microprocessore e dei blocchi funzionali utilizzati per realizzare il progetto e l'altra analitica della componente software, soffermandosi sui passaggi più importanti dell'applicazione e quindi del codice.

Capitolo 1 - Standard IEE 802.11

La IEEE 802 Local And Metropolitan Area Network Committee venne fondata nel 1980 con l'obiettivo di realizzare il progetto IEEE 802 (Fig [1.1]), uno standard per le Local Area Network (LAN) e le Metropolitan Area Network (MAN) in armonia con il modello ISO/OSI (v.di Appendice A). Nacquero diversi comitati che parallelamente lavoravano allo sviluppo di protocolli di comunicazione di livello Fisico e Data Link, come ad esempio Ethernet (802.3) e Token Ring (802.5), con l'obiettivo comune di fornire un'interfaccia univoca verso il livello Rete. Per ottenere questo risultato il livello Data Link venne suddiviso in due sotto-livelli: il sotto-strato Medium Access control (MAC) differente per ogni protocollo e il sotto-strato Logical Link Control (LLC) comune a tutti i protocolli.

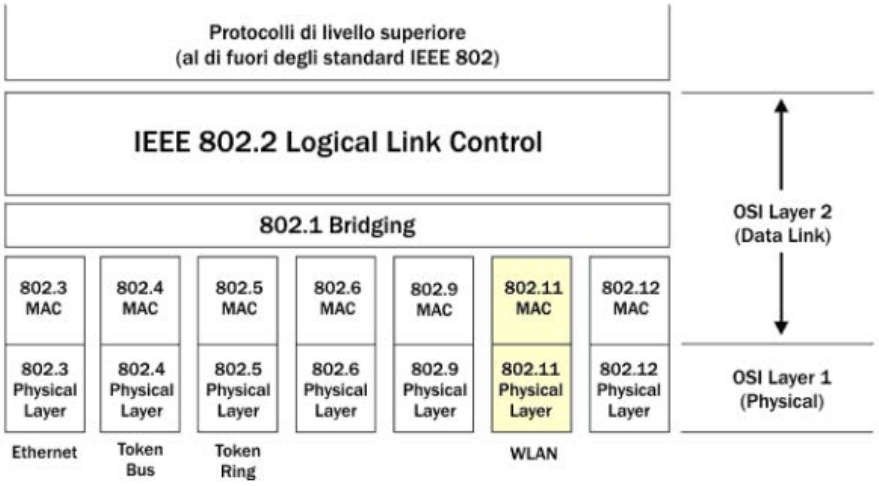


Fig.1.1 - Protocolli 802 http://www.comlab.uniroma3.it/Sistemi1/tesiWLAN21.pdf

Nel 1987 una commissione internazionale costituita da professionisti provenienti sia dal mondo accademico che dal mondo aziendale, la *802.11 Wireless Local Area Network Standards Working Group* incaricata di sviluppare uno standard globale per le reti wireless che lavorano nella banda dei 2.4 GHz, con data rates 1 e 2 Mbps. Solo nel 1997 fu definitivamente approvato col nome di IEEE 802.11.

Lo sviluppo di nuove tecnologie e il conseguente superamento dei limiti tecnologici hanno garantito il miglioramento del protocollo IEEE 802.11 o Legacy Mode, portando alla luce standard più performanti in termini di data rates, latenze e range di trasmissione.

Oggi il termine IEEE 802.11, noto come Wi-Fi (Wireless Fidelity) Alliance, definisce un insieme di protocolli per lo scambio di dati in una WLAN (*Wireless Local Area Network*), una rete circoscritta geograficamente che utilizza le radio frequenze per la comunicazione di dati. I dispositivi associati a queste reti possono lavorare nel campo 2.4-2.4835 GHz o 5.725 – 5.785 GHz della banda ISM (*Industrial-Scientific-Medical band*). Questi standard definiscono gli strati più bassi del modello ISO/OSI, in particolare forniscono le specifiche per quanto riguarda il *Medium Access Control (*MAC*)* sublayer, che si occupa della gestione della comunicazione, e del *Physical (*PHY) layer, incaricato di fornire un canale per lo scambio di informazioni. (Fig[1.2])

Gli studi riguardanti il Wi-Fi e lo sviluppo di nuove tecnologie hanno permesso la nascita di nuovi standard appartenenti alla famiglia 802.11, che nel tempo ha mantenuto invariata la gestione della comunicazione, definendo però differenti tecniche di modulazione del segnale a livello PHY, ognuna dipendente dalla modalità di trasmissione utilizzata.

Sicuramente la caratteristica su cui si è insistito maggiormente nella definizione dei nuovi standard riguarda l'interoperabilità, ovvero la capacità di poter scambiare frame tra dispositivi che implementano tecniche trasmissive differenti.

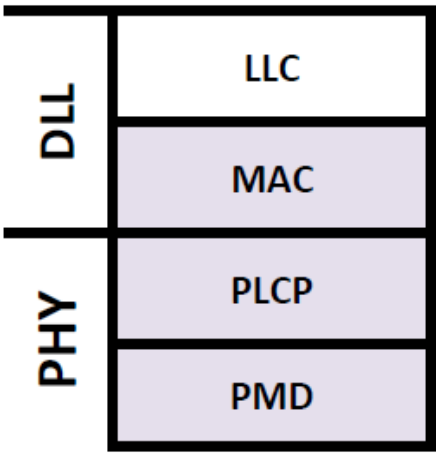


Fig.1.2- Livello PHY e DLL del Protocollo 802.11 Standard[10]

Architetture di una rete Wi - Fi

L'elemento base di una rete WLAN (Wireless Local Area Network) è la stazione (spesso abbreviata come STA, un qualsiasi dispositivo in grado di ricreare il protocollo 802.11. Le stazioni possono essere classificate in mobili ad esempio i telefoni cellulari o statiche quando sono connesse al sistema di distribuzione (DS) cablato, come ad esempio l'Access Point.
Un insieme di stazioni può interconnettersi formando un *Basic Service Set* (BSS), un gruppo di dispositivi che operano con le stesse caratteristiche di accesso al mezzo, quali modulazione, radio frequenza e data rates.

Lo standard 802.11 Legacy Mode definisce i protocolli per due tipologie di reti:

- Peer to peer (P2P)
- Client/Server.

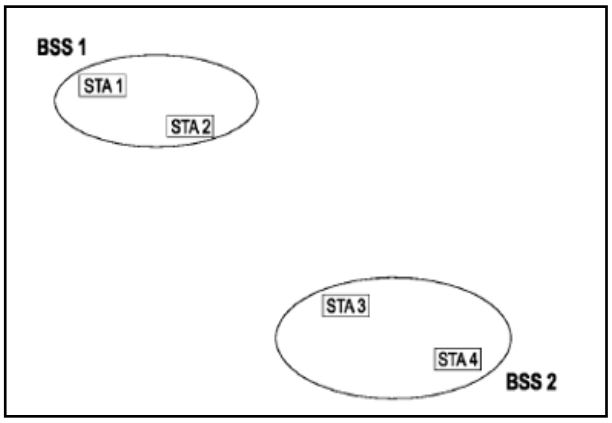


Fig.1.3 - Rappresentazione di due Reti BSS

Nel primo caso Fig [1.3] è possibile costituire una Independent *Basic Service Set* (IBSS) nota anche con il nome di "Rete Ad Hoc", nella quale due o più stazioni possono comunicare direttamente entro un certo raggio.

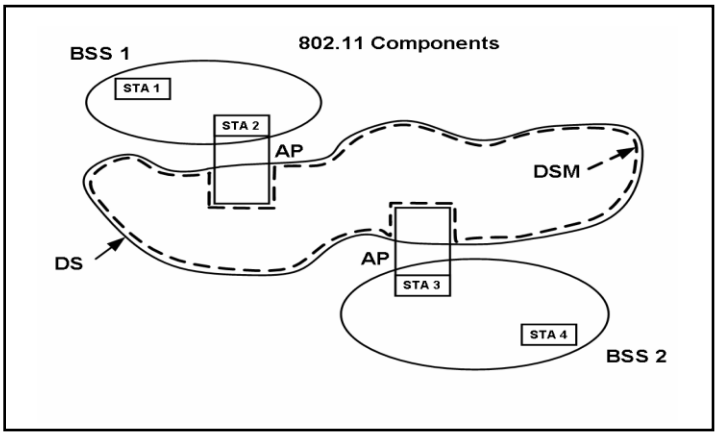


Fig.1.4 - Rappresentazione di un sistema BSS Infrastructured

Nel secondo caso Fig [1.4] è possibile costituire una BSS di tipo *Infrastructured*, superando di fatto i limiti fisici legati alla massima distanza di trasmissione ammissibile tra due STA che si scambiano frame tramite tecnologia wireless. Risulta quindi possibile porre in comunicazione più BSS utilizzando un AP che ha il compito di permettere l'accesso al DS.

Physical Layer

Il Physical Layer (PHY), lo strato più basso del modello ISO/OSI, ha due compiti: associare il frame proveniente dallo strato MAC ad un Overhead costituito da un preambolo e da un'intestazione, e trasmetterlo attraverso l'etere, utilizzando la tecnica di modulazione più adeguata. L'Overhead contiene delle informazioni fondamentali per garantire l'interoperabilità, la STA ricevente in questo modo può adattarsi al suo standard per scambiare informazioni.

Per soddisfare entrambe le funzioni lo strato PHY è suddiviso in due sub-layer: il Physical Medium Dependent (PMD) sub-layer, lo strato più vicino al mezzo di trasmissione che si occupa della modulazione del segnale e il Physical Layer Convergence Procedure (PLCP) sub-layer che permette di interfacciare il substrato MAC con quello PMD. Generalmente la porzione PLCP del frame è inviata ad un data rates minore, questa procedura garantisce che il messaggio sia ricevuto con successo. La scelta di inviarlo ad una velocità più bassa garantisce una migliore immunità ai disturbi, il carico che trasporta è particolarmente sensibile, sono contenute informazioni

riguardanti sia la sincronizzazione con la rete che la sua gestione, fornisce ad esempio la tecnica di modulazione implementata dalla STA e il data rates supportato.

Le tecniche principali di modulazione adottate dalla famiglia di standard 802.11 sono la Direct Sequence Spread Spectrum (DSSS) e la Orthogonal Frequency Division Multiplexing (OFDM).

Tecniche DSSS

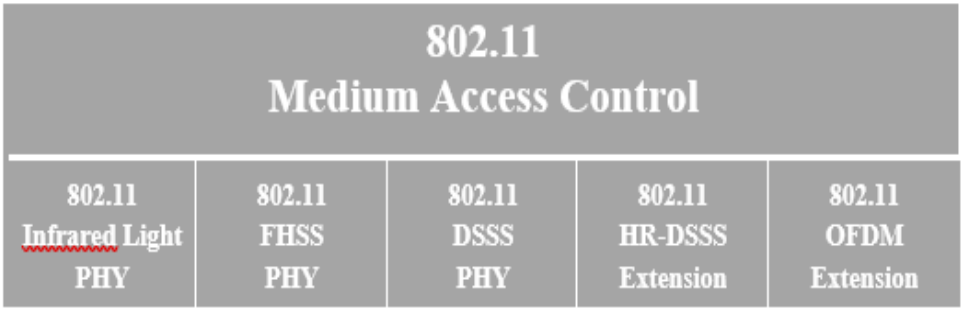


Fig.1.5 - Tecniche di modulazione ammesse dalla famiglia 802.11

La tecnica DSSS, implementata per la prima volta nello standard Legacy Mode e poi successivamente estesa a tutti gli standard, si basa sul concetto di ampliare lo spettro del segnale trasmesso, in modo tale che il ricevitore sia in grado di discriminare tra segnale utile e rumore, garantendo una maggiore robustezza alle interferenze.

Questa tecnica trasmette ogni singolo bit ad una determinata frequenza, l'ampliamento dello spettro è permesso applicando al valore logico alto e basso, una predeterminata sequenza ridondante di 11 bit o chips, definita sequenza di Barkley. Di seguito viene definita la sequenza per identificare lo zero e l'uno logico:

- 0 → 01001000111
- 1 → 10110111000

Inizialmente vennero previsti due tipi di modulazione basati sullo sfasamento dell'onda trasmessa.

La prima fu la **Differential Binary Phase Shift Keying** (DBPSK) con data rate 1 [Mb/s], che si basa sulla traslazione di 180° del segnale inviato per trasmettere un livello logico basso e sul mantenimento della fase per trasmettere un livello logico alto, riuscendo ad inviare un simbolo ogni microsecondo. Un simbolo corrisponde a 11 chips.

La modulazione **Differential Quaternary Phase Shift Keying** (DQPSK)con data rate 2 [Mb/s] invece sfrutta la possibilità di traslare il segnale inviato di 0°,90°,180° e 270° permettendo di inviare in un colpo solo due simboli ogni microsecondo.

DBPSK	
Diff. fase	Simbolo
0°	0
180°	1

DQPSK	
Diff. fase	Simbolo
0°	00
90°	01
180°	11
270°	10

Tabella 1 - Simboli trasmessi in base alle modulazioni DBPSK e DQPSK

Lo standard IEEE 802.11b noto anche **High-Rate DSSS** (HR/DSSS) mantiene invariata la tecnica di modulazione, sostituendo la sequenza di Barkley con la Complementary Code Key (CCK), i valori binari reali sono rimpiazzati da valori complessi quaternari (± 1, ± j), raggiungendo data rate di 5.5[Mb/s] o 11[Mb/s]. Questa tecnologia permette di codificare più bit in un simbolo.

La versione "b" a livello PLCP integra due tipi di Overhead per migliorare le prestazioni: con preambolo breve e con preambolo lungo, la lunghezza dell'intestazione rimane invariata. Questo stratagemma serve soprattutto per gestire in maniera più efficace la rete. I frame di controllo e i frame video real-time, un tipo di dati sensibili al ritardo, sono inviati con il preambolo breve con la tecnica DBPSK o DQPSK per compatibilità tra i dispositivi. La parte del frame legata agli strati superiori del modello ISO/OSI è trasmessa secondo lo standard implementato nell'AP.

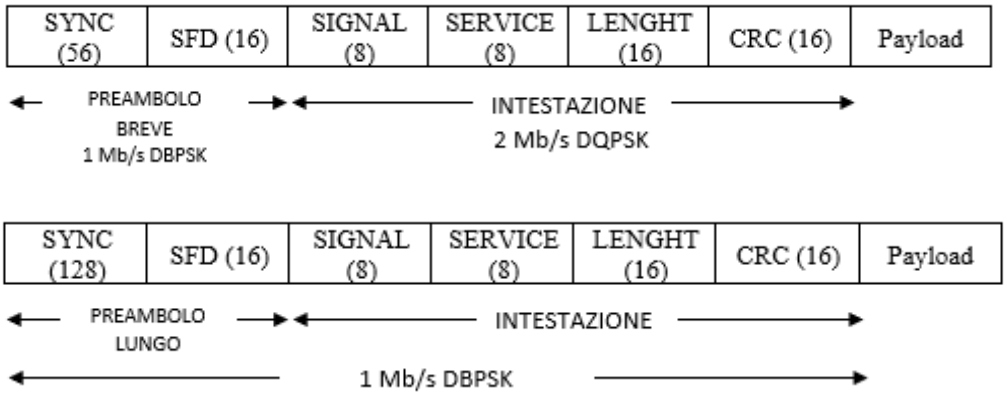


Fig. 1.6 - a) Frame PLCP con Preambolo breve.
b) Frame PLCP con Preambolo lungo compatibile con 802.11

Tecnica OFDM

Un data rates più alto fu possibile con l'introduzione della tecnica Orthogonally Frequency Division Multiplexing (OFDM) a partire dallo standard IEEE 802.11g in Europa e IEEE 802.11a in America. Questa tecnica si basa sulla divisione della banda disponibile in molti canali di banda relativamente stretta, a ciascuno dei quali corrisponde una sotto-portante che sarà modulata in maniera indipendente. Il flusso dei dati inviati è suddiviso in più parti trasmesse in parallelo con velocità massima raggiungibile complessivamente 54 [Mbps]. La tecnica OFDM permette di inviare un simbolo ogni 4 microsecondi, quindi in un simbolo sono codificati fino a 216 bit dati. Ad ogni simbolo sono inoltre associati 72 bit di correzione, per un totale dunque di 288 bit per simbolo divisi tra 48 sotto-portanti, quindi 6 bit per sotto-portante.

La nascita di questo standard ebbe come conseguenza problemi di retrocompatibilità con le versioni precedenti che non implementavano la tecnologia OFDM. Si definirono quindi quattro livelli PHY differenti per garantire la convergenza con lo standard "b" e Legacy Mode. A livello PLCP, la versione "g" presenta come la versione "b" l'overhead sia con preambolo lungo che corto, ma sempre trasmesso con le tecniche DSSS. La parte dati del frame invece può essere trasmessa sia con tecnica DSSS se in rete sono presenti STA non aggiornate alla versione "g", altrimenti sono trasmessi con tecnica OFDM.

La diffusione e il successo dello standard Wi-Fi hanno portato delle conseguenze in merito alla gestione delle frequenze nel range dei 2.4 GHz (Fig. [1.7]). L'affollamento delle reti, soprattutto nelle aree urbane, ha creato dei problemi di interferenza che furono risolti dapprima con l'introduzione di 14 canali distinti di ampiezza 22 MHz ciascuno, ad una distanza di 5MHz l'uno dall'altro. In Europa vennero identificati tre canali non sovrapposti per la trasmissione, l'1, 6, 11,

distanti 25 MHz l'uno dall'altro, ciò significava che 3 AP potevano coesistere nella stessa area senza interferenza.

La densità di utilizzo della banda dei 2.4 GHz portò ad implementare, a partire dallo standard IEEE 802.11n l'utilizzo della banda ISM dei 5 GHz, in cui i canali non sovrapposti sono 19 e presentano un'ampiezza di banda di 20 MHz.

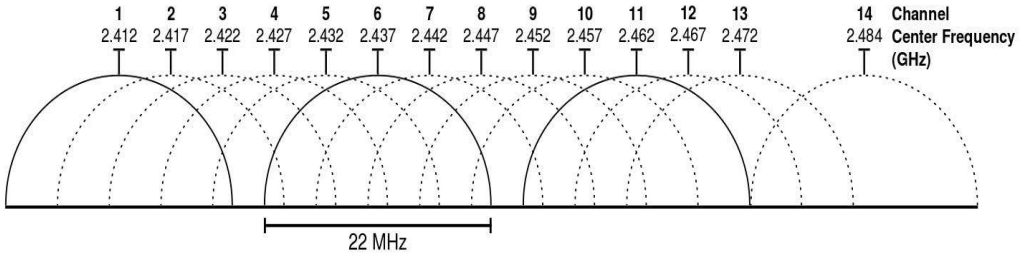


Fig.1.7: Canali di 22 [MHz] disponibili nella banda dei 2.4 [GHz] - https://www.netspotapp.com/it/wifi-channel-scanner.html

La versione "n" continua ad utilizzare la modulazione OFDM (1 simbolo ogni 4 microsecondi) però introduce nuove tecniche per migliorare il data rates come ad esempio l'aumento del numero di sotto-portanti, portandole complessivamente da 48 a 52 o l'utilizzo di canali da 20 MHz. Questi miglioramenti permisero di raggiungere i 65 [Mb/s] in trasmissione. Fino alla definizione dello standard "g" le STA erano classificate come sistemi *Single Snput Single Output* (SISO) perché utilizzavano sia in trasmissione che in ricezione una singola antenna. La nascita dello standard "n" portò delle migliorie a livello Fisico che incrementarono notevolmente le prestazioni, i dispositivi iniziarono ad utilizzare fino a 4 antenne e vennero classificati come sistemi *Multiple Input Multiple Output* (MIMO) portando il data rate massimo a 260[1] [Mb/sec]. Questo standard prevedeva inoltre la tecnica opzionale *Channel Bonding* che consiste nell'utilizzare due canali adiacenti di 20 MHz definendo dunque un unico canale con larghezza 40 MHz (Fig.[1.8]). L'unione tra i due canali permette di incrementare ulteriormente il numero di sotto-portanti, arrivando a ottenere 108 poiché si riesce a ricavare 4 sotto-portanti aggiuntive (52+52+4), garantendo una velocità fino a 540 [Mb/s] con quattro antenne in trasmissione.

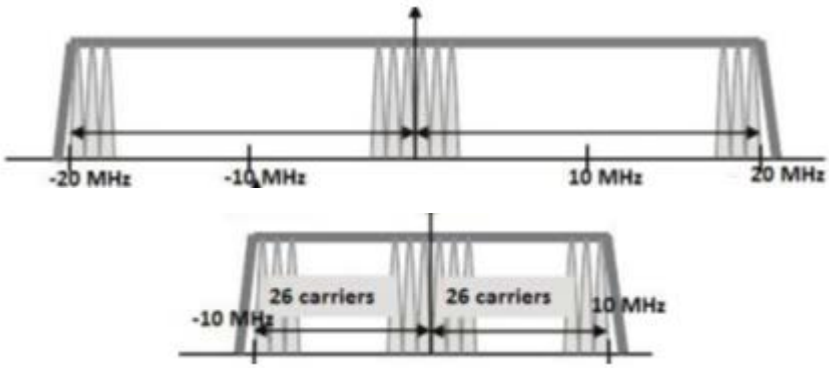


Fig 1.8 - Canali da 20 MHz e 40 MHz nella banda 2.4 GHz

[1] Con l'utilizzo di quattro antenne.

Tecnologia MIMO

La tecnologia MIMO garantì notevoli miglioramenti nell'ambito delle comunicazioni.

wireless, la trasmissione e ricezione di segnali multipli permise di aumentare sensibilmente sia il throughput che la distanza di trasmissione negli ambienti indoor, in cui i vari ostacoli fisici riducono la potenza del segnale fino a renderlo indistinguibile dal rumore.

Le antenne multiple sono necessarie negli ambienti in cui i dispositivi in comunicazione non si vedono reciprocamente, quindi il segnale non arriva direttamente ma tramite riflessioni multiple su superfici metalliche. Il ricevitore percepisce gli effetti negativi del multipath, ricevendo copie del segnale originale sfasate nel tempo che possono dare luogo ad interferenza distruttiva o costruttiva in maniera del tutto casuale. L'utilizzo di antenne ricetrasmittenti amplifica notevolmente questo effetto, lo standard 802.11n sfrutta a proprio vantaggio il multipath per migliorare la qualità della trasmissione in termini di velocità.

Esistono differenti tecniche MIMO che si contraddistinguono per la difficoltà di implementazione. Il **Transmit Beamforming** è un metodo che permette di realizzare sempre un'interferenza di tipo costruttiva cercando di far arrivare al ricevitore i segnali in fase. Lo **Space Time Block Coding** migliora l'affidabilità della trasmissione perché trasmette lo stesso segnale riordinato su un'antenna differente da quella iniziale sfasato nel tempo (in un momento successivo) garantendo la correzione dell'errore ma dimezzando il data rates. **L'equalizzatore MIMO** è una tecnica utilizzata nei dispositivi che in trasmissione presentano una singola antenna e in ricezione due. Il segnale inviato raggiungerà le antenne riceventi tramite percorsi multipli che possono causare sia interferenza distruttiva che costruttiva. La distanza tra le due antenne però garantisce che il segnale captato non generi interferenza distruttiva su entrambe le antenne, quindi attraverso particolari tecniche di somma applicate ai segnali captati si potrà ottenere nuovamente il segnale originale.

Di seguito un confronto schematico tra le caratteristiche salienti degli standard della famiglia IEEE 802.11, precedentemente descritti (Tabella 2).[2]

802.11 PHY Standard							
Protocollo	Frequenza [GHz]	Banda [MHz]	Data Rate[1] [Mbps]	MIMO	Modulazione	Range [m]	
						In	Out
802.11	2.4	22	1 - 2	1	DSSS	20	100
b	2.4	22	1 - 11	1	DSSS(SISO)	35	140
g	2.4	20	6 - 54	1	DSSS-OFDM (SISO)	38	140
n	2.4 - 5	20	6.5 - 65	4	DSSS-OFDM (MIMO)	70	250
		40	13.5 - 135				

Tabella 2: Caratteristiche principali degli standard 802.11, 802.11b, 802.11g, 802.11n

[2] Valore minimo e massimo con singola antenna trasmittente e ricevente

Capitolo 2 - Medium Access Control Layer

Il sotto-strato MAC del livello Data Link svolge i seguenti compiti:

- **Assemblaggio dei Frame**
- **Gestione della mobilità**
- **Accesso al mezzo**

Assemblaggio dei Frame

Ogni frame[3] è costituito da un *MAC Header* (Fig. [2.1a]) che fornisce informazioni sulla gestione della rete, un F*rame Body* (Fig. [2.1b]) di lunghezza variabile tra 0 - 2312 byte che rappresenta il payload proveniente dai livelli superiori del modello ISO/OSI ed infine un CRC di 4 byte.

I campi più significativi del *MAC Header* sono: il *Frame Control* e i campi *Duration/ID* e

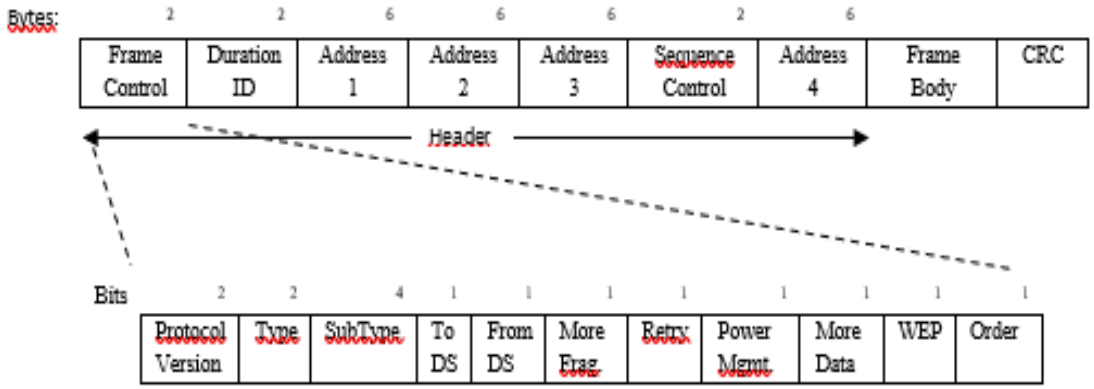


Fig 2.1 - a) MAC Header del frame 802.11. b) Campo Control del MAC Header

Sequence Control entrambi forniscono informazioni sulla lunghezza del *Frame Body* e della sua eventuale frammentazione. Il F*rame Control* è di fondamentale importanza per la rete, i sotto-campi *Type* e *Subtype* servono per distinguere tra i tre tipi di frame possibili: Gestione, Controllo e Dati. I Flag *To DS* e *From DS* indicano se il frame è diretto o proveniente dal DS ed abilitano, in base all'impostazione dei bit, i quattro campi *Address* che definiscono gli indirizzi fisici o MAC address, la cui importanza sarà chiarita nel successivo capitolo.

Frame di Controllo

In questo paragrafo sono descritti i Frame di controllo Acknoledgment, Clear To Send e Request To send, successivamente saranno introdotti i frame di gestione utilizzati per il processo di accesso di una nuova STA ad una rete già esistente.

[3] Il frame è una struttura composta da un numero definito di bit, ciascuno dei quali presenta uno specifico significato.

Frame Acknolewdgment (ACK): Permettere di capire se la STA in trasmissione è riuscita ad inviare con successo il frame, rappresenta un feedback positivo sull'avvenuta ricezione. Non ricevere questo messaggio significa che il pacchetto inviato dall'AP è andato in contro ad una collisione.

Frame Clear To Send (CTS) - Request To Send (RTS): Sono necessari per poter applicare in sicurezza la modalità di accesso al mezzo di tipo Collision Avoidance (CA). Questa tecnica permette alle STA che non si trovano nello stesso raggio di copertura radio, quindi che non sono in visibilità reciproca, di comprendere se il mezzo trasmissivo è occupato. La STA trasmittente, dopo essersi accertata che il mezzo trasmissivo è libero, invece di trasmettere direttamente il payload invia un frame RTS, mentre la STA in ricezione (AP) risponde con un frame CTS. Il terminale nascosto, cioè quello che non trasmette ma è sempre in ascolto, è rappresentato dalla STA C in Fig[2.2a]. La ricezione del frame CTS permette di capire a STA C che il mezzo è occupato e che deve astenersi dal trasmettere per un tempo definito dal campo Duration (Fig. [2.2b])

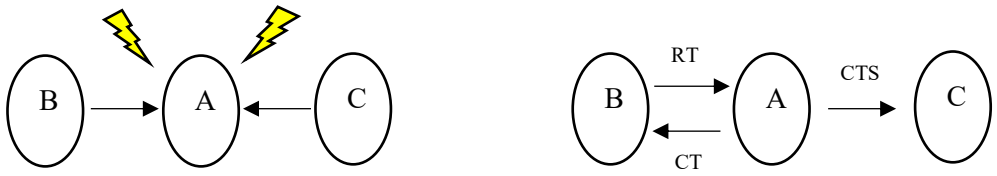


Fig. 2.2 - Il primo schema rappresenta la collisione tra i frame della STA C e B mentre il secondo schema rappresenta il meccanismo RTS-CTS.

Frame di Gestione: Accesso ad una BSS

Una STA che decide di accedere ad un BSS esistente deve fornire le proprie credenziali all'AP tramite **il processo di autentificazione,** una volta superata questa fase è necessario che ottenga le informazioni di sincronizzazione dall'AP tramite **il processo di associazione**.

Esistono due tecniche per accedere ad una rete, la prima è chiamata **scansione passiva** (Fig [2.3a]), l'AP invia ogni 100 [msec] un frame di Beacon in cui è contenuto il suo SSID, cioè il nome della rete a cui associarsi, un Timestamp e altre informazioni come ad esempio il data rate. (Fig [2.4])

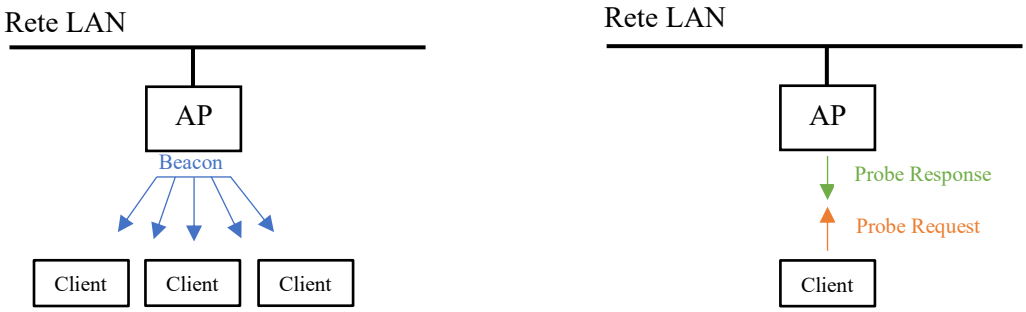


Fig 2.3 - a) Scanning passivo b) Scanning attivo

La STA inizierà il processo di associazione e nell'eventualità che ci siano più AP nella stessa WLAN si assocerà con l'AP che invia un Beacon con livello di potenza più alto.

La seconda tecnica è invece chiamata **scansione attiva** (Fig [2.3b]) e permette un'associazione più rapida ma energeticamente meno efficiente perché la STA prova direttamente a cercare un AP trasmettendo il frame di Probe Request e aspettando per il frame di Probe Response dall'AP.

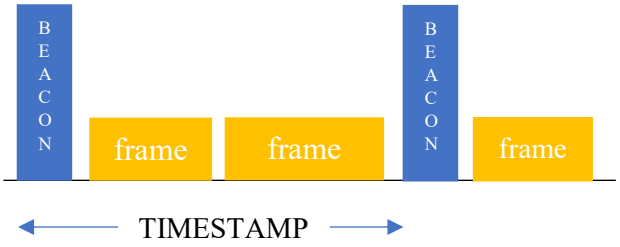


Fig.2.4 - Rappresentazione dell'intervallo di tempo Timestamp

In un sistema a chiave condivisa l'AP invia un testo crittografato, se la STA ha la chiave di decriptazione corretta, può inviare il frame di associazione che permette all'AP di allocare risorse e rispondere con il data rate supportato dalla rete.

Una STA associata può decidere di de-autentificarsi da una rete tramite i frame disassociazione e deautentificazione, comunicando all'AP la fine di un'associazione prima dello spegnimento. Per ragioni di manutenzione o aggiornamento è possibile che sia necessario spegnere l'AP, quindi tutte le STA si troveranno in uno stato di disassociazione. Anche in questo caso viene notificato il cambio di stato tramite un frame poco prima dello spegnimento.

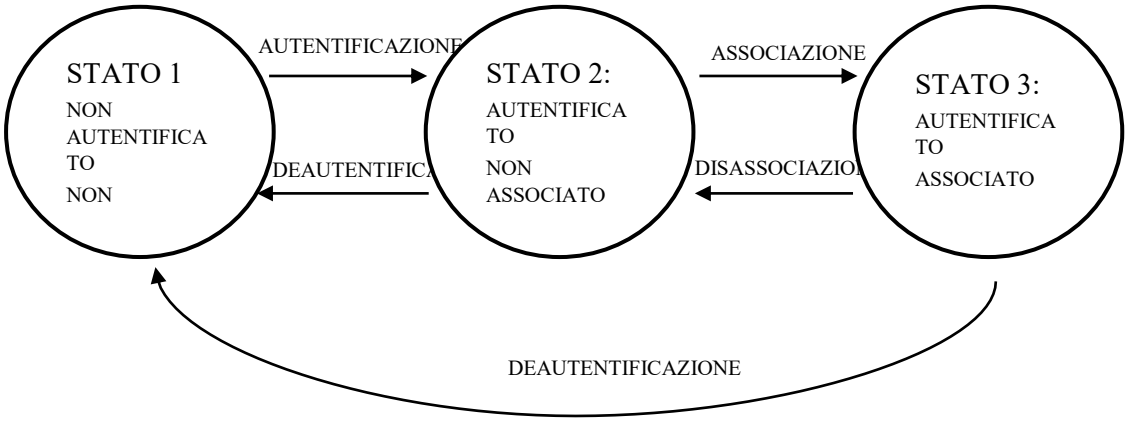


Fig 2.5 - Macchina a stati che descrive il processo di autentificazione/associazione/deautentificazione

MAC: Collision Avoidance

Il compito più importate ed anche il più complicato che deve essere svolto dal sotto-strato MAC è sicuramente il controllo dell'accesso al mezzo. La famiglia di standard IEEE 802.11 utilizza

come metodo fondamentale di accesso il Distributed Cordination Function (DCF), che si basa sul Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA).

La modalità DCF è stata progettata per ridurre al minimo la probabilità di collisione tra frame inviati contemporaneamente da STA differenti. La tecnica utilizzata si basa sull'attesa di un certo intervallo di tempo, sul meccanismo dei pacchetti ACK, RTS, CTS e sull'algoritmo di back-off.

Il momento più critico per la trasmissione si verifica quando il mezzo risulta libero, subito dopo che una STA ha terminato la propria comunicazione e due STA che vogliono inviare un messaggio trasmettono contemporaneamente il loro frame, generando una collisione.

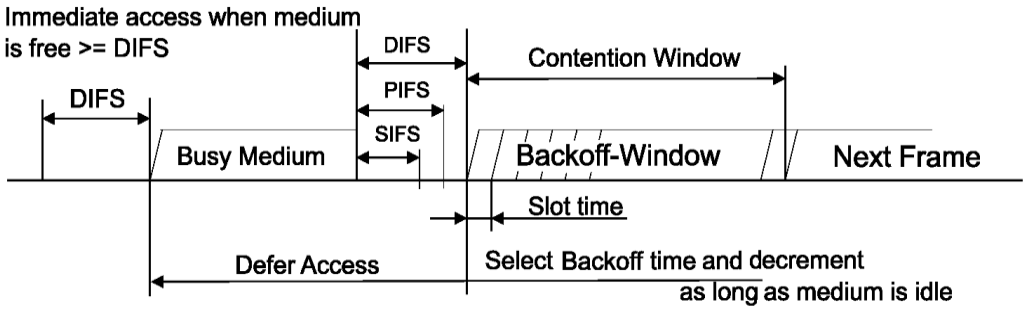


Fig 2.6 - Rappresentazione dei tempi di attesa

Il problema della contesa del mezzo viene superato utilizzando l'algoritmo Exponential Backoff che si basa sull'assegnazione di un intervallo di tempo, costituito da un numero di slot time casuale. L'intervallo assegnato rappresenta il tempo che la STA deve attendere prima di poter prenotare il mezzo, questo valore deve essere inferiore alla durata massima della Contention Window (CW). Fig.[2.6], cioè il massimo intervallo di tempo in cui è possibile prenotare il mezzo. Uno slot time viene definito come il tempo minimo che l'informazione impiega a propagarsi attraverso il canale, attraversare lo strato PHY e arrivare allo strato MAC cambiando successivamente la modalità di funzionamento della STA da ascolto a trasmissione.

Quando il conteggio del tempo di back-off di una STA arriva a zero, inizia il proprio turno di trasmissione per la STA che prenota il mezzo. Il processo di trasmissione inizia con l'invio dei frame RTS/CTS con le modalità descritte precedentemente, in particolare il campo Duration/ID del frame RTS/CTS contiene il tempo per cui il mezzo sarà impegnato. La STA che ha prenotato il mezzo può iniziare la trasmissione vera e propria del payload.

Un pacchetto è stato inviato con successo se la STA riceve in risposta un frame di ACK. Nel caso fosse necessaria un'operazione di frammentazione del frame, al fine di mantenere la sincronizzazione con l'AP ed evitare problemi di collisione, l'invio di un pacchetto ACK è necessario ogni volta che viene trasmesso un frammento.

La gestione efficace del meccanismo CSMA/CA è stata possibile grazie alla definizione di tre intervalli di tempo la cui durata dipende dal grado di priorità del frame, ad esempio i frame di controllo, cioè quelli a priorità maggiore, dovranno attendere il tempo minore

- **Short Inter Frame Space** (SIFS) è l'intervallo più breve ed è utilizzato quando una STA ha prenotato il mezzo e deve sapere se il frame inviato è arrivato con successo. Questa caratteristica permette che una STA per tutta la durata della trasmissione mantenga sempre la priorità sulle altre STA anche se dovesse essere frammentato il frame.

- **Point Cordinator IFS** (PIFS) è l'intervallo utilizzato quando una STA lavora in modalità PCF, garantisce un accesso prioritario al mezzo oppure serve per inviare un frame che annunci il cambio di canale.
- **Distributed IFS** (DIFS) è l'intervallo utilizzato quando una STA lavora in modalità DCF e corrisponde al tempo minimo in cui il mezzo rimane a riposo.

	SIFS [µs]	PIFS [µs][2]	DIFS [µs][3]	Slot time [µs]
802.11	10	30	50	20
802.11b	10	30	50	20
802.11g[4]	10	19 o 30	28 o 50	9 o 20
802.11n[4] (2.4 GHz)	10	19 o 30	28 o 50	9 o 20
802.11n (5 GHz)	16	25	34	9

Tabella 3 - Intervalli di tempo SIFS, PIFS, DIFS definiti per la famiglia di standard 802.11.

Per comprendere meglio il meccanismo della prenotazione del mezzo quando si utilizza la tecnica CSMA/CA di seguito viene fornito un esempio pratico Fig. [2.7].
La STA A è la prima che impegna il mezzo trasmissivo, durante l'invio del frame le STA B, C, D decidono di trasmettere un messaggio ma non possono perché gli era stato comunicato tramite RTS il tempo necessario da attendere prima di poter considerare il mezzo a riposo.
Terminata la comunicazione della STA A e dopo aver atteso un DIFS viene assegnato alle STA B, C, D tramite algoritmo di Backoff, un intervallo di attesa casuale, inizia il periodo di contesa.
La STA C vince la contesa e inizia la trasmissione del proprio frame perché le è stato assegnato un valore più basso, al termine della comunicazione e dopo aver atteso un DIFS ricomincia il periodo di contesa che vede come protagonisti STA B, D che ripartono dal valore di Backoff residuo, ma anche la STA E a cui viene assegnato un valore di back-off casuale.
Le successive finestre di contesa (CW) seguono le stesse modalità della prima descritta.

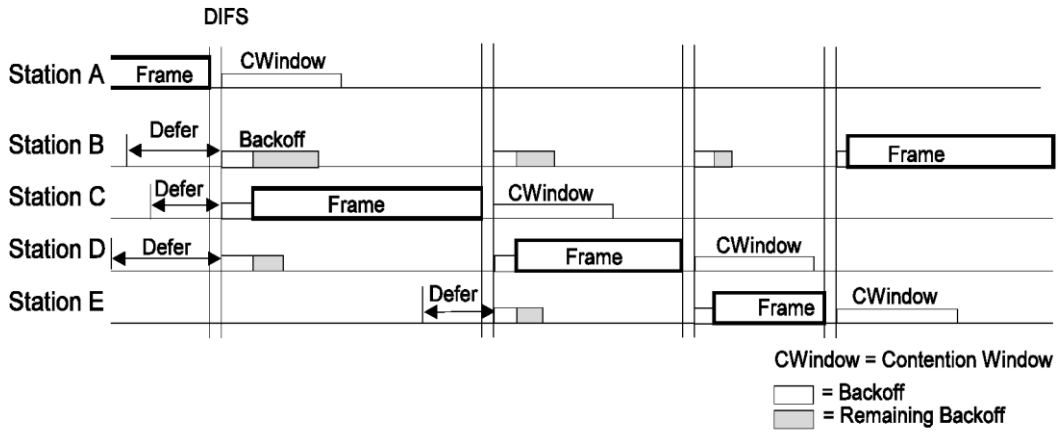


Figura 2.7 - Esempio di un caso di accesso al mezzo con modalità DCF.

MAC: Carrier Sense

Il Carrier Sense è una procedura necessaria a determinare se il mezzo trasmissivo è occupato o in stato di riposo, nel caso di comunicazioni wireless ed in particolare per la famiglia IEEE 802.11 sono definite due tecniche spesso usate in combinazione: il Carrier Sense virtuale implementato dal sotto-livello MAC che consiste nell'aggiornamento del Network Allocation Vector (NAV) e il Physical Carrier Sense.

Il NAV può essere considerato come un contatore e fornisce una previsione temporale sulla durata della trasmissione in corso, indicando per quanto sarà occupato il mezzo, questa informazione è reperita dal campo Duration dei pacchetti RTS/CTS.

Il NAV associato al RTS rappresenta il tempo necessario a trasmettere il frame, i pacchetti ACK e CTS ed i tre SIFS relativi, mentre il NAV relativo al CTS è ridotto coerentemente. Le STA non in trasmissione che ricevono l'RTS e successivamente il CTS aggiornano il contatore del NAV e aspettano la fine dell'intervallo per iniziare una nuova contesa. La STA che ha inviato la richiesta di trasmissione legge l'indirizzo contenuto nel campo address del pacchetto CTS e capisce che non deve aggiornare il suo NAV.

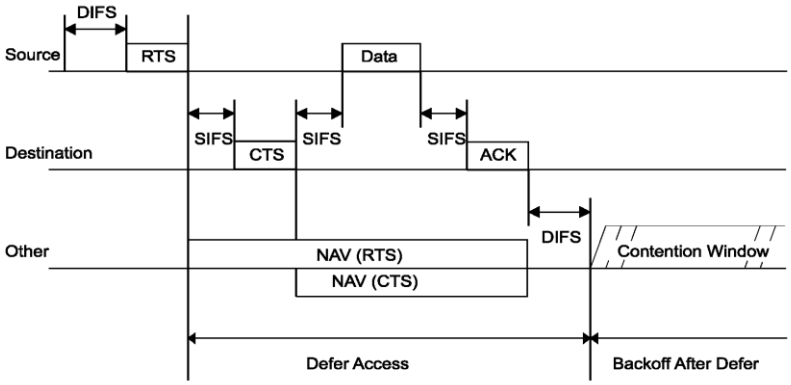


Fig 2.8 - Rappresentazione dell'intervallo di attesa NAV.

La Fig. [2.9] evidenzia il meccanismo del NAV nel caso di frammentazione del payload. Come precedentemente descritto, la trasmissione del frame dati ha inizio solo dopo lo scambio dei pacchetti RTS-CTS, il loro campo Duration indica a tutte le STA in ascolto quanto tempo è necessario affinché la trasmissione del primo pacchetto vada a buon fine.

Se la trasmissione del primo frammento dati non ha successo il mezzo torna rapidamente libero. In caso di esito positivo, il pacchetto ACK inviato dall'AP in risposta al frammento, contiene al suo interno il tempo necessario ad inviare il prossimo frammento dati.

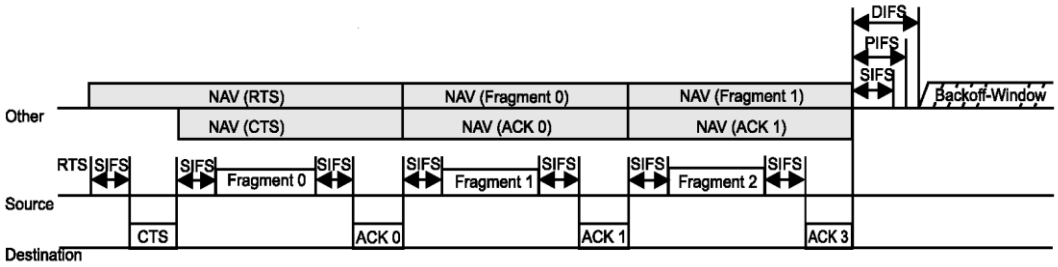


Fig 2.9 - Rappresentazione della trassmisione di frammenti

Il processo descritto si ripete finché la STA in trasmissione non termina la comunicazione o si verifica un errore.
Il Physical Carrier Sense invece è implementato a livello PHY dal sub-layer PLCP attraverso la funzione *Clear Channel Association* che valuta se il mezzo è occupato analizzando o il livello del segnale o i bit ricevuti. Questo metodo è più efficace perché si basa sulla rilevazione dei dati ma è decisamente più lento rispetto al primo che però può provocare falsi allarmi a causa della presenza di interferenze.

Point Coordination Function

Il Point Coordination Function (PCF) è un meccanismo di accesso al mezzo costruito al di sopra del DCF, permette un controllo più stringente sulla trasmissione. Una STA speciale, generalmente individuata con l'AP, quando è abilitata a diventare PCF gestisce l'accesso al mezzo garantendo il trasferimento di un frame senza contesa, quindi i messaggi di RTS e CTS non sono necessari in questo meccanismo.

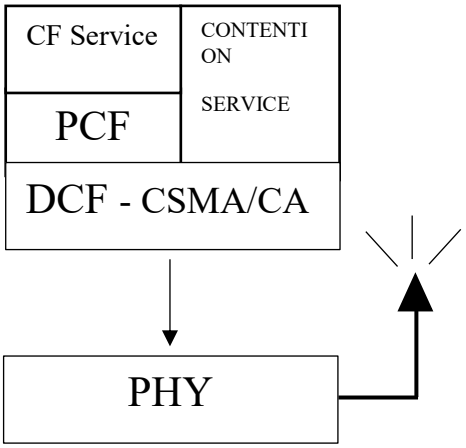


Fig 2.10 -Struttura dei meccanismi di accesso al mezzo

Il PCF è una tecnica opzionale per avere un controllo di tipo centralizzato, se attivata si alternano periodi liberi da contesa a periodi in cui è abilitato il meccanismo DCF precedentemente descritto.

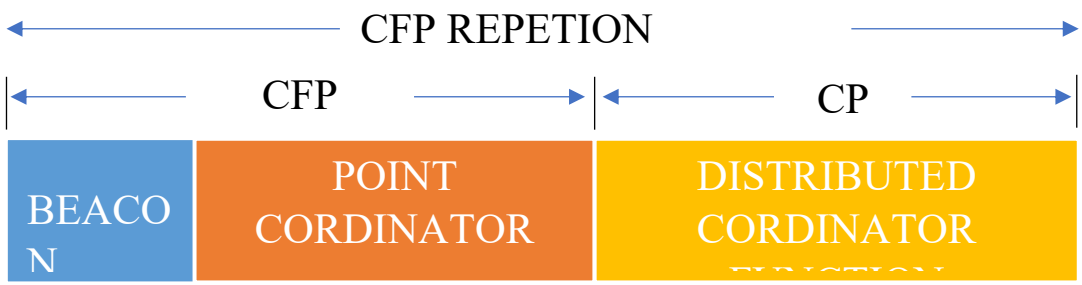


Fig 2.11 - Coesistenza di CFP e CP

All'interno di un Contention Free Period repetition interval, sono definite due porzioni di tempo, il Contention Free Period (CFP) regolato secondo la modalità PCF e il Contention Period (CP), gestito secondo il metodo DCF (Fig [2.11]). L'AP in base alle condizioni di traffico può decidere quanto tempo del CFP repetition interval allocare per il PCF.

Dopo essere trascorso un PIFS dall'inizio del Contention Free Period (CFP), se il mezzo trasmissivo è libero, Il PCF ne prende il controllo e invia un frame di Beacon a tutte le stazioni in ascolto. Le STA che ricevono questo messaggio aggiornano il loro NAV con il valore *CFPMaxDuration* che indica la durata massima del CFP. Dopo aver trasmesso il Beacon il PCF può inviare uno dei seguenti frame:

- **Frame Dati** trasmessi ad una particolare STA, o a tutte le STA.
- **Frame di CF-Poll** inviato ad una STA, indica che quella STA ha il permesso di inviare un singolo Frame a chiunque voglia. Se la STA non deve inviare dati dovrà trasmettere un frame data con tutti i bit a zero.
- **Frame Dati+CF-Poll** inviati contemporaneamente ad una STA questa è un'operazione di *piggyback* e permette di ridurre l'overhead in rete, quindi aumenta l'efficienza della trasmissione.
- **Frame Dati+CF-ACK+CF-Poll**, usato quando un PCF dopo essere passato un SIFS deve comunicare alla STA che ha ricevuto correttamente il frame da lei inviato e contemporaneamente deve inviare dati e abilitare nuovamente la comunicazione della STA verso se stesso. Anche in questo caso abbiamo un'operazione di *piggyback*.
- **Frame CF-ACK**, usato dall'AP, dopo essere passato un SIFS ha la necessità di comunicare alla STA che ha ricevuto correttamente il messaggio da lei inviato
- **Frame CF End** trasmesso per identifica la fine del CFP, il PCF non ha ulteriori frame da inviare e STA da interrogare.

PCF: Scambio dati con una STA

In questo sotto-paragrafo è presentato lo scambio di frame sotto la gestione del PCF, messaggi multipli trasmessi da e verso l'AP durante il CFP.

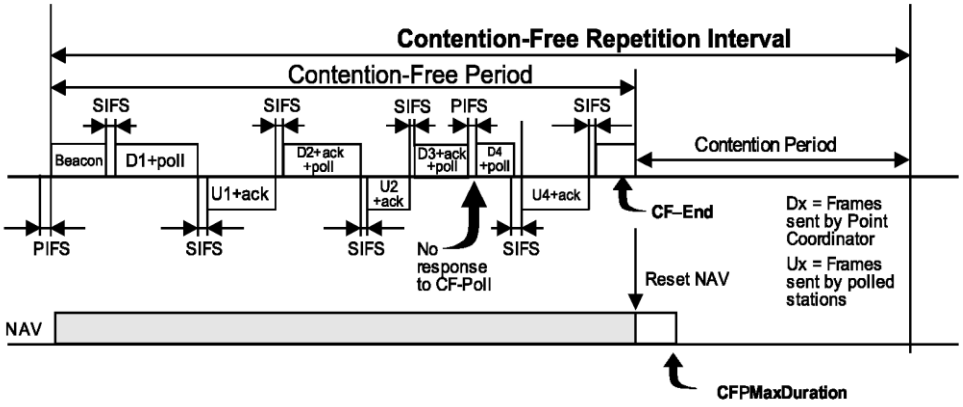


Fig 2.12 - Scambio di frammenti supervisionati da un AP durante il CFP

L'AP prende il controllo della gestione dell'accesso al mezzo durante la finestra di contesa, dopo essere passato un PIFS da quando il mezzo è libero. La prima operazione è la trasmissione di un Beacon frame che contiene la durata della CFP, tutte le STA in ascolto impostano il NAV al valore *CFPMaxDuration*. Una volta comunicato che non è più possibile accedere al mezzo secondo la modalità DCF, l'AP invia frame di polling, interroga a turno le STA e se deve trasmettere verso di loro qualche informazione, incapsula il campo dati nel frame CF-Poll. La CFP può concludersi per due motivi: l'AP invia un CF-END perché non ha più frame da trasmettere e STA da interrogare oppure quando il timer del NAV arriva a zero.

Power Saving

La dissipazione di energia dei dispositivi che utilizzano gli standard della famiglia IEEE 802.11 è un argomento di grande importanza. In una WLAN le STA associate ad un AP sono alimentate da una batteria e dunque preservare la carica quando non sono coinvolte nello scambio di informazioni è fondamentale.

In una BSS di tipo Infrastructured che presenta accesso al mezzo distribuito per poter garantire il Power Saving (PS) è necessario che la STA informi l'AP impostando a uno il bit nel sotto campo *Power Managment*, appartenente al campo *Control* del frame a livello MAC. Dopo aver ricevuto il messaggio, l'AP aggiorna la lista delle STA che si trovano in modalità risparmio di energia all'interno della rete e salva in un buffer i dati in arrivo, diretti verso le STA che si trovano in questa modalità. Questa impostazione mette le STA in "ascolto" (Listen Interval) all'interno della rete. Periodicamente nella WLAN viene trasmesso un frame di Beacon contenente una Traffic Indication Map (TIM) che informa i dispositivi in modalità PS se l'AP deve trasmettere dei frame dati verso di loro. Dopo aver ricevuto il frame di Beacon se una STA capisce che ci sono pacchetti per lei avvia la procedura di contesa del canale trasmettendo un PS-Poll Frame.

Nelle reti infrastructured in cui la gestione dell'accesso al mezzo è controllata dall'AP, la modalità PS si distingue solo per un aspetto rispetto al caso precedentemente descritto. Ad intervalli regolari l'AP trasmette un frame di Beacon contente il Delivery Traffic Information Map (DTIM),

le STA che sono abilitate a ricevere questo tipo di messaggio si riattivano e l'AP trasmette un frame multicast.

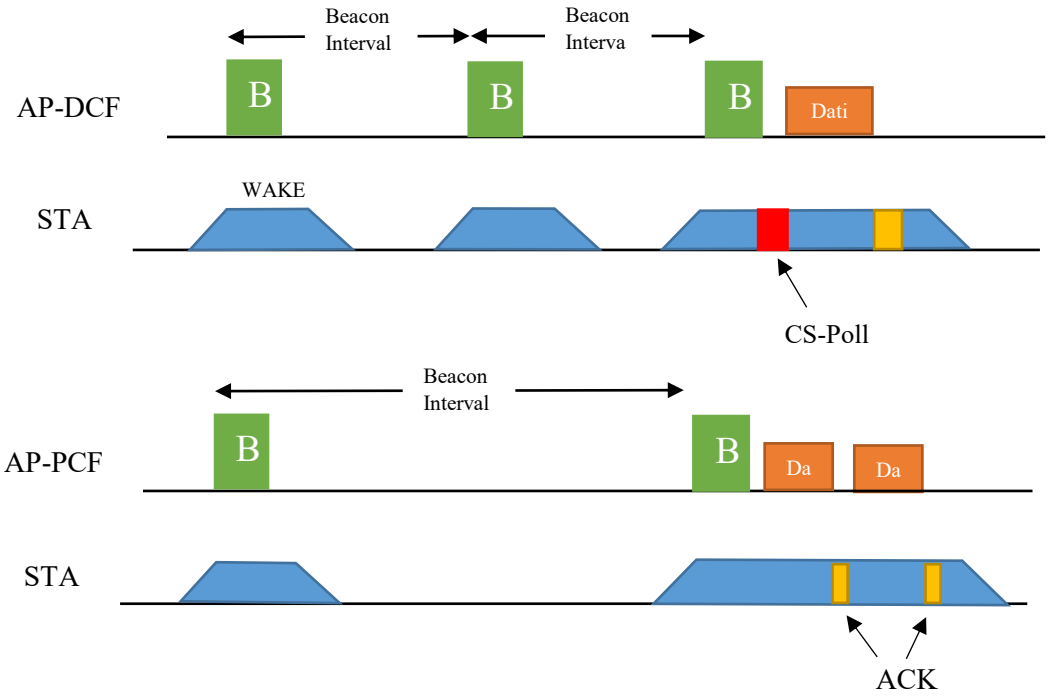


Fig 2.13 - Power Saving in reti Infrastructured con gestione dell'accesso casuale (DCF) e centralizzato (PCF)

Capitolo 3 - Internet Protocol

IPv4 Header

IP è un protocollo che fornisce un servizio di trasporto datagrammi di tipo ***best-effort*** e ***connectionless***. Best-effort significa che non esiste una garanzia sul fatto che il pacchetto IP arrivi a destinazione, se ad esempio ci sono problemi in un nodo router, l'algoritmo che governa la gestione degli errori elimina l'ultimo datagramma arrivato. L'affidabilità delle trasmissioni è garantita dagli strati superiore come ad esempio da TCP.

Il termine connectionless indica che IP non fornisce informazioni sullo stato dei datagrammi tra loro collegati, ogni datagramma è gestito indipendentemente dagli altri, se le risorse di rete permettono di portare il pacchetto a destinazione verrà consegnato, altrimenti sarà scartato. Supponiamo che due datagrammi (**1**,**2**) siano trasmessi alla stessa destinazione, ognuno è indirizzato indipendentemente e quindi possono seguire percorsi differenti, i problemi che possono verificarsi sono molteplici: Ad esempio **2** può arrivare prima di **1** oppure entrambi vengono duplicati o ancora un datagramma può essere alterato. La soluzione è rappresentata dai protocolli di livello superiore come ad esempio il già citato TCP.

I compiti svolti da IP riguardano l'identificazione dell'host, l'incapsulamento e l'instradamento dei pacchetti di livello trasporto Fig [3.1] e la consegna dei dati nell'ordine in cui sono arrivati al livello superiore. L'Header IPv4 ha una parte fissa di 20 byte e una opzionale di 40 byte ma

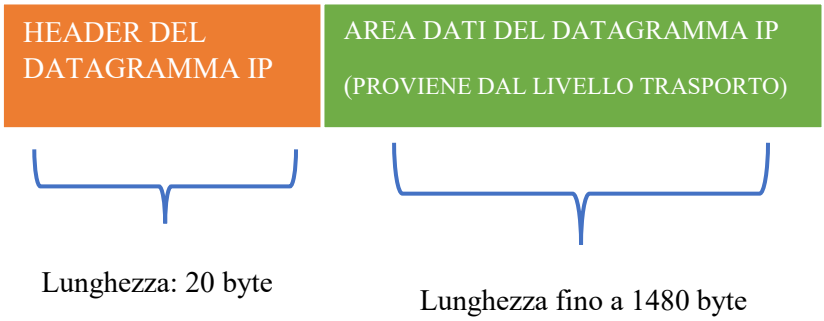


Figura 3.1 - Incapsulamento di un Header IPv4 in un pacchetto Ethernet

raramente usata. I bit sono trasferiti a partire dallo 0 fino al 31 rispetto alla Fig [3.2], questo ordine di trasmissione è chiamato Big Endian, fa riferimento a tutti i valori binari trasmessi dai protocolli TCP/IP e indica il modo in cui sono ordinati i bit che transitano in rete. Questo formato si distingue dal Little Endian usato generalmente nelle CPU dei computer, in cui i bit sono ordinati nel modo opposto, dall'ultimo al primo. Quindi per mettere in comunicazione host differenti attraverso la rete è necessario eseguire una doppia conversione tra formati.

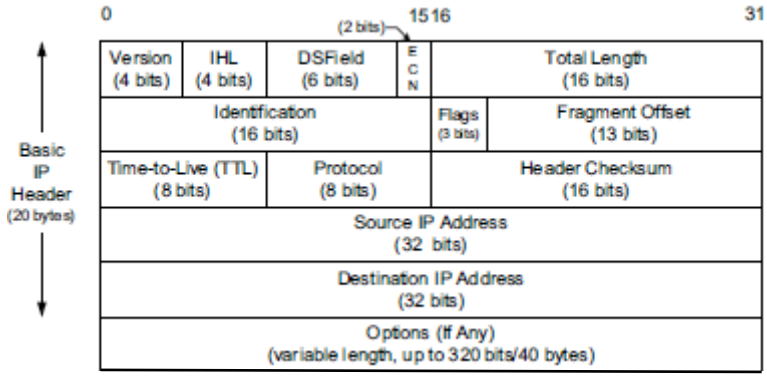


Fig 3.2 - Header IPv4

Per capire come sono trasmessi i datagrammi IPv4 è necessario capire quali sono le caratteristiche dell'header e quindi conoscere il significato dei suoi campi. Di seguito una descrizione dettagliata dei bit utili a definire il protocollo IPv4.

Il campo **Version** (4 bit) può essere popolato con il valore 4 se stiamo utilizzando indirizzi IPv4 o con il valore 6 se stiamo utilizzando indirizzi IPv6

Internet Header Lenght (**IHL**) è un numero a 4 bit che indica la dimensione dell'header.

Type of Service (8 bit) racchiude il campo **Differentiated Service Field** (**DSField**) e il campo **Explicit Congression Notification** (**ECN**), entrambi forniscono delle informazioni sulla qualità del trasporto a livello dei router, a cui i datagrammi IP dovrebbero essere soggetti durante il trasporto. I router basandosi su questi valori, definiscono il percorso, per garantire l'instradamento ottimale, e la precedenza di trasmissione dei pacchetti in coda.

Il campo **Total Lenght** (16 bit) rappresenta la dimensione di header e payload, è importante perché alcuni sotto-livelli che trasportano i datagrammi IPv4 non trasmettono accuratamente la lunghezza della parte dati del messaggio.

Il campo **Identification** permette di identificare ogni datagramma inviato ad un host.

A volte è necessario frammentare un pacchetto in diverse unità. Affinché sia possibile discriminare i frammenti appartenenti ad un datagramma da quelli appartenenti ad un altro, il trasmettitore incrementa di una unità un contatore interno ogni volta che spedisce un frammento e copia questo valore nel campo Identification.

I frammenti appartenenti allo stesso datagramma quindi, presentano nel header IPv4 un valore differente nel campo Identification ma stesso IP address, quindi un certo datagramma è univocamente definito dalla coppia di valori Identification ed IPv4 di provenienza.

I frammenti seguono percorsi differenti per arrivare all'host in ricezione e vengono incapsulati nuovamente nel datagramma originale solo una volta arrivati a destinazione. Discriminare tra un frammento e un datagramma intero è possibile grazie al flag **More Fragment** e al campo **Fragment Offset**.

Il primo frammento di un datagramma è univocamente individuato dal bit di *More Fragment* impostato ad uno e quello di *Fragment Offset* impostato a zero.

L'ultimo frammento di un datagramma IP invece è univocamente definito dal campo *More Fragment* impostato a zero e quello di *Fragment Offset* diverso da zero.

L'host in ricezione quindi può stabilire la dimensione del datagramma solo quando arriva l'ultimo frammento, poiché ogni frammento contiene nel campo Total Lenght solo la propria dimensione. Quando arriva l'ultimo pacchetto l'host somma il valore di *Fragment Offset* con la lunghezza del payload in termini di bit. Se nel processo di instradamento viene perso un frammento e quindi non arriva a destinazione, l'host avvia un timer appena riceve il primo frammento per evitare latenze. Se il conteggio arriva a zero e non sono arrivati tutti i frammenti allora il datagramma cade e bisogna ripetere la procedura di trasmissione.

Il campo **Time-to-Live** (**TTL**) definisce attraverso quanti router può passare un datagramma prima di essere scartato, quindi rappresenta un limite massimo per il pacchetto. Questo valore a 8 bit viene modificato dai nodi in ricezione della rete. Valori tipici sono 64, 128, 255, e ogni volta che il datagramma attraversa un router viene decrementato di uno. Se il campo raggiunge il valore 0 il pacchetto viene scartato e viene inviata una notifica al trasmettitore. Questo meccanismo permette di ridurre la congestione di rete evitando che un datagramma sia bloccato all'interno di un loop di ritrasmissioni. Storicamente questo campo era definito come il tempo massimo di vita di un datagramma IP espresso in secondi, i router però potevano eseguire al massimo decrementi di una unità perché la permanenza al loro interno di un pacchetto non superava un secondo. Si decise quindi di definire una strategia basata sul numero di passaggi attraverso i router

Il campo **Protocol** si riferisce al tipo di dati presenti all'interno del payload del datagramma, quindi definisce da quale protocollo di livello Trasporto arriva il pacchetto. Valori tipici sono 0 per L'Internet Control Message Protocol (ICMP), 6 per Transmission Control Protocol (TCP) e 17 per User Datagram Protocol (UDP).

Il campo **Header Checksum** fornisce un controllo sui bit del solo campo header, questo ci permette di capire il motivo per cui non viene eseguito a livello Rete un controllo sull'errore del payload, questo compito spetta solo ai protocolli di livello superiore come ad esempio TCP o UDP. Avere un checksum separato per dati ed header presenta il vantaggio di ridurre la congestione in rete, in modo tale che i router non siano soggetti a ritardi.

Source IP Address e **Destination IP address** presentano rispettivamente l'indirizzo IPv4 in formato binario dell'host in trasmissione e dell'host in ricezione. Questi valori rappresentano un identificativo per gli host e sono necessari per poter instradare correttamente il datagramma.

Indirizzi IP

Internet Protocol (IP) è un protocollo di livello Network del modello ISO/OSI e rappresenta il mattone su cui si basano la rete internet e tutti i protocolli di livello trasporto, come ad esempio Transport Control Protocol (TCP) o User Datagram Protocol (UDP).

L'indirizzo IP è sicuramente la caratteristica più importante dei protocolli Internet ed è usato per identificare le interfacce di rete dei dispositivi connessi. Esistono due tipologie di indirizzi IP: indirizzi IPv4 e IPv6. D'ora in poi ci riferiremo esclusivamente al primo tipo perché sono quelli utilizzati per lo sviluppo del progetto, quindi ne verrà fornita una breve descrizione e la loro applicazione nelle LAN.

Un indirizzo IP è costituito da 32 bit ma convenzionalmente è rappresentato da quattro numeri in formato decimale, ognuno compreso tra 0-255 e separato da un punto, quindi capiamo che potranno esistere circa 4.3 miliardi di indirizzi.

La struttura degli indirizzi IP si basa sul concetto di *net number*, una porzione di bit contigui della parte iniziale, e *host number*, la rimanente parte.

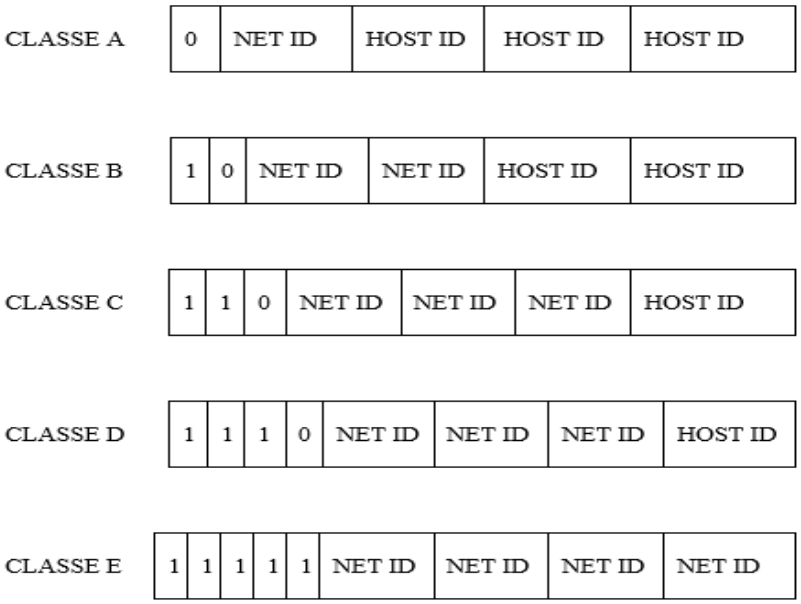


Fig 3.3 - Classi degli Indirizzi IP

La nascita di reti di grandezza differente ha portato alla definizione di classi di cui tre principali, una multicast ed una sperimentale che si differenziano per il numero di *host* che possono contenere. Le tre classi principale: la A, la B e la C presentano rispettivamente 24, 16 e 8 bit per gli *host*. La definizione dei primi tre bit va ad indicare il tipo di classe.

Ad esempio 200.110.12.0 è un indirizzo IP di classe C in notazione decimale.

Il rapido sviluppo di Internet e la nascita di numerose reti LAN mise in forte difficoltà l'utilizzo del metodo degli indirizzi IP in quanto era necessario alloccare un *network number* per ogni nuova porzione di rete che voleva usufruire del protocollo Internet.

Il metodo risultò subito poco efficiente perché significava allocare solo una piccola parte di tutti gli indirizzi possibili in una rete. La LAN più piccola appartenente quindi agli indirizzi di classe C permetteva di definire al massimo 255 dispositivi, dunque se un utente privato decideva di realizzare una rete personale con N dispositivi, a cui sarebbero stati associati N indirizzi IP, significava che tutti gli altri 255-N indirizzi IP che potevano essere utilizzati non sarebbero mai stati allocati. Questo problema venne risolto definendo un numero di rete per una certa area che sarebbe poi stata suddivisa localmente creando delle sotto-reti con un proprio *subnet address*.

L'indirizzo IP di ogni *host* risulta così costituito: un campo *network address* che identifica la rete e quindi il tipo di classe, un campo *subnet address* che identifica la sottorete a cui appartiene l'*host* ed infine un campo *host address*.

Viene definita la *subnet mask* un numero di 32 bit che rappresenta l'estensione della sotto-rete, definendo di fatto il numero di *host* che possono associarsi alla stessa sotto-rete. Una Subnet Mask è associata ad una Subnet e su tale base si calcolano gli indirizzi IP utilizzabili. Se il campo Host ID di un indirizzo è formato da H bit, allora sarà possibile utilizzarne 2^H-2 perché 2^H rappresenta le combinazioni possibili utilizzando H bit nel campo *host*. Bisogna sottrarre due perché uno è l'indirizzo della rete non usabile e l'altro è l'indirizzo di broadcast che tutte le reti devono avere.

Le *subnet mask* sono utilizzate dall'*host* per capire se il pacchetto deve essere instradato all'interno della stessa rete o verso reti esterne passando per il router, che esegue un'operazione di bitwise con l'indirizzo IP del host destinatario.

	Codifica decimale	Codifica Binaria
Indirizzo IP	200.110.12.0	11001000.11011100.00001100.00000000
Subnet mask	255.255.255.224	11111111.11111111.11111111.11100000
Operazione di Bitwise		11001000.11011100.00001100.00000000

Tabella 4 - Operazione di subnetting

La tabella mostra un esempio esplicativo riguardante i concetti di submask e subnet. Dalla subnet mask 255.255.255.224 possiamo capire che la rete a cui appartiene l'indirizzo IP 200.110.12.0 è costituita da 30 dispositivi, infatti può accogliere 2^5 -2=30 host. Gli indirizzi IP associabili agli host di questa rete vanno dal 200.110.12.0 a 200.110.12.30.

Address Resolution Protocol

Il protocollo IP è stato progettato per garantire l'interoperabilità tra un ampio scenario di reti fisiche, per fare questo si serve dell'indirizzo IP che permette l'interconnessione tra gli *host* di una LAN con gli *host* di altre reti eterogenee, dunque possiamo riassumere la sua funzionalità con l'instradamento del pacchetto, ovvero definisce il percorso per interconnettere due *host* uno sorgente e l'altro destinatario appartenenti a sotto-reti differenti, in questo caso si parla di instradamento indiretto, oppure appartenenti alla stessa sotto-rete e in questo caso si parla di instradamento diretto.

In questa trattazione sarà trascurata la descrizione dell'instradamento indiretto e ci focalizzeremo su quello diretto poiché i dispositivi connessi al sistema di comunicazione sviluppato nel progetto appartengono alla stessa LAN.

L'instradamento diretto è basato sul riconoscimento da parte degli *host* dell'indirizzo MAC noto anche come Hardware Address, un numero a 6 byte univocamente assegnato dal produttore di ogni scheda di rete ethernet o wireless. Un problema frequente che riguarda la comunicazione è la mancata conoscenza dell'indirizzo MAC dell'host destinatario. Il solo indirizzo IP non è sufficiente a stabilire lo scambio di frame in una rete basata sull'utilizzo di indirizzi hardware, ogni protocollo progettato per operare con la scheda di rete deve necessariamente fare uso dell'indirizzo hardware.

Abbiamo visto nel secondo capitolo che questa informazione è necessaria per riempire i campi *address* del frame a livello MAC, la conoscenza degli indirizzi hardware di host sorgente e destinatario permette di inviare il payload.

Frame ARP

L'Address Resolution Protocol (ARP), un protocollo a livello Network permette di ottenere l'indirizzo MAC a partire da quello IP, utilizzando due frame differenti: L'ARP Request e L'ARP Reply. Un frame ARP viene incapsulato in un frame di sotto-livello Medium Access Control. Fig [3.4]

Fig 3.4 - Incapsulamento di un pacchetto ARP

Facendo riferimento al protocollo Wi-Fi, questi messaggi sono in un primo momento inviati dalla STA all'AP quindi incapsulati in un MAC header della famiglia 802.11 e poi in secondo momento incapsulati in un MAC header Ethernet e spediti attraverso la rete cablata al default gateway che ha l'indirizzo MAC della STA con cui si vuole scambiare informazioni. Un secondo tecnica possibile è quella di inviare direttamente però sempre passando attraverso L'AP un ARP request alla STA che appartiene alla WLAN.

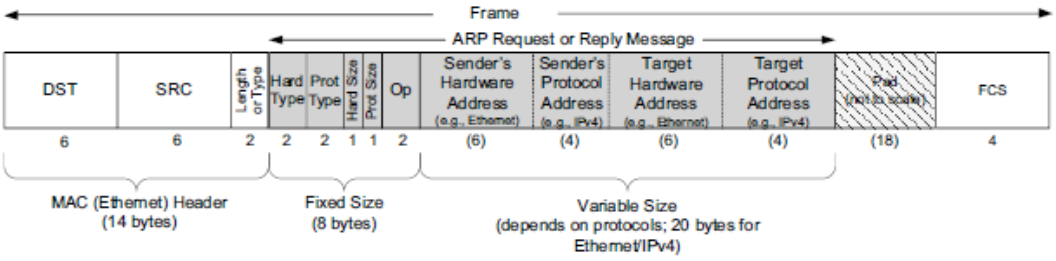


Fig 3.5 - Formato di un pacchetto ARP

I primi due campi dell'ARP frame destinato ad una rete Ethernet fanno riferimento all'indirizzo MAC dell'host mittente e dell'host destinatario, poiché questo ultimo valore non è noto si popola con ff:ff:ff:ff:ff:ff, che equivale ad imporre tutti i bit ad uno e significa che sarà trasmesso un messaggio broadcast a tutti i dispositivi collegati in rete. Il campo **Lenght/Type** è riempito con il valore 0x0806 che identifica un messaggio ARP (request o reply). **Hard Type** indica il tipo di MAC address che si sta cercando mentre il campo Prot Type indica il protocollo di livello rete usato. **Hard Size** e **Prot Size** fanno rispettivamente riferimento alla dimensione in termini di byte del MAC address e del protocol address. Il campo **OP** indica un ARP request o un ARP replies. Questa informazione è necessaria per individuare il tipo di operazione che si vuole eseguire perché entrambi i pacchetti ARP hanno la stessa dimensione. I successivi quattro campi **Sender's Hardware Address**, **Sender's Protocol Address**, **Target Hardware** e **Target Protocol** fanno riferimento all'indirizzo MAC e al tipo di protocollo di livello Rete usato da host in trasmissione e host destinatario. Per un ARP request il campo Target Hardware è riempito con il valore 0. É interessante notare che queste informazioni sono ridondanti perchè presenti sia nel MAC Header che nel frame ARP.

Indirizzamento diretto

Quando utilizziamo un servizio Internet il nostro dispositivo deve capire come contattare il processo server a cui siamo interessati, in primo luogo deve determinare se appartiene ad una rete locale o ad una rete remota. Se appartiene ad una rete remota è necessario disporre di un router se invece presenta la stessa *subnet mask,* è possibile usare direttamente il protocollo ARP.

Utilizziamo un esempio per capire come funziona il meccanismo dell'instradamento diretto, supponiamo che il processo server si trovi all'interno della stessa rete cablata locale in cui è presente anche il processo Client.

1. A livello Applicazione del modelllo ISO/OSI Il client genera un messaggio da trasmettere all'host che presenta interfaccia di rete 200.110.12.0.

2. L'Applicazione chiede al protocollo TCP di stabilire una connessione con 200.110.12.0.

3. Supponendo che il dispositivo host presenti una scheda di rete Ethernet, utilizzando il protocollo ARP deve essere in grado di risolvere l'indirizzo IPv4 del destinatario nel suo indirizzo MAC.

4. Viene inviato in rete un messaggio di ARP request contenente l'indirizzo IPv4 del server ad ogni host presente in rete. In pratica si pone la seguente domanda: "Se il tuo indirizzo IPv4 è uguale a 200.110.12.0, rispondi con il tuo indirizzo MAC".

5. Il messaggio ARP è inviato a tutti, anche ai dispositivi che presentano interfaccia di rete IPv6 o quelli che presentano un indirizzo IPv4 diverso da quello richiesto. Chi si accorge di non possedere l'indirizzo cercato scarta il messaggio, chi invece contiene 200.110.12.0 risponde con un messaggio di ARP replies. La risposta contiene l'indirizzo IPv4 per una controverifica e l'indirizzo MAC. Questo pacchetto non è inviato in broadcast ma solo a chi ha posto la domanda. L'host in ricezione salva l'indirizzo MAC dell'host in trasmissione per eventuali usi futuri.

6. Il client una volta ricevuta l'ARP reply invia direttamente i datagrammi al destinatario incapsulandoli in un frame Ethernet.

Capitolo 4 - Transmission Control Protocol

Controllo di flusso

Il Transmission Control Protocol (TCP) è un protocollo a livello Trasporto del modello ISO/OSI, appartiene alla suite dei protocolli di internet. Viene utilizzato generalmente insieme al protocollo di rete IP per sopperire alla mancanza di affidabilità della comunicazione e migliorare dunque la qualità del servizio.

TCP fornisce un servizio di tipo "connection-oriented" che ha come obiettivo la realizzazione di un canale di trasporto "end-to-end" astratto: l'host in ricezione e l'host in trasmissione devono stabilire una connessione prima di poter scambiare dati e, la comunicazione è full-duplex, ossia il flusso di informazioni è supportato in entrambe le direzioni contemporaneamente, ma solo dopo aver stipulato la connessione.

La proprietà di affidabilità della comunicazione è garantita dall'utilizzo simultaneo di tecniche differenti. TCP attua un processo di impacchettamento, convertendo il flusso di dati a livello Applicazione in un certo numero di segmenti che possono essere trasportati da IP. Ad ogni segmento è associato sia un *sequence number* che un *acknowledgment number*, una coppia di numeri binari a 32 bit che garantiscono un controllo sui segmenti dati ricevuti e inviati. Il sequence number è considerato un valore di off-set rispetto al primo byte inviato. L'*acknowledgment number* è inviato dall'host in ricezione per informare l'host in trasmissione che un determinato segmento è stato correttamente ricevuto. Il valore presente in questo campo è costituito dal *sequence number* del pacchetto ricevuto incrementato di una unità.

Il seguente esempio aiuterà a capire meglio l'utilità del *sequence number*: supponiamo di voler inviare 2000 byte frammentati in 4 segmenti da 500 byte, Supponiamo per semplicità che i allora i rispettivi campi *sequence number* conterranno il valore 0, 500, 1000, 1500. I pacchetti non arriveranno in maniera ordinata al destinatario ma lo faranno in ordine sparso perché come abbiamo già accennato il protocollo IP non è in grado di garantire questo servizio. Si potranno presentare tre situazioni: potrà essere ricevuto il segmento con *sequence number* identico (**1**), inferiore (**2**) o superiore (**3**) a quello atteso.

1. Se alla prima transazione arriva il segmento con *sequence number* identico a quello atteso, esso verrà inviato a livello Applicazione del modello ISO/OSI.

2. Se arriva un segmento con *sequence number* minore di quello atteso, ad esempio attendo il 1500 ma arriva il 500 significa che quest'ultimo è un duplicato e quindi viene scartato.

3. Se arriva un segmento con *sequence number* maggiore di quello atteso, ad esempio se attendendo il 500 ma arriva il 1500 significa che i dati precedenti o sono andati persi oppure sono in attesa di essere ricevuti. Per definire i due casi si utilizza un timer, se dopo un certo tempo non è arrivato il pacchetto atteso, esso viene considerato perduto.

Stop & Wait vs Go - back N

Per garantire il controllo sul flusso di dati, il protocollo TCP utilizza la tecnica di ritrasmissione Go back N, un'evoluzione della tecnica Stop & Wait.

La tecnica *Stop & Wait* prevede che per ogni segmento dati inviato sia necessario attendere un riscontro dal ricevitore tramite ricezione di un pacchetto di ACK. Si tratta di un sistema a bassa efficienza di trasmissione perché può inviare solo un segmento alla volta.

Gli header associati ai segmenti inviati con questo meccanismo, naturalmente, presentano i campi sequence number e acknowledgement number in modo tale da ricevere un riscontro tra ciò che si invia e ciò che si riceve. Se il segmento atteso non arriva però entro un tempo limite (*time-out*) si considera perso, e il trasmettitore deve inviare nuovamente il pacchetto dati.

La tecnica Go-back N implementata da TCP riesce a migliorare sensibilmente la bassa efficienza del sistema Stop & Wait grazie al concetto di finestra scorrevole.

A livello di software, questa tecnica è realizzata tramite una coppia di puntatori: uno dei quali punta al primo pacchetto inviato ma non riscontrato del buffer, l'altro punta all'ultimo pacchetto non riscontrato. La struttura finestra è definita in questo modo sia in ricezione che in trasmissione e permette un accesso diretto al livello Applicazione.

Fig 4.1 - Meccanismo della finestra lato trasmettitore

Si definisce larghezza della finestra l'entità variabile che definisce il numero di pacchetti che possono essere inviati e quelli che sono in attesa di conferma.

La Fig [4.1] mostra la finestra del trasmettitore nell'istante in cui il pacchetto 3 è stato inviato correttamente ed è stato ricevuto il riscontro, il pacchetto 7 invece è pronto ad essere inviato ma deve attendere di entrare nella finestra.

I dati sono inviati dal trasmettitore al ricevitore mentre gli ACK sono trasmessi nel verso opposto.

Nell'esempio che stiamo considerando l'host in trasmissione sta aspettando l'ACK relativo al pacchetto 4 per poterlo eliminare dalla memoria: una copia deve essere infatti sempre mantenuta finché non si è certi della corretta ricezione. Il riscontro del pacchetto 4 permette alla finestra di scorrere di un pacchetto, cosìcchè il pacchetto 7 possa essere inviato. Se uno dei pacchetti della sequenza non trova riscontro, il trasmettitore, dopo aver aspettato un certo tempo (time-out) invia tutta la sequenza a partire dal pacchetto successivo all'ultimo ricevuto correttamente. In pratica il trasmettitore torna indietro di N pacchetti

Il TCP è definito un protocollo a finestra con ACK cumulativi. Generalmente il ricevitore non trasmette un ACK ad ogni pacchetto ricevuto ma lo fa solo una volta che ha ricevuto un gruppo di segmenti dati. Questo implica che, una volta ricevuto correttamente un ACK riferito ad un certo segmento dati, siano automaticamente convalidati anche tutti i segmenti dati precedenti e rilasciati dal buffer per via della sequenzialità dei dati garantita dal protocollo TCP. Se un pacchetto dati che appartiene alla catena inviata viene perso, viene re-inviata l'intera sequenza.

Il protocollo TCP, realizzato per supportare e scambiare messaggi in entrambe le direzioni, permette di fornire un riscontro sulla corretta ricezione di un pacchetto dati inserendo nel campo *acknowledgment number* un valore che identifica il *sequence number* del successivo byte

appartenente al pacchetto che si vuole ricevere. Questa operazione è chiamata *piggyback* Fig. [4.2].

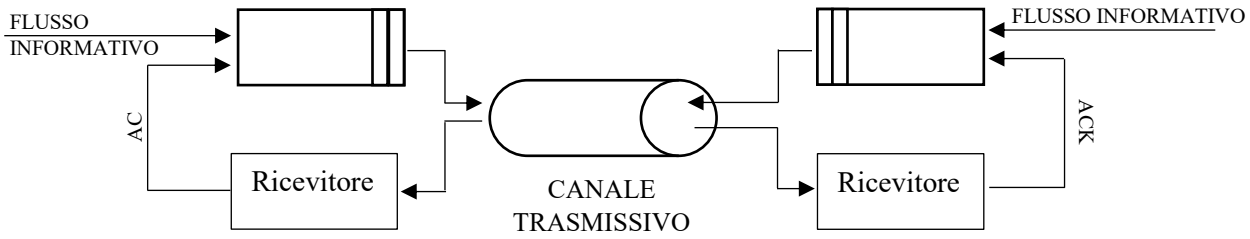


Fig 4.2 - Operazione di Piggybacking

TCP HEADER

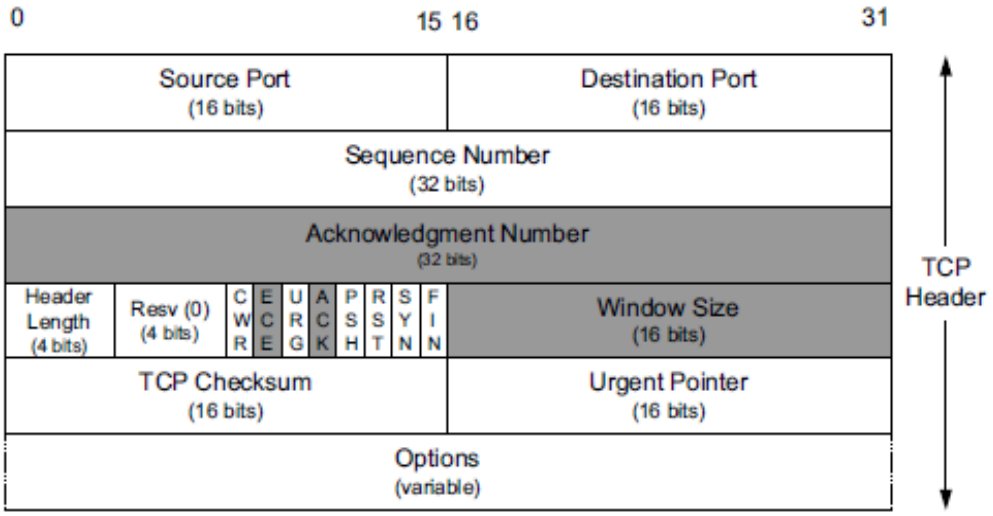


Fig 4.3 - TCP Header

Un header (Fig [4.3]) del protocollo TCP contiene due valori a 16 bit che definiscono la **porta di destinazione** e **porta sorgente**. Questi due campi insieme agli indirizzi IP dei due host in comunicazione permettono di identificare in maniera univoca la connessione. La combinazione di una porta e di un indirizzo IP è chiamata socket.

Il campo **Sequence Number** è improntato all'identificazione di ogni byte del segmento dati, rappresenta la posizione occupata dal primo byte in ogni scambio di pacchetti. Questo valore una volta raggiunto il numero massimo di 2^{32} - 1 torna a zero. Il campo **Acknowledgment Number** invece identifica il byte consequenziale che il ricevitore aspetta dal segmento successivo. Questo valore, quindi, è uguale al *sequence number* dell'ultimo byte ricevuto +1.

I campi flag permettono di accedere a funzioni speciali quali: frame di inizio e fine connessione che si ottengono abilitando rispettivamente **SYN** e **FIN**, abilitazione del campo *acknowledgment number* impostando ad 1 il bit di **ACK**.

Il campo **Window Size** permette di variare dinamicamente la dimensione della finestra di trasmissione. Il ricevitore comunica al trasmettitore quanti byte può ricevere, cioè quanto spazio libero c'è nel buffer in ricezione. Il trasmettitore, invece, comunica al ricevitore quanti byte presenti nel buffer può inviare senza attendere la ricezione del riscontro.

Nel protocollo è previsto un campo **Checksum** che permette di identificare la presenza di errore sui bit in trasmissione. Il trasmettitore in base ai bit presenti nel campo dati, calcola questo valore e lo invia insieme al payload. Il ricevitore una volta ricevuto il pacchetto calcola il *checksum* sui bit in ricezione e lo confronta con quello inviato, se i due coincidono trasmette nel segmento successivo il campo a*cknowledgment number* per notificare la corretta ricezione, se invece non coincidono significa che il messaggio è stato corrotto e quindi non trasmette nulla e attende che venga ritrasmesso lo stesso frame.

Connessione

Prima di stabilire la connessione e poter quindi scambiare il flusso di dati tra i processi applicativi è necessario che le entità locali TCP comunichino con i processi applicativi. E' possibile identificare due tipologie di processi in una comunicazione che utilizza una connessione TCP:

- Il processo applicativo lato server esegue una *passive open*, creando le socket necessarie all'ascolto e informa il livello TCP che è pronto ad accettare nuove connessioni.

- Il processo applicativo lato client esegue invece un *active open*, creando di fatto un canale virtuale tra la sua porta e la porta di destinazione del server.

Il procedimento per stabilire una connessione tra due host è chiamato *three way handshake*, in quanto è necessario scambiare tre segmenti prima di poter scambiare dati. La figura [4.4] schematizza i tre passaggi, di seguito spiegati nel dettaglio:

1. Il client estrae un *sequence number* casuale, ad esempio il 67803 e trasmette un segmento costituito solo dall'header TCP con il flag di SYN abilitato.

2. Il server riceve il segmento con il campo SYN e decide se accettare la connessione TCP. In caso positivo genera un messaggio con i flag SYN-ACK impostati ad 1 che permette lo scambio dati in entrambi le direzioni. Estrae un *sequence number* casuale, ad esempio 5608 e lo invia insieme al riscontro 67804.

3. Il client riceve il messaggio SYN-ACK e trasmette al server il riscontro 5609 per il *sequence number* 5608. La connessione è definitivamente abilitata, il primo pacchetto dati è inviato con questo segmento.

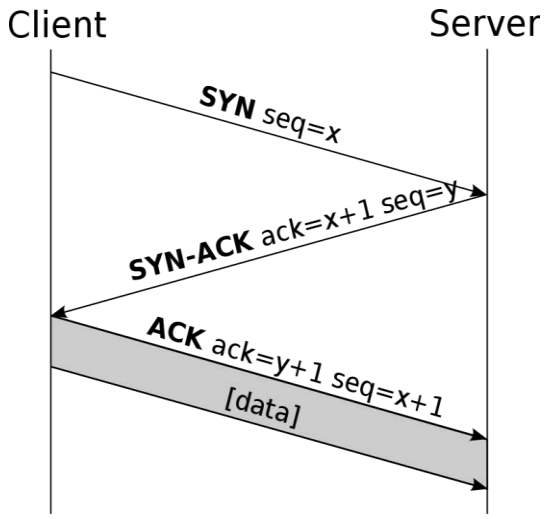


Fig 4.4 - Procedimente Three-way handshake

Il procedimento per poter terminare una connessione invece è chiamato *four way handshake*. Il suo nome deriva dalla natura del canale TCP che è considerato come l'unione di due canali di flusso monodirezionali piuttosto che un unico canale bidirezionale. La chiusura di uno dei due canali permette infatti di poter ancora inviare dati nel senso opposto.

Per poter chiudere il canale di comunicazione in entrambe le direzioni, è necessario che host client e host server si scambino quattro segmenti TCP. Questo procedimento è simile al *three way handshake* con la differenza che il messaggio per terminare la connessione presenta i campi FIN-ACK abilitati.

Interazione client/server in TCP: le socket

Il termine socket in informatica descrive un'astrazione che permette di leggere o scrivere dati da trasmettere o ricevere attraverso un canale di comunicazione che collega due host diversi. Una socket è un'interfaccia (API) tra il livello trasporto e il livello applicazione che fornisce informazioni utili per implementare il protocollo, come ad esempio identifica chi è il processo server e chi quello client. (Fig [4.5])

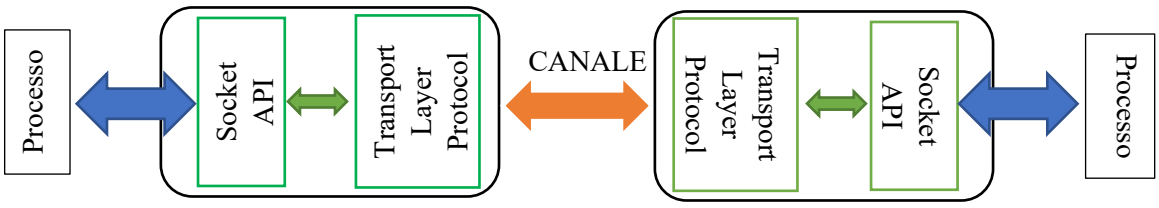


Fig 4.5 - Interfaccia Socket

Le socket possono essere raggrupate in tre categorie: datagram, stream, raw:

- Le **Datagram Socket** non sono improntate alla connessione e sono utilizzate da UDP.
- Le **Stream Socket** sono improntate alla connessione e sono utilizzate da TCP.
- Le **Raw Socket** sono utilizzate per accedere al livello network.

In questa trattazione analizzeremo quali sono i passaggi necessari per creare un canale di connessione virtuale tra un client e un server remoto attraverso l'implementazione delle stream socket.

1. Il primo passo riguarda la realizzazione di una socket sia lato client sia lato server, quest'ultimo si pone in ascolto su una porta.

Fig 4.6 a - Realizzazione della Socket Client e Server

2. Il client chiede al server di poter effettuare una connessione.

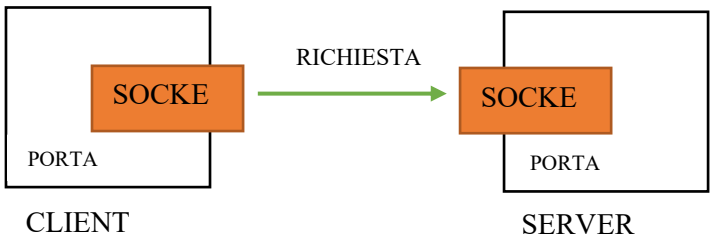


Fig 4.6 b - Richiesta di connessione

3. Scambio di dati attraverso il canale di comunicazione.

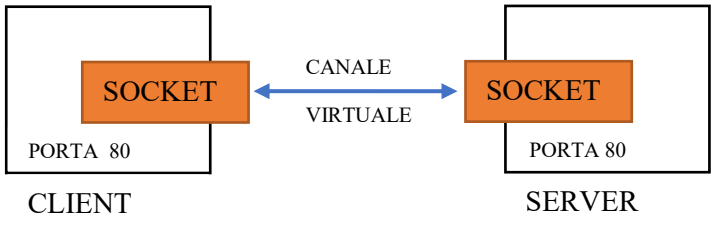


Fig 4.6 c - Definizione del canale virtuale e scambio dati

4. Una volta concluso lo scambio dati, il client lo comunica al server di chiudere la connessione.

Implementazione delle socket in linguaggio C lato server

In questo paragrafo analizzeremo come è realizzato il canale virtuale, studiando le API in linguaggio C per sistema operativo Windows. Questi concetti ci permetteranno successivamente di realizzare il progetto oggetto della tesi.

La libreria, che contiene le API per poter utilizzare le socket, è inclusa nel seguente header file **winsock.h**. Di seguito verranno elencati e spiegati i passaggi per realizzare la comunicazione.

1. Per poter utilizzare la libreria winsock.h nel sistema operativo di Windows è necessario eseguire la sua abilitazione chiamando la function **WSAstartup** che ha in ingresso due parametri **wVersionRequested** e **wsaDATA**. wVersionRequested indica la versione più recente delle window sockets che può essere usata dall'applicazione. se la versione richiesta dall'applicazione è maggiore o uguale della minore delle versioni supportata dalla winsock la chiamata ha successo e la winsock restituisce le informazioni dettagliate nella struttura **WSADATA**.

```
/* Inizializzazione della libreria Socket */
WORD wVersionRequested = MAKEWORD(2,2);
WSADATA wsaData;
wsastartup = WSAStartup(wVersionRequested, &wsaData);
```

Fig 4.7 a - Listato 1/6 per la definizione del processo server,

2. La seconda operazione da fare è dichiarare e poi definire la struttura **socket_addr_in** per il server, popolando i campi con: lunghezza della struttura in termini di byte, famiglia degli indirizzi e indirizzo IP.

   La figura mostra una porzione del listato per poter creare un canale di comunicazione

```
#define port 100

struct sockaddr_in{
short int sin_family; // AF_INET
unsigned short int sin_port; // numero di porta
struct in_addr sin_addr;
}

/* Popola i campi della struttura sockaddr_in */

struct sockaddr_in Server;

Server.sin_family = AF_INET;
Server.sin_addr.s_addr = htonl(INADDR_ANY);
Server.sin_port = htons(port);
```

Fig 4.7 b - Listato 2/6 per la definizione del processo server.

   virtuale server-client; si possono inoltre notare gli elementi di cui è composta la struttura

socketaddr_in definita nella libreria winsock.h. Per prima cosa si assegna al tipo struct sockaddr_in la variabile server, successivamente si popolano i campi. Il campo family definisce la famiglia di indirizzi utilizzati, AF_INET rappresenta la famiglia di indirizzi IPv4. Il campo successivo indica da quali indirizzi IP il server può accettare le connessioni, INADDR_ANY indica che è possibile accettare qualsiasi indirizzo client. La function htonl() converte un numero dal formato dell'host al formato accettato dalla rete di tipo long, cioè il Big-Endian. Il campo sin_port invece è popolato con il numero di porta precedentemente definito e la function htons() converte un numero dal formato dell'host al formato accettato dalla rete di tipo short, cioè il Big-Endian.

3. Dal lato server è possibile definire la socket attraverso la function **socket(int family, int type, int protocol)**. Il primo campo rappresenta la famiglia di indirizzi da utilizzare, ad esempio se volessimo utilizzare la famiglia IPv4 dovremmo popolare il campo *family* con AF_INET. Il secondo parametro in ingresso invece indica il tipo di socket utilizzata distinguendo tra Datagram, Stream, Raw. L'ultimo valore di input indica il tipo di protocollo livello trasporto utilizzato, quindi è possibile selezionare o il protocollo TCP o UDP. Il valore di ritorno che può essere un valore sia positivo che negativo: rappresenta il socket descriptor che fa da referente per la socket creata e può essere passato ad altre API. In caso di esito negativo è possibile impostare un controllo di tipo condizionale per indicare che la creazione della socket non è andata a buon fine.

```
/* Creazione della Socket che si porrà in ascolto di richieste del Client*/
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sd < 0)
printf("Errore: sd socket non creata\n");
else printf("sd socket creata\n");
```

Fig 4.7 c - Listato 3/6 per la definizione del processo server.

Questa porzione di listato individua la creazione della socket di tipo stream per la famiglia di indirizzi IPv4 e protocollo di trasporto di tipo TCP, la quale restituisce 1 in caso di esito positivo e stampa a schermo "sd socket creata", altrimenti restituisce 0 e stampa un messaggio di errore.

4. Il Client può accedere al canale di comunicazione solo se l'indirizzo IP del server e la porta che individua il processo sono associati alla socket. La function **bind(int sockds, struct sockaddr_in *localaddr, int localaddrlen),** definita solo lato server, mette in relazione questi elementi. Il primo campo individua il *socket descriptor*, il secondo invece rappresenta il puntatore alla struttura Server in cui sono specificati il numero di porta locale e l'indirizzo IP locale a cui il sistema operativo deve collegare il socket. L'operatore di cast è necessario perché esistono strutture differenti che variano in base al family address. La funzione ritorna 0 se ha successo, -1 se si è tentato di stabilire una connessione con un numero di porta già in uso.

```
if(bind(sd,(struct sockaddr *)&Server,sizeof( Server)) < 0)
printf("Errore function bind()");
```

Fig 4.7 d - Listato 4/6 per la definizione del processo server.

Questa porzione di listato rappresenta la function bind() inserita in un controllo di tipo condizionale, in caso di esito negativo stampa a schermo un messaggio di errore.

5. In seguito alla creazione del canale di comunicazione è necessario che il server attenda di essere contattato da un client. Il server si pone in "ascolto" chiamando la funzione **listen(int socket, int backlog)**. Il primo campo individua la socket che si pone in ascolto, invece il secondo campo rappresenta il numero di connessioni che può accettare il server. Listen() resistuisce 0 in caso di successo e -1 in caso di errore.
   Questa porzione di listato rappresenta la function listen(), in caso di esito positivo

```
/* La socket è "ascolto" tramite la listen() */
sd = listen(sd, 1);
if (sd < 0)
    printf("Errore durante la chiamata di listen().\n");
else
    printf("La Socket ora è in Ascolto\n");
```

Fig 4.7 e - Listato 5/6 per la definizione del processo server.

   restituisce 0 e stampa a schermo un messaggio che indica la buona riuscita dell'operazione, altrimenti restituisce -1 e stampa un messaggio di errore.

6. Il server, per poter accettare la connessione con un client chiama la function **accept(int sockds, struct sockaddr_in *client_addr, int *clientaddrlen),** che fino a quel momento si trovava in uno stato di attesa bloccante. Il campo **sockds** rappresenta il *socket descriptor* della socket creata precedentemente, il secondo argomento indica un puntatore alla struttura di tipo **sockaddr_in** che attraverso la function viene popolato con indirizzo IP e numero di porta utilizzati dal client. L'ultimo termine rappresenta un puntatore alla dimensione in termini di byte della struttura del client. La funzione restituisce un *socket descriptor* che fa riferimento ad un'altra socket che si crea dopo che la connessione è stata accettata a seguito del *three-way handshake*. La comunicazione tra client e server sarà mantenuta su questa socket mentre la precedente rimane ancora attiva e in attesa che nuovi processi client decidano di connettersi. Se accept() restituisce -1 significa che nessun client si è collegato.
   Questa porzione di listato rappresenta la creazione di una nuova struttura di tipo

```
struct sockaddr_in Client; // struttura per il client
int fd; // socket descriptor per il client
int Client_Len;
Client_Len = sizeof(Client); // definisce la dimensione della struttura client
if ((fd = accept(sd, (struct sockaddr *)&Client,&Client_Len)) < 0)
    printf("Errore durante la chiamata di accept().\n");
printf("IP Client: %s\n",inet_ntoe(Client.sin_addr)); //indirizzo IP client
```

Fig 4.7 f - Listato 6/6 per la definizione del processo server.

   sockaddr_in popolato con dimensione, indirizzo IP e numero di porta utilizzato dal client. Viene realizzata una nuova socket su cui si scambieranno informazioni il client e il server che sarà identificata con un nuovo socket socket descriptor (fd). La function accept() restituisce 0 in caso di esito positivo e stampa a schermo l'indirizzo IP del client altrimente resistuisce -1 e stampa un messaggio di errore. La funzione inet_ntoe(Client.sin_addr) converte l'indirizzo IP del client dal formato a 32 bit di rete a stringa ASCII terminante con il carattere null.

Implementazione delle socket in linguaggio C: lato client

Per poter scambiare dati con il processo server, è necessario che anche il processo client definisca una socket che rappresenterà il canale virtuale, ed è pertanto necessario scrivere un nuovo listato. Di seguito sono evidenziati i passaggi principali affinché i due host in comunicazione possano aprire la connessione, scambiarsi informazioni, e chiudere la connessione.

1. Il primo passo è abilitare la libreria *winsocket.h* con le stesse modalità viste per il server nel sotto-capitolo precedente: si definisce quindi il tipo int socket descriptor per la nuova socket che si andrà a realizzare attraverso la function **socket(int family, int type, int protocol)**. Il primo campo rappresenta la famiglia di indirizzi da utilizzare: dato che il server utilizza la famiglia IPv4 dovremo popolare il campo *family* con AF_INET. Il secondo parametro in ingresso, invece, indica il tipo di socket che dovremmo identificare con le Datagram socket. L'ultimo valore di input indica il tipo di protocollo livello trasporto utilizzato, nel nostro caso TCP. Il valore di ritorno, che può essere sia un valore positivo che negativo, rappresenta il socket descriptor che fa da referente per la socket creata e può essere passato ad altre API. In caso di esito negativo è possibile impostare un controllo di tipo condizionale per indicare che la creazione della socket non è andata a buon fine ()

```
/* Creazione della Socket che si porrà in ascolto di richieste del Client*/
sd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sd < 0)
printf("Errore: sd socket non creata\n");
else printf("sd socket creata\n");
```

Fig 4.8 a - Listato 1/4 per la definizione del processo client.

2. Successivamente bisogna creare la struttura sockaddr_in e definire le specifiche del server con cui comunicare, quindi indirizzo IP, numero di porta e socket descriptor. Chiamando la function **connect(sd ,(struct sockaddr_in *)&Server, sizeof(Server))** si stabilisce la connessione tra le due socket.

```
struct sockaddr_in Server;
memset(&Server, 0, sizeof(Server));
Server.sin_family = AF_INET;
Server.sin_addr.s_addr = inet_addr("127.10.10.11"); // IP del server
Server.sin_port = htons(100); // Server port
// Effettua la connessione
if (connect(sd, (struct sockaddr *)&Server, sizeof(Server)) < 0)

    printf( "Connessione fallita.\n" );
```

Fig 4.8 b - Listato 2/4 per la definizione del processo client

Il campo family definisce la famiglia di indirizzi utilizzati, AF_INET rappresenta la famiglia di indirizzi IPv4. Il campo successivo indica da quali indirizzi IP il server può accettare le connessioni, il secondo argomento indica che il client vuole comunicare con un determinato server: in questo caso con il server che fa capo all'indirizzo IP 127.10.10.11. La function inet_addr() converte una stringa ASCII in cui è presente il carattere "." in una sequenza di bit che sia accettato dalla rete, cioè il Big-Endian. Il campo sin_port invece è popolato con il numero di porta in cui è definito il processo del server. La function htons() converte un numero dal formato dell'host al formato accettato dalla rete di tipo short, cioè il Big-Endian.

3. In seguito al processo di connessione di tipo *three-way handshake* è possibile inviare messaggi con la function **int send(int sock, char *buffer, int len, int flag**) e ricevere messaggi con la function **int recv(int sock, char *buffer, int len, int flag).** Il primo campo di entrambe le funzioni indica la socket in cui si svolgerà la comunicazione rispettivamente per il client e per il server. Il secondo argomento rappresenta per send() il puntatore al buffer in cui sono immagazzinati i dati da inviare e per recv() il puntatore al buffer in cui salvare i valori ricevuti. L'ultimo termine fa riferimento alla dimensione dei dati inviati.

```
int send (int sd, char *buffer, int len, int flags)

int recv (int sock, char *buffer, int len, int flags)
```

Fig 4.8 c - Listato 3/4 per la definizione del processo client

4. L'ultimo step è chiudere la connessione in entrambe le direzioni chiamando la funzione **close(int socket)**, il cui unico argomento rappresenta la socket da chiudere.

```
close(fd);
```

Fig 4.8 d - Listato 4/4 per la definizione del processo client

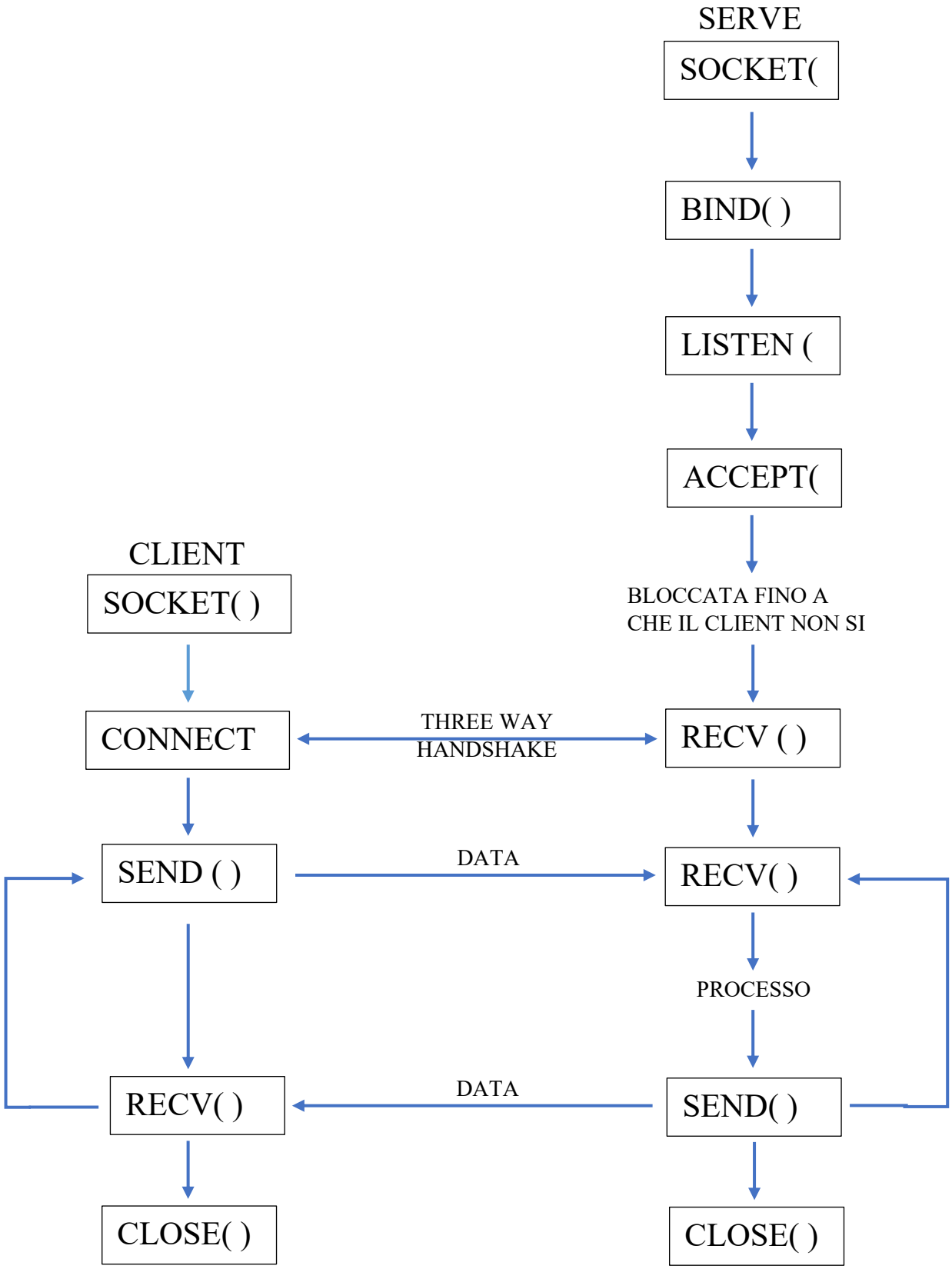


Fig. 4.9 - Diagramma a blocchi definizione processo Server e Client

Il diagramma di Fig [4.9] rappresenta in maniera schematica i passaggi che sono stati descritti precedentemente per aprire e chiudere una socket ed utilizzarla per scambiare le informazioni attraverso il canale virtuale. Ogni blocco rappresenta le API implementate in C per eseguire questi passaggi. Ripercorrendo brevemente i passaggi necessari ad instaurare la comunicazione possiamo notare la definizione della socket sia per il processo Client che per il processo Server, la richiesta di connessione da parte del Client con la successiva accettazione con il metodo three-way handshake. Dopo la trasmissione dati, il canale di comunicazione viene chiuso da entrambi i lati.

Capitolo 5 - Sistema Atmel

Il kit Atmel SAMD21 Xplained Pro in fig. [5.1] è un sistema embedded, cioè una piattaforma hardware dotata di differenti periferiche gestite dal microprocessore programmabile **ATSAMD21J18A**.

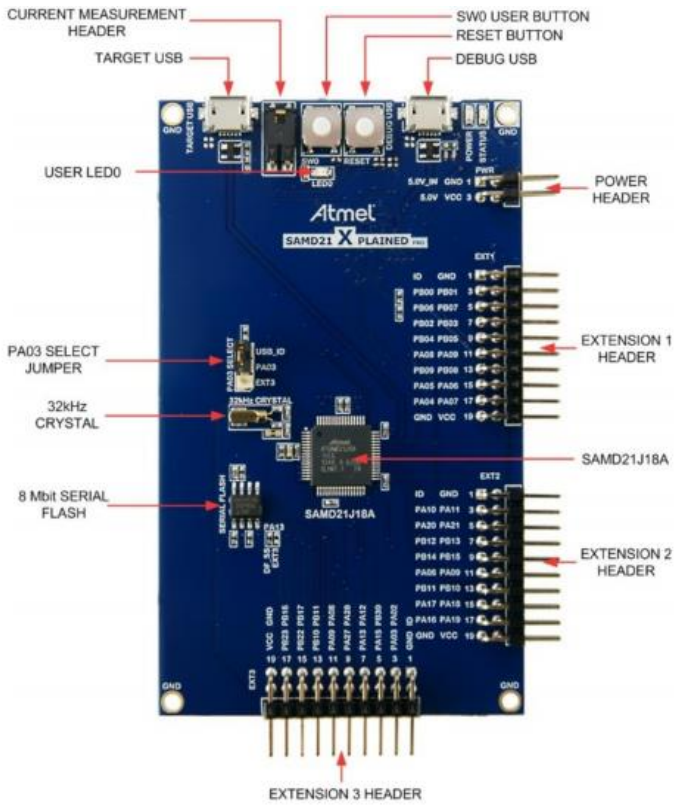


Fig 5.1 - Microcontrollore Atmel SAMD21 XPLAINED PRO
con descrizione delle periferiche.

Questo sistema permette di realizzare una certa applicazione, lavorando con Il sistema di sviluppo **Atmel Studio**.

ATSAMD21J18A è un microprocessore alimentato a 3,3 [V], appartiene alla famiglia ARM Cortex M0 e presenta un'architettura RISC a 32 bit. La CPU può lavorare fino a 48[KHz] garantendo consumi molto bassi.

Il suo Reduced Instruction Set Computer (RISC) rappresenta un set di istruzioni minimo, necessario per svolgere trasferimenti memoria-memoria, registro-registro e operazioni logiche o aritmetiche in un solo ciclo di clock. Questa scelta progettuale permette di realizzare unità di controllo molto semplici, in modo da garantire più spazio per i registri interni.

La periferica più importante, presente nel kit SAMD21 Xplained Pro, è **l'Atmel Embedded Debugger** (**EDBG**), un dispositivo USB composto da tre interfacce: un debugger, una Virtual COM Port e una Data Gateway Interface (DGI). (Fig [5.2])
Il debugger permette di programmare il microcontrollore ed eseguire il debug del codice, la Virtual COM Port è connessa all'interfaccia UART, e fornisce un modo pratico di comunicare

con l'applicazione tramite un terminale virtuale, infine il DGI è costituito da due interfacce di comunicazione la SPI e I2C, che garantiscono lo scambio di dati con il microprocessore.

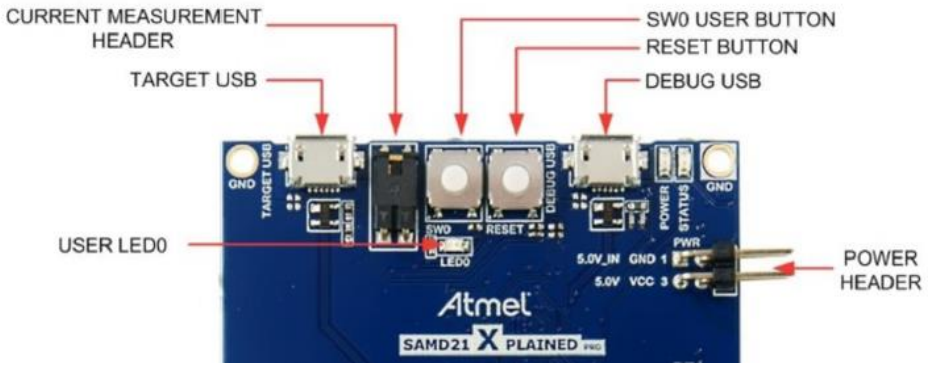


Fig 5.2 - Periferiche del microcontrollore Atmel SAMD21 XPLAINED PRO

Le periferiche principali presenti nel kit di sviluppo sono il quarzo da 32 [KHz] che può essere usato come sorgente di clock, i due pulsanti meccanici, uno che può essere programmato via software e l'altro per il reset del kit, i due led di Status Control e l'ingresso micro USB.

GPIO

Il microprocessore ATSAMD21J18A possiede 64 pins, gestiti dai **I/O Pin Controller** (**PORT**), ogni PORT gestiste un gruppo di pin e ne può controllare fino a 32 contemporaneamente.

Ogni linea di I/O del PORT è associata ad un pin presente sulla board del microcontrollore, i quali sono definiti con una particolare nomenclatura: **PXy** in cui X indica il PORT e y invece rappresenta un numero a due cifre (es. PA10). Sul kit, di fianco ad ogni pin è fornita questa indicazione.

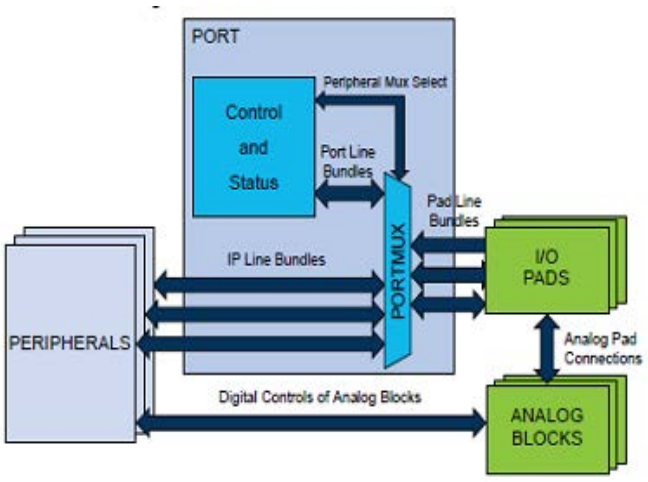


Fig 5.3 - Diagramma a blocchi del PORT

Ciascun I/O pin può essere assegnato ad una periferica presente nel MCU, abilitando la porta corretta del multiplexer, o usato come GPIO, gestito dall'applicazione. In questa ultima modalità

di utilizzo, ogni pin può essere programmato per funzionare sia come input, che come output digitale o analogico, quindi è possibile selezionare i loro driver mode.

La configurazione dei pin avviene attraverso i registri periferici del PORT, la direzione di funzionamento è gestita da un particolare bit del registro **DIR**, impostarlo ad "1" significa configurarlo come output, altrimenti lavorerà come un input pin. Il registro **OUT** invece permette di impostare il livello iniziale del pin, al bit "1" è associato un livello logico alto, mentre al bit "0" è associato al livello logico basso. Il registro Pin Configuration (**PINCFG**) è usato invece per impostare la drive mode del pin.

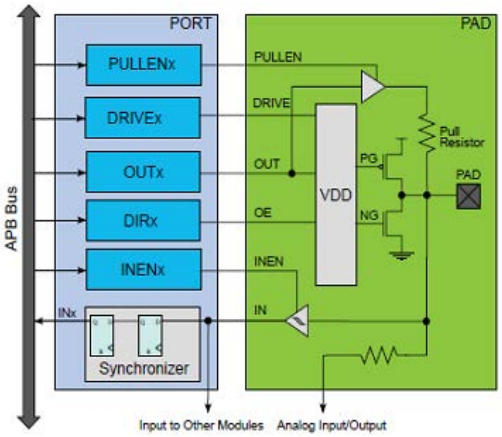


Fig 5.4 - Diagramma a blocchi dei registri del PORT

In ingresso al PORT sono presenti due segnali di clock: il clock principale di sistema, che utilizza il bus locale IOBUS per accedere al PORT e, il clock APB, derivato dal clock principale, che invece permette di configurare i registri descritti. (Fig. [5.4])

SERCOM

Nel kit SAMD21 Xplained pro è possibile implementare fino a 6 moduli per la comunicazione seriale, chiamati anche **SERCOM (Serial Communication Interface)**, che possono supportare tre modalità di funzionamento: SPI, USART e I2C. (Fig. [5.5])

Ogni SERCOM è costituito da un sistema che permette la trasmissione e la ricezione di dati controllato da un modulo **TX/RX DATA**, basato su di una unità **BAUD/ADDR** che permette sia di selezionare il clock adeguato a stabilire la velocità che di lavoro che individuare l'indirizzo corretto del dispositivo in ricezione, nell'eventualità che i dispositivi in comunicazione siano molteplici. L'unità **CONTROL/STATUS** mi permette invece, di selezionare una delle tre modalità di scambio informazioni (SPI, I2C, USART) e impostare il dispositivo, come master o come slave della comunicazione.

Per utilizzare un SERCOM bisogna configurare gli I/O pin necessari all'interfaccia selezionata utilizzando il PORT**.** Nei successivi due paragrafi, sono definiti i moduli SERCOM SPI e SERCOM USART, e quali sono le operazioni necessarie per abilitarli. Questi blocchi funzionali sono utilizzati per l'esecuzione del progetto.

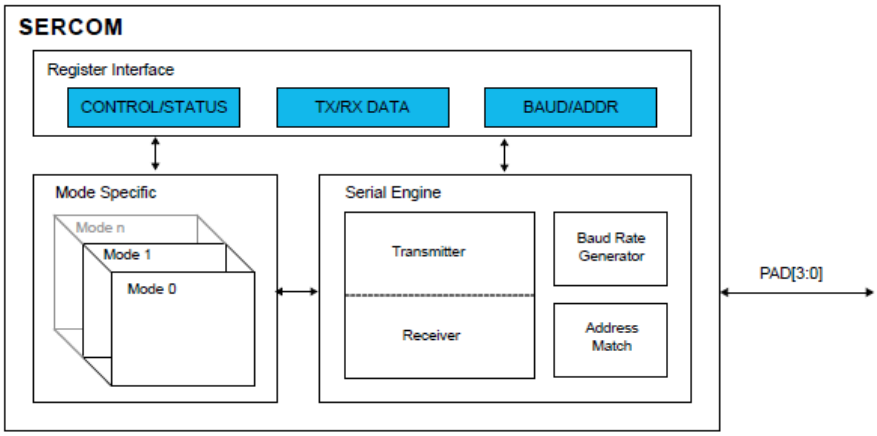


Fig 5.5 - Schema a blocchi del modulo SERCOM.

SERCOM SPI

Il **Serial Peripherical Interface** (SPI) è un'interfaccia per il trasferimento dati ad alta velocità di tipo sincrono e full-duplex, permette la comunicazione tra un dispositivo master ed uno o più dispositivi slave, connessi in parallelo. Lo scambio di informazioni è gestito attraverso 4 linee di collegamento (Fig. [5.6]):

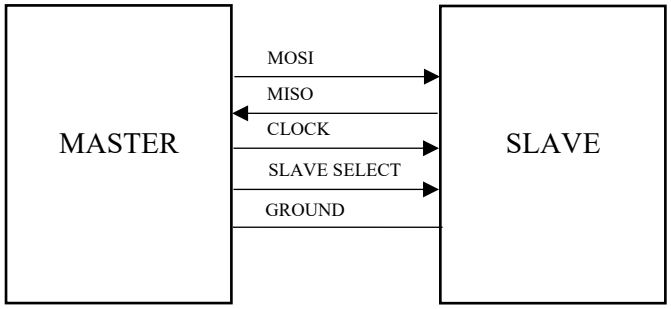


Fig. 5.6 - Diagramma a blocchi delle linee di collegamento di una interfaccia di comunicazione SPI

1. **MOSI**: Master Output Slave Input è la linea dedicata al trasferimento dati dal master allo slave.

2. **MISO**: Master Input Slave Output è la linea dedicata al trasferimento dati dallo slave al master.

3. **SCK**: Serial Clock è la linea in cui viene trasmesso il segnale di clock, generato dal master.

4. **SS**: Slave Select è una linea pull-up controllata dal master per abilitare la comunicazione.

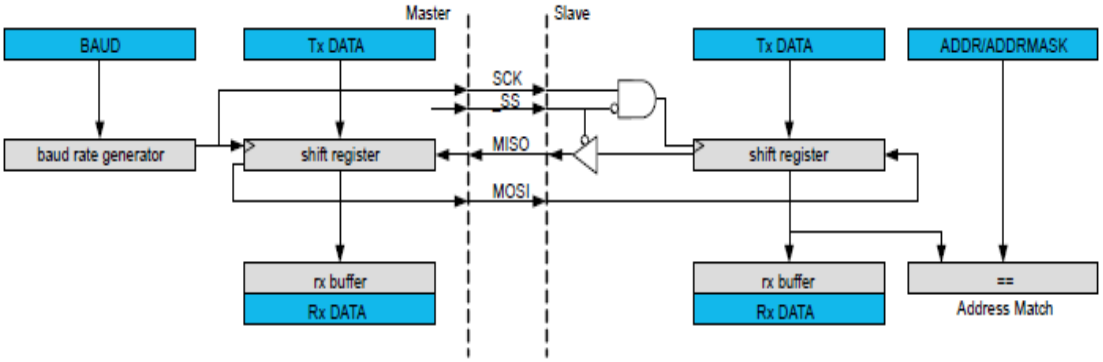


Fig. 5.7 - Schema a blocchi dei registri che gestiscono la comunicazione in un modulo SERCOM SPI

I dati da inviare sono inizializzati dai **Tx DATA** *Register* dei dispositivi in comunicazione che li caricano nei rispettivi registri a scorrimento. Se lo slave non ha informazioni utili da inviare, perché il master lo deve ancora interrogare, carica nello shift register o un dummy byte o l'ultimo valore presente nel **Tx DATA** *Register,* il master scarta i valori sapendo che non devono essere salvati. Lo slave può inviare dati utili solo in seguito alla richiesta del master. Quando la linea SS torna alta dopo un primo scambio, i valori del registro richiesto sono precaricati nello shift register.

Il master è l'unico che può iniziare lo scambio di informazioni, interroga uno slave abilitando la linea SS dedicata, che risulta quindi una linea *active low.* Entrambi i dispositivi iniziano a spedire i bit corrispondenti al carattere da inviare nel registro a scorrimento. Il master contemporaneamente genera il clock, la cui frequenza è impostata tramite il registro **BAUD** e il segnale è trasmesso sulla linea dedicata SCK. Il MSB presente nello shift register del master viene inviato sulla linea MOSI, contemporaneamente il MSB corrispondente allo shift register dello slave è invece inviato sulla linea MISO. L'ordine in cui sono inviati i bit del carattere non è predeterminato, è possibile scegliere se inviare prima il bit più significativo MSB o quello meno significativo LSB.

Ogni volta che viene trasmesso un carattere dal master, lo slave lo campiona e lo salva nel suo rx buffer, un comportamento analogo è assunto dal master.

E' importante sottolineare che può essere inviato solo un byte alla volta quando è abilitata la linea SS, successivamente torna a riposo, nel suo stato di alta impedenza. In questa configurazione, un nuovo byte viene caricato nello shift register, e quando la linea SS torna nello stato logico basso sarà trasmesso. Come già sottolineato tutto questo avviene ad alta velocità, la frequenza tipica del clock trasmesso è dell'ordine decina dei MHz.

I registri **STATUS,** permettono di avere un controllo più spinto sulla gestione degli errori o dello stato dei due buffer in ricezione. I bit appartenenti a questo registro possono generare un interrupt, che segnala al sistema se il buffer va in overrun, se è vuoto o pieno.

L'interfaccia SPI può essere impostata per trasmettere o ricevere in quattro modalità differenti, è fondamentale dal punto di vista del successo della comunicazione che il master e lo slave utilizzino la stessa modalità. Generalmente è il master che deve adeguarsi allo slave selezionando il modo opportuno, poichè la modalità con cui lo slave stabilisce la comunicazione è predeterminata dal costruttore del dispositivo.

Le quattro modalità, sono impostate definendo i bit appartenenti ai campi **CPHA** e **CPOL** del registro **CTRL.** Il registro CTRL possiede ulteriori funzionalità: permette di definire la lunghezza del carattere inviato, è possibile decidere se impostarlo ad 8 o 9 bit, tramite il campo **CHSIZE,**

la seconda funzionalità invece permette di impostare i quattro pin del modulo **SERCOM** per ricevere o inviare dati, abilitando il campo **DIPO** (*data input pinout*) si imposta la linea di trasmissione mentre abilitando il campo **DOPO** (*data output pinout*) si imposta la linea di ricezione.

Il campo **CPOL** (Clock Polarity), se impostato a "1", indica che il segnale di clock è trasmesso sfasato di 180°, rispetto al caso in cui il bit è impostato a "0".
Il campo **CPHA,** (Clock Phase) permette di decidere se campionare i bit, che transitano sulla linea di trasmissione **MISO** o **MOSI,** quando il segnale di clock è sul fronte di salita o discesa.

Le quattro modalità ammissibili, definite dal microcontrollore Atmel, sono date dalla quattro combinazione possibili tra i campi **CPOL** e **CPHA** e sono riassunte nella seguente tabella.

MODE	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Tabella 5 - Modalità funzionamento dell'interfaccia SPI

SERCOM USART

Universal Syncronous-Asyncronous Receiver Transmitter (**USART**) è un protocollo per il trasferimento dati a bassa velocità, di tipo asincrono se si decide di lavorare in modalità full-duplex, o sincrono lavorando in modalità half-duplex.

Questo protocollo, sviluppato per trasmettere i caratteri del codice **ASCII**, è utilizzato per avere un feedback visivo del codice che sta compilando, per capire, nel caso di si verifichino problemi software durante la fase di esecuzione del codice, se fosse necessario indagare ulteriormente ed eseguire il debug. Per poter utilizzare in maniera efficace il protocollo UART, è necessario disporre di un programma che simula un terminale virtuale, come ad esempio Termite o Tera Term, in questo modo i messaggi di debug possono essere stampati sullo schermo.

Un'altra applicazione tipica del protocollo UART riguarda l'invio di comandi, predefiniti nel codice, al microcontrollore utilizzando un software che simula un terminale virtuale. Digitando una sequenza di caratteri sulla tastiera, si può attivare un led o gestire un processo.

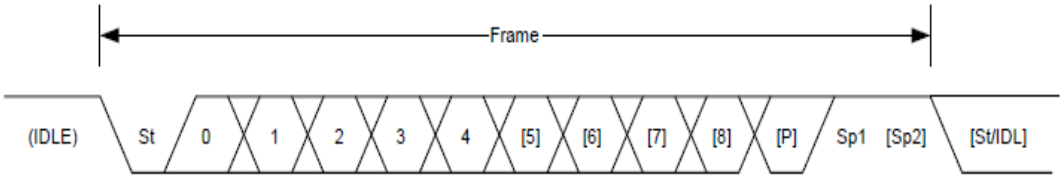


Fig. 5.8 - Invio di un carattere tramite attraverso la linea di trasmissione o ricezione.

Il frame base (Fig. [5.8]), inviato tramite la linea dedicata, è costituito da 10 bit, il primo bit definisce l'inizio del messaggio, 1 byte rappresenta il payload, mentre l'ultimo bit definisce la

fine del messaggio. Eventualmente si può decidere di avere un controllo sull'errore impostando un bit di parità definendo il campo **PMODE** del registro **CTRL.**

Prima di inviare un byte di informazione utile, è necessario impostare la velocità di trasmissione tramite il registro **BAUD.** Successivamente saranno caricati i singoli bit nel registro a scorrimento e poi inviati a partire dal più significativo MSB o meno significativo LSB. Questa scelta è possibile impostando il campo **DORD** del registro **CTRL**.

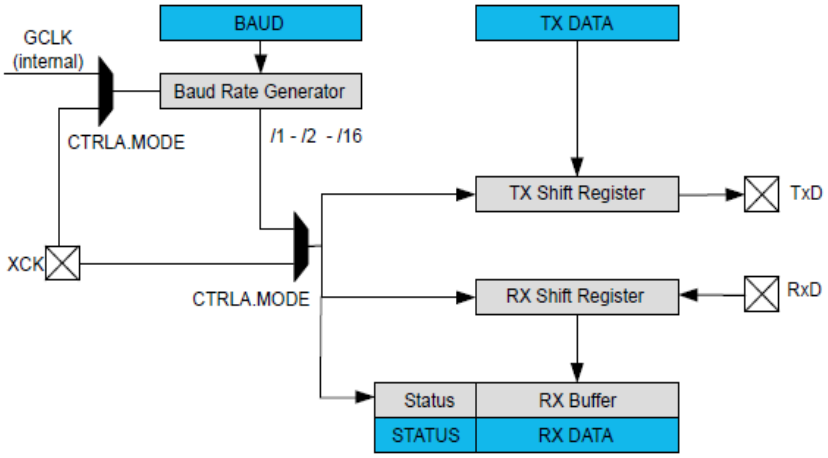


Fig. 5.9 - Schema a blocchi dei registri che gestiscono la comunicazione in un modulo SERCOM USART

WINC 1500 XPLAINED PRO

Il kit WINC 1500 Xplained Pro (Fig. [5.10]), un'estensione per i kit di sviluppo della famiglia SAM D21, può essere facilmente inserito in uno degli extension header del microcontrollore SAM D21 Xplained Pro, implementando i protocolli di comunicazione IEEE 802.11 b/g/n.

Fig. 5.10 - Modulo ATWINC1500 Xplained Pro

ATWINC1510-MR210PB Xplained Pro è un sistema embedded, alimentabile nel range di tensioni 3,0V- 3,6V. Il nucleo del modulo è costituito da un microprocessore Cortus APS3 a 32-bit con velocità di clock 40 [Mhz], che esegue il firmware del modulo, quindi anche le API che gestiscono il livello PHY e MAC. Il microprocessore è molto versatile perché può supportare diverse modalità di lavoro, come ad esempio la modalità AP e STA, e fino alla versione del firmware 19.5.2, era in grado di funzionare come client in una rete P2P.

A livello MAC, è in grado di frammentare e assemblare il frame, creare i differenti tipi di messaggi definiti dalla famiglia di protocolli 802.11 e supportare le modalità di funzionamento come STA e come AP.

A livello PHY, è in grado di ricreare le principali tecniche di modulazione spiegate nel primo capitolo, lavora in canali con ampiezza di banda di 20 MHz nella fascia di frequenze ISM dei 2.4 GHz e garantisce anche il carrier sense.

La gestione della periferica è permessa attraverso l'interfaccia SPI e l'utilizzo di 4 ulteriori pin, che permettono l'abilitazione del modulo (Fig. [5.11]). La Tabella 5 individua la corrispondenza che esiste tra le 8 linee di I/O che si interfacciano con il modulo ed i rispettivi pin sia lato microprocessore sia lato *extension header*.

Pin on Extension Header	Pin on ATWINC15x0-MR210xB module	Function
5	4	RESET_N
6	11	WAKE
9	13	IRQ_N
10	22	CHIP_EN
15	16	SPI_SSN
16	15	SPI_MOSI
17	17	SPI_MISO
18	18	SPI_SCK

Tabella 6 - Corrispondenza tra i pin del modulo ATWINC1500 Xplained Pro e del microprocessore Cortus-APS3.

Il modulo è stato progettato per funzionare come microcontrollore slave della comunicazione, quindi è possibile controllarlo tramite un microcontrollore master. ATWINC1500 è stato programmato per lavorare nella modalità SPI 0: il clock è trasmesso non sfasato, dunque il registro **CPOL** è definito con livello logico basso e, il campionamento avviene sui fronti di salita, impostando il livello logico basso anche per il bit del registro **CPHA**. La massima frequenza di lavoro è impostata a 48 [MHz]

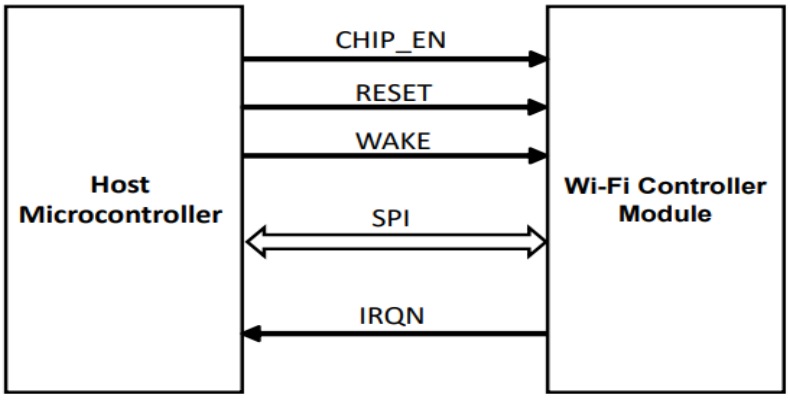


Fig 5.11 - Linee di conessione tra l'MCU master e il modulo Wi-Fi.

ATWINC1510-mr210PB implementa due interfacce UART. (Fig. [5.11])

UART 1 permette di eseguire l'upgrading del firmware e mandare messaggi di debug. La trasmissione e la ricezione di informazioni sono abilitate tramite i pin J14 e J19 del microprocessore che sono a loro volta rispettivamente collegati ai pin 6 e 4 dell'*extension header* oppure è possibile saldare dei piedini sulla traccia DEBUG UART, come si vede in figura [5.12].

UART 2 permette di impartire al modulo Atmel 1500 Xplained pro i comandi AT tramite un terminale virtuale direttamente dal pc garantendo la possibilità di eseguire una serie di comandi base. Questa modalità è disponibile solo dalle versioni del firmware precedenti la 19.5.2. I pin per la trasmissione e la ricezione lato microprocessore sono il J1 e J26 che sono collegati ai GPIO_6 e GPIO_4.

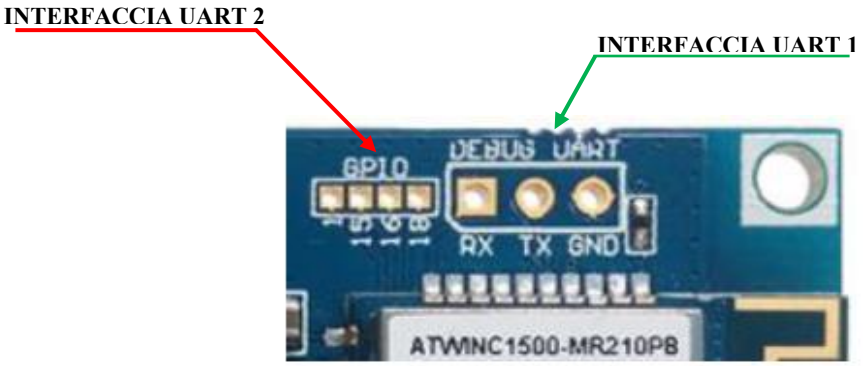


Fig 5.12 - Interfacce hardware UART 1 e UART 2

Per poter utilizzare i protocolli comunicativi UART è necessario impostare i seguenti parametri:

- Baud rate: 115200
- Data: 8 bit
- Parità: No
- Stop bit: 1 bit
- Controllo: No

Architettura del driver

Il Wireless Inteface Network Controller (WINC) **Software Driver** è una libreria C, costituita da un insieme di API che permettono di eseguire: le applicazioni realizzate per il MCU slave e gestire le operazioni nella WLAN. **Il WINC Software HOST Driver** (Fig. [5.13]), implementato

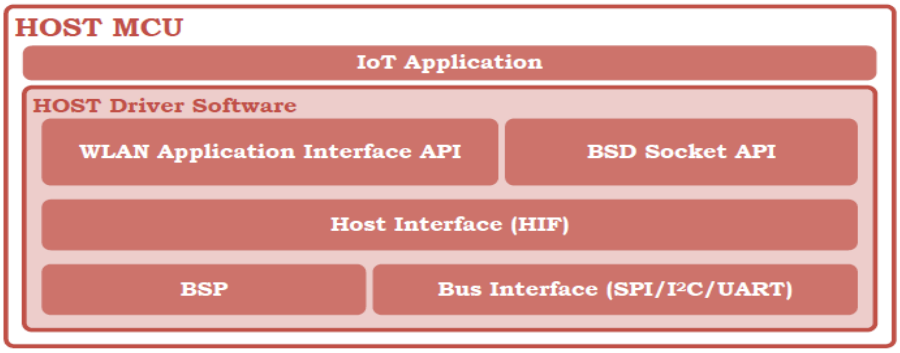


Fig. 5.13 - Architettura del Software Driver.

nel MCU master, è costituito dalle seguenti unità: **WLAN API**, **Socket API**, **Board Support Package API**, **Host Interface** e **Serial Bus Interface**.

L'unità **WLAN API** definisce un'interfaccia per realizzare le applicazioni di gestione del modulo Wi-Fi all'interno della WLAN, come ad esempio scansione degli AP, impostazione dei parametri per il funzionamento da STA e abilitazione della modalità Power Saving. Questa interfaccia è definita nel file header *m2m_wifi.h*.

L'unità **Socket API** fornisce un metodo che permette al MCU master di comunicare con MCU remoti tramite il protocollo internet, utilizzando un meccanismo basato sulle socket BSD. Questa interfaccia è definita nel header file *socket.h*.

L'unità **Host Interface (HIF)** gestisce la comunicazione tra il Driver e il firmware WINC, includendo quindi la gestione degli interrupt e gli accessi diretti alla memoria **(DMA)**. Questa interfaccia è definita nel header file *m2m_hif.h*

L'unità **Board Support Package** astrae le funzionalità di uno specifico MCU master, rendendo di fatto possibile estendere il porting del codice ad un'ampia gamma di MCU. Le funzionalità principali definite nel header file *nm_bsp.h* includono assegnazione dei pin, sequenza di accensione/spegnimento e reset.

L'unità **Serial Bus Interface**, definita nell'header file *nm_bsp_wrapper.h*, fornisce le interfacce di comunicazione I2C, SPI e UART tra il MCU master e il MCU slave, implementando le operazioni basilari di accesso al bus quali scrittura e lettura.

Architettura di ATWINC1500

L'architettura del sistema embedded ATWINC1500 (Fig. [5.14]) è costituita da due livelli: uno software ed uno hardware gestiti da un microprocessore APS3S-Cortus a 32 bit.

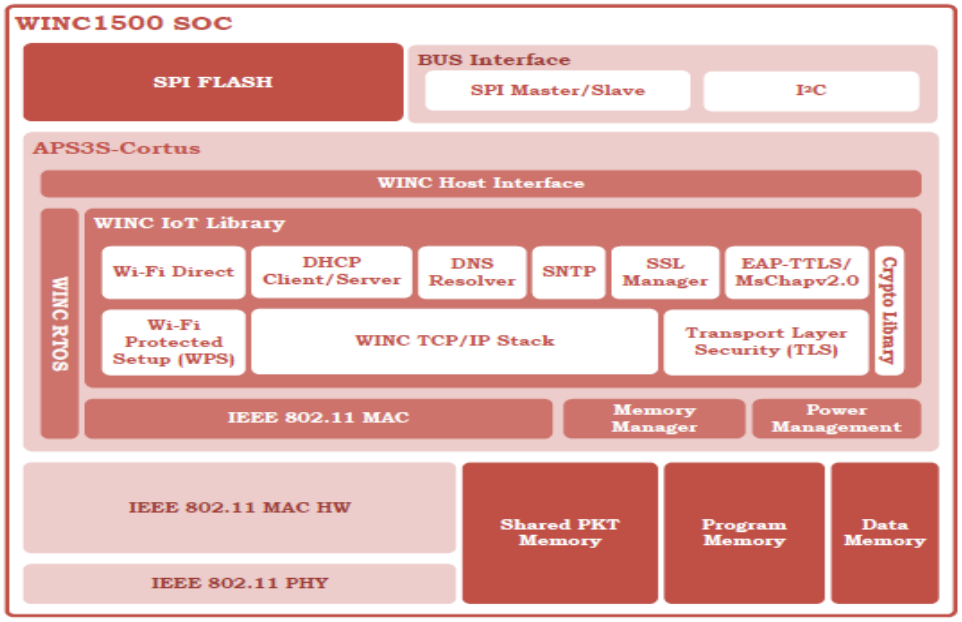


Fig. 5.14 - Architettuta di ATWINC1500

Il livello software implementa la **libreria IoT** (Internet of Thing), costituita da un insieme di file header che permettono di realizzare un set di protocolli nel firmware del WINC. I protocolli comunicativi definiti permettono sia di accedere alle applicazioni di Internet (TCP/IP/Wi-Fi) che scambiare informazioni a livello locale, come ad esempio accedere alla memoria flash da 4 MB o comunicare con un MCU host.

Il livello hardware implementa le **Interfacce Bus:** I2C e SPI permettono il passaggio dei segnali, generati dal software, attraverso le linee dedicate, per scambiare informazioni con il MCU host. L'interfaccia **SPI FLASH** permette invece l'accesso alla memoria flash. Sono inoltre installati i dispositivi per realizzare le tecniche di modulazione supportate dal modulo, quali ad esempio DSSS e OFDM, l'antenna per aumentare la distanza di trasmissione e il carrier sensing a livello PHY.

Inizializzazione del WINC

Dopo aver alimentato il WINC, è necessario eseguire una sequenza di operazioni che garantiscono un corretto funzionamento del modulo e contemporaneamente permettono di poter accedere e utilizzare con successo le API che trasmettono i frame tipici del protocollo Wi-Fi.

Non portare a termine queste operazioni significa che il WINC non potrà essere utilizzato per scambiare informazioni attraverso il canale Wi-Fi.

Inizializzare il modulo significa eseguire l'API *nm_bsp_init()* che include le seguenti operazioni:

- Esecuzione della sequenza di Reset del WINC comandando i gpio di chip_enable, wake e reset.

- Abilitazione della linea di interrupt, in modo tale che il WINC risulti agli occhi del MCU master una sorgente di interrupt che segnala l'esecuzione delle applicazioni del software WINC.

- Esecuzione della function di ritardo *nm_bsp_sleep().*

In seguito all'inizializzazione del modulo è necessario chiamare la function *m2m_wifi_init* che avvia il Driver per gestire correttamente gli eventi del firmware eseguendo i seguenti passaggi:

- Inizializzazione del bus I2C, UART o SPI in base all'interfaccia che si decide di utilizzare per comunicare con il driver.

- Assicurarsi che la versione del firmware e del driver siano compatibili, altrimenti è necessario l'aggiornamento del firmware.

- Inizializzazione del HIF e del modulo che assembla e deassembla i frame per renderli compatibili con il protocollo IEEE 802.11.

Il WINC Software Driver utilizza il protocollo HIF per garantire l'interazione tra le applicazioni del MCU master e il firmware del WINC. Le applicazioni possibili si dividono in due categorie: controllo delle operazioni di Wi-Fi e gestione delle socket. Nel primo gruppo sono presenti le seguenti operazioni: la scansione dei canali, l'identificazione della rete o i processi di connessione e disconnessione. Nel secondo gruppo invece sono presenti le operazioni di trasferimento dati attraverso il canale Wi-Fi. Questi pacchetti sono inviati dopo aver connesso il modulo ATWINC1500 alla rete.

Il MCU master, grazie all'utilizzo del Driver, riesce a garantire le operazioni fondamentali per poter utilizzare il Wi-Fi in modo asincrono, cioè quando l'applicazione chiama un API per richiedere un determinato servizio, il sistema non si blocca ad attendere la risposta. Questo comportamento è fondamentale ma soprattutto necessario quando si realizzano operazioni che richiedono un tempo significativo di attesa, come ad esempio la connessione di una STA ad un AP.

In Fig.[5.15] è analizzato il modo in cui il WINC Software Driver gestisce lo scambio di informazioni tra i due MCU. L'applicazione invia una richiesta allo slave (ATWINC1500), il frame è incapsulato nel formato previsto da HIF.

L'HIF, prima di trasmettere l'informazione, genera un interrupt per comunicare ad ATWINC1500 di prepararsi a gestire un nuovo evento. Successivamente il firmware del WINC analizza la richiesta e svolge le operazioni necessarie.

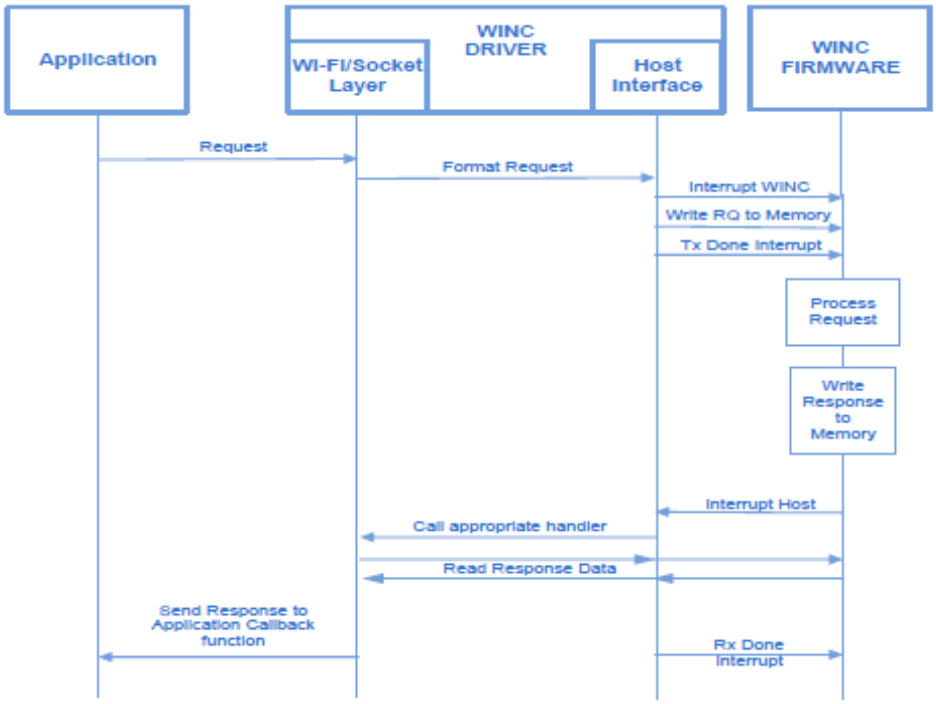


Fig. 5.15 - Gestione degli eventi in seguito alla richiesta di eseguire un API.

Quando il WINC conclude il compito richiesto, invia un interrupt al Master attraverso la linea dedicata. Il Master periodicamente chiama l'API function *m2m_wifi_handle_events( )* che individua gli eventi in attesa di essere eseguiti. Questi eventi rappresentano le risposte che il modulo WINC fornisce al master dopo aver eseguito un determinato compito. Il tipo di evento ricevuto definisce l'API function che deve essere chiamata per poterlo gestire correttamente.

Modalità di funzionamento del WINC

WINC prevede tre modalità di funzionamento (Fig. [5.16]): **Stazione**, **Access Point** e **Sniffer**. In questo sistema embedded non è prevista la possibilità di poter gestire due modalità di funzionamento contemporaneamente, ad esempio STA e AP. Prima di cambiare stato, è necessario disabilitare la modalità attiva e transitare attraverso l'Idle Mode o stato "di riposo".

Idle Mode: Dopo aver inizializzato il WINC chiamando l'API *m2m_wifi_init()*, il dispositivo rimane in uno stato di riposo, in attesa di un comando che gli permetta di cambiare modalità di funzionamento o di aggiornare i parametri di configurazione. Se non riceve nessuna operazione da eseguire il modulo entra in modalità risparmio energetico.

Il WINC entra in modalità **Stazione** quando il MCU Master richiede all'AP di connettersi chiamando l'API *m2m_wifi_connect*. Il dispositivo esce dallo stato di inattività quando riceve una richiesta di disconnessione dall'AP; o quando il Master decide di terminare la connessione chiamano l'API *m2m_wifi_disconnect*.

Il WINC entra in modalità **Access Point** a chiave condivisa o ad accesso libero quando il Master invoca l'API *m2m_wifi_enable_ap*, in questo stato è permesso ad una sola STA di connettersi ed ottenere un indirizzo IP dal WINC DHCP server. La modalità AP termina quando il Master chiamando l'API *m2m_wifi_disable_ap.*

Il WINC entra in modalità promiscua o **Sniffer mode** quando il Master chiama l'API *m2m_wifi_enable_monitoring_mode*. Il modulo può attivare questo stato senza connettersi ad alcun AP. Applicando dei particolari filtri ai pacchetti, captati nel canale wireless, Il WINC può decidere quali tenere e quali far cadere. Il Master per uscire da questa modalità chiama l'API function *m2m_wifi_disable_monitoring_mode*.

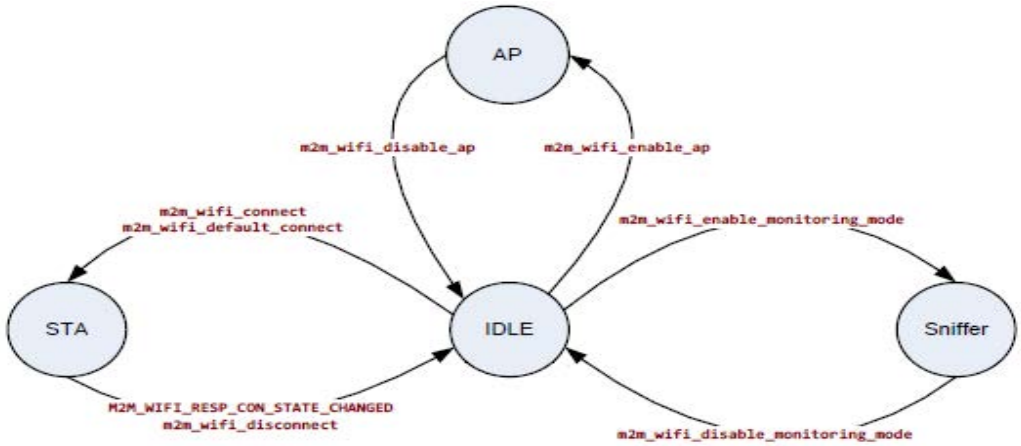


Fig 5.16 - Macchina a stati delle modalità in cui può operare il WINC

Power Saving mode

La modalità Power Saving permette di disabilitare temporaneamente i moduli dell'ATWINC1500 che si occupano di ricevere e trasmettere frame nel canale wireless. I dati, inviati alla STA, vengono immagazzinati dall'AP. Il dispositivo ATWINC1500 periodicamente, alla ricezione del frame di Beacon, si "sveglia", si sincronizza con l'AP e riceve i dati che gli spettano.

Il firmware del WINC supporta quattro modalità di risparmio energetico: Manual, Automatic, H_Automatic e Deep Automatic. Ciascuna modalità può essere abilitata dal microcontrollore Master utilizzando le seguenti API: *m2m_wifi_set_sleep_mode* inizializza una delle quattro modalità e, *m2m_wifi_set_lsn_int* permette di configurare l'intervallo di attesa, esprimendolo in periodi di Beacon o Timestamp (100 msec).

Manual

La modalità Manual è completamente gestita dal MCU Master e viene attivata chiamando l'API *m2m_wifi_request_sleep*. In questo stato il modulo Wi-Fi non può svegliarsi autonomamente per poter ricevere i frame di Beacon ma deve essere abilitato dal Master. Periodi di disattivazione prolungati, fanno disassociare la STA dalla WLAN e, se la connessione dovesse cadere, il WINC può accedere nuovamente alla rete solo in seguito al successivo evento di "risveglio". L'AP avvia un timer interno ogni volta che viene stabilita un'associazione, se periodicamente non riceve un frame di keep-alive considera la connessione terminata. Il Master potrebbe scegliere l'intervallo di tempo in cui il modulo è "dormiente", facendo un compromesso tra il massimo risparmio di energia ottenibile, e il tempo minimo necessario all'AP per non considerare il WINC disconnesso.

La modalità automatica è preferibile quando il WINC deve scambiare raramente frame con la rete oppure quando deve inviarli frequentemente in un determinato intervallo, ma tra un intervallo e l'altro trascorre molto tempo.

Il vantaggio principale che assicura questa modalità rispetto a quella in cui il modulo non è alimentato sta nella prontezza della risposta del WINC. Ogni volta che si alimenta il WINC, è necessario caricare in memoria il firmware, mentre nella modalità Power Save risulta già caricato.

H Automatic

La modalità H_Automatic permette il "risveglio" periodico del WINC per captare i frame di Beacon. La STA che abilita questo stato, disattiva i moduli che gestiscono il sistema a livello PHY e MAC, il segnale di clock che gestisce gli eventi del microprocessore rimane invece attivo, permettendo alla STA appena svegliatasi di trasmettere un frame.

Deep Automatic

Nella modalità Deep Automatic il MCU slave disabilita i clock del sistema prima di "addormentarsi" ed avvia un timer che ne programma il "risveglio". Il WINC, eventualmente, può svegliarsi nel caso in cui il MCU master richieda l'esecuzione di un'applicazione chiamando l'API dedicata al processo.

Applicazioni: Connessione ad un Access Point

La prima applicazione che ho realizzato si poneva come obiettivo la dimostrazione del fatto che un modulo ATWINC1500 Xplained Pro, funzionando in modalità STA, sia in grado di eseguire la scansione delle reti wireless presenti nella sua zona.

Ho deciso di dividere l'applicazione in due sotto-parti per mantenere una distinzione marcata tra i due processi, e poterli descrivere più chiaramente: la prima parte riguarda la scansione della WLAN mentre la seconda riguarda la connessione alla rete.

Il programma, realizzato da ATMEL ed eseguito dal microcontrollore, permette di trovare gli access point presenti nella zona e di associarsi ad uno di essi.

L'associazione ad un AP è permessa solo se sono noti il suo identificativo (SSID) e la sua chiave di accesso. Inserendo nel header file *main.h* le informazioni relative alla WLAN domestica gestita da "TP-LINK_BB0FEO", il modulo WI-Fi si è connesso all'AP.

In questo progetto il microcontrollore abilita l'interfaccia SERCOM USART, in modo da ricevere un riscontro visivo sia in caso di errore, realizzando dunque un debug visivo, che sullo stato dei processi dell'applicazione quali: scansione, autentificazione e associazione. Le informazioni, raccolte dal modulo, sono stampate a schermo sul Terminal Window, un software che simula un terminale virtuale.

Scansione

Il dispositivo Wi-Fi è stato progettato per lavorare solo con le frequenze intorno ai 2.4 [GHz] della banda ISM, nella quale il numero di canali è variabile a seconda della regione geografica: i canali disponibili in Asia sono quattordici mentre quelli disponibili in Nord America ed Europa sono undici ma, di questi undici, solo tre non presentano una sovrapposizione di banda. I canali scelti per il Nord America sono l'1, il 6 e l'11 mentre quelli per l'Europa sono l'1, il 5, e il 9.

Il processo di scansione della WLAN si articola nei seguenti passaggi:

1. Il modulo ATWINC1500 invia dei frame broadcast di Probe-Request per individuare tutti gli AP presenti nelle vicinanze.
2. Gli AP che captano il frame inviano dei messaggi di Probe-Response contenente alcune informazioni sulla loro WLAN, come ad esempio il loro SSID.
3. Il modulo ATWINC1500 riceverà i frame di Probe-Response, analizzerà i risultati dell'operazione di scanning, e selezionerà la rete a cui vuole accedere facendo un confronto tra la SSID che conserva in memoria e quelle ricevute.
4. Una volta individuato l'AP con cui vuole connettersi, ATWINC1500 invierà un ulteriore Probe-Request contenente il nome della rete e la password di accesso. Successivamente inizierà il processo di associazione.

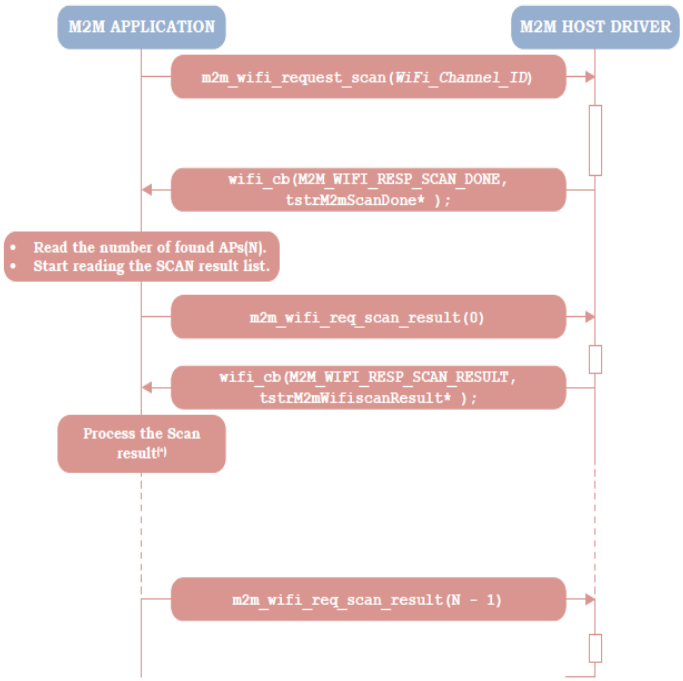


Fig. 5.17 - API function chiamte dal MCU master per eseguire la scansione del canale.

Il diagramma logico in Fig. [5.17] evidenzia quali sono le API da chiamare per eseguire la scansione del canale dopo aver inizializzato il modulo Wi-Fi e il WINC Software Driver.

La prima applicazione eseguita riguarda il ritrovamento dell'AP a cui desidero connettermi; successivamente ho inserito nel programma il nome della SSID, ma non la chiave di accesso. Non conoscendo il canale wireless in cui opera il mio AP, ho inserito il comando **M2M_WIFI_CH_ALL** nell'API ***m2m_request_scan()***, in modo tale che eseguisse la scansione su tutti i canali della banda 2.4 [GHz]. E' possibile, nell'eventualità che sia noto il canale in cui opera il proprio AP, inserire nell'argomento di ***m2m_request_scan*** **(WiFi_Channle_ID)** il numero del canale**.**

Il modulo Wi-Fi esegue la scansione, ritorna il numero e il nome degli AP trovati, il WINC Driver Software comunica il risultato ritornando la function **wifi_cb()**, e stampa a schermo il risultato.

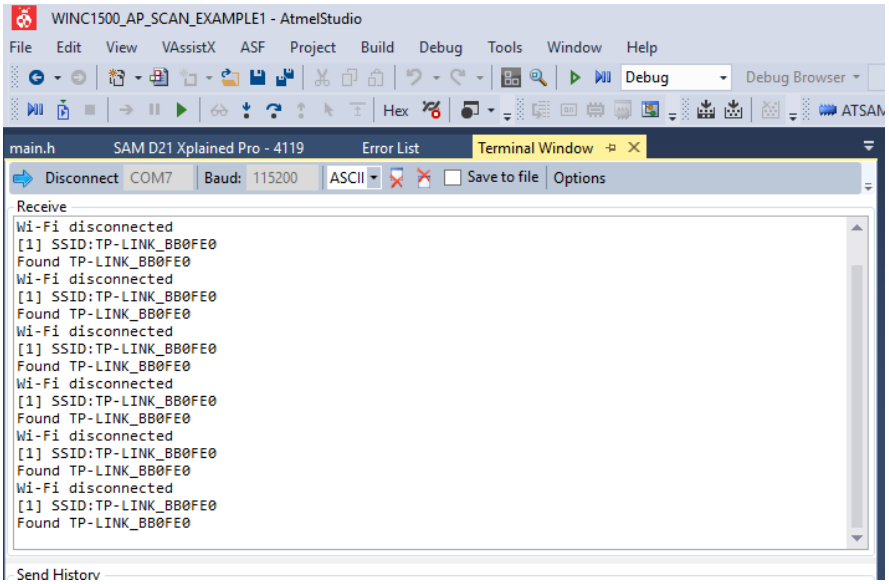


Fig. 5.18 - scansione della rete WLAN gestita dall'AP TP-LINK_BB0FE0

Siccome non è stata inserita la chiave di accesso il processo degenera in un loop infinito.

Associazione

Il processo di connessione ad una rete a chiave condivisa, si articola nei seguenti passaggi:

1. Il modulo ATWINC1500 inizia il processo di autenticazione inviando un frame di richiesta di Autenticazione, contenente il nome dell'AP e il campo *challenge text* lasciato appositamente vuoto. La richiesta di Autenticazione è necessaria per capire se la STA ha le caratteristiche adeguate ad essere aggiunta alla rete dall'AP.
2. L'AP risponde immediatamente fornendo il *challenge text.*
3. Il modulo Wi-Fi utilizza la chiave condivisa della rete per decriptare il *challenge text*, una volta terminata l'operazione trasmette la soluzione all'AP
4. L'AP a sua volta decripta il *challenge text* e lo confronta con quello appena ricevuto, se i due coincidono accetta l'autenticazione della STA e risponde con un frame risposta di Autentificazione, altrimenti invia un messaggio di errore e termina automaticamente il processo di associazione.
5. La STA quindi invia un frame di richiesta di Associazione a cui risponderà l'AP con un frame risposta Associazione.

Il diagramma logico in figura [5.19] evidenzia quali sono le API da chiamare per connettersi ad una rete a chiave condivisa. Gli argomenti passati a *m2m_wifi_connect()* sono: il tipo di sicurezza

adottato per proteggere la rete, il nome della BSS, il canale in cui opera e il campo Key contenente la password per decriptare il testo.

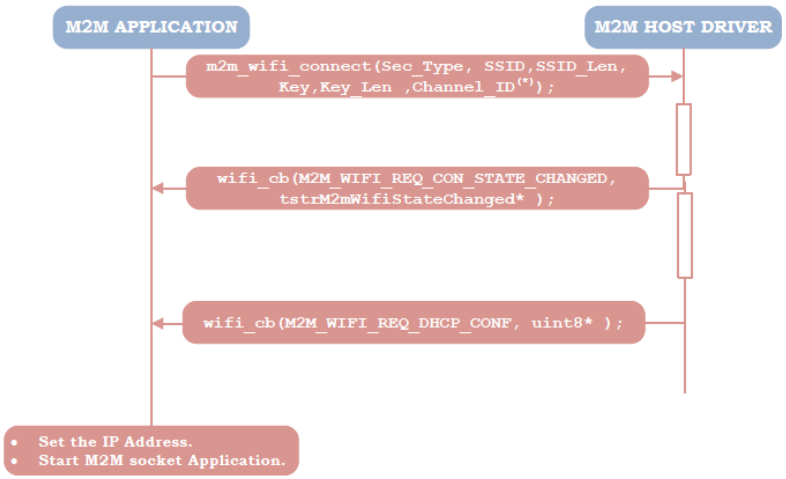


Fig. 5.19 - API function chiamte dal MCU master per eseguire la scansione del canale.

La prima chiamata di **wifi_cb()** informa il *master* sul cambio di stato, ora la STA è associata. La seconda chiamata di **wifi_cb()** comunica l'assegnazione di un indirizzo IPv4 all'interfaccia di rete del modulo. La richiesta di un indirizzo IP è eseguita in maniera automatica dopo aver ottenuto lo stato di connessione.

La figura [5.20] mostra il feedback visivo delle operazioni compiute dal microcontrollore durante l'esecuzione del codice per scansionare e accedere ad una rete. Il primo blocco di informazioni riguarda l'abilitazione dell'interfaccia seriale SPI, vengono comunicate le informazioni reperibili nella memoria del modulo Wi-Fi, quali versione e data di realizzazione del Software Driver WINC e del WINC Firmware, ID del chip e la data in cui è stato compilato il codice. Il secondo blocco di informazioni invece riguarda le operazioni di scansione della rete, connessione all'AP e assegnazione dell'indirizzo IP.

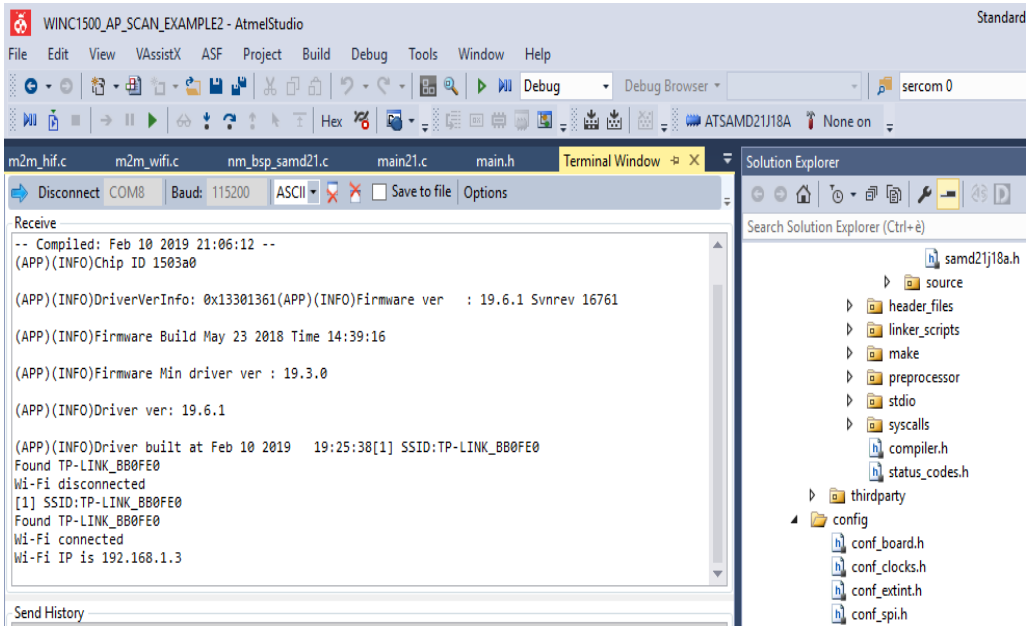


Fig. 5.20 - Associazione con la rete WLAN e assegnazione di un indirizzo IPv4

Applicazioni: comunicazione Client/Server

TCP è un protocollo di livello trasporto che garantisce la corretta ricezione di tutti i byte trasmessi. L'affidabilità di questo sistema si fonda sullo scambio di frame ACK incapsulati nelle risposte del ricevitore. La trasmissione dei messaggi in entrambe le direzioni è possibile solamente una volta realizzato il canale di comunicazione virtuale full-duplex di tipo end to end. Le estremità del canale o socket sono definite da quattro campi: l'indirizzo IP del destinatario o del ricevitore, il loro numero di porta e dal protocollo utilizzato.

La seconda applicazione da me eseguita riguarda la comunicazione tra un processo server e un processo client che realizza un canale virtuale in modo che i due processi si scambino i dati secondo il protocollo TCP. É possibile sviluppare il progetto secondo due modalità: utilizzando due MCU oppure attraverso un solo MCU e un terminale virtuale che può svolgere il compito di TCP/IP server o client.

In questo progetto è stata seguita la prima strada: entrambi i MCU funzionano in modalità STA e devono connettersi alla WLAN per poter scambiare informazioni ed in particolare accendere un led. La prima operazione è l'associazione con l'AP secondo le modalità viste nel paragrafo precedente.

Il software realizzato da ATMEL e utilizzato per questa applicazione permette di abilitare il MCU sia in modalità server che in modalità client. Scrivendo nel Terminal Window un messaggio preceduto dalla sequenza **"<<"** viene eseguita un'operazione locale, come ad esempio la creazione di una socket per il processo client. É possibile accedere ad un processo Server remoto, come ad esempio il controllo di un led, digitando la sequenza **">>"** prima di un messaggio. In questo modo è possibile comandare un led.

Processo Client

Dopo aver inizializzato il modulo ATWINC1500, secondo le modalità viste nei precedenti paragrafi è possibile associarsi all'AP TP-LINK_BB0FE0, questo permette l'assegnazione di un indirizzo IPv4 alla STA che di conseguenza la renderà riconoscibile in rete. Nel *main.c* vengono eseguite le API **socketInit()** e **registerSocketCallback():** la prima inizializza la libreria socket mentre la seconda si occupa di gestire in maniera asincrona gli eventi legati all'utilizzo delle socket.

L'ultima API restituisce informazioni riguardo la connessione client/server, se si vuole, ad esempio, stipulare una connessione con il server si possono generare due eventi: nel caso in cui il server accetti la richiesta il MCU master viene avvertito della corretta connessione; nel caso in cui sia il processo server presenti 30 secondi di inattività, viene fornito un resoconto negativo.

Digitando in Terminal Window il comando "<<connect 192.168.1.13", il microcontrollore esegue alcune function che gli permetteranno di capire che l'operazione richiesta è un processo locale per la creazione della socket. Successivamente viene eseguito l'API **connect()** che permette l'associazione con il microcontrollore in remoto. La risposta a questa richiesta viene comunicata nel **SOCKET_MSG_CONNECT** e inviata a

registerSocketCallback( ) che stampa a schermo "*socket_cb: connect success*.".(Fig. [5.21])

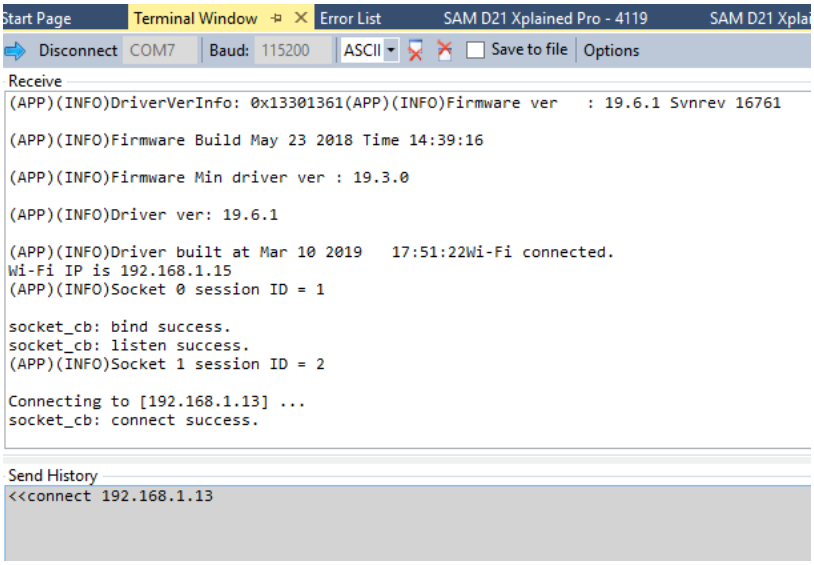


Fig. 5.21 - Connessione stabilita tra processo Client e Server

É ora possibile inviare messaggi al server facendo seguire alla sequenza ">>" il testo che si desidera trasmettere in un pacchetto TCP/IP. É possibile inoltre comandare un led in accensione o spegnimento digitando i codici *">>control ledon*" e *">>control ledoff*". Questi comandi vengono analizzati dal software e passati all'API **send( )** che contiene quattro campi di input: il socket ID, caratterizzato dal numero di porta e dall'indirizzo IPv4, il payload e la sua dimensione in termini di byte. Dopo che il WINC ha trasmesso i dati, ritorna un evento del tipo **SOCKET_MSG_SEND** gestito dal'API registerSocketCallback( ) che informa il MCU sul numero di byte inviati, se questo evento torna zero o un valore negativo significo che non è stato ricevuto un segmento di acknowledgement.

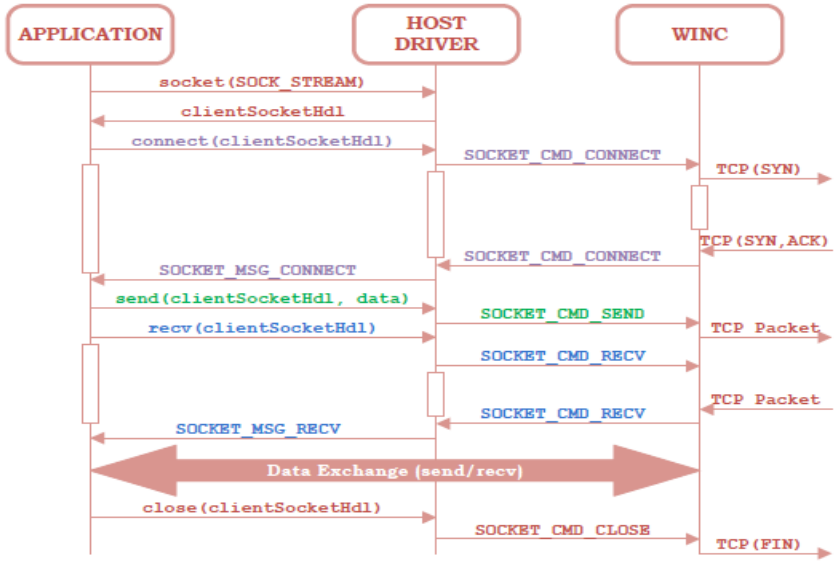


Fig. 5.22 - API function chiamte dal MCU master per realizzare il processo client e scambiare dati con il processo server.

Ogni volta che l'API connect( ) va a buon fine è necessario chiamare l'API **recv( )** altrimenti i dati ricevuti vengono immagazzinati nel firmware del WINC utilizzando le risorse del sistema in maniera non appropriata finché non è chiusa la connessione tra i due processi**.** Per questa API viene allocato un buffer in maniera tale che quando viene individuato un evento del tipo **SOCKET_MSG_RECV** i dati ricevuti vengono immagazzinati in questo array.

Processo Server

Il processo Server, eseguito sul secondo microcontrollore, si distingue dal processo client perché dopo aver inizializzato il sistema e realizzato la socket che permette di accedere al canale di comunicazione virtuale, è necessario chiamare l'API **bind().** Vengono eseguite delle operazioni che permettono di associare alla socket un indirizzo IPv4, in questo caso 192.168.1.13 e un numero di porta. Quando l'API bind() è conclusa si genera un messaggio del tipo **SOCKET_MSG_BIND** gestito dal'API registerSocketCallback( ) che permette di mettere il sistema in uno stato di "ascolto" verso le connessioni dei processi Client.

Il sistema attende di essere contattato e quindi si pone in stato di listening chiamando la socket API **listen()**, a schermo sono visualizzato i messaggi "*socket_cb: bind success*." E "*socket_cb: listen success."* che indicano la corretta esecuzione delle API bind() e listen() (Fig. [5.23]). Nell'eventualità che ci fosse una richiesta di connessione, questa viene accettata automaticamente senza dover chiamare la socket API **accept().**

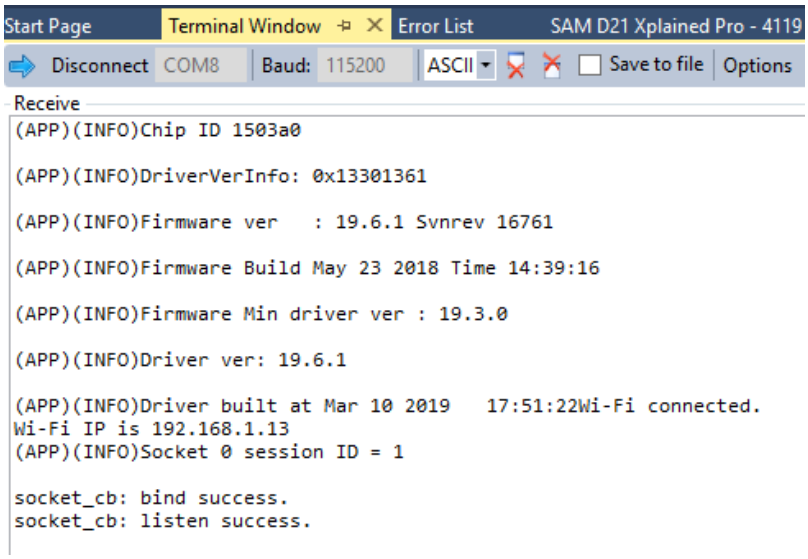


Fig. 5.23 - Creazione delle socket, il processo server si pone in stato di ascolto.

La modalità "ascolto" genera un evento del tipo **SOCKET_MSG_LISTEN**: questo messaggio aggiorna il MCU sullo stato del processo server, quindi indica se il WINC, che agisce da server, è pronto ad accettare nuove connessioni. Questa modalità genera un secondo evento: il WINC, dopo aver comunicato che può accettare nuove connessioni manda un messaggio del tipo **SOCKET_MSG_ACCEPT** che una nuova connessione è stata accettata. Entrambi gli eventi sono gestiti dal WINC software Driver tramite la socket API registerSocketCallback( ).

Una volta terminato lo scambio di informazioni, il canale virtuale può essere chiuso chiamando la socket API close(), e le risorse messe a disposizione per la socket vengono rilasciate. Dato che il canale di comunicazione è full-duplex, questa operazione deve essere svolta sia dal processo server che dal processo client.

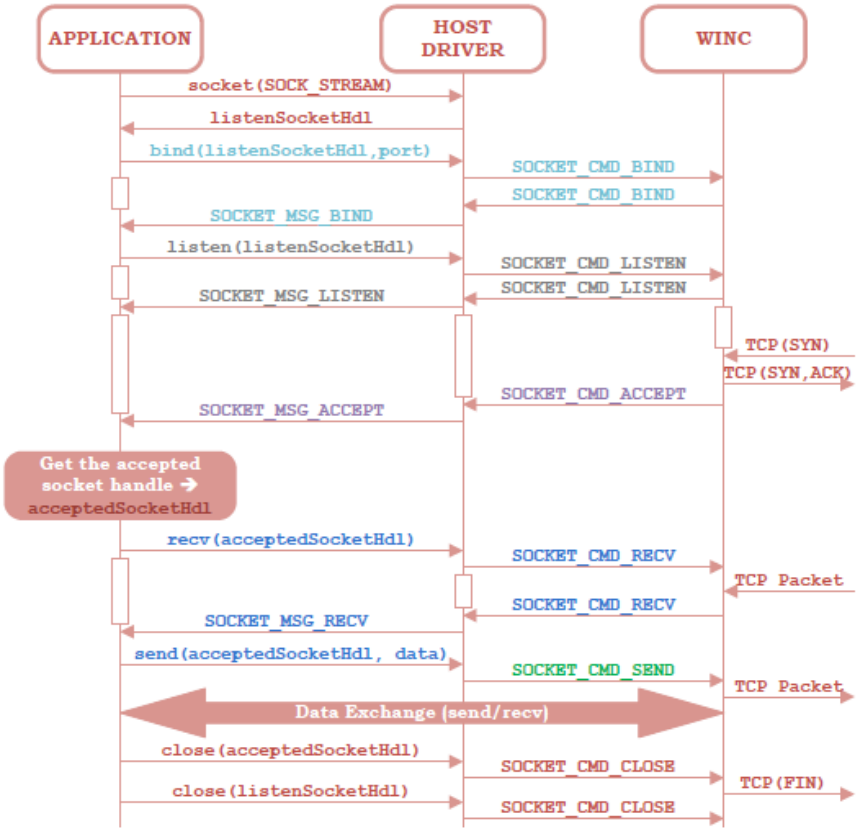


Fig. 5.24 - API function chiamte dal MCU master per realizzare il processo server e scambiare dati con il processo client

Capitolo 6 - Progetto Cypress

Sistema Cypress: CY8CKIT-059 PSoC 5LP

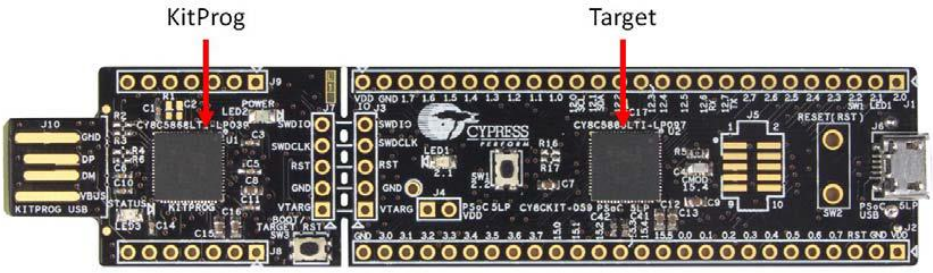


Figura 6.1 - CYCKIT-059 PSoC 5LP Prototyping kit.

Il **CY8CKIT-059 PSoC 5LP Prototyping Kit**, un sistema embedded programmabile, è costituito da due moduli separabili**: il microcontrollore Target ed il KitProg**. (Fig [6.1])

Il **KitProg** è un modulo fondamentale perché permette di eseguire la programmazione del microprocessore Target e il debug del codice. Il kitProg inoltre fornisce un'interfaccia USB-UART e USB-I2C tra i due moduli, le linee dedicate alla comunicazione sono collegate a specifici pin del microcontrollore Target: ad esempio, UART_RX e UART_TX, le linee dedicate alla trasmissione e ricezione dati del protocollo UART, sono collegate rispettivamente ai pin **P12_6** e **P12_7**. Separando il KitProg dal Prototyping Kit, è possibile programmare ed eseguire il debug di qualsiasi dispositivo appartenente alle famiglie PSoC 3, PSoC 4 e PSoC 5, collegandolo in maniera opportuna.

Il **microcontrollore Target** è un sistema embedded costituito da un dispositivo della serie PSoC5 LP. Il nucleo del modulo, il microprocessore ARM-Cortex M3, gestisce le periferiche software e hardware come ad esempio: il pulsante e il led programmabili, l'ingresso Micro-USB, e i due Expansion Header, costituiti da 20 I/O pin ciascuno.

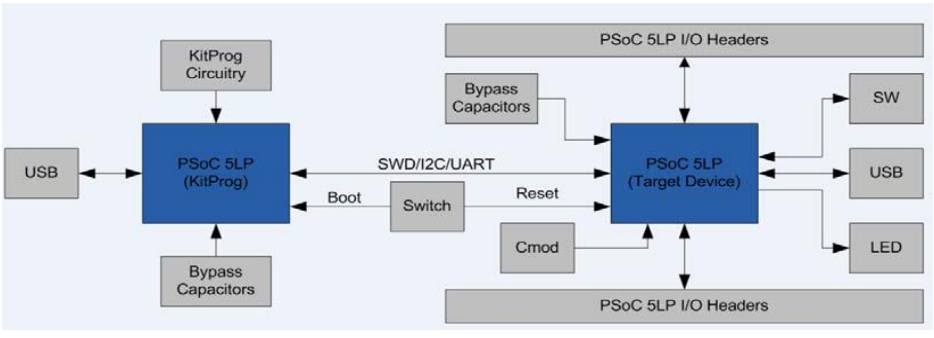


Figura 6.2 - Diagramma a blocchi di CYCKIT-059 PSoC 5LP Prototyping Kit

La maggior parte delle applicazioni, realizzabili con il CY8CKIT-059 PSoC 5LP Prototyping Kit, richiedono 5,0 [V] di alimentazione. L'energia elettrica, necessaria ad alimentare i progetti che si desiderano sviluppare con il microcontrollore, può essere acquisita secondo tre differenti modalità: attraverso la porta USB disponibile lato KitProg, utilizzando la porta micro-USB presente dalla parte del microcontrollore Target oppure collegando una sorgente di alimentazione esterna ai pin $V_{DD,}$ situati sull'extension header. É importante osservare che gli ultimi due modi

definiti per alimentare il kit di sviluppo, non trasferiscono energia al KitProg che risulta quindi non alimentato.

Architettura piattaforma PSoC 5LP

I dispositivi della famiglia PSoC 5LP (Fig. [6.3]) sono costituiti dai seguenti moduli: **sotto-sistema digitale programmabile, sottosistema analogico programmabile, sistema dei Clock, blocco di Power System, I/O System.** Di seguito è fornita una breve spiegazione dei moduli, per approfondimenti si rimanda al manuale tecnico del PSoC 5LP [16], [17].

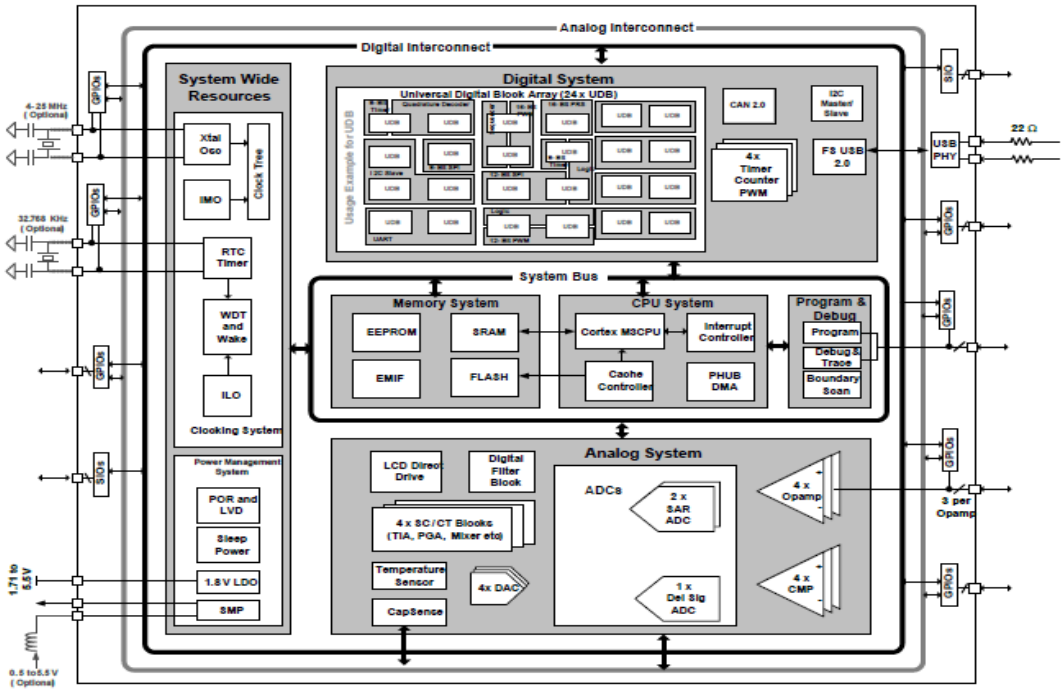


Fig. 6.3 - Architettura del dispositivo PSoC 5LP

Sotto-sistema digitale programmabile

La potenza dell'architettura del PSoC 5LP risiede nella sua flessibilità. É possibile infatti realizzare facilmente un'applicazione complessa perché non bisogna relazionarsi direttamente con i registri. L'ambiente di sviluppo, PSoC Creator 4.2, fornisce un'interfaccia grafica che permette di configurare in maniera più rapida e intuitiva i registri. Questo programma mette a disposizione una libreria di elementi connessi alla matrice di Universal Digital Block (**UDB**).

Il sotto-sistema digitale programmabile garantisce, grazie al **sistema digitale di interconnessione** (**DSI**), la trasmissione di un segnale digitale da una periferica a qualsiasi I/O pin, permette inoltre la programmazione della matrice di UDB per realizzare una periferica digitale standard, come ad esempio il blocco PWM o UART, oppure di crearne una personalizzata, come ad esempio l'interfaccia di un sensore.

Il componente principale del sotto-sistema digitale programmabile è l'Universal Digital Block (UDB) (Fig. [6.4]), costituito dai seguenti componenti: Due **Programmable Logic Device** (**PLD**), cioè una doppia matrice di porte logiche che implementano le funzioni booleane, un

Datapath, il cui elemento fondamentale è l'ALU, un modulo **Status and Control** ed uno **Clock and Reset Control**.

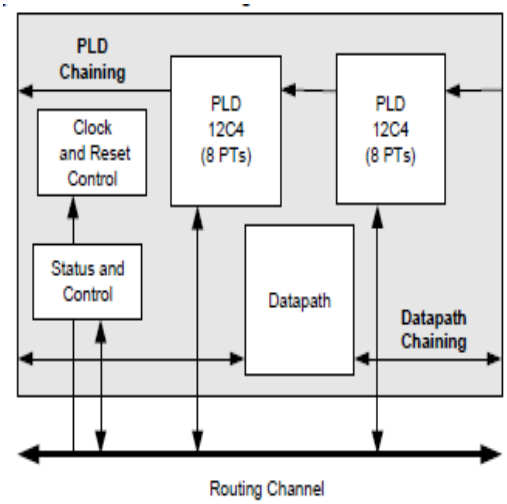


Fig. 6.4 - Schema a blocchi di un UDB

I 2 **Programmable Logic Device** sono usati per implementare macchine a stati, tavole di look-up e possono essere configurati per svolgere funzioni aritmetiche.

Il **Datapath** è il modulo principale del blocco UDB perché comprende un'Unità Logico Aritmetica (ALU) a 8-bit. Il Control Store è un altro componente chiave all'interno del modulo, immagazzina le microistruzioni usate per realizzare l'instruction set dell'ALU[4] e permette di selezionarle real-time durante l'esecuzione di ogni ciclo.

Il modulo **Status/Control** coordina l'interazione tra il firmware e le operazioni svolte dal blocco UDB.

Come già accennato gli UDB permettono la realizzazione sia di periferiche digitali personalizzate che di già esistenti, di seguito elenco quelle standard classificandole in tre gruppi.

- **Periferiche di comunicazione**: I2C, UART, SPI.
- **Funzioni**: PWM, Timer, Counter.
- **Porte Logiche**: OR, AND, NOT, XOR.

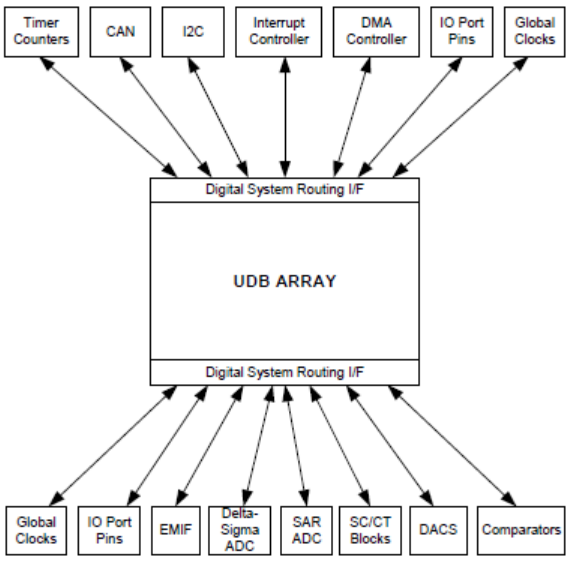


Fig. 6.5 - Sotto-sistema digitale programmabile

Collegando più UDB è possibile formare una matrice interconnessa, chiamata anche UDB Array. L'UDB array e il **Digital System Interconnection** (DSI) costituiscono il sotto-sistema digitale.

[4] ALU instruction set: somma, sottrazione, incremento, decremento, AND logico, OR logico, XOR logico.

Il DSI permette, tramite un sistema di bus l'interazione tra i segnali generati dai blocchi UDB, e le periferiche esterne come ad esempio: DMA, interrrupt e I/O pin.

Sotto-sistema analogico programmabile

Il sotto-sistema analogico programmabile, combinando tra loro i blocchi funzionali[5] programmabili, sia standard che avanzati, permette la realizzazione di specifiche applicazioni il cui compito è quello di processare i segnali analogici che vengono indirizzati tramite un sistema di bus. L'architettura del sistema di bus, costituito da **Analog Global Bus**, **Analog MUX Bus** e **Analog Local Bus**, garantisce un'elevata flessibilità, rende di fatto possibile l'instradamento dei segnali analogici sia verso differenti blocchi funzionali che verso i GPIO. (Fig.[6.6])

L'**Analog Global** (**AG**) **Bus** è costituito da due schiere di otto bus ciascuna, una disposta sul lato sinistro e l'altra sul lato destro. Gli otto AG Bus presenti sul lato sinistro sono collegati sia ai blocchi funzionali analogici che ai GPIO, discorso analogo per gli AG bus del lato destro. (Fig. [6.7])

L'**Analog MUX Bus** (**AMUXBUS**) è costituito da due bus che possono essere connessi insieme. Un sistema di switch analogici permette il collegamento tra l'AMUXBUS e i GPIO. (Fig.[6.7])

Ci sono otto **Analog Local Bus** (**abus**), disposti in due gruppi distinti, quattro nel lato sinistro e quattro nel lato destro, direttamente connessi ai blocchi funzionali analogici ma non ai GPIO. La loro implementazione nel sistema permette di ridurre il consumo degli AG bus.

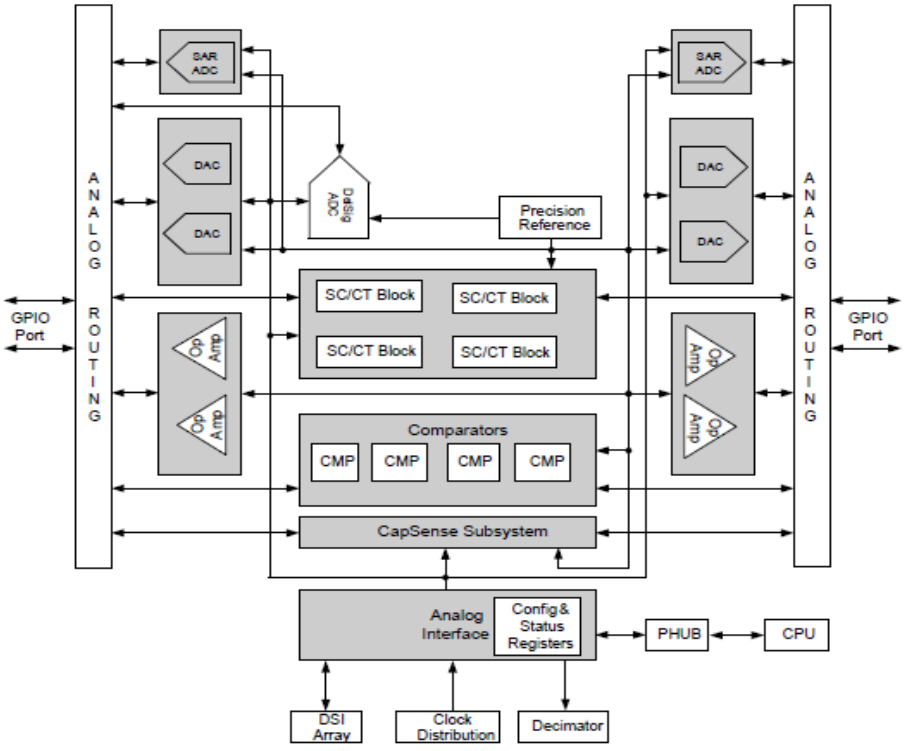


Fig. 6.6 - Sotto-sistema analogico programmabile.

[5] I blocchi funzionali analogici sono: DAC (4), comparatori (4), CapSense (2), Delta Sigma ADC (1) e opamp (4), switched capacitor (4).

Gli **Analogic Switches** e i **Multiplexer** sono invece utilizzati per instradare i segnali tra i blocchi funzionali analogici oppure verso i GPIO. Ogni GPIO è costituito da due switch analogici, uno permette la connessione con l'AMUXBUS e l'altro con gli AG bus.

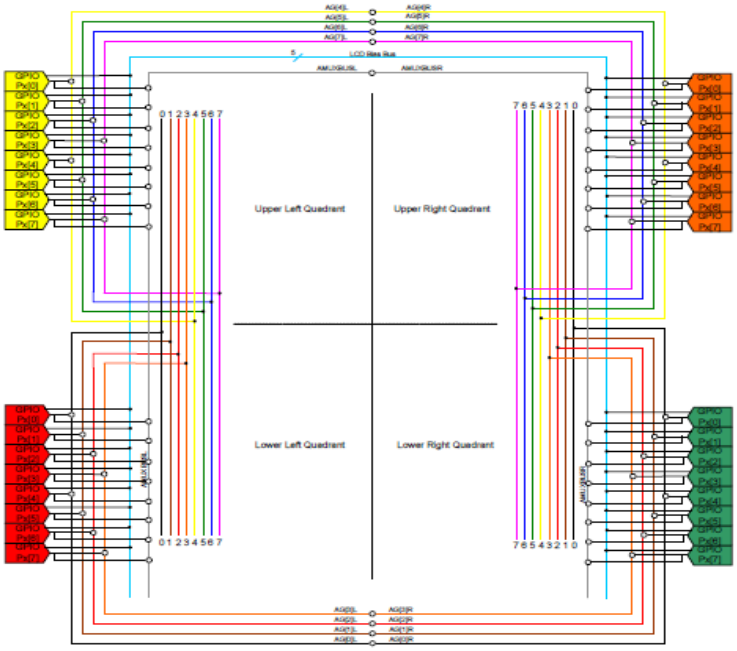


Fig. 6.7 - Architettura parziale del sistema di bus: lo schema comprende l'AG bus e l'AMUXBUS.

Sistema dei clock

Il sistema dei clock genera, divide e distribuisce il segnale di clock in tutte le periferiche del PSoC che lo richiedono, permettendo quindi di raggiungere elevate prestazioni in ogni progetto. Questo sistema include sette sorgenti di clock e permette di crearne di aggiuntive, a frequenze differenti, utilizzando i blocchi UDB e il 16-bit clock divider. Un'applicazione può essere ad esempio la realizzazione del baud rate desiderato per il modulo UART.

1. **Internal Main Oscillator** (**IMO**) è la sorgente di clock utilizzata nelle maggior parte delle applicazioni, riesce a erogare segnali di clock alle seguenti frequenze: 3, 6, 12, 24, 48, 74 MHz, garantendo una precisione del 1%.

2. **Phase-Locked Loop** (**PLL**) fornisce segnali di clock a bassa frequenza ma elevata precisione che però possono essere moltiplicati per estendere il campo di funzionamento in frequenza, valori tipici ricadono nel range di frequenze 24-80 MHz.

3. **Clock Doubler** fornisce in uscita un segnale di clock con frequenza doppia rispetto alla frequenza del segnale in ingresso.

4. **Internal-Low Speed Oscillator** (**ILO**) genera segnali di clock a 1 KHz, 33 KHz e 100 KHz, garantendo un consumo di energia davvero basso. ILO è generalmente utilizzato nelle applicazioni di supervisione come ad esempio il watchdog timer.

5. **MHz External Oscillator** (**MHzECO**) eroga segnali di clock ad elevata frequenza e precisione spinta utilizzando un cristallo esterno, valori tipici di frequenza sono individuabili nel range 4-25 MHz.

6. **32.768 KHz External Oscillator** (**32KHzECO**) utilizza un cristallo esterno che garantisce consumi di energia davvero bassi, collegato direttamente allo sleep timer per monitorare il PSoC quando si funziona in modalità risparmio energetico.

7. Il **Digital System Interconnect (DSI)** utilizzato per collegare al sistema di distribuzione dei clock i segnali di clock provenienti da sorgenti esterne collegate agli I/O pins.

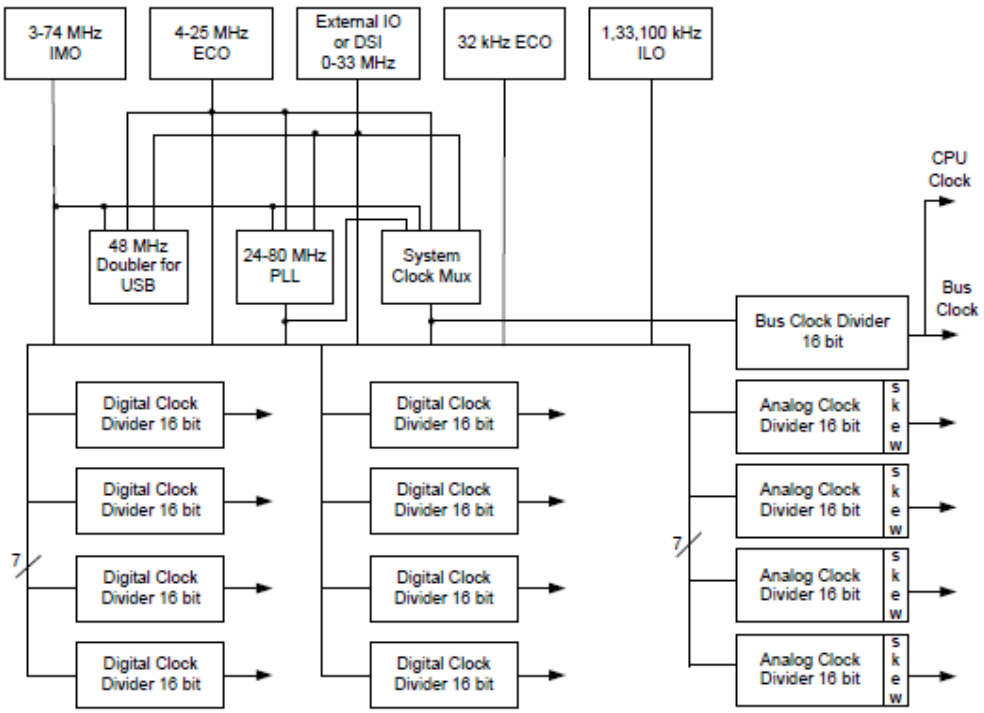


Fig. 6.8 - Sistema di distribuzione del Clock

Power System

Nei dispositivi PSoC 5LP l'alimentazione (Fig. [6.9]) del sotto-sistema digitale e del sotto-sistema analogico è separata, in particolare i loro circuiti logici sono alimentati da due regolatori a 1,8 [V], rispettivamente $\mathbf{V_{CCD}}$ e $\mathbf{V_{CCA}}$., che funzionano solo quando il microcontrollore è in modalità attiva. Nel sistema sono presenti tre ulteriori regolatori di tensione: **Sleep mode regulator**, necessario per garantire le operazioni offerte dal microprocessore quando il sistema lavora in modalità sleep, **I2C regulator** che alimenta la logica del sistema di comunicazione, ed infine

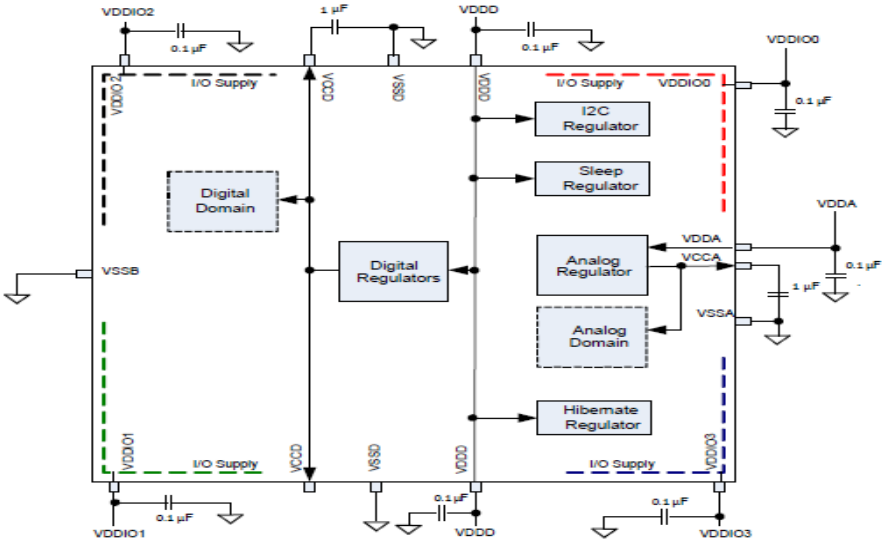


Fig. 6.9 - Sistema di alimentazione dei dispositivi PSoC 5LP.

l'hibernate regulator che garantisce la potenza necessaria a mantenere in vita il sistema quando il microcontrollore funziona in modalità hibernate.

Il sistema di distribuzione dell'energia all'interno del dispositivo PSoC 5LP è alimentato dai pin analogici $\mathbf{V_{DDA}}$, pin digitali $\mathbf{V_{DDD}}$. Ulteriori quattro pin, i $\mathbf{V_{DDIOx}}$, alimentano un numero specifico di I/O pin.

Sistema di I/O

Gli I/O pin permettono un'elevata flessibilità alla programmazione, ogni GPIO infatti può essere programmato per funzionare sia come pin digitale che come pin analogico, ma può anche cambiare il suo modo di operare dinamicamente, mentre il codice è in esecuzione chiamando le API dedicate. Un pin può funzionare come input, output o modalità bidirezionale, cioè prevede entrambi i versi.

Sono disponibili 8 drive mode per poter controllare i pin, di seguito sono elencate e descritte brevemente.

Le modalità **High Z** fa riferimento al comportamento dei circuiti logici quando non sono alimentati, il loro stato non è determinabile perché si trovano a potenziale flottante. Il rumore esterno e le cariche statiche presenti sugli oggetti circostanti alterano lo stato del componente in maniera non prevedibile. Per evitare un funzionamento non corretto del pin è necessario collegarlo ad una fonte esterna. Generalmente questa modalità è usata con i blocchi funzionali ADC.

La modalità Resistive **Pull-up/Pull-Down** è utilizzata per forzare il pin ad assumere un valore logico definito, invece di rimanere flottante, per evitare che il pin capti i segnali di disturbo e assuma uno stato casuale.

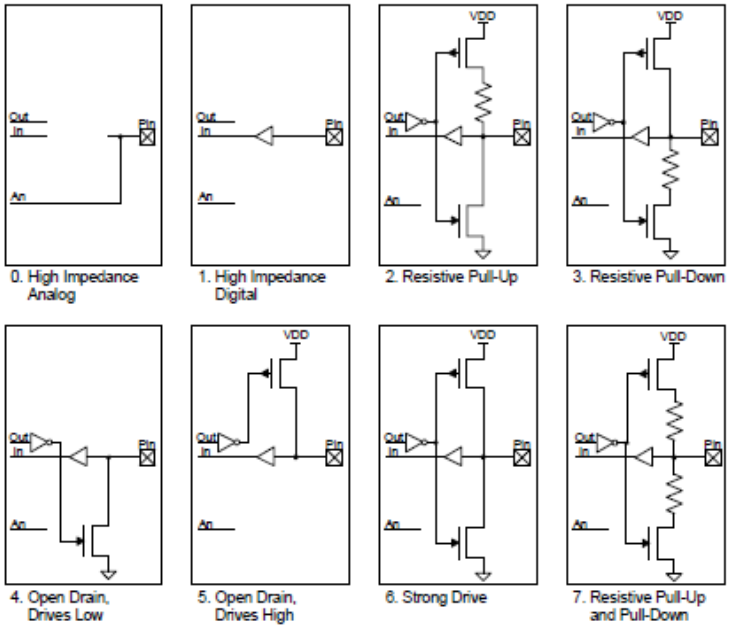


Fig. 6.10 - Modalità di pilotaggio dei GPIO

La modalità **Open Drain Drives Low** forza l'uscita a GND quando impongo un livello logico alto, altrimenti il pin si trova in uno stato di alta impedenza, risultando flottante. Generalmente si aggancia una resistenza di pull-up per definire il potenziale con certezza. La modalità **Open Drain Drives High** si differenzia perché l'uscita è forzata a V_{CC}.

La modalità **Strong Drive** permette di forzare la tensione sul pin a V_{CC} quando impongo un livello logico alto, altrimenti è forzata a GND.

La modalità **Pull-Up and Pull-Down** prevede due resistenze, una collegata a V_{CC} e l'altra a GND.

Applicazione: Connessione con ATWINC1500

ATWINC1500 è un dispositivo intuitivo da utilizzare poiché Atmel ha realizzato un ampio sistema di funzioni API, definite in 47 file divisi equamente tra file header e file sorgente, che permettono la realizzazione di progetti basati sulle più note applicazioni in ambito Internet, come ad esempio l'invio di e-mail, o trasmissione dati secondo i protocolli TCP/IP.

Di fatto ATWINC1500 può essere collegato ad un'ampia varietà di microcontrollori, prodotti da Atmel o realizzati da altri produttori, quindi, implementando i principali protocolli di comunicazione su cui si basa la rete Internet, rende smart i sistemi embedded già esistenti. ATWINC1500 quindi fornisce un'interfaccia che permette di integrare i principali standard della famiglia IEEE 802.11, cioè il "b", il "g" e l "n".

Le funzionalità di un microcontrollore sono facilmente estendibili grazie al lavoro svolto da Atmel nel realizzare la **libreria winc**: un microprocessore master che implementa le API function presenti nella libreria winc richiede al microprocessore APS3S-Cortus di ATWINC1500 Xplained Pro, la PCB in cui è integrato, di eseguire una qualsiasi operazione di STA o di AP. Il

comando transita, sotto forma di bit, attraverso un'interfaccia di comunicazione, la più usata è la SPI ma il modulo Wi-Fi supporta anche la I2C. La connessione di un qualsiasi MCU al modulo ATWINC1500 è possibile configurando opportunamente i file sorgente, che gestiscono a livello hardware la periferica SPI, ogni produttore di microcontrollori definisce differentemente come interagisce la logica programmabile dei propri MCU e come le funzioni API sono definite.

La connessione tra ATWINC1500 Xplained Pro e il microcontrollore CY8CKIT-059 PSoC 5LP Prototyping Kit, o più in generale con qualsiasi microcontrollore, è possibile eseguendo il porting di due file sorgente della libreria winc: **nm_bus_wrapper.c** e **nm_bsp_sam21.c**.

Eseguire il porting significa modificare uno o più file all'interno di una libreria esistente utilizzando le API function di un MCU, non designato ad interagire con la libreria. Entrando maggiormente nel dettaglio, eseguire il porting del codice significa: scrivere un software che abilita il MCU ad interagire con l'hardware di una periferica esterna, in questo caso il modulo ATWINC1500.

Il file **nm_bus_wrapper.c** descrive il modo in cui Atmel SAMD21 Xplained Pro gestisce il modulo SERCOM SPI per ricevere e trasmettere i dati al dispositivo Wi-Fi, quindi definisce il funzionamento dei pin che permettono la comunicazione, cioè i pin MISO, MOSI, CLK e SS.

Il file **nm_bsp_sam21.c** descrive come Atmel SAMD21 Xplained Pro gestisce l'abilitazione del modulo ATWINC1500: definisce le function che inizializzano il modulo, lo resettano, lo fanno operare in modalità sleep e lo disattivano. Questo file realizza le operazioni descritte gestendo i pin di WAKE, IRQN, RESET e CHIP_EN

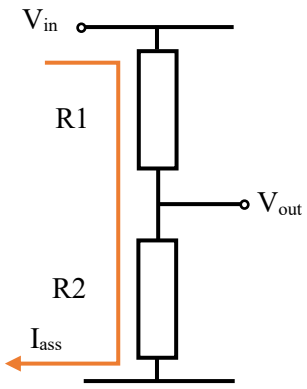
Fortunatamente non è necessario eseguire il porting di tutti gli altri file sorgente della libreria winc perché le operazioni che può svolgere il modulo Wi-Fi sono comandate a livello software dal WINC Host Driver, definito nel MCU master. Abilitando i pin di comunicazione, il Driver trasmette dei valori esadecimali, scritti in prestabiliti registri di APS3S-Cortus che permettono l'esecuzione di una determinata operazione come ad esempio la scansione dei canali della banda di frequenze 2.4 GHz.

Interfaccia Hardware

ATWINC1500 Xplained Pro è un modulo alimentato a 3,3 [V], per poter funzionare in maniera ottimale richiede che i segnali forniti in ingresso ricadano nel range 3,0 - 3,6 [V] mentre CY8CKIT-059 PSoC 5LP Prototyping Kit fornisce in uscita segnali a 5,0 [V].

Il primo problema da risolvere riguarda la realizzazione dell'interfaccia hardware necessaria ad abbassare i livelli logici dei segnali, corrispondenti ai pin MOSI, SCK, SS, CHIP_EN, RESET e WAKE, in uscita dal PSoC. Bisogna inoltre alzare i livelli logici dei segnali, corrispondenti ai pin MISO e IRQN, in uscita dal modulo ATWINC1500.

Abbassare il livello logico del segnale è possibile utilizzando un partitore di tensione (Fig. [6.11]). Scegliendo adeguatamente le resistenze i due microcontrollori possono interagire in modo opportuno, scambiandosi informazioni. La scelta delle resistenze si basa sulla definizione di due grandezze: il livello di tensione, che si vuole ottenere sul nodo intermedio, e il valore di corrette assorbita. Siccome la tensione è imposta, cioè deve essere selezionata nel range di tensioni tra i 3,0 [V] e i 3,6[V], ho un sistema ad un grado di libertà.



Generalmente il progettista sceglie il valore di corrente assorbita dal microcontrollore master più bassa possibile, in questo caso si è imposto un valore di 0,17 [mA].

$$\begin{cases} V_{out} = V_{in} * \dfrac{R_2}{R_2 + R_1} \;\; Partitore\ di\ tensione \\ V_{in} = (R_1 + R_2) * I_{ass} \;\; Legge\ di\ Kirkoff \end{cases}$$

Fig. 6.11 - Partitore di tensione

Risolvendo il sistema di equazioni imponendo la tensione a 3,3 [V], è possibile calcolare i seguenti valori di resistenza:

- R_1 = 11,763 [KΩ]
- R_2 = 17,647 [KΩ]

I valori di resistenze disponibili in laboratorio più vicine ad R_1 e R_2 sono:

- R_1 = 12,00 [KΩ]
- R_2 = 18,00 [KΩ]

Il nuovo valore di tensione calcolato è V_{out} = 3,0 [V].

Trasmettendo i segnali attraverso questa interfaccia hardware il modulo Wi-Fi non rispondeva ai comandi: ho eseguito quindi una misura con l'oscilloscopio digitale (Fig. [6.12]) del segnale MOSI sia in uscita del PSoC, rappresentato in figura dal grafico blu, che in ingresso ad ATWINC1500, rappresentato dal grafico rosa, e ho osservato un comportamento fortemente capacitivo per cui il segnale non riesce a raggiungere il valore di regime.

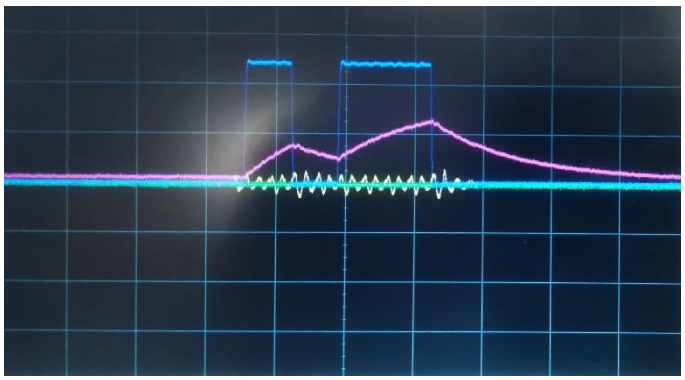


Fig. 6.12 - Trasmissione di un byte sulla linea MOSI in uscita del PSoC (grafico blu) e in ingresso di ATWINC1500 (segnale rosa).

Questa scelta di R_1 e R_2 è risultata quindi non idonea perché la costante di tempo del condensatore, in ingresso alla linea MOSI era troppo elevata. Un sensibile miglioramento è stato ottenuto

imponendo la corrente assorbita a 1,00 [mA]. Calcolando le equazioni del sistema sono stati individuati i nuovi valori delle resistenze, R_1 = 1,13 [KΩ] e R_2 = 2,87 [KΩ].

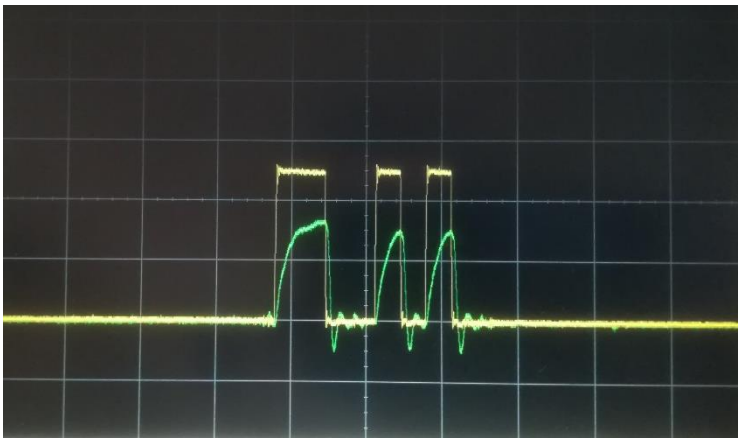


Fig. 6.13 - Trasmissione di un byte sulla linea MOSI in uscita del PSoC (grafico giallo) e in ingresso di ATWINC1500 (grafico verde).

In fig. [6.13] è stata ripetuta la misura del segnale MOSI con l'oscilloscopio digitale, possiamo notare che il comportamento capacitivo è stato notevolmente ridotto. Riducendo ulteriormente il valore delle resistenze R_1 e R_2 la costante di tempo del condensatore diminuisce e, i fronti di salita del segnale MOSI in ingresso ad ATWINC1500 risultano più ripidi.

Un ulteriore miglioramento è stato apportato inserendo, prima del partitore, un line driver SN74LS07 (hex buffer Fig. [6.14]), alimentato a 5,0 [V] da una sorgente esterna, cioè dalla Power Board MB-V2. Questa fonte di alimentazione è dotata anche di un'uscita a 3,3 [V] usata per alimentare il modulo ATWINC1500 Xplained Pro.

Nel progetto dell'interfaccia hardware è stato utilizzato un secondo line driver SN74LS07, necessario ad innalzare il livello di tensione delle linee MISO e IRNQ da 3,3 [V] a 5,0 [V].

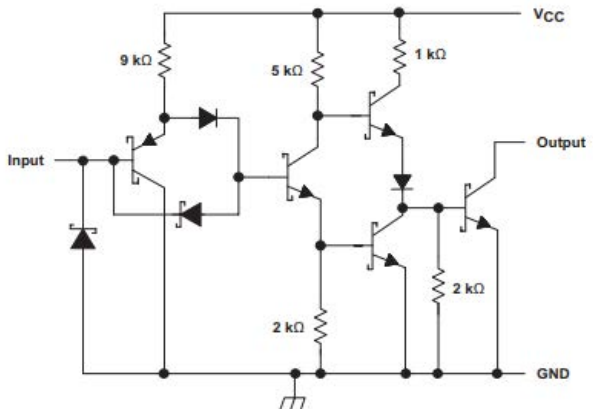


Fig. 6.14 - Schema elettronico del line driver SN74LS07

Di seguito lo schema elettrico del circuito.

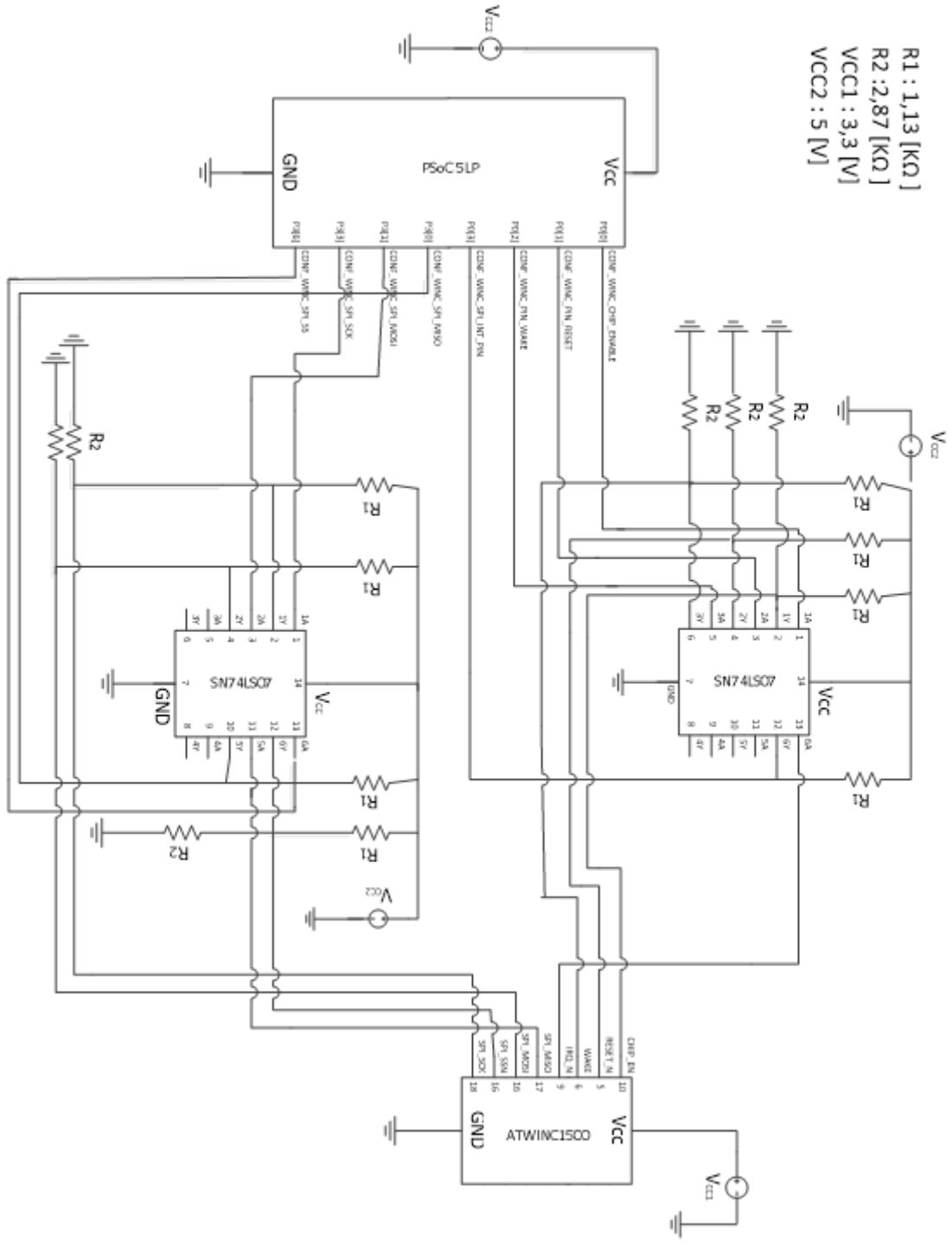


Fig. 6.15 - Schema elettrico dell'interfaccia hardware che permette la comunicazione tra ATWINC1500 e PSOC 5LP.

Ho realizzato il circuito su una Breadbord e il risultato finale è stato questo (Fig. [6.16])

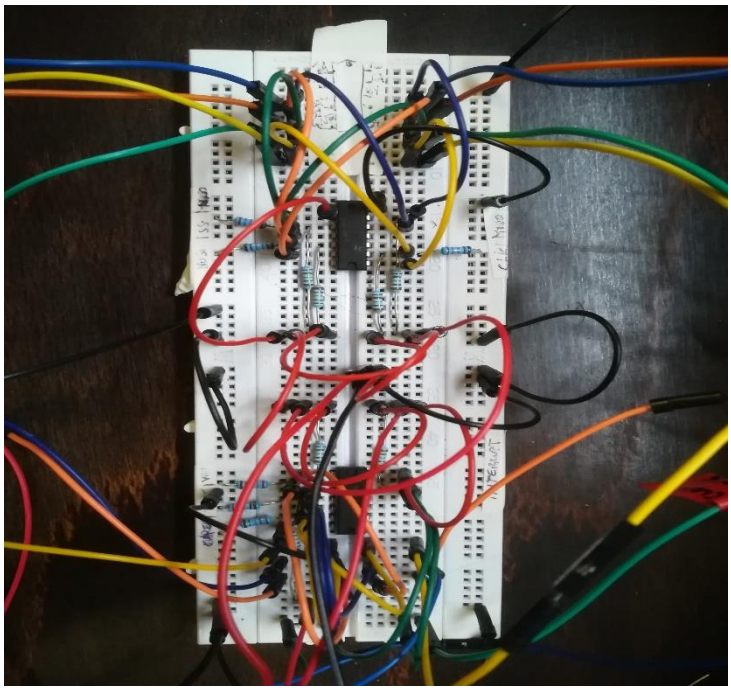


Fig. 6.16 - Interfaccia hardware realizzata su breadbord

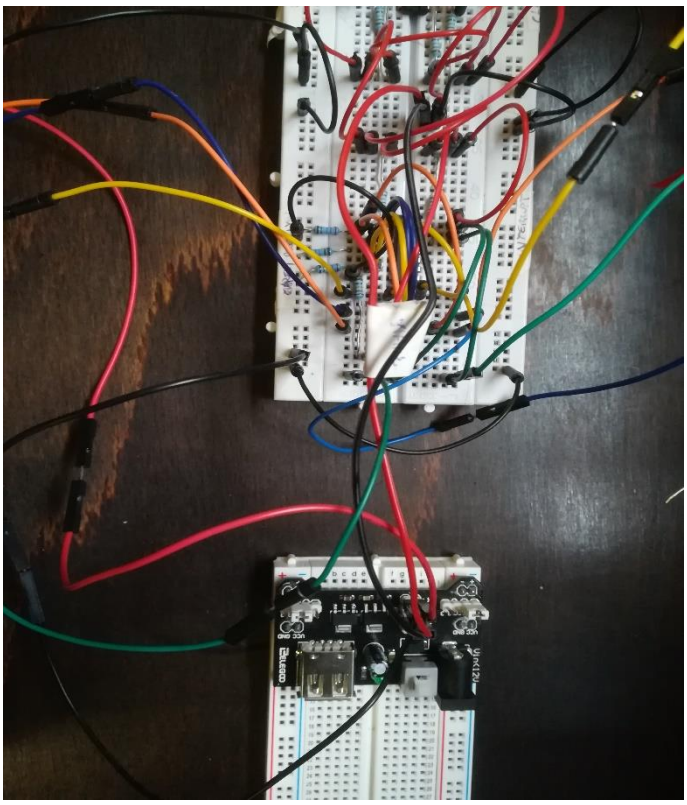


Fig. 6.17 - Interfaccia Hardware realizzata su breadbord e Power Board

Questa configurazione è risultata spesso problematica a causa dei falsi contatti tra i jumper e la Breadboard, quindi ho deciso di realizzare lo stesso circuito su una tavoletta millefori. (Fig. [6.18]) Saldando a stagno i componenti ho ottenuto questo risultato.

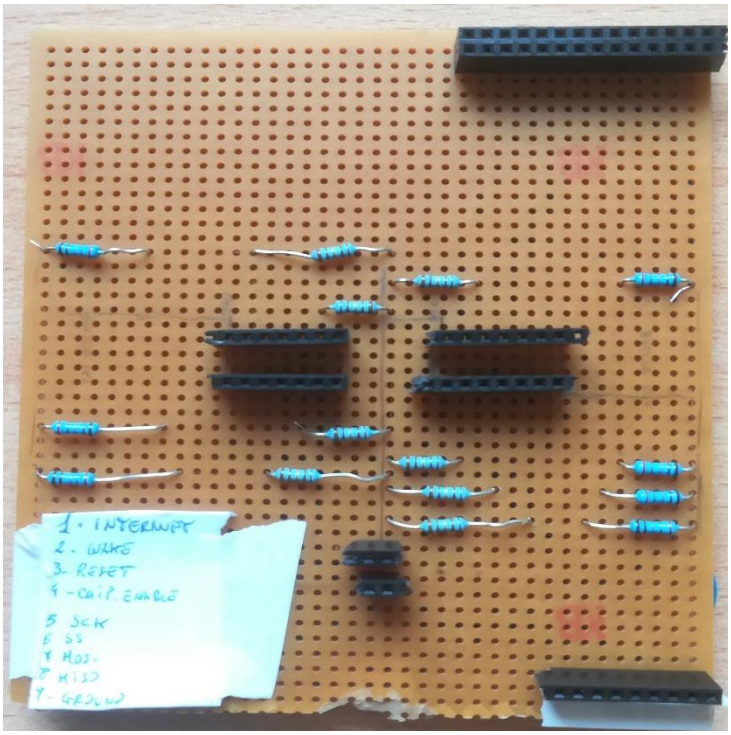


Fig. 6.18 - Interfaccia Hardware realizzata su tavoletta millefori.

Interfaccia Software

L'interfaccia software è stata realizzata eseguendo il porting dei file sorgente nm_bsp.c e nm_bus_wrapper.c, questo aspetto è approfondito nell'APPENDICE C, e configurando i registri delle periferiche utilizzate attraverso l'interfaccia grafica di PSoC Creator 4.2.(Fig. [6.19])

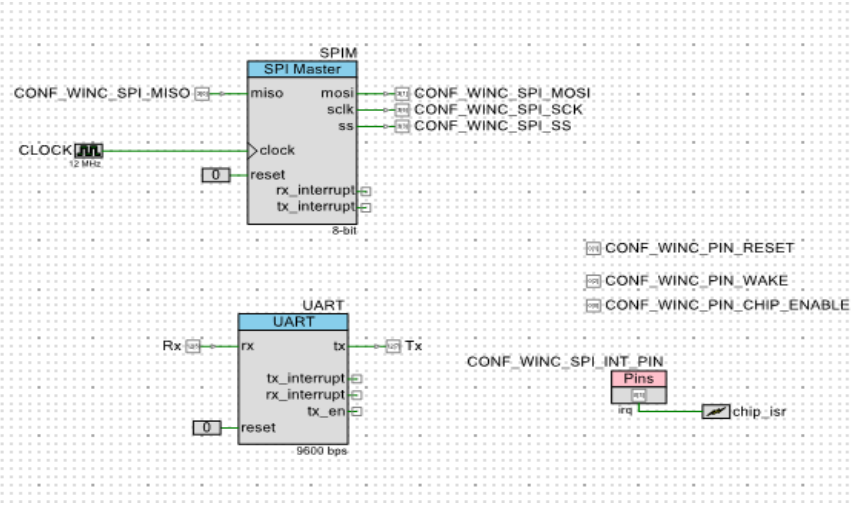


Fig. 6.19 - Componenti che abilitano le periferiche a livello software presenti nel file TopDesign.cysch

L'interfaccia grafica permette di accedere ad una libreria personalizzabile dotata di numerosi blocchi funzionali, realizzati dagli sviluppatori della Cypress, che rendono molto semplice la programmazione delle periferiche. Un sistema di *drag & drop* permette di rilasciare i blocchi funzionali nell'ambiente di lavoro principale, un file *TopDesign.cysch,* in modo da abilitare le periferiche a livello software.

Dalla libreria, selezionando il blocco SPIM è possibile configurare gli UDB adibiti alla comunicazione seriale SPI per scambiare dati tra il PSoC che assume il ruolo di master e il modulo Wi-Fi che assume il ruole di slave. Per poter configurare correttamente il blocco SPIM è necessario selezionare un componente Clock, impostandolo a 12 MHz, e quattro componenti pin, definendoli: **CONF_WINC_SPI_MOSI**, **CONF_WINC_SPI_MISO**, **CONF_WINC_SPI_SS**, **CONF_WINC_SPI_SCK**.

Selezionando il blocco SPIM (Fig.[6.20]) è possibile impostare le condizioni che permettono la corretta comunicazione tra i due microcontrollori. In questo progetto, il master deve soddisfare le condizioni pre-impostate dello slave, bisogna quindi abilitare la modalità SPI 0, trasmettere sulla linea MOSI un byte ogni transazione ed inviare i bit a partire da quello più significativo (MSB).

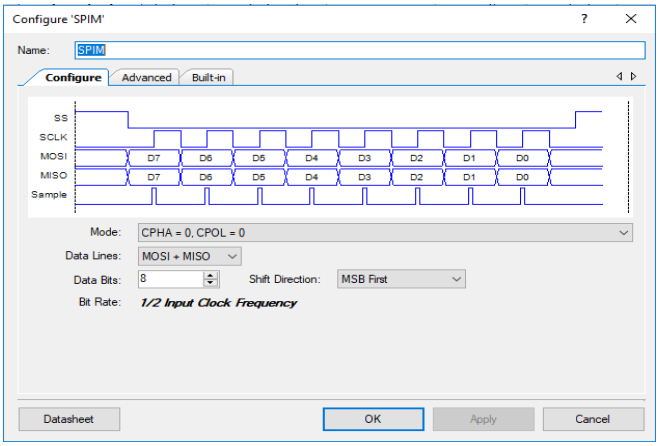


Fig. 6.20 - Impostazioni del modulo SPI

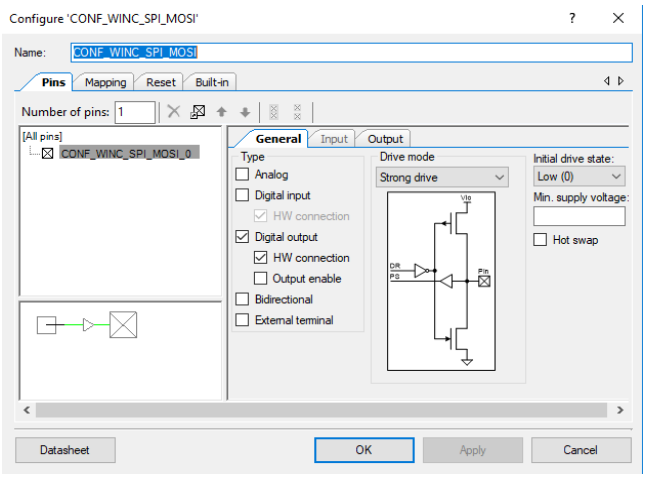


Fig. 6.21 - Impostazione dei pin MOSI, SCK, SS, RESET, WAKE,CHIP_ENABLE

I pin CONF_WINC_PIN_SCK, CONF_WINC_ PIN _MISO e CONF_WINC_ PIN _SS sono impostati come uscite digitali (*Digital Output*) con modalità di funzionamento Strong Drive. (Fig. [6.21])

Secondo la stessa modalità sono definiti anche i pin CONF_WINC_ PIN _RESET, CONF_WINC_ PIN _CHIP_ENABLE, CONF_WINC_ PIN _WAKE che gestiscono rispettivamente la linea Reset, Chip Enable e Wake.

I pin CONF_WINC_SPI_MISO e CONF_WINC_SPI_INT_PIN sono entrambi configurati come ingressi digitali ed utilizzano come modaità di funzionamento la Pull-Up.(Fig. [6.22])

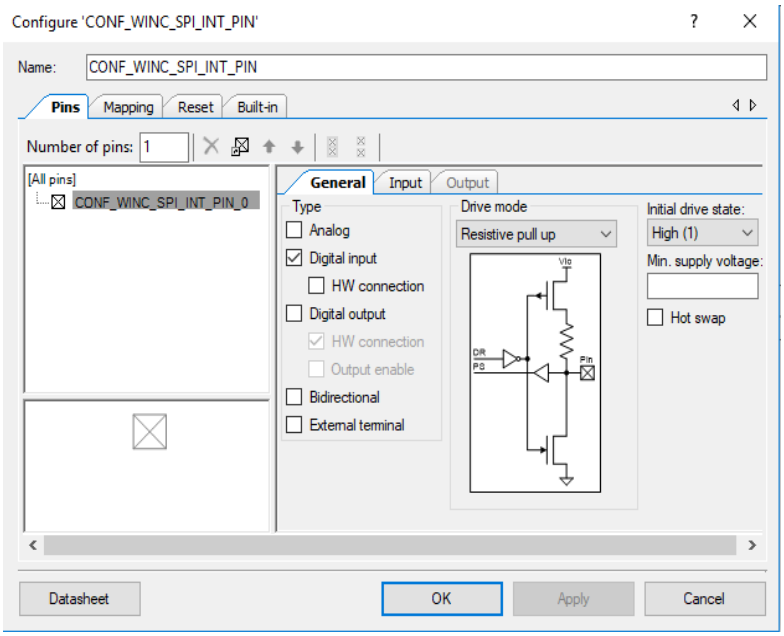


Fig. 6.22 - Impostazioni del pin che gestisce la linea INTERRUPT

Gestire gli eventi di interrupt provenienti dall'omonima linea gestita da ATWINC1500 è possibile agganciando un blocco funzionale di interrupt, definito chip_isr, al pin CONF_WINC_SPI_INT_PIN e abilitarlo sui fronti di discesa (Fig. [6.23]).

Ovviamente è necessario definire e dichiarare a livello software la routine di servizio che gestisce questi eventi.

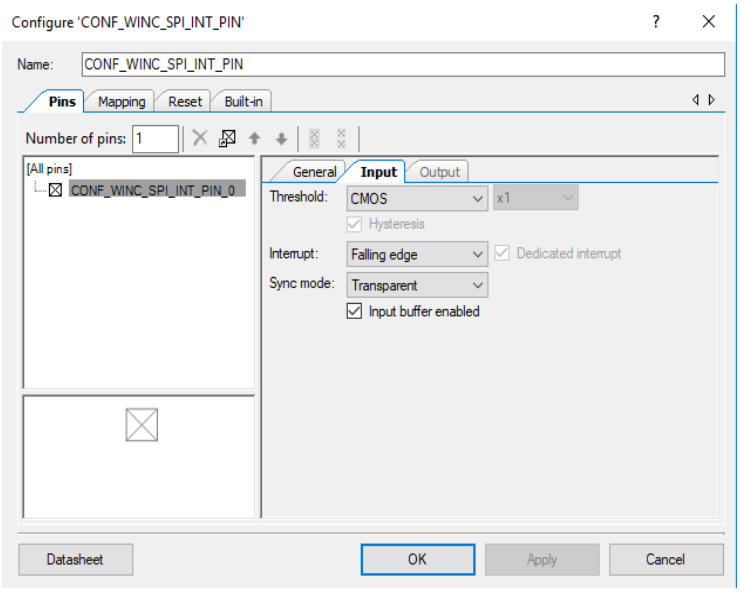


Fig. 6.23 - Impostazione dell'interrupt

In questo progetto è abilito inoltre il blocco funzionale UART. In questo modo è possibile ricevere un riscontro visivo sia in caso di errore, realizzando dunque un debug visivo, che sullo stato dei processi dell'applicazione quali ad esempio: scansione, autentificazione e associazione della STA. Le informazioni, raccolte dal modulo, sono stampate a schermo su Termite RS232 Terminal, un software che simula un terminale virtuale.

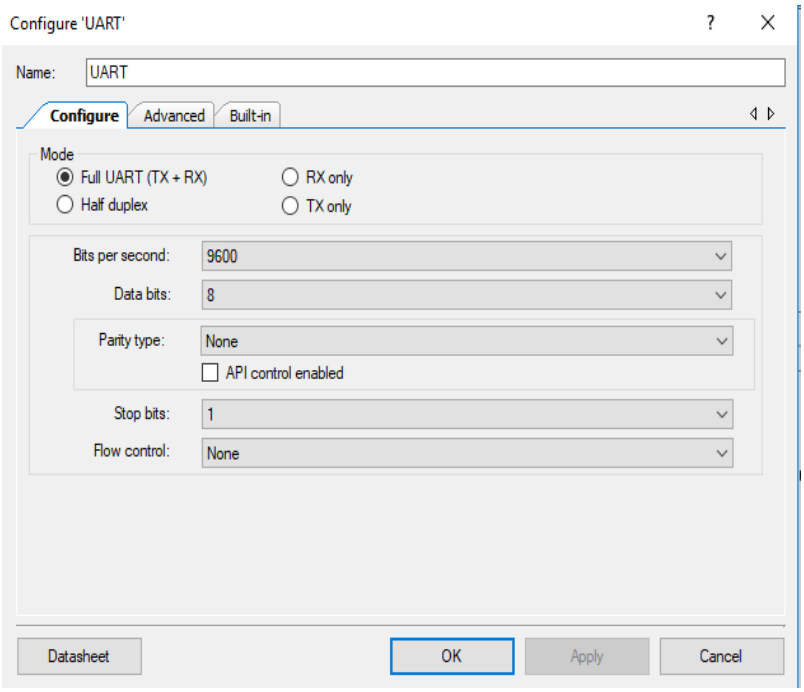


Fig. 6.24 - Impostazioni della periferica UART

Per gestire al meglio la comunicazione, sono state impostati seguenti valori: Baud-Rate 9600, nessun bit di parità, 8 bit trasmessi alla volta.(Fig. [6.24])

Il passaggio successivo è stato associare i pin alle porte. (Fig. [6.25])

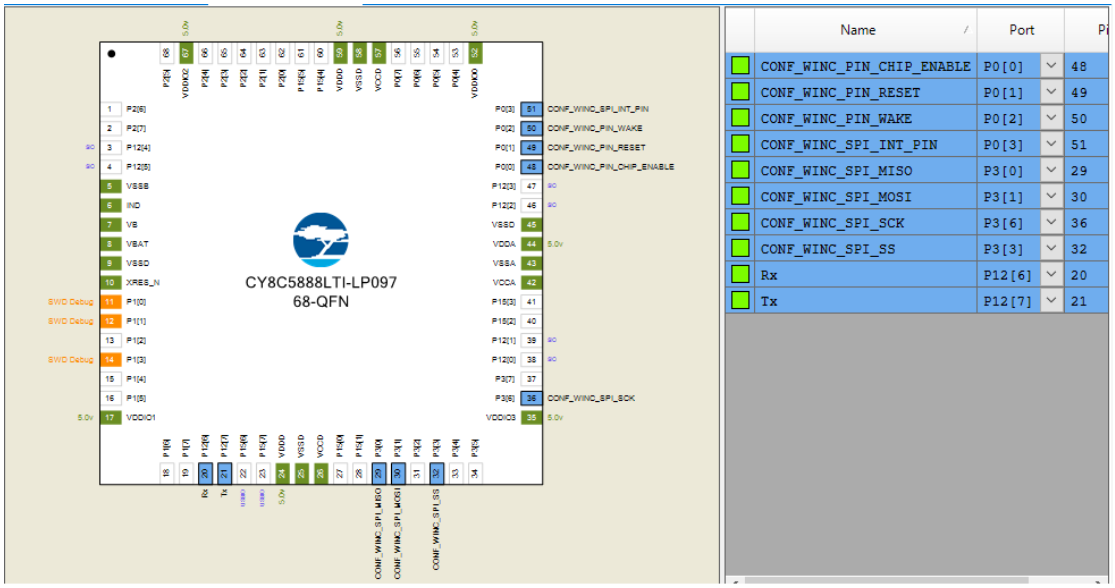


Fig. 6.25 - Associazione dei pin alle porte.

L'operazione finale è stata eseguire il *building* del progetto e successivamente scaricare il firmware nel microprocessore, quindi ho aperto il terminale virtuale e ho visualizzato sullo schermo le informazioni comunicate direttamente da ATWINC1500 al PSoC come ad esempio la versione del firmware di ATWINC1500 ed il suo chip ID. (Fig. [6.26])

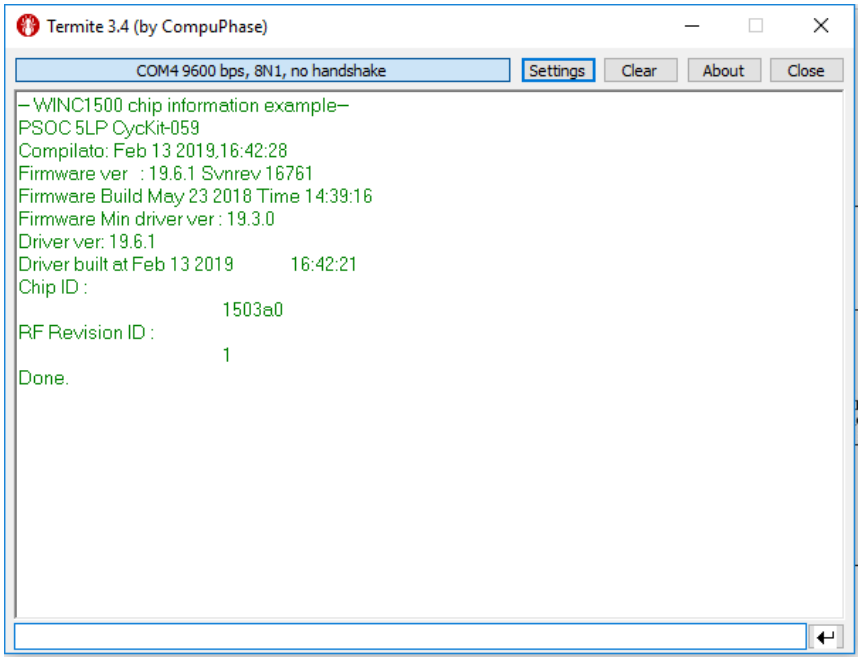


Fig. 6.26 - Risposta di ATWINC1500

Conclusione

L'obiettivo dell'attività di tesi è stato lo studio della tecnologia Wi-Fi e della sua implementazione nei dispositivi embedded, in particolare è stato indagato a livello software come la libreria winc, sviluppata per interfacciarsi con ATWINC1500, interagisce con il microcontrollore ATSAMD21 per realizzare un'applicazione di scambio informazioni basata sui protocolli Internet TCP/IP. Dopo aver studiato il codice che permette ad una STA di scansionare e accedere ad una rete wireless locale, ho utilizzato il modulo Wi-Fi per eseguire la connessione con l'AP di una WLAN. Il passo successivo è stato quello di realizzare un sistema di comunicazione wireless utilizzando due moduli ATWINC1500 e due microcontrollori ATSAMD21 Xplained pro, li ho fatti interagire comandando l'accensione e lo spegnimento di un led.

Dopo aver analizzato come il modulo Wi-Fi interagisce con il microcontrollore della Atmel ho progettato l'interfaccia hardware per realizzare la comunicazione tra ATWINC1500 e il microcontrollore CY8CKIT-059 PSoC 5LP. Per implementare nel microcontrollore della Cypress le funzionalità del protocollo IEEE 802.11 ho eseguito il porting della libreria winc, successivamente ho realizzato l'interfaccia hardware su di una tavoletta millefori per eliminare i problemi dei falsi contatti, dovuti ai numerosi jumper presenti sulla breadboard.

Attualmente, utilizzando il microcontrollore della Cypress è stato possibile interrogare ATWINC1500 e ricevere risposta in merito ad alcune informazioni proprie del modulo quali identificativo del chip e versione del firmware implementata da APS3S-Cortus, il microprocessore del modulo Wi-Fi. É necessario tuttavia indagare il motivo per cui il microprocessore del modulo Wi-Fi non riesca a comandare la linea di interrupt, e quindi generare i segnali necessari a comunicare al microcontrollore PSoC 5LP il tipo di evento Wi-Fi che dovrà gestire. Un primo approccio alla risoluzione del problema riguarda uno studio più approfondito sulla gestione dei GPIO da parte del PSoC, sembrerebbe che il modulo Wi-Fi non riesca ad imporre il livello logico voluto sulla linea. Se il problema non dovesse riguardare la gestione del pin, è necessario analizzare ed eventualmente modificare il file nm_bsp.c appartenente alla libreria winc, che gestisce l'interazione del microcontrollore con le linee di reset, wake, interrupt e chip enable.

Un test necessario da eseguire tra due microcontrollori ATSAMD21 che sperimentano la comunicazione wireless, riguarda la misurazione del data rates del modulo Wi-Fi. Questa prova è fondamentale perché ci permette di capire il tempo necessario ad inviare un frame il cui payload è composto da pochi bytes. L'esito è cruciale, infatti influenzerebbe la decisione di implementare il sistema di comunicazione wireless basato sui moduli ATWINC1500 nel progetto del caricabatteria wireless. Lo scopo del test è quello di comprendere se l'overhead del frame 802.11 comprometta negativamente l'efficienza della trasmissione quando si decide di inviare solo pochi byte di dati utili, come nel caso delle informazioni trasmesse dall'attuale sistema di comunicazione utilizzato dal caricabatteria wireless del laboratorio. Per eseguire questo test si può scrivere a livello software un ciclo for che trasmetta 100 volte un pacchetto dati e, attivare un timer che si arresti dopo aver eseguito la 100-esima iterazione. Successivamente si divide il tempo calcolato per il numero di iterazioni per calcolare il periodo medio di aggiornamento e si verifica che effettivamente sia minore della frequenza media di trasmissione dei chip attualmente usati dalla scheda di controllo del caricabatteria wireless.

Se il test dovesse avere esito positivo l'obiettivo successivo riguarda la sostituzione del sistema di comunicazione del caricabatterie wireless del laboratorio. Sarà necessario quindi adattare la libreria winc e il modulo ATWINC1500 al microcontrollore che gestisce il sistema di controllo e di condizionamento dell'energia.

ACRONIMI

ACK: Acknowledgment.

AP: Access Point.

API: Application Programming Interface

ARP: Address Resolution Protocol.

BSS: Basic Service Set.

CFP: Contention Free Period.

CSMA/CA: Carrier Sense Multiple Access / Collision Avoidance.

CTS: Clear To Send.

CW: Contention Window.

DCF: Distributed Cordination Function.

DIFS: Distributed Inter Frame Space.

DS: Distribution System.

IBSS: Independent Basic Service Set.

IP: Internet Protocol.

MAC: Medium Access Control.

MAN: Metropolitan Area Network.

MCU: Micro-Cotroller Unit.

MIMO: Multiple Input Multiple Output.

NAV: Network Allocation Vector.

OFDM: Orthogonally Frequency Division Multiplexing.

PIFS: Point Cordinator Inter Frame Space.

PLCP: Physical Layer Convergence Procedure.

PMD: Physical Medium Dependent.

RTS: Ready To Send.

SERCOM: Serial Communication Interface.

SIFS: Short Inter Frame Space.

SPI: Serial Peripheral Interface.

SSID: Service Set ID

STA: Stazione.

TCP: Transport Protocol Layer.

UART: Universal Asyncronous Receiver Transmitter.

WINC: Wireless Interface Network Controller

WLAN: Wireless Area Network.

Bibliografia

[1] IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area network - Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,2007.

[2] Meraki White Paper, “802.11n Technology” (February 2011).

[3] A Comparison between IEEE 802.11a, b, g, n and ac Standards 1Ramia Babiker Mohammed Abdelrahman, 2Amin Babiker A. Mustafa, 3Ashraf A. Osman. IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 5, Ver. III (Sep. – Oct. 2015), PP 26-29 www.iosrjournals.org

[4] A simple model of IEEE 802.11 Wireless LAN, flaitao Wu, Yoiig Peng, Keping Long, Shiduan Cheng. 2001 International Conferences on Info-Tech and Info-Net. Proceedings (Cat. No.01EX479)

[5] TCP/IP Illustrated, Volume 1 The ProtocolsSecond Edition-Kevin R. Fall W. Richard Stevens.

[6] WINC1500 Xplained Pro User's Guide.

[7] Documento tecnico Atmel SAM D21E/ D21G/ D21J SMART-ARM.

[8] Documento tecnico Atmelsamd21 modulo SERCO SPI.

[9] Documento tecnico Atmelsamd21 modulo SERCO UART.

[10] Documento tecnico Line Driver SN74LS07.

[11] Documento tecnico Power BoardMB-V2.

[12] Documento tecnico ATWINC1500-MR210PB.

[13] ATWINC1500 Wi-Fi Network Controller - Software Design Guide.

[14] CY8CKIT_059 PSoC 5LP Protoyping Kit Guide.

[15] PSoC3/5 Reference Book-Edward H. Currie and David Van Ess-March 29, 2010.

[16] PSoC® 5LP: CY8C58LP Family Datasheet.

Sitografia

[1] http://cs.unibo.it/~ghini/didattica/reti_lpr/TAC006a.pdf

[2] http://www.swprog.com/articoli/csc.php

[3] http://www.di.uniba.it/~reti/LabProRete/Interazione(TCP)Client-Server_Portabile.pdf

[4] https://web.archive.org/web/20090201053710/http://wireless-center.net/WLANs-WPANs/1436.html

[5] http://www.swprog.com/articoli/csc.php

[6] http://ww1.microchip.com/downloads/en/appnotes/atmel-42630-atwinc1500-wifi-network-controller-station-mode_applicationnote_at12264.pdf

[7] https://www.researchgate.net/profile/Daniela_Laselva/publication/240627696_The_IEEE_80211_Medium_Access_Control_MAC_S72333_PhD_Seminar_on_Radio_Communications/links/0046353380d3f28554000000/The-IEEE-80211-Medium-Access-Control-MAC-S-72333-PhD-Seminar-on-Radio-Communications.pdf

[8] http://disi.unitn.it/locigno/teaching-duties/wmvn/02_802.11_MAC_Layer.pdf

[9] http://tesi.cab.unipd.it/27068/1/Da_802.11g_a_802.11n_innovazioni_e_ottimizzazioni.pdf

[10] http://tesi.cab.unipd.it/42079/1/NasoCaterina578399.pdf

APPENDICE A - Modello ISO/OSI

Il progetto di tesi si basa sull'utilizzo dei protocolli che permettono di gestire lo scambio di informazioni all'interno di una rete locale, quindi è necessario come primo approccio descrivere la struttura di base su cui si appoggia lo stack di protocolli utilizzato. Il modello di riferimento è l'ISO/OSI, che fornisce uno schema per riconoscere e quindi confrontare i differenti protocolli. Questo modello, poiché è stato sviluppato in seguito alla nascita di alcuni standard comunicativi, non rappresenta un vincolo da seguire o un sistema a cui è necessario uniformarsi per la realizzazione di un protocollo, è considerato piuttosto una linea guida.

L'ISO/OSI è un'architettura basata su sette livelli in cui è possibile scambiare informazioni solo tra i livelli adiacenti, infatti vengono fornite delle interfacce che descrivono quali dati un livello deve fornire al successivo o al precedente, ma non viene indicata la modalità con cui farlo.

Ogni livello del modello ISO/OSI del dispositivo in trasmissione opera come se fosse direttamente connesso con il livello corrispondente appartenente al dispositivo in ricezione, in quanto ogni livello aggiunge delle parti all'informazione principale che possono essere interpretate solo dal livello omonimo ricevente

Al fine della comunicazione sono importanti ma non necessari tutti i livelli. Ad ogni livello, il frame viene arricchito con ulteriori informazioni aggiungendo nuovi campi. Il messaggio originale partendo dallo strato Applicazione può essere incapsulato in un overhead, costituito da un header e un trailer oppure gli si può aggiungere solo un campo header. L'header è il preambolo del messaggio mentre il trailer rappresenta la parte finale, agganciata alla coda.

Fig. A.1 - Schema a blocchi dei livelli del modello ISO/OSI.

Di seguito una breve spiegazione delle funzionalità aggiunte da ciascun livello del modello.

LIVELLO FISICO: Gestisce la trasmissione della sequenza di bit attraverso il mezzo di comunicazione, modula il segnale da trasmettere ed applica i livelli di tensione necessari per generare il flusso di informazione.

LIVELLO CONNESSIONE DATI: Frammenta i dati da trasmettere e li incorpora in una struttura chiamata frame ed aumenta l'affidabilità della trasmissione.

LIVELLO RETE: Gestisce l'instradamento ottimale dei messaggi.

LIVELLO TRASPORTO: Fornisce un canale di comunicazione end-to-end, chi riceve è convinto di parlare direttamente con chi trasmette l'informazione, non è conscio di ciò che succede nel mezzo. Il trasmettitore inoltre frammenta i dati inviandoli attraverso strade parallele, in modo da non congestionare la rete, chi li riceve invece avrà il compito di riassemblarli in modo adeguato.

LIVELLO SESSIONE: Inserisce un checkpoint dopo aver trasmesso un certo quantitativo di dati.

LIVELLO PRESENTAZIONE: Esegue la crittografia dei dati.

LIVELLO APPLICAZIONE: Può essere un terminale virtuale, un servizio http, l'invio di una mail o il trasferimento di un file.

APPENDICE B - Inizializzazione WINC

In questa appendice è discussa la porzione di codice che inizializza il modulo Wi-Fi.

La funzione API **nm_bsp_init( )** (Fig. [B.1]) abilita il modulo ATWINC1500 a ricevere i segnali inviati dal microcontrollore master: configura i pin CHIP_ENABLE, WAKE e RESET come uscite digitali pilotate in *Strong mode*. esegue successivamente il reset dei pin RESET e CHIP_ENABLE imponendo le uscite relative nello stato logico alto per 10 [msec] e poi nuovamente nello stato logico basso.

```
/* Initialize the BSP. */
nm_bsp_init();
```

Fig. B.1 - Funzione API che inizializza il modulo.

Il WINC Host Driver esegue l'API **m2m_wifi_init()** (Fig. [B.2]) passando come argomento la struttura del tipo *tstrWiFiParam* costituita da due puntatori a funzione, *tpfAppwificb* che gestisce le notifiche in risposta alle operazioni di Wi-Fi e *tpfAppwifiMoncb* che invece monitora e consegna i pacchetti ricevuti attraverso l'interfaccia Wi-Fi. La funzione m2m_wifi_init() svolge due compiti: abilita la comunicazione seriale SPI definendo la struttura necessaria a configurare il master e i suoi pin e, inizializza il modulo HIF. Successivamente vengono richieste al modulo Wi-Fi alcune informazioni come ad esempio la versione del driver e del firmware.

Tutti gli eventi notificati al Driver sono gestiti eseguendo in loop infinito l'API

```
/* Initialize Wi-Fi parameters structure. */
memset((uint8_t *)&param, 0, sizeof(tstrWifiInitParam));

/* Initialize Wi-Fi driver with data and status callbacks. */
param.pfAppWifiCb = wifi_cb;
ret = m2m_wifi_init(&param);
if (M2M_SUCCESS != ret) {
	printf("main: m2m_wifi_init call error!(%d)\r\n", ret);
	while (1) {
	}
}
```

Fig. B.2 - Funzione API che abilitano la periferica SPI.

m2m_wifi_handle_events() (Fig. [B.3]). Quando si verifica un interrupt esterno, proveniente da ATWINC1500, il modulo Host Interface (HIF) legge il control register per identificare l'evento. Dopo aver letto il campo dati relativo all'evento, il Driver esegue la funzione corretta. L'API function m2m_wifi_cb() gestisce le notifiche relative alla configurazione del Wi-Fi e la connessione, m2m_i_cb() invece gestisce gli eventi a livello Trasporto e Rete, infinem2m_ota_cbhandles() gestisce gli eventi di aggiornamento del firmware Over-the Air.

```
while (1) {
    /* Handle pending events from network controller. */
    while (m2m_wifi_handle_events(NULL) != M2M_SUCCESS) {
    }
}
```

Fig. B.3 - Funzione API che gestisce gli eventi di interrupt esterni generati da ATWINC1500.

APPENDICE C - Porting Librerie

nm_bsp_psoc5lp.c

```
#include "C:\Users\giori\OneDrive\Desktop\Librerie psoc\winc_library\bsp\include\nm_bsp.h"
#include "C:\Users\giori\OneDrive\Desktop\Librerie psoc\winc_library\common\include\nm_common.h"
#include "conf_winc.h"
#include "project.h"

static tpfNmBspIsr gpfIsr;

CY_ISR(chipisr)
{
    chip_isr_ClearPending();
    if (gpfIsr) {
        gpfIsr();
    }
}

/*
 * @fn     init_chip_pins
 * @brief  Initialize reset, chip enable and wake pin
 */
static void init_chip_pins(void)
{

    /*
    *   Questa configurazione del PSoC avviene tramite l'interfaccia grafica
    *   Impostare i pin:CHIP_ENABLE, WAKE, RESET come uscite digitali
    *   drive mode:Strong
    */

}

```

Fig. C.1 - Listato 1/4 del file sorgente nm_bsp_psoc5lp.c

```
/*
 * @fn     nm_bsp_init
 * @brief  Initialize BSP
 * @return 0 in case of success and -1 in case of failure
 */
sint8 nm_bsp_init(void)
{
    gpfIsr = NULL;

    /* Initialize chip IOs. */
    init_chip_pins();
    nm_bsp_reset();
    chip_isr_StartEx(chipisr);
    return M2M_SUCCESS;
}

/**
 * @fn     nm_bsp_deinit
 * @brief  De-iInitialize BSP
 * @return 0 in case of success and -1 in case of failure
 */
sint8 nm_bsp_deinit(void)
{

    CONF_WINC_PIN_CHIP_ENABLE_Write(0);
    CONF_WINC_PIN_RESET_Write(0);

    return M2M_SUCCESS;

}

```

Fig. C.2 - Listato 2/4 del file sorgente nm_bsp_psoc5lp.c

nm_bsp_psoc5lp.c

```
66  /**
67   *  @fn     nm_bsp_reset
68   *  @brief  Reset NMC1500 SoC by setting CHIP_EN and RESET_N signals low,
69   *          CHIP_EN high then RESET_N high
70   */
71  void nm_bsp_reset(void)
72  {
73      CONF_WINC_PIN_CHIP_ENABLE_Write(0);
74      CONF_WINC_PIN_RESET_Write(0);
75      nm_bsp_sleep(100);
76      CONF_WINC_PIN_CHIP_ENABLE_Write(1);
77      nm_bsp_sleep(10);
78      CONF_WINC_PIN_RESET_Write(1);
79      nm_bsp_sleep(10);
80
81  }
82
83  /*
84   *  @fn     nm_bsp_sleep
85   *  @brief  Sleep in units of mSec
86   *  @param[IN]  u32TimeMsec
87   *              Time in milliseconds
88   */
89  void nm_bsp_sleep(uint32 u32TimeMsec)
90  {
91
92      while (u32TimeMsec--) {
93          CyDelay(1);
94      }
95
96  }
97
```

Fig. C.3 - Listato 3/4 del file sorgente nm_bsp_psoc5lp.c

```
98  /*
99   *  @fn     nm_bsp_register_isr
100  *  @brief  Register interrupt service routine
101  *  @param[IN]  pfIsr
102  *              Pointer to ISR handler
103  */
104 void nm_bsp_register_isr(tpfNmBspIsr pfIsr)
105 {
106 /*Impostazione dell'interrupt da interfaccia grafica:
107 collegarlo al pin chip_irq (pull-up) e impostare initial state high ,
108 abilitarlo sui fronti di discesa*/
109
110  gpfIsr = pfIsr;
111
112 }
113
114 /*
115  *  @fn     nm_bsp_interrupt_ctrl
116  *  @brief  Enable/Disable interrupts
117  *  @param[IN]  u8Enable
118  *              '0' disable interrupts. '1' enable interrupts
119  */
120 void nm_bsp_interrupt_ctrl(uint8 u8Enable)
121 {
122     if(u8Enable) {
123
124         chip_isr_Enable();
125     }
126
127     else{
128
129         chip_isr_Disable();
130
131     }
132 }
```

Fig. C.4 - Listato 4/4 del file sorgente nm_bsp_psoc5lp.c

nm_bus_wrapper_psoc5lp.c

```
#include <stdio.h>
#include "..\..\..\Desktop\Librerie psoc\winc_library\bsp\include\nm_bsp.h"
#include "..\..\..\Desktop\Librerie psoc\winc_library\common\include\nm_common.h"
#include "..\..\..\Desktop\Librerie psoc\winc_library\bus_wrapper\include\nm_bus_wrapper.h"
#include "conf_winc.h"
#include <stdbool.h>
#include "project.h"

#define NM_BUS_MAX_TRX_SZ   256


sint8 spi_rw(uint8* pu8Mosi, uint8* pu8Miso, uint16 u16Sz);

tstrNmBusCapabilities egstrNmBusCapabilities =
{
    NM_BUS_MAX_TRX_SZ
};

#ifdef CONF_WINC_USE_SPI

static inline sint8 spi_rw_pio(uint8* pu8Mosi, uint8* pu8Miso, uint16 u16Sz)
{
    uint16_t rxd_data = 0;
    uint8 u8Dummy = 0;
    uint8 u8SkipMosi = 0, u8SkipMiso = 0;
    uint16_t txd_data = 0;

```

Fig. C.5 - Listato 1/6 del file sorgente nm_bus_wrapper_psoc5lp.c

```

    if(((pu8Miso == NULL) && (pu8Mosi == NULL)) ||(u16Sz == 0)) {
        return M2M_ERR_INVALID_ARG;
    }

    if (!pu8Mosi) {
        pu8Mosi = &u8Dummy;
        u8SkipMosi = 1;
    }
    else if(!pu8Miso) {
        pu8Miso = &u8Dummy;
        u8SkipMiso = 1;
    }
    else {
        return M2M_ERR_BUS_FAIL;
    }

```

Fig. C.6 - Listato 2/6 del file sorgente nm_bus_wrapper_psoc5lp.c

nm_bus_wrapper_psoc5lp.c

```
47      while (u16Sz) {
48
49          txd_data = *pu8Mosi;
50
51          while(!(SPIM_STS_TX_FIFO_EMPTY & SPIM_ReadTxStatus()) )
52              ;
53
54          SPIM_WriteTxData(txd_data);
55
56          while(CONF_WINC_SPI_SS_Read()==1)
57          ;
58          rxd_data=SPIM_ReadRxData();
59
60              if(SPIM_ReadRxStatus() & SPIM_STS_RX_FIFO_OVERRUN){
61
62              SPIM_ClearRxBuffer();}
63
64          *pu8Miso = rxd_data;
65
66          u16Sz--;
67          if (!u8SkipMiso)
68              pu8Miso++;
69          if (!u8SkipMosi)
70              pu8Mosi++;
71      }
72
73      while(!(SPIM_STS_SPI_DONE & SPIM_ReadTxStatus()))
74          ;
75
76      return M2M_SUCCESS;
77  }
78
```

Fig. C.7 - Listato 3/6 del file sorgente nm_bus_wrapper_psoc5lp.c

```
78
79  sint8 spi_rw(uint8* pu8Mosi, uint8* pu8Miso, uint16 u16Sz)
80  {
81
82      {
83          return spi_rw_pio(pu8Mosi, pu8Miso, u16Sz);
84      }
85  }
86
87  /*
88  *   @fn     nm_bus_init
89  *   @brief  Initialize the bus wrapper
90  *   @return M2M_SUCCESS in case of success and M2M_ERR_BUS_FAIL in case of failure
91  */
92  sint8 nm_bus_init(void *pvinit)
93  {
94      sint8 result = M2M_SUCCESS;
95      nm_bsp_reset();
96      nm_bsp_sleep(1);
97
98      return result;
99  }
100
```

Fig. C.8 - Listato 4/6 del file sorgente nm_bus_wrapper_psoc5lp.c

nm_bus_wrapper_psoc5lp

```c
/*
*   @fn     nm_bus_ioctl
*   @brief  send/receive from the bus
*   @param[IN]  u8Cmd
*                   IOCTL command for the operation
*   @param[IN]  pvParameter
*                   Arbitrary parameter depending on IOCTL
*   @return M2M_SUCCESS in case of success and M2M_ERR_BUS_FAIL in case of failure
*   @note   For SPI only, it's important to be able to send/receive at the same time
*/
sint8 nm_bus_ioctl(uint8 u8Cmd, void* pvParameter)
{
    sint8 s8Ret = 0;
    switch(u8Cmd)
    {
        case NM_BUS_IOCTL_RW: {
            tstrNmSpiRw *pstrParam = (tstrNmSpiRw *)pvParameter;
            s8Ret = spi_rw(pstrParam->pu8InBuf, pstrParam->pu8OutBuf, pstrParam->u16Sz);
        }
        break;
        default:
            s8Ret = -1;
            UART_PutString("invalid ioclt cmd\n"); //errore
            break;
    }

    return s8Ret;
}
```

Fig. C.9 - Listato 5/6 del file sorgente nm_bus_wrapper_psoc5lp.c

```c
/*
*   @fn     nm_bus_deinit
*   @brief  De-initialize the bus wrapper
*/
sint8 nm_bus_deinit(void)
{
    sint8 result = M2M_SUCCESS;

    nm_bsp_deinit();

    return result;
}

/*
*   @fn         nm_bus_reinit
*   @brief      re-initialize the bus wrapper
*   @param [in] void *config
*                   re-init configuration data
*   @return     M2M_SUCCESS in case of success and M2M_ERR_BUS_FAIL in case of failure
*/
sint8 nm_bus_reinit(void *config)
{
    return M2M_SUCCESS;
}

```

Fig. C.10 - Listato 6/6 del file sorgente nm_bus_wrapper_psoc5lp.c