Università degli Studi di Ferrara
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Progettazione e Realizzazione di un Sistema di Test per Dispositivi di Identificazione a RadioFrequenza

Relatore:
Dott. MIRCO ANDREOTTI

Correlatore:
Dott.Ing. ANGELO COTTA RAMUSINO

Tutor:
Ing. PAOLO SOAVI

Laureando
DANIELE MANDRIOLI

Anno Accademico 2006-2007

Indice

Abstract

This thesis describes the design and development of a test system for Radio Frequency IDentification devices (RFID devices). The current project has been carried out by using a Field Programmable Gate Array (FPGA) whose function has been developed in a VHSIC Hardware Description Language based environment (Very High Speed Integrated Circuit Hardware Description Language, VHSIC-HDL). My task was to create the FPGA programming code written in VHDL and it has been executed in the research and development laboratories of Digitek a leading company in electronics (Infomobile Research and Development Department, Digitek, Concordia s. Secchia, MO).

Introduzione

La presente tesi tratta la progettazione e la realizzazione di un sistema di test per dispositivi di Identificazione a RadioFrequenza (*Radio Frequency IDentification*, RFID) impiegando un FPGA (*Field Programmable Gate Array*) la cui funzione è stata progettata usando un ambiente di sviluppo basato su VHSIC Hardware Description Language (*Very High Speed Integrated Circuit Hardware Description Language*, VHSIC-HDL). Il lavoro è stato quello di realizzare il codice di programmazione della FPGA attraverso l'uso del linguaggio VHDL, ed è stato svolto nei laboratori Ricerca e Sviluppo Infomobile presso l'azienda Digitek di Concordia s. Secchia (MO).

Digitek è una società leader nel settore elettronico con un'altissima specializzazione nei campi di acquisizione e trasmissione dati, ed sistemi di controllo e visualizzazione.

L'azienda sviluppa e realizza, strumenti di bordo, sistemi di controllo cambio, cruscotti e sistemi di *infotainment*, tester di diagnosi, controllo elettronico alimentazione GPL/Metano, telediagnosi, sistemi di pagamento pedaggi e controllo accessi, video sorveglianza.

I sistemi RFID permettono l'acquisizione di dati e l'identificazione di qualsiasi oggetto, persona o animale a cui siano applicati. I sistemi RFID funzionano in assenza di contatto tra lettore e ricevitore. Questa loro caratteristica permette il miglioramento dell'efficienza nella raccolta dei dati, superando i limiti imposti dai sistemi di identificazione attualmente utilizzati (codici a barre, bande magnetiche e smart card), dovuti agli ambienti industriali ostili, bui, sporchi, con umidità e temperature elevate. Inoltre l'assenza di contatto fra lettore e ricevitore ne permette l'utilizzo nel caso in cui non può esserci contatto diretto con l'oggetto da identificare. I sistemi di Identificazione a RadioFrequenza sono costituiti da due unità distinte: un lettore e un ricevitore chiamato comunemente transponder o tag. Il lettore interroga il tag con le opportune query, e il tag per ogni messag-

gio ricevuto correttamente, risponde con un altro messaggio che verrà inviato al lettore, che a sua volta lo rimanda al sistema informatico di controllo.

Il sistema di test sviluppato in questa tesi, si colloca nel lettore e si occupa di verificare il corretto funzionamento del tag, controllando che i messaggi di ritorno siano corretti e del tipo atteso. Tramite un interfaccia visuale è possibile vedere le percentuali di messaggi corretti ricevuti, e stabilire se il tag funziona correttamente e verificare l'area di copertura dell'antenna del lettore. Se la zona dove ci si aspetta di rilevare il tag non è correttamente illuminata dall'antenna, si potrebbe verificare il non funzionamento del sistema. La verifica dell'area di copertura, consente di individuare il posizionamento corretto dell'antenna. Il modulo di test consiste in una scheda elettronica collegata al lettore, su cui è montata la FPGA e i necessari componenti di servizio (alimentazioni, temporizzazioni, porte di I/O).

I dispositivi FPGA sono dispositivi digitali la cui funzionalità è programmabile via software. Si tratta infatti di dispositivi standard la cui funzionalità è definita dall'utente sul campo e non impostata dal produttore che può quindi realizzarli su larga scala a basso prezzo. La loro genericità e le loro prestazioni, li rende adatti a un gran numero di applicazioni come elettronica di consumo, comunicazioni, automotive, aerospaziale e biomedicale. Questi dispositivi consentono la realizzazione di prototipi in tempi brevi. Il VHDL è il linguaggio di alto livello, scelto per realizzare il codice di programmazione della FPGA. VHDL è stato scelto perché adatto a modellare sistemi digitali sia per scopi di simulazione che per scopi di sintesi su dispositivi reali. Il maggior pregio di questo linguaggio è la possibilità di descrivere un sistema digitale senza vincolarsi a dispositivi particolari. L'utilizzo di FPGA e VHDL consentono, di ridurre il *time to market* ovvero il tempo necessario affinchè un prodotto arrivi sul mercato.

La presente tesi si sviluppa nei seguenti capitoli:

Nel 1° capitolo sono descritte le caratteristiche dei linguaggi di descrizione di hardware, spiegando le motivazioni che hanno portato al loro sviluppo. Sono analizzati inoltre gli stadi del flusso di progetto tipico della programmazione VHDL.

Nel 2° capitolo è descritto il funzionamento dei dispositivi FPGA.

Nel 3° capitolo è presentata una panoramica sui sistemi RFID, descrivendo le loro caratteristiche e il principio di funzionamento.

Nel 4° capitolo è descritto il funzionamento del sistema di test ed è analizzata

l'architettura di rete del sistema RFID utilizzato.

Nel 5° capitolo è affrontata tutta la descrizione del sistema digitale implementato in VHDL, con la descrizione dei singoli blocchi e delle macchine a stati che li costituiscono.

Nel 6° capitolo sono descritti l'ambiente di sviluppo e i software che sono stati utilizzati in tutte le fasi del presente progetto. Riportando inoltre la descrizione della scheda prototipale e una breve analisi dei test di funzionamento eseguiti sul sistema finale.

Nel 7° ed ultimo capitolo riporteremo le conclusioni con la descrizione dei risultati ottenuti e le future espansioni previste.

Capitolo 1

Il linguaggio VHDL

In questa capitolo viene presentato il linguaggio VHDL per spiegare come da una descrizione di un circuito si possa ottenere un file binario che caricato su un dispositivo specifico permetterà l'implementazione della rete logica desiderata. Lo scopo non è quello di spiegare la sintassi e tutte le funzionalità del linguaggio ma quello di dare una base teorica dello strumento che è alla base della moderna progettazione elettronica, in campo digitale.

1.1 Come nascono i linguaggi HDL

Negli ultimi 40 anni l'evoluzione delle tecnologie VLSI[1] hanno permesso di integrare, un numero sempre più elevato di transistor all'interno di un singolo chip, portando ad un grande aumento della complessità dei circuiti. La progettazione e la descrizione di circuiti, con un numero enorme di transistor non era più possibile usando i disegni schematici, cioè creando il sistema come una rete di porte logiche (*gate level*).

La necessità di strumenti, con un più alto livello di astrazione rispetto al *gate level*, portò allo svilupparsi di una serie di meta-linguaggi il cui scopo è fornire una descrizione del progetto che si intende realizzare. Molti di questi linguaggi si svilupparono, in modo indipendente nelle università, per scopi didattici oppure all'interno di grandi aziende (ad esempio IBM) per gestire i nuovi grandi progetti. Nascono così i linguaggi di descrizione hardware, identificati con il termine generico HDL: Hardware Description Language. Il loro scopo è di fornire una descrizione non ambigua di un circuito elettronico, che possa essere interpretato

[1] Very Large Scale Integration

univocamente da tutti i vari progettisti. È anche possibile disegnare diagrammi di macchine a stati, utilizzando direttamente il formalismo grafico, che verranno tradotti in una forma HDL. Il primo HDL è stato PALASM (*PAL ASseMbler*) che serviva a specificare reti logiche per dispositivi programmabili tipo PAL[2]. Un linguaggio simile è ABEL (*Advanced Boolean Equation Language*). Questi linguaggi permettono di descrivere il sistema attraverso equazioni logiche, tabelle della verità e diagrammi a stati in forma testuale.

Le diversità tra i vari linguaggi che si svilupparono, portò alla necessità di disporre di uno standard per la progettazione, la sintesi, la verifica e la documentazione dei sistemi digitali.

1.2 Come nasce VHDL

Il VHDL (*VHSIC Hardware Description Language*) nasce agli inizi degli anni 80, dalla richiesta del dipartimento della difesa americano(*Departiment Of Defese*, DOD) di un linguaggio HDL per il progetto VHSIC (*Very High Speed Integrated Circuits*)[1].

Il VHDL è stato pensato per risolvere, i molteplici problemi che si presentano nel progetto, nello scambio di informazioni e nella documentazione di hardware digitale. Infatti il dipartimento della difesa americano richiese per le forniture di circuiti integrati, migliaia di pagine di documentazione a partire dalla commessa, fino al collaudo finale del componente. Quando un componente deve poi essere sostituito sono richiesti grandi sforzi per ricostruirne il presunto comportamento.

Nasce così il progetto VHSIC dove vengono stabiliti le caratteristiche e gli obbiettivi del VHDL. Il VHDL è un linguaggio di descrizione hardware di alto livello adatto a modellare sistemi digitali sia per scopi di simulazione che per scopi di sintesi su dispositivi reali. Il concetto su cui si fonda tale linguaggio è che deve servire per la descrizione del sistema dal progetto iniziale fino alla porta logica. La descrizione deve includere il comportamento, le tempistiche e le caratteristiche strutturali dei dispositivi digitali. Il VHDL è nato quindi, come linguaggio per la documentazione dei sistemi digitali, poi solo in un secondo momento sono stati introdotti i programmi di sintesi. Per questo non tutti i costrutti del VHDL sono sintetizzabili ad esempio, i tipi di dato float, le stringhe

[2]Programmable Array Logic

e le operazioni relative all'accesso su files, non hanno una diretta corrispondenza in hardware anche perché non avrebbero significato.

Nonostante la presenza di numerosi linguaggi di descrizione di hardware, nessuno di essi era mai stato accettato come standard dall'industria prima dell'avvento del VHDL. Infatti, i linguaggi precedenti erano strettamente legati alla tecnologia usata o alle loro case produttrici.

Il maggior pregio del VHDL è la possibilità di descrivere un sistema digitale a prescindere dalla tecnologia con cui esso è realizzato (indipendanza tecnologica).

1.3 Perché si usa VHDL

Il VHDL rispetto agli altri linguaggi è uno standard pubblico, è stato infatti riconosciuto come standard nel 1987 (IEEE-1076-1987) ed è stato successivamente aggiornato nel 1993 (IEEE-1076-1993), ulteriori informazioni sui vari aggiornamenti sono disponibili in appendice A.

Sotto la spinta del governo statunitense che richiese esplicitamente le descrizioni VHDL di ogni ASIC[3] acquistato, anche le industrie private richiedono a loro volta le descrizioni dei circuiti acquistati dai loro fornitori. Queste motivazioni fanno si che la scelta del VHDL sia ormai obbligata per garantire la diffusione e comprensione del proprio lavoro.

Il VHDL può supportare differenti metodologie di progetto (top-down oppure bottom-up) ed è indipendente dalla tecnologia e dai processi di realizzazione. Per questo VHDL viene utilizzato da industrie che operano in ambiti differenti e con diversi problemi progettuali.

Una descrizione del sistema può essere sviluppata ad un livello astratto e successivamente sintetizzata a livello gate, a seconda del processo tecnologico scelto. Il VHDL permette rappresentazioni circuitali a vari livelli di complessità, da descrizioni puramente comportamentali, fino a descrizioni al livello di porte logiche. Qualunque sia il livello di astrazione scelto, il VHDL consente di simularne il comportamento, con la possibilità di simulazioni di dispositivi che presentano nei vari sottosistemi delle descrizioni, ad alto livello ed in altri a livello gate. Ciò consente di sostituire o modificare la descrizione di un modello

[3] Application Specific Integrated Circuit, cioè un circuito integrato creato appositamente per risolvere un'applicazione di calcolo specifica

intervenendo su un singolo blocco.

Decomporre i progetti in sottosistemi è quindi importante quanto la capacità di descrizioni dettagliate ed accurate. La possibilità di creare packages e librerie, è insita nel linguaggio facilitando così la suddivisione del lavoro e la possibilità di sperimentare nuove soluzioni all'interno di un progetto già sviluppato. Questi elementi del linguaggio permettono il riutilizzo del codice in vari progetti. Il ciclo di sviluppo di un dispositivo diviene simile a quello di un programma software.

Infine, poichè il VHDL è uno standard, tutti i circuiti descritti in questo linguaggio possono essere simulati su un qualunque simulatore VHDL. Ciò significa che un modello sviluppato da una azienda funzionerà anche presso altre aziende che utilizzano differenti programmi di simulazione. Nel caso però si utilizzino librerie ottimizzate per una particolare architettura, non sarà possibile portarne il codice su altri dispositivi. I sottosistemi che realizzano un circuito possono così svilupparsi indipendentemente, essere riutilizzati, ed eventualmente essere acquistati presso terzi. Si riducono così gli sforzi necessari ad integrare le varie parti di un progetto.

1.4 Il flusso di progettazione

La progettazione di un sistema microelettronico utilizza solitamente un approccio di tipo top-down il quale parte dal livello di astrazione più alto e va verso il più basso, suddividendo il progetto iniziale in parti sempre più elementari. Questa non una regola fissa ma dipende dal tipo di applicazione che si vuole realizzare.

Infatti, potrebbe essere necessario partire dalla progettazione di una cella elementare o se si è fortunati saltare alcuni livelli del normale uso di progettazione perché sono magari disponibili componenti già realizzate.

Dal punto di vista operativo questi livelli si traducono in una serie di passi attraverso i quali il progetto evolve e anche se questi non vengono direttamente realizzati, è bene conoscerne il loro funzionamento. Lo sviluppo di un sistema parte dalla sua specifica iniziale con l'obiettivo di realizzare fisicamente un dispositivo.

Le attività coinvolte nel progetto dipendono dal tipo di dispositivo, ma si pos-

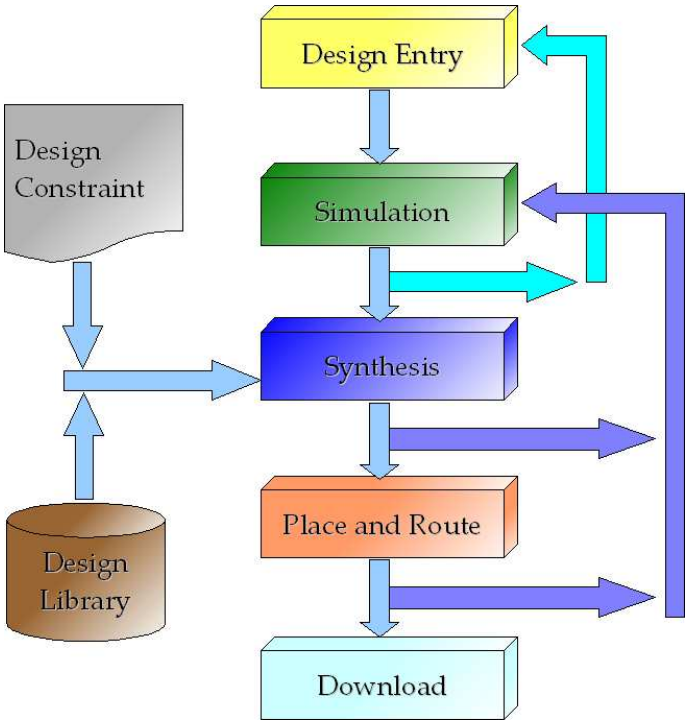


Figura 1.1: Il flusso di progettazione

sono individuare delle fasi fondamentali tipiche di tutti i progetti, visibili anche in Fig.1.1:

1. **Modellazione:** si disegna il sistema costruendo un modello software, cioè una sua rappresentazione fatta con opportuni strumenti grafici o linguaggi testuali. Inizialmente il modello può anche essere molto astratto, cioè privo di dettagli sull'implementazione.

2. **Simulazione:** il simulatore esegue il modello, cioè ne simula il comportamento affinchè si possa verificare il corretto funzionamento. Se l'esito è negativo si torna a correggere il modello nella fase precedente. Se la modellazione non è orientata alla realizzazione di un dispositivo, ma solo alla documentazione e/o simulazione, il progetto può terminare qui.

3. **Sintesi:** per implementare fisicamente il sistema, è necessario descriverne la struttura. La sintesi traduce il modello in una netlist, cioè la rete logica realizzata con blocchi logici elementari (porte e registri) corrispondente al modello. In questa fase il modello, benchè simulato correttamente, potrebbe

rivelarsi inadeguato, perché alcuni costrutti del linguaggio potrebbero non essere orientati alla sintesi o non essere supportati dal sintetizzatore.

4. **Implementazione:** consiste nella fase di *place and route*, dove la netlist viene ulteriormente elaborata, per produrre un formato dipendente dal dispositivo utilizzato. Si possono calcolare esattamente i ritardi di propagazione delle linee di interconnessione, per inserirli nel modello ed eseguire una simulazione più completa. Per un ASIC non programmabile, i files risultanti dall'implementazione servono alla *silicon foundry*[4] che dovrà produrre fisicamente il dispositivo, mentre per un dispositivo programmabile, tali files vengono caricati sulla logica programmabile scelta.

1.4.1 I livelli di modellazione

Un progettista riesce ad essere efficiente utilizzando un linguaggio descrittivo che è il più possibile vicino alla propria visione del sistema da modellare. Il modello software sviluppato necessita per poter essere interpretato correttamente a livello hardware di una ulteriore elaborazione. La soluzione consiste nell'utilizzo di specifici traduttori che trasformano il modello di alto livello, in uno di livello inferiore. L'utilizzo dei traduttori consente ai progettisti di concentrarsi sullo sviluppo di un modello di alto livello, senza preoccuparsi della piattaforma hardware su cui verrà effettivamente implementato il progetto. Nei sistemi hardware si è soliti parlare di sintetizzatore analogo al compilatore in un ambiente di sviluppo per software.

Il VHDL supporta tre livelli di astrazione:

1. **Descrizione Funzionale o Comportamentale**[5]**:** è al livello più alto e definisce cosa il sistema deve fare per questo il progettista utilizza descrizioni algoritmiche, fatte con istruzioni procedurali simili a quelle dei linguaggi di programmazione software.

2. **Descrizione Dataflow :** cosiddetta perché contiene una rappresentazione del flusso dei dati fatta per mezzo di equazioni logiche, in cui si descrive l'effetto dei segnali in uscita da un flusso in risposta a eventi su segnali in ingresso al flusso.

[4] E' la fabbrica che si occupa della produzione di circuiti integrati
[5] In Inglese è indicata col termine *Behavioural*

3. **Descrizione Strutturale o Gate Level :** è al livello più basso, dove il sistema viene rappresentato direttamente nella sua struttura come rete di porte logiche, indicando i componenti del sistema e le interconnessioni tra di essi. Questa descrizione è equivalente a quella che si ottiene con un disegno schematico, solo che non è grafica ma testuale.

Il modello può contenere descrizioni di tipo diverso per parti diverse del sistema. Inoltre è evidente che se di una parte del sistema viene data una descrizione strutturale, il funzionamento interno dei sottosistemi corrispondenti deve comunque essere definito da qualche parte nel progetto o in moduli esterni. Per poter realizzare il dispositivo una descrizione funzionale o dataflow deve essere tradotta in una descrizione strutturale. La traduzione automatica di tutti i costrutti funzionali permessi dal VHDL non è però sempre possibile e dipende dal supporto offerto dal software di sintesi utilizzato.

1.4.2 I livelli di simulazione

La simulazione viene eseguita andando a stimolare il dispositivo o alcune sue parti con tutti i possibili ingressi verificando se i segnali in uscita sono quelli attesi. Si crea quindi un file comunemente chiamato $testbench$, che si occupa di fornire gli stimoli necessari.

La possibilità di simulare un sistema mentre alcune sue parti sono ancora in via di sviluppo si paga in termini di accuratezza dei risultati ottenuti. Minori sono le informazioni sull'hardware che vengono fornite al simulatore, maggiore è la possibilità che il risultato sia affetto da errori, un esempio significativo è dato dalla mancata conoscenza dei tempi di risposta dei vari componenti.

Principalmente si possono distinguere tre livelli di simulazione dipendenti dal grado di accuratezza che si vuole ottenere:

1. **Simulazione pre-sintesi:** questa simulazione viene eseguita sul codice originale ed è pertanto molto veloce ma non tiene conto dei ritardi di commutazione delle porte logiche, che non sono ancora state definite. I risultati che si ottengo in questa fase sono quindi ideali.

2. **Simulazione post-sintesi:** questa simulazione è eseguita dopo la sintesi del codice. Il codice originale viene sostituito con la rete logica risultante dalla sintesi. Questa simulazione tiene conto dei vari ritardi di commutazione

delle porte logiche che ci sono forniti dal costruttore del dispositivo finale, oppure sono scelti dal progettista nel caso manchino ancora i dettagli implementativi.

3. **Simulazione post-place and route:** questa simulazione è eseguita tenendo conto del dispositivo finale su cui andrà programmata la rete logica. Vengono considerati i ritardi relativi alle lunghezze delle connessioni tra le porte logiche dovute alle loro posizioni finali sul chip. Nel caso i vincoli sulle tempistiche siano molto ristretti, l'utilizzo di percorsi di connessione più o meno lunghi può portare a ritardi non trascurabili. Il tempo di questa simulazione è molto lungo ma i risultati rispecchiano, il funzionamento del progetto finale realizzato nel dispositivo.

1.4.3 La sintesi

La sintesi è la trasformazione attraverso opportune regole della descrizione di un sistema hardware di alto livello, in un altra descrizione dello stesso sistema in termini di blocchi elementari interconnessi.

Tramite costrutti di poche righe si descrivono circuiti di migliaia di porte logiche, consentendo al progettista di concentrarsi sul comportamento del sistema e non sui dettagli della loro implementazione.

La sintesi del codice potrebbe non essere efficiente, complicando inutilmente la logica risultante se la descrizione VHDL è scritta male poichè è il sintetizzatore a decidere l'implementazione al *gate level*. Per ottimizzare il codice è quindi preferibile una conoscenza approfondita della struttura interna del dispositivo specifico dove verrà utilizzato il codice. L'utilizzo di librerie ottimizzate dal costruttore del dispositivo permettono di avere un risultato garantito, a scapito però della portabilità.

Il processo di sintesi dipende dalla tecnologia nella quale si vuole realizzare l'implementazione, dagli obiettivi che si pone il progettista e dai vincoli imposti dall'ambiente. Gli obiettivi specificano l'importanza dei diversi parametri di costo con cui valutare la bontà di un'implementazione, fra questi la velocità di esecuzione, l'area occupata dal circuito e il consumo. I vincoli specificano le condizioni imposte da tutto ciò che circonda il circuito, come il tempo di arrivo dei valori in ingresso o il massimo ritardo di quelli in uscita.

A sintesi ultimata si vuole ottenere un implementazione che sia coerente con

le specifiche, realizzabile nella tecnologia specificata e di costo minimo.

La semantica dei linguaggi di descrizione, di contro, è spesso ambigua, alcune descrizioni se simulate esibiscono comportamenti dissimili da quelli dell'hardware sintetizzato (es. una porta logica in fase di simulazione non presenta ritardo tra l'ingresso e l'uscita mentre una volta sintetizzata questo è presente).

Occorre definire un sottoinsieme del linguaggio che sia sintetizzabile e degli stili di descrizione in grado di eliminare le ambiguità. A questo scopo è meglio avere la conoscenza di base di come le singole istruzioni del linguaggio sono poi tradotte dal sintetizzatore in reti logiche elementari. Il rispetto dei vincoli richiede che il sintetizzatore sia in grado di valutare accuratamente i ritardi di propagazione ed ottimizzarli identificando i cammini critici.

L'ottimizzazione è mirata a raggiungere un compromesso tra l'area complessiva occupata dal circuito sintetizzato e il ritardo di propagazione sui cammini. Di solito tra tutte le implementazioni che soddisfano i parametri, al sintetizzatore è richiesto di scegliere quella ottima, cioè quella che minimizza una funzione di costo globale.

Si possono identificare vari livelli di sintesi dovuti ai diversi livelli di astrazione:

1. **High-level:** fornisce una descrizione a livello logico partendo da descrizioni comportamentali di alto livello.
2. **Logic-level:** fornisce una descrizione strutturale al livello logico partendo da una descrizione comportamentale.
3. **Low-level o Silicon compilation:** fornisce una descrizione del *layout* partendo da una descrizione low-level

Si possono avere così strumenti per la sintesi architetturale che da un algoritmo o un data-flow creano delle unità funzionali interconnesse ottimizzate. La sintesi a livello logico è in grado di implementare ad esempio una macchina a stati finita complessa a livello comportamentale oppure può generare da un insieme di equazioni logiche, l'interconnessione delle porte logiche ottimizzando il loro numero. La sintesi a livello più basso converte un insieme di porte logiche in celle interconnesse ottimizzando l'area occupata e i ritardi.

Commercialmente esistono diverse aziende che sviluppano sintetizzatori di alto livello, ma che non producono logiche programmabili. Sintetizzatori in grado di generare automaticamente il *layout* a partire da descrizioni comportamentali sono disponibili solo per applicazioni molto specifiche.

1.4.4 Implementazione

Una volta che il progettista ha definito il chip in VHDL ed ha fatto tutte le sue simulazioni si procede al trasferimento del progetto logico sul supporto fisico. Per questo procedimento esistono tre possibili approcci:

1. Realizzare mediante *Silicon Foundry* un circuito integrato con un aproccio *full custom* dove l'utente ha il controllo anche del disegno dei singoli transistor;

2. Realizzare un circuito integrato *semi-custom* dove l'utente ha la libertà di collegare tra loro celle elementari con strutture definite dalla *Silicon Foundry*;

3. Utilizzare una logica programmabile.

La prima soluzione prevede la compilazione del progetto logico fino al livello di singole disposizioni di transistor, questo approccio richiede un forte intervento umano ma permette una grande ottimizzazione dei ritardi e lo sfruttamento massimo della tecnologia di integrazione scelta, consetendo di raggiungere prestazioni elevate. Sono progetti *full custom* tipicamente i processori (Intel, AMD,etc...). Chiaramente chi si può permettere un approccio del genere può creare prodotti altamente performanti caricando di molto i costi fissi e i costi di sviluppo ma riducendo poi i costi ricorrenti come il prezzo per ogni *die*[6] di silicio integrato. E' tuttavia prevedibile un ciclo di progettazione piuttosto lungo adatto a prodotti di lunga vita commerciale.

La seconda soluzione prevede l'utilizzo di tecnologie semi-lavorate. Si tratta in pratica di array di porte logiche in gran parte già definiti, dove tramite la definizione delle interconnessioni a livello di chip si implementa la logica voluta. I costi di sviluppo si riducono a discapito delle prestazioni e dell'ottimizzazione, che rimangono comunque molto elevate. Si ha anche un certo livello di risorse non utilizzabili a causa della struttura a griglia predefinita che quindi fanno

[6]La sottile piastrina di silicio sulla quale è stato ricavato un circuito integrato.

salire il costo per pezzo prodotto. I chip video sono ad esempio spesso dei chip in tecnologia *semi-custom*, così come i *chip-set* delle mother-board.

La terza soluzione utilizza dei particolari chip già integrati in cui si hanno una serie di blocchi logici elementari riprogrammabili e collegabili tra loro secondo le esigenze del progetto finale. I vantaggi sono evidenti: si compra il chip e la scheda che ne permette la programmazione, si genera tramite un ambiente di sviluppo VHDL il file di configurazione, ed il chip è già pronto per l'applicazione sul campo.

In questo modo si azzerano praticamente i costi fissi (non ci sono maschere da realizzare per l'integrazione dei chip), ma si innalza di molto il costo per singolo chip. Queste logiche infatti sono ampiamente ridondanti per permettere la riprogrammazione, ed esiste quindi sempre un rapporto considerevolmente sfavorevole tra numero di transistor globali che compongono il chip fisico e numero di transistor equivalenti del progetto logico in esso implementato.

E' chiaro tuttavia che questo è un compromesso non solo accettabile,ma in molti casi anche l'unica possibilità di sviluppare un chip in proprio per chi non può disporre dei centinaia di migliaia di euro necessari a creare le maschere per l'integrazione diretta di un chip.

Inoltre, c'è da considerare che in certi ambiti industriale, biomedicali e nella creazione di prototipi, la produzione conta solo pochi chip (migliaia, nel migliore dei casi) e che conseguentemente non è assolutamente pensabile l'adozione delle tecniche *full-custom* o *semi-custom* che hanno bassi costi per pezzo ma altissimi costi di progettazione che andrebbero a ricadere sui pochi chip prodotti.

1.4.5 Verifica

Altri aspetti fondamentali in un flusso di progettazione sono l'analisi e la verifica dei vari passi del progetto e la simulazione globale del lavoro. La maggior percentuale di tempo impiegata nella realizzazione di un chip è dovuta alla verifica della correttezza del progetto stesso.

Supponendo di procedere con un approccio top-down ogni qualvolta si scende di livello è necessario verificare che non siano stati commessi errori nella sintesi. Inizialmente, quando i simulatori non erano molto efficienti il metodo seguito per verificare la correttezza dei risultati era quello di completare la progettazione

fino alla produzione di un prototipo del dispositivo e a questo punto testarne il funzionamento.

Lo sviluppo dei simulatori ha dato un notevole contributo alla riduzione dei costi di progettazione permettendo di testare il dispositivo prima di realizzarlo fisicamente.

Oggi è possibile tramite *demo-board* più o meno complesse, caricare il file di implentazione sulla logica, e verificare i risultati. I vari produttori di logica programmabili mettono a disposizione queste schede a prezzi contenuti tanto che è possibile utilizzare anche per uso hobbistico.

1.5 Caratteristiche di VHDL

Il VHDL è stato realizzato seguendo una filosofia comune ai moderni linguaggi di programmazione. Il VHDL deriva pesantemente da ADA[7] sia come concetti che come sintassi.

La ragione per cui il DOD richiese che la sintassi fosse il più possibile basata su ADA, fu per evitare di re-inventare concetti che erano già stati ampiamente testati in campo militare con lo sviluppo di ADA. Come ADA il VHDL è quindi fortemente tipato e *case insensitive*.

Naturalmente il VHDL ha molte caratteristiche che in ADA non troviamo come l'estensione del tipo booleano con tutte le relative operazioni che vengono normalmente usate nei sistemi digitali. Un'altra caratteristica è la possibilità di indicizzare un array in entrambe le direzioni, sia discendente che ascendente perché entrambe le convenzioni sono usate in hardware.

Si pone anche particolare attenzione all'aspetto concorrenziale del sistema che si andrà a descrivere caratteristica intrinseca dei circuiti digitali. VHDL è quindi un linguaggio di programmazione parallela dove vari blocchi di codice vengono eseguiti contemporaneamente, a differenza di un normale linguaggio di programmazione dove le istruzioni vengono eseguite sequenzialmente.

[7] Linguaggio di programmazione software sviluppato verso la fine degli anni '70 su iniziativa del Dipartimento della Difesa (DOD) degli Stati Uniti.

1.6 Concetti chiave del VHDL

Un sistema hardware si può sempre rappresentare in termini di un oggetto che riceve degli input, esegue delle operazioni e produce degli output. In generale sarà costituito da uno o più sottosistemi rappresentabili sempre in questo schema. Quindi per descrivere un sottosistema si devono definire:

1. La sua interfaccia esterna, ossia gli ingressi e le uscite che rappresentano le sue relazioni con gli altri sottosistemi o con l'esterno;

2. Il suo funzionamento interno, cioè che cosa fa il sistema e/o come lo fa.

In VHDL il modello di un sottosistema è detto design entity ed è costituito da due parti: una entity che descrive l'interfaccia, cioè come viene visto dall'esterno e una architecture che descrive il funzionamento. Un semplice esempio è rappresentato in Fig.1.2.

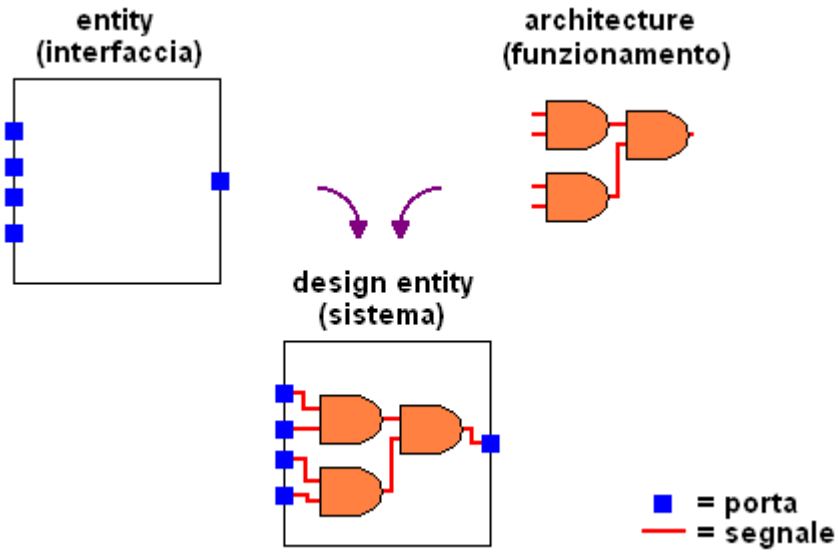


Figura 1.2: Esempio di entità

Nella descrizione hanno un ruolo fondamentale i segnali, rappresentazione dei conduttori, che spostano i dati tra parti diverse del sistema. Le porte dell'interfaccia esterna sono segnali particolari perché spostano i dati da e verso l'esterno.

I packages e le librerie sono moduli che possono contenere porzioni di codice isolate dal resto della descrizione, allo scopo di rendere il codice riutilizzabile. Le

entity, le architecture, i package (e i package body) sono detti design units. Un modello praticamente è costituito da design units, al cui interno si costruisce la descrizione dettagliata del sistema. Il modello di un sistema si descrive in un file sorgente che contiene istruzioni VHDL in formato testo.

L'utilizzo del linguaggio VHDL è fondamentale costituendo uno strumento di specifica univoco in quanto standardizzato. Nel prossimo capitolo, è analizzato il funzionamento generale dei dispositivi logici programmabili.

Capitolo 2

Le logiche programmabili

In questo capitolo verranno presentate le logiche programmabili, descrivendo il loro funzionamento, e perché attualmente vengano utilizzate negli ambiti più svariati. Alla fine è presentata la FPGA utilizzata nel presente lavoro di tesi.

2.1 Vantaggi delle logiche programmabili

I dispositivi FPGA (*Field Programmable Gate Array*) sono dispositivi digitali la cui funzionalità è programmabile via software.

Non si tratta della classica programmabilità di un processore ma della possibilità di riconfigurare le risorse logiche disponibili per implementare un progetto complesso a livello di singole interconnessioni tra porte logiche.

Le FPGA sono elementi che presentano caratteristiche intermedie rispetto ai dispositivi ASIC (*Application Specific Integrated Circuit*) e a quelli con architettura PAL (*Programmable Array Logic*). L'uso di tali componenti comporta tuttavia alcuni vantaggi rispetto agli ASIC, si tratta infatti di dispositivi standard la cui funzionalità non viene impostata dal produttore che può quindi produrli su larga scala a basso prezzo.

La loro genericità li rende adatti a un gran numero di applicazioni come elettronica di consumo, comunicazioni, automotive, aerospaziale, militare e biomedicale. Sono programmati direttamente dall'utente finale, consentendo la diminuzione dei tempi di progettazione, di verifica mediante simulazioni e della prova sul campo dell'applicazione tramite prototipazione rapida.

Il grande vantaggio rispetto agli ASIC è che permettono di apportare eventuali modifiche o correggere errori semplicemente riprogrammando il dispositivo

in qualsiasi momento. Di contro sono antieconomici per applicazioni con elevati volumi di produzione (milioni di pezzi), poichè il prezzo unitario del dispositivo è superiore a quello degli ASIC che permettono in questi casi di ammortizzare gli elevati costi di progettazione.

Gli sviluppi tecnologici e la crescente capacità di integrazione hanno portato al continuo incremento del numero di porte logiche integrabili su un unico chip tanto che gli ultimi modelli affiancano ai blocchi per la realizzazione di logica combinatoria e sequenziale, memorie dedicate, sommatori e moltiplicatori hardware, complessi sistemi di clock multifase, interfacce di I/O ad elevate prestazioni oltre a veri e propri microprocessori *embedded* (nei prodotti di fascia più alta), tanto da consentire la realizzazione di interi sistemi sul chip (SoC, System on Chip).[2]

Generalmente le FPGA vengono programmate con linguaggi di alto livello come il Verilog o il VHDL di cui si è parlato approfonditamente nel capitolo 1. Con questi linguaggi ci si può permettere di usare librerie di componenti predefiniti e lavorare con blocchi e macro-blocchi di alto livello. Non bisogna comunque dimenticare la possibilita di usare la modalità *schematic-entry* che consente un approccio veloce e semplificato a tale tecnologia collegando tra loro blocchi già forniti dal costruttore. Molte case costruttrici forniscono gratuitamente sistemi di sviluppo (*demo-board* e software) per supportare tutta la loro gamma di prodotti.

2.2 Descrizione di un dispositivo FPGA

Fra i dispositivi logici programmabili PLD (Programmable Logic Devices) i circuiti FPGA hanno sicuramente la struttura più flessibile. Un dispositivo FPGA è un circuito a semiconduttore che contiene sia componenti logici che interconnessioni programmabili.

I componenti logici programmabili (o blocchi logici) permetto di implementare le porte logiche elementari, le funzioni combinatorie più complesse o anche semplici funzioni matematiche. Tramite un blocco logico si può quindi rappresentare una qualunque funzione logica.

L'architettura tipica di un dispositivo FPGA, è caratterizzata da una matrice di blocchi logici programmabili, interconnessi fra loro con connessioni programmabili. La struttura prevede una serie di *pad* di I/O esterni con posizione fissata ma sempre con funzionalità programmabile. La FPGA di esempio in Fig.2.1, ha

9 blocchi logici e 24 I/O *pads*.

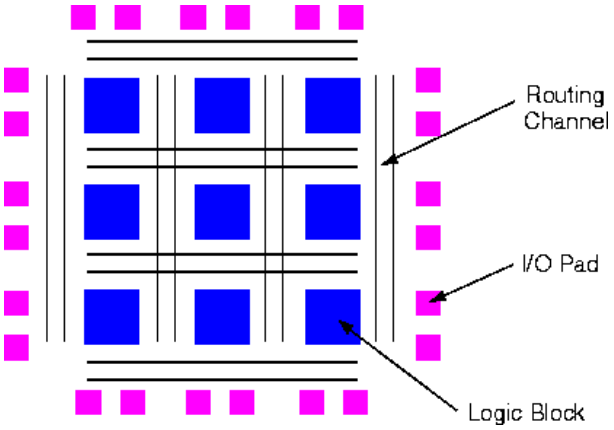


Figura 2.1: Struttura di un dispositivo FPGA

Per realizzare i blocchi logici le varie case costruttrici hanno sviluppato nel tempo diverse soluzioni, a partire dalla più semplice realizzazione di funzioni booleane attraverso la connessione di porte NAND e NOR fino all'utilizzo di memorie statiche.

La tecnologia più largamente impiegata negli ultimi anni è quella che prevede la realizzazione dei logic blocks attraverso LUT (*look-up tables*) a N ingressi che permettono l'implementazione di tutte le 2^N funzioni logiche di N variabili. In Fig. 2.2 è raffigurato, un blocco logico elementare, dotato di quattro ingressi alla *lookup table* (LUT) e un flip-flop [3].

Ai quattro ingressi va aggiunto l'ingresso del segnale di clock separato dagli altri ingressi perché utilizza percorsi preferenziali per poter raggiungere tutti i blocchi con il minimo ritardo. Attualmente vengono prodotti LUT con 6 o più ingressi.

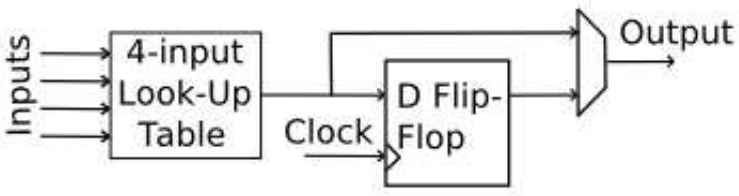


Figura 2.2: Esempio di blocco logico elementare

Ogni ingresso è accessibile da un lato del blocco logico e ogni uscita può essere connesso con ogni filo presente nel canale ad esso adiacente tramite un blocco di interconnesione programmabile denominato *switch box*.

Per ogni intersezione di un canale verticale ed uno orizzontale si inserisce uno *switch box*. Ogni filo collega un unico blocco logico con uno *switch box*. L'architettura di esempio in Fig.2.3 mostra che ogni volta un filo entra nello *switch box* ci sono tre *switch* programmabili che permettono di connetterlo ai tre fili dei canali adiacenti. La Fig. 2.4 mostra un esempio della topologia di uno *switch box* dove

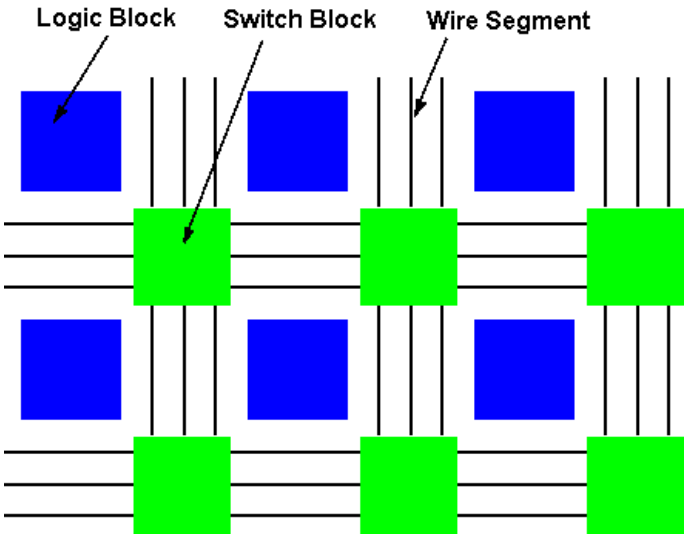


Figura 2.3: Architettura di un dispositivo FPGA

si vede in dettaglio quali connessioni è possibile creare per ogni incrocio.

In realtà le FPGA possono includere, blocchi DSP[1] generici, processori *embedded*, memorie e blocchi di I/O ad elevata velocità. Sulla base di quanto descritto emerge che per bassi e medi volumi di produzione conviene utilizzare i dispositivi FPGA. Anche per generare progetti su ASIC tipicamente si realizza un prototipo sub-ottimo su FPGA per effettuare verifiche sulla validità del progetto stesso e una volta concluso si trasferisce sul circuito ASIC per la produzione su larga scala.

Tecnologie per programmare i dispositivi FPGA

La configurazione della FPGA avviene tramite la programmazione degli *switch box* e dei vari blocchi logici del dispositivo. Sono state perciò sviluppate diverse

[1]Digital Signal Processor

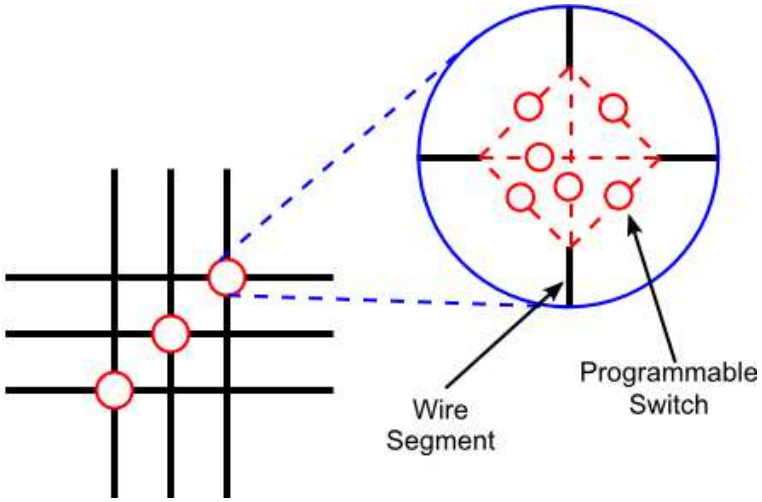


Figura 2.4: Topologia di uno switch box

soluzioni per la realizzazione della configurazione delle interconnessioni, dalle cui proprietà dipendono molte delle caratteristiche strutturali, funzionali ed economiche degli FPGA attualmente in commercio.

Presentiamo di seguito le diverse tecnologie usate per realizzare la programmazione della FPGA.

Tecnologia SRAM

La soluzione è basata sull'utilizzo di RAM statiche in cui viene memorizzata la configurazione della FPGA.

Quando nella cella di SRAM è memorizzato un valore di tensione alto, il *pass gate* si comporta come un interruttore chiuso e può essere utilizzato per connettere due linee, mentre quando la cella memorizza un valore basso, il transistor presenta una elevata resistenza tra le linee impedendone il collegamento come raffigurato in Fig. 2.5.

A causa della natura volatile della SRAM, la FPGA deve essere configurata al momento dell'alimentazione del chip. I dispositivi basati su questa tecnologia necessitano quindi di una memoria esterna permanente per la conservazione dei bit di programmazione.

Il maggior inconveniente di questa tecnologia risulta l'elevata occupazione di area, compensata però dalla rapida ed efficiente riprogrammabilità del dispositivo.

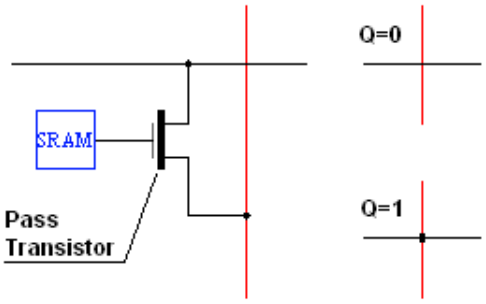
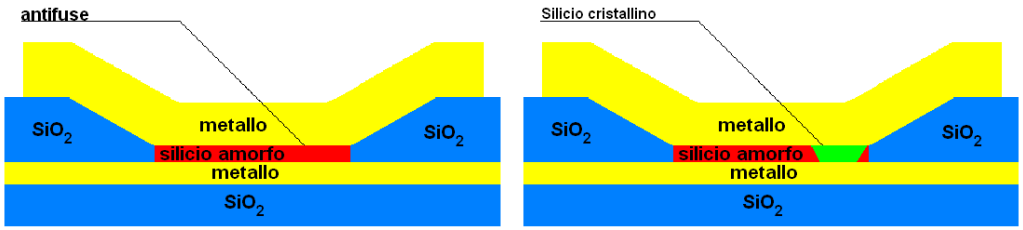


Figura 2.5: Tecnologia SRAM

Tecnologia antifuse

La soluzione prevede l'utilizzo di *antifuse devices* per il collegamento dei vari blocchi logici.

Un *antifuse* è un dispositivo a due terminali che presenta, prima della programmazione, un'elevata resistenza tra i suoi capi comportandosi come un circuito aperto, in Fig. 2.6(a) è rappresentata la sezione di un transistor MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*) realizzato con tecnologia *antifuse*. La parte in rosso è composta da silicio amorfo che si comporta come un materiale isolante. Applicando tensioni molto alte ai terminali del dispositivo *antifuse* si crea un cammino permanente scarsamente resistivo, evidenziato in verde in Fig. 2.6(b), trasformando il silicio amorfo in silicio policristallino e realizzando in questo modo le interconnessioni sulla FPGA. I principali vantaggi di questa tec-



(a) Circuito aperto. (b) Circuito chiuso .

Figura 2.6: Sezione circuito antifuse

nologia sono le dimensioni estremamente ridotte, la resistenza in serie all'*antifuse* relativamente bassa e capacità parassite molto inferiori rispetto ad altre tecnologie. Va aggiunta anche la resistenza intrinseca di questa tecnologia alle radiazioni, per cui questa tecnologia risulta particolarmente impiegata in condizioni ambientali difficili: come sistemi spaziali, militari, nucleari ed ospedalieri.

La programmazione di dispositivi che utilizzano questa tecnologia non è reversibile, tale limitazione può rappresentare uno svantaggio nel caso sia richiesta la riprogrammazione del dispositivo.

Tecnologia a gate flottante

La tecnologia a gate flottante (*floating gate*) si basa sull'utilizzo di transistor a gate flottante che forniscono un meccanismo di programmazione reversibile e non volatile. La tecnica consiste nel mantenere l'informazioni in porzioni di conduttori elettricamente isolati in grado di mantenere la carica. In Fig. 2.7 è rappresentata

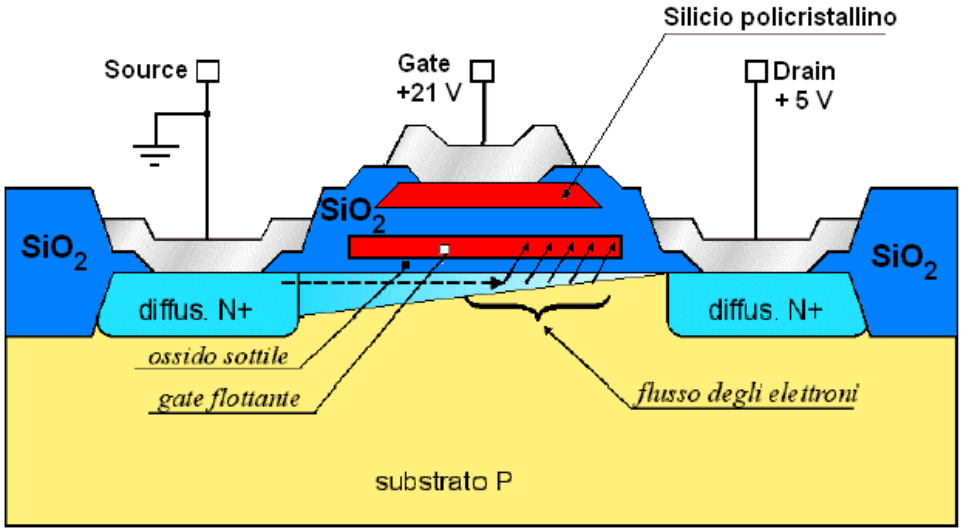


Figura 2.7: Tecnologia a gate flottante

la sezione della cella di memoria: è composta uno strato di materiale conduttore racchiuso entro due strati di ossido di silicio, uno più spesso ed uno sottostante molto sottile. Lo strato conduttore completamente racchiuso nell'ossido viene detto gate flottante.

Per scrivere il bit da memorizzare nella cella si fornisce tensione al transistor Mosfet di tipo sottostante (si applicano +5 V fra drain e source) dopodiché si polarizza l'elettrodo di gate con una tensione Vgs sufficientemente elevata (anche qualche decina di Volt) da costringere gli elettroni che transitano nel canale fra drain e source a perforare per effetto tunnel la sottile barriera di ossido isolante e raggiungere lo strato di gate flottante. Gli elettroni raggiunto questo strato vi rimangono confinati in quanto non sono in grado di proseguire verso il gate so-

vrastante a causa del maggior spessore dell'ossido. Una volta tolta la tensione di programmazione gli elettroni rimarranno intrappolati all'interno del gate flottante grazie all'elevatissima resistività dell'ossido che è in grado di garantire una *data retention* di molte decine di anni.

La lettura del contenuto della cella viene effettuato polarizzando normalmente il transistor, applicando una tensione di 5 V fra gate e source e leggendo la tensione in uscita: se il gate flottante non contiene elettroni il Mosfet sarà in conduzione, mentre se lo strato di conduttore annegato nell'ossido contiene elettroni il Mosfet risulterà interdetto, a causa della maggiore tensione di soglia di gate necessaria per la conduzione.

Come per la tecnologia basata su SRAM il maggior vantaggio di questa soluzione è rappresentato dalla sua riprogrammabilità, inoltre non necessita di una memoria esterna permanente per programmare il chip, la realizzazione su silicio richiede però processi tecnologici più complessi rispetto a quello CMOS (*Complementary MOS*) standard.

2.3 Caratteristiche della logica utilizzata

La scelta della FPGA del presente progetto, è ricaduta sulla serie ProASIC3 [4] prodotta dalla Actel, e in particolare il modello A3P125 con package VQ100[2]. Questa versione è dotata di 125000 porte logiche oltre a vari blocchi specifici come memorie RAM, ROM e un PLL[3] interno. Questa logica rispetta i requisiti richiesti cioè basso costo per singolo chip, non volatilità, basso consumo e intervallo di temperatura di funzionamento molto ampio.

Altre caratteristiche pubblicizzate dal costruttore sono visibili in Fig.2.8 dove viene messo in evidenza l'unione di alcune caratteristiche dei dispositivi ASIC con quelle delle logiche programmabili FPGA *SRAM based* [4].

La Fig. 2.9 mostra il diagramma a blocchi dell'architettura della serie ProAsic, dove è possibile identificare i vari blocchi logici e notare come già in FPGA economiche, siano presenti blocchi ottimizzati per funzioni specifiche.

[2]La sigla identifica un package plastico ultrasottile con 100 pin disposti sui 4 lati (Quad-edged Flat Pack.)

[3]Phase Locked Loop

[4]FPGA veloci e di basso costo, ma perdono la programmazione in mancanza di alimentazione.

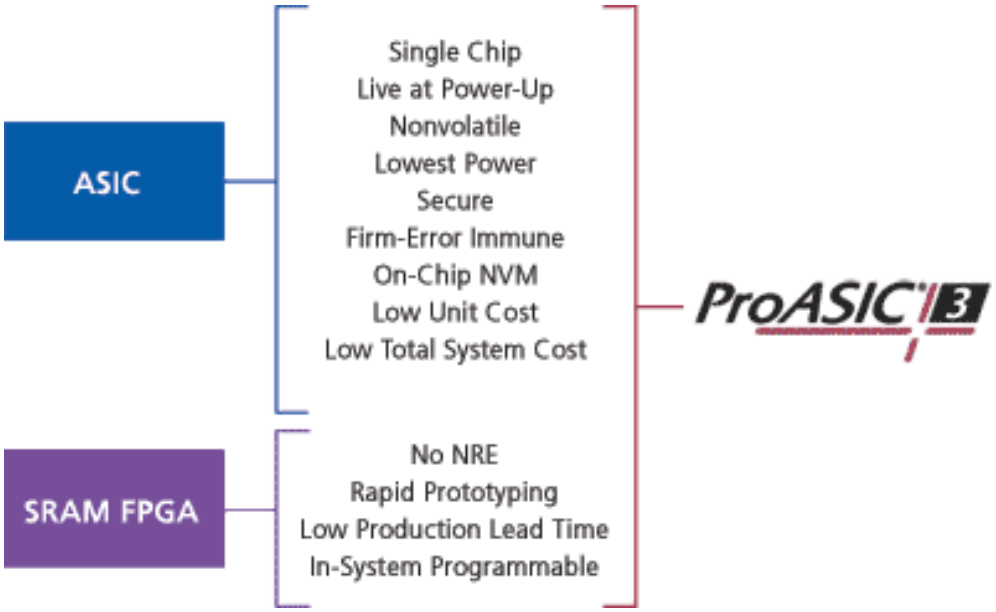


Figura 2.8: Caratteristiche della serie ProASIC3

1. Memoria dedicata e registi;
2. VersaTile elements, cioè il termine utilizzato dall'Actel per rappresentare un LUT a 3 ingressi, oppure un flip-flop di tipo D;
3. Interfaccia I/O avanzata, capace di utilizzare varie tensioni da 1,5 V fino a 5 V;
4. Charge Pumps, o pompa di carica cioè circuiti che inalzano le tensioni, per permettere il funzionamento a tensioni variabili;
5. Flash ROM, è una memoria ROM programmabile dall'utente, utile per memorizzare valori costanti;
6. JTAG(IEEE1532), è l'interfaccia standard per la programmazione;
7. Switch box;
8. PLL interno, che consente di generare il segnale di clock.

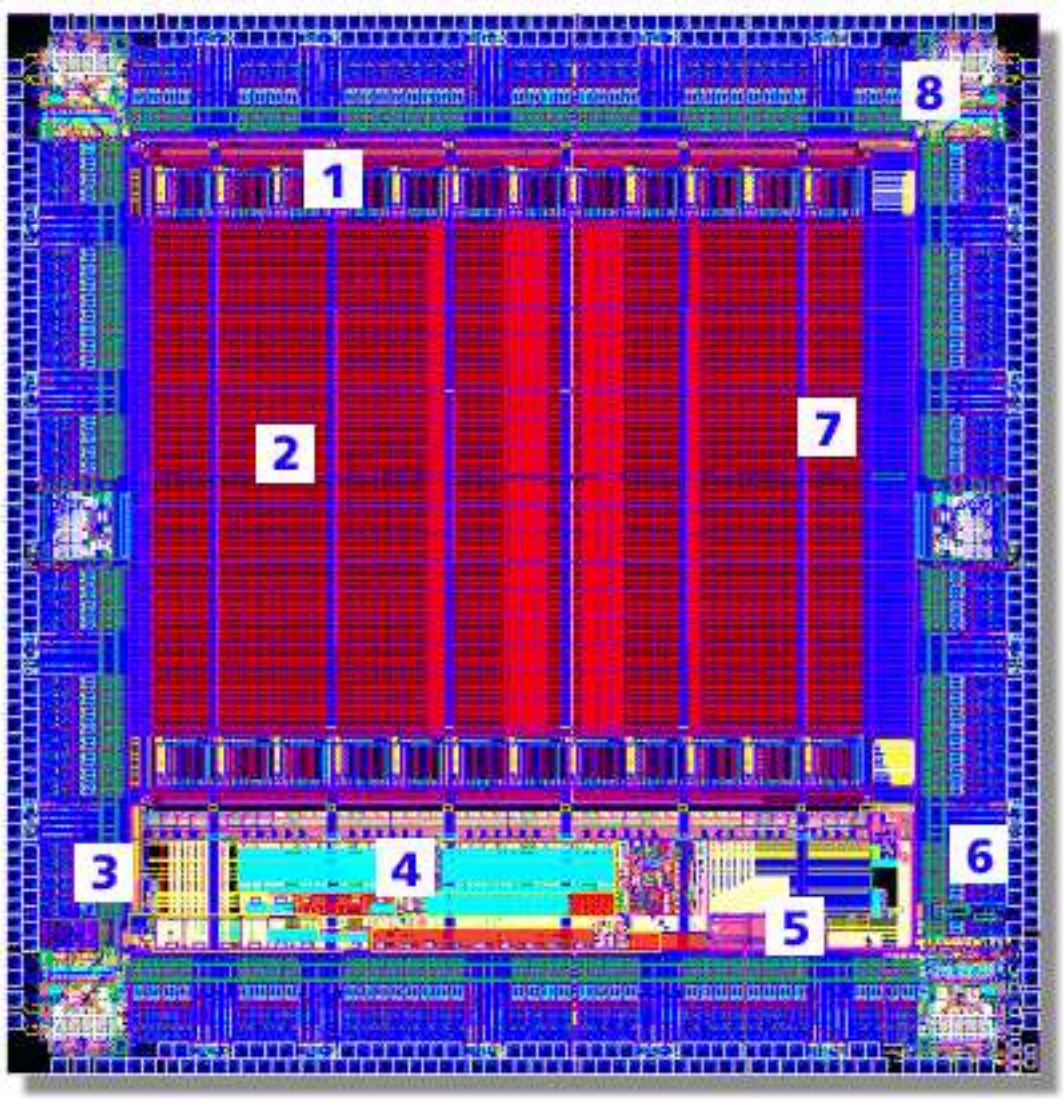


Figura 2.9: Architettura a blocchi ProASIC3

Capitolo 3

I dispositivi RFID

In questo capitolo sono introdotti i sistemi di Identificazione a RadioFrequenza (RFID), elencandone le caratteristiche, il funzionamento di base e le possibili applicazioni.

3.1 Composizione e funzionamento di un sistema RFID

RFID è acronimo di Radio Frequency IDentification traducibile in Identificazione a RadioFrequenza, ed è una tecnologia per la identificazione automatica [5].

La configurazione tipica dei sistemi RFID mostrata in Fig. 3.1 si può considerare formata da tre oggetti principali:

- Il tag o transponder che è situato sull'oggetto, persona, o animale da identificare;
- Il lettore o transceiver interroga il tag leggendo o scrivendo dati nella sua memoria;
- Il sistema di elaborazione dati.

Il sistema si basa sulla lettura a distanza di informazioni contenute in un tag RFID usando dei lettori RFID. Un tag è in grado di ricevere e di trasmettere via onde radio le informazioni contenute nel chip ad un lettore RFID.

Un tag RFID è costituito da:

- Un microchip che contiene dati;
- Un'antenna;

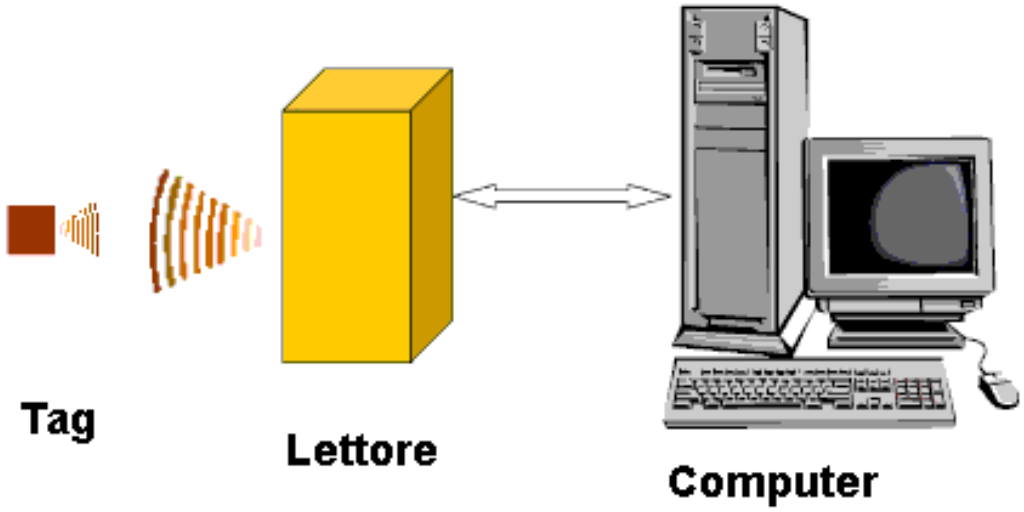


Figura 3.1: Composizione del sistema RFID

- Può essere dotato o meno di una batteria.

Il tag consiste normalmente di un microchip, che immagazzina i dati ed un elemento d'accoppiamento o antenna; in Fig. 3.2 è mostrato un tag. Il lettore con-

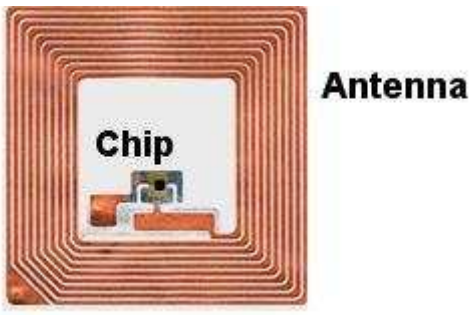


Figura 3.2: Conformazione di un tag

tiene un modulo a radiofrequenza per trasmettere e ricevere i dati, un antenna di accoppiamento, un alimentatore e una interfaccia per la comunicazione con il sistema di acquisizione dati.

Il tag è completamente spento quando non è all'interno della zona di copertura del lettore e viene attivato ed interrogato soltanto quando è all'interno di tale zona. L'alimentazione richiesta per attivare il tag è fornita completamente dall'onda a radiofrequenza attraverso l'unità d'accoppiamento, nel caso di tag completamente passivi. Nel caso di tag semiattivi, l'alimentazione viene fornita in parte dalle batterie e in parte dall'accoppiamento elettromagnetico con il lettore. In ogni caso, non è mai richiesto il contatto tra lettore e tag.

Principi di funzionamento del tag

Tutto il sistema opera rispettando i principi dell'accoppiamento induttivo, sia il lettore che il tag sono dunque dotati di elementi di accoppiamento. L'accoppiamento induttivo permette il trasferimento di energia dal lettore al tag. Una parte del flusso del campo magnetico generato dall'antenna del lettore, si concatena con l'antenna del tag che si trova ad una certa distanza dal lettore. Due o più circuiti si dicono mutuamente accoppiati quando, al variare della corrente in uno di essi, corrispondono variazioni di flusso che vanno anche a concatenarsi con gli altri circuiti, originando f.e.m. indotte di mutua induzione. La mutua induttanza è la misura quantitativa dell'accoppiamento fra i due conduttori.

Fissate le caratteristiche dei due conduttori, vale a dire il raggio della spira e il numero di spire, il coefficiente di accoppiamento dipende unicamente dalla distanza fra i due conduttori. La legge di Faraday-Lenz stabilisce che quando il flusso magnetico concatenato con il circuito varia nel tempo, si genera in tale circuito una f.e.m. indotta che si oppone a tali variazioni e risulta uguale alla derivata rispetto al tempo del flusso cambiato di segno:

$$e = -\frac{d\phi}{dt}$$

Lo schema in Fig. 3.3 rappresenta il circuito equivalente del tag e del lettore, che si studia come il modello di un trasformatore dove al primario vi è l'antenna del lettore e al secondario l'antenna del tag.

Il lettore è composto dall'induttanza L_1 e dalla resistenza R_{L1} che rappresentano l'induttanza e la resistenza dell'antenna del lettore. Lo scopo del lettore è massimizzare il flusso magnetico e quindi la corrente, perciò bisogna aggiungere la capacità C_1 in serie all'induttanza per creare un impedenza totale che tende a zero alla frequenza di risonanza, in tale modo la corrente sarà limitata solo dalla resistenza R_{L1} del circuito.

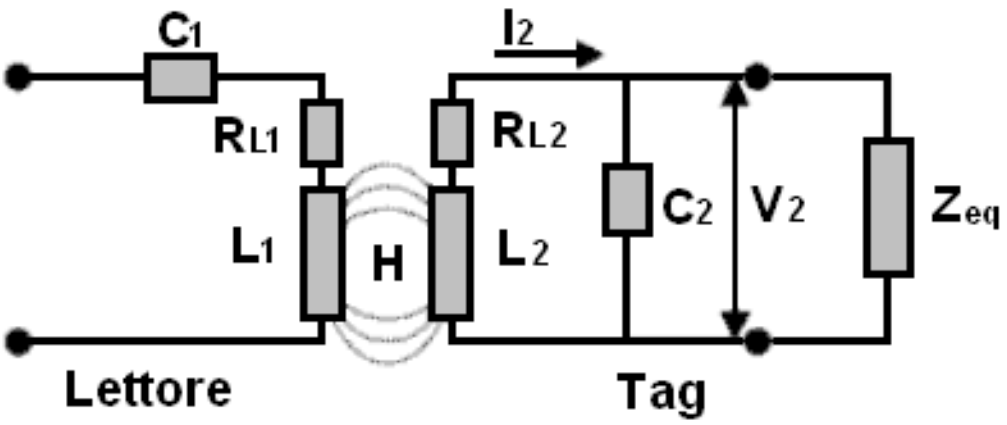


Figura 3.3: Circuito equivalente del tag e del lettore

Il tag è composto dalla resitenza R_{L2} e dall'induttanza L_2 dovuti alle caratteristiche della sua antenna. In questo caso bisogna inserire una capacità C_2 in parallelo, in modo da rendere l'impedenza totale alla frequenza di risonza di valore molto grande, cosi da massimizzare la tensione V_2 che dipenderà solo dalla resistenza R_{L2}. Zeq rappresenta il carico del tag dovuto ai consumi del chip di controllo, del modulatore e della memoria. La tensione V_2 indotta nell'antenna del tag è utilizzata come tensione di alimentazione del microchip del tag stesso.

Un flusso che varia nel tempo nel conduttore 1 causa mutuo accopiamento tra i due conduttori e induce una tensione nel conduttore 2. La corrente variabile i_1 che scorre nel conduttore 1 genera un flusso magnetico $\phi 1$, che in parte si concatena sia con l'avvolgimento 1 che con l'avvolgimento 2, provocando una f.e.m. indotta da fornire una d.d.p. v_2 nell'avvolgimento 2.

3.2 Classificazione dei sistemi RFID

I sistemi RFID possono essere classificati in base al tipo di alimentazione, alla frequenza di lavoro, e al raggio di azione.

Classificazione in base ai requisiti d'alimentazione: I tag attivi sono alimentati da batterie, i tag semiattivi sono alimentati da batterie solo per la parte di trasmissione e i tag passivi non hanno nessuna fonte di alimentazione interna ma traggono l'energia per attivarsi dall'onda radio inviata dal lettore che li interroga.

Classificazione in base al raggio di azione: Il raggio d'azione di questi sistemi, può variare da pochi centimetri fino a 15 m di distanza. I sistemi RFID a piccolo raggio, necessitano per il loro funzionamento che il tag sia molto vicino al lettore, con distanze di pochi centimetri. I tag utilizzati sono di solito di tipo passivo, visto che la vicinanza facilita la fornitura di elevate quantità di potenza.

Questi sistemi sono usati nelle applicazioni che richiedono elevati requisiti di sicurezza ma non un ampio raggio d'azione, come ad esempio sistemi di porte con chiusura elettronica e sistemi di pagamento. I sistemi a medio raggio scrivono e leggono fino alla distanza di 1 metro, tutti quelli con portata superiore a 1 metro sono classificati come a lungo raggio con distanze solitamente comprese tra i 5-15 metri. I tag per questi sistemi sono semiattivi o attivi.

Classificazione in base alla frequenza: Le frequenze utilizzate spaziano tra valori che vanno da 135 kHz (onde lunghe) a 5.8 GHz (micro onde), esistono comunque delle norme ben precise che stabiliscono le bande di trasmissioni utilizzabili visto che la loro disponibilità dipende dai vari stati. Le frequenze utilizzate più standard sono:

- 125/134 kHz
- 13,56 MHz
- 868/915 MHz UHF *Ultra high frequency* (UHF)
- maggiore di 2,4 GHz *Ultrawide band*

I tag a 125 kHz ed a 13,56 MHz sono previsti dalle norme ISO come passivi, mentre per i tag RFID UHF e Ultrawide band esistono attivi, semiattivi e passivi.

3.3 Campo di applicazione

I sistemi RFID permettono di superare i limiti imposti dai sistemi di identificazione attualmente utilizzati, (codici a barre, bande magnetiche e smart card) dovuti all'utilizzo in ambienti industriali ostili, bui, sporchi e con umidità e temperature elevate. Si tratta dunque di dispositivi molto flessibili con elevate potenzialità di applicazione. Inoltre l'assenza di contatto fra lettore, e ricevitore ne permette l'utilizzo nel caso in cui non possa esserci contatto diretto con l'oggetto da identificare. Le dimensioni, le forme e i rivestimenti possono essere

di svariato tipo in base alle varie necessità.

Negli ultimi anni la procedura di riconoscimento automatico ha suscitato molto interesse e si sta sviluppando in ogni settore industriale da quello di acquisto e distribuzione di servizi logistici a quello industriale, manifatturiero e metalmeccanico.

Riportiamo una sintesi delle applicazioni delle tecnologie RFID:

Logistica:

- Stoccaggio e spedizione delle merci.
- Identificazione pallets, stazioni di lavoro.
- Magazzini automatici.

Gestione:

- Gestione, tracciabilità ed inventario documenti (biblioteche, archivi, studi notarili, tribunali, etc.)
- Gestione movimentazione item

Sanità e farmaceutico:

- Controllo somministrazione farmaci
- Controllo flusso materiali nelle sale chirurgiche
- Gestione cartelle cliniche
- Monitoraggio e controllo pazienti

Industriale:

- Robotica e automazione
- Controllo nei processi di produzione
- Gestione dati nel controllo qualità
- Identificazione utensili nelle macchine automatiche

Automotive:

- Immobilizzatori per auto, motocicli, barche con transponder implementato nella chiave di accensione

- Sistemi di sicurezza
- Controllo del parco auto e loro manutenzione
- Localizzazione di automezzi
- Sistemi di pagamento pedaggi

Anticontraffazione:

- Etichette con transponder implementato contro la contraffazione di beni di Lusso (abbigliamento, borse, cinture, liquori, profumi...)
- Identificazione passaporti e/o documenti personali
- Certificati di origine

Agro-alimentare, animali e piante:

- Identificazione animali con transponder iniettato (normativa ISO 11784/11785)
- Controllo filiera e qualità agro-alimentare
- Registrazione e controllo caratteristiche di origine
- Protezione prodotti tipici, anti contraffazione
- Identificazione univoca animali e piante

Rilevazione transiti/presenze e Ticketing:

- Sistemi di controllo a mani libere
- Apertura porte, cancelli, serrande garage
- Sistemi di controllo accessi
- Gestione parcheggi

In questo capitolo è stato analizzato il funzionamento e la composizione generare dei sistemi RFID, nel capitolo seguente analizzeremo in dettaglio le caratteristiche del sistema RFID utilizzato nel presente lavoro tesi.

Capitolo 4

Funzionamento del sistema di test

In questo capitolo è descritto il funzionamento del sistema di test ed è analizzata l'architettura di rete del sistema RFID utilizzato in questo progetto descrivendo le caratteristiche delle codifiche e dei protocolli che sono poi stati implementati in VHDL per la realizzazione del sistema digitale.

4.1 Requisiti del sistema di test

Nel capitolo 3 sono stati descritti i componenti che formano un sistema RFID cioè il *transponder* o *tag*, il lettore e il sistema di elaborazione dati.

In particolare il lettore del sistema RFID utilizzato si compone dei seguenti elementi:

1. Un antenna esterna orientabile;
2. un modulatore/demodulatore di frequenza;
3. un alimentatore.

Il contenitore che racchiude il modulatore e l'alimentatore viene identificato con il termine di boa, la Fig. 4.1 rappresenta uno schema di tutto il sistema.

Il lettore non fa altro che modulare i dati che riceve dal sistema informativo per trasmetterli al tag e demodulare la risposta del tag per inviarla al sistema informativo. Da questo punto di vista si comporta come un *bridge* permettendo la trasmissione di dati che viaggiano su mezzi di trasmissione differenti come l'aria e il cavo di rame.

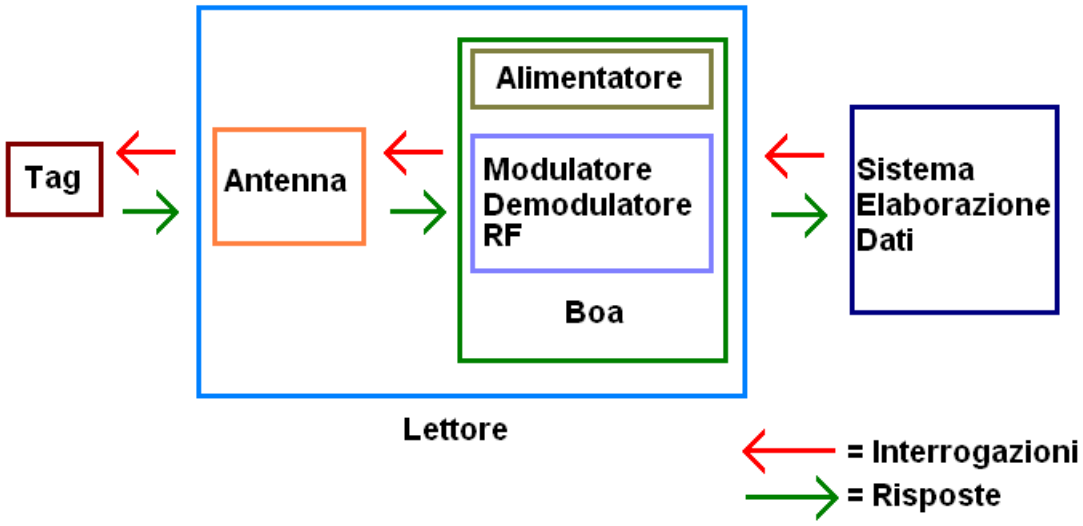


Figura 4.1: Scomposizione del sistema RFID.

Il sistema è utilizzato attualmente in applicazioni logistiche per lo stoccaggio e la spedizione merci e l'antenna è posizionata come in Fig .4.2.

Per via della distanza tag-antenna superiore ai 4.5 m, il tag utilizzato è del tipo semiattivo e può funzionare solamente se viene interrogato dal lettore. Il tag non può quindi mai comunicare in maniera autonoma con il lettore ma solo rispondere ai messaggi ricevuti correttamente. Il funzionamento del tag utilizzato è analizzato al paragrafo 4 di questo capitolo.

Tramite il sistema di elaborazioni dati e alcune *utility* specifiche di test è quindi possibile interrogare il tag e verificare il funzionamento del sistema. Il software di test permette di verificare se l'antenna è orientata correttamente tramite la visualizzazione sul monitor delle percentuali di efficienza e di copertura.

L'utilizzo di questo strumento è quindi indispensabile in fase di installazione e orientazione dell'antenna.

Per le caratteristiche dell'applicazione finale tale strumento risulta però scomodo e poco funzionale per i seguenti motivi:

- Il computer remoto di controllo spesso può essere lontano dal lettore decine o centinaia di chilometri;
- L'antenna del lettore è disposta in luogo difficilmente raggiungibile e nello specifico si tratta di circa 4.5 m di altezza.

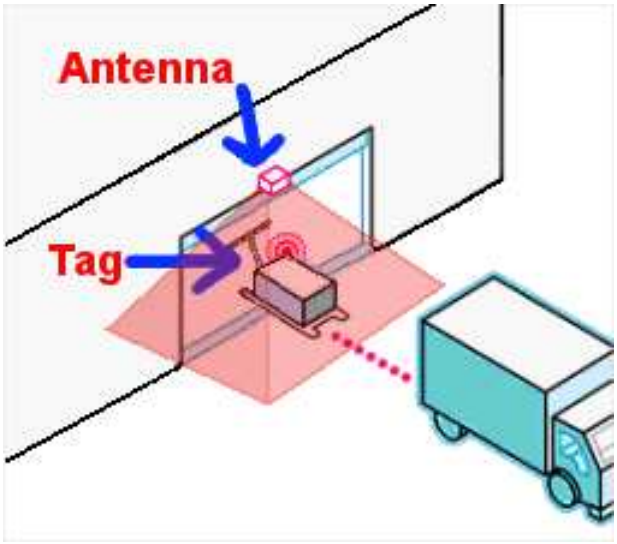


Figura 4.2: Esempio di possibile installazione dell'antenna.

L'alternativa più veloce consiste nel collegare al lettore un computer portatile e utilizzare un *utility* di test. Questa opzione risulta comunque svantaggiosa perché:

- Costosa: per via dell'utilizzo di computer portatili di piccole dimensioni ed elevata resistenza;
- Lenta: il collegamento dei cavi e il caricamento dei software fa perdere tempo;
- Poco praticabile da un solo operatore: i display lcd sono scarsamente visibili in condizioni ambientali con forte luce e un solo operatore impiega molto più tempo per la lettura dei dati con cui deve regolare l'antenna.

Ecco che nasce l'esigenza di disporre di uno strumento integrato nella boa che consenta una veloce verifica sul corretto posizionamento dell'antenna e di tutto il sistema RFID.

Un orientazione poco precisa dell'antenna può ridurre significativamente l'area di copertura a terra rendendo difficoltosa o anche errata la raccolta delle informazioni del tag.

Il sistema di test sviluppato deve poter essere integrato nel lettore originale utilizzando uno slot di espansione dedicato e permettere tramite un apposita interfaccia visiva semplificata una interpretazione immediata dei dati di funzionamento.

La scheda di test viene sistemata all'interno della boa tra il modulatore di frequenza e il sistema di acquisizione dati, come in Fig 4.3.

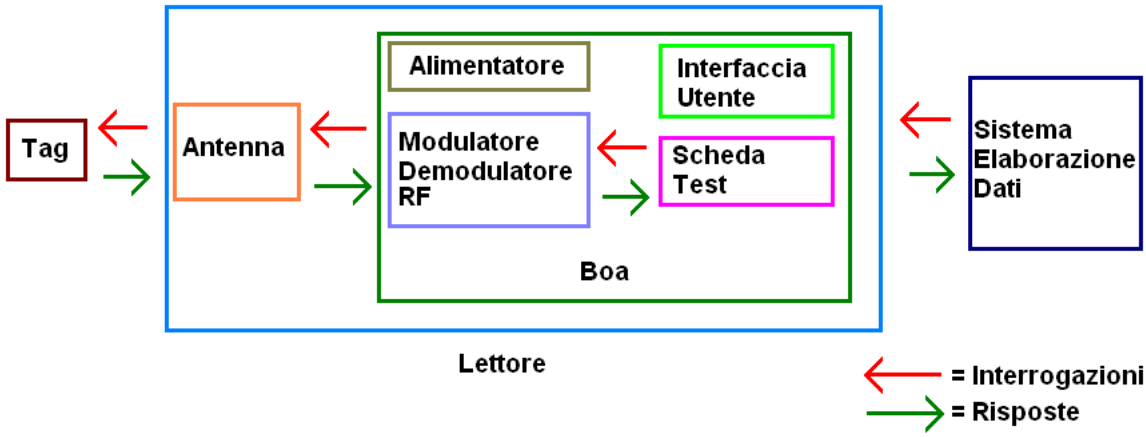


Figura 4.3: Posizione scheda di test e interfaccia utente.

4.2 Funzionamento del sistema di test

L'interfaccia visiva di cui è stata dotata la Boa è in grado di fornire informazioni sia in fase di funzionamento, che di installazione.

Si compone essenzialmente di 6 LED (*Light Emitting Diode*) ad alta efficienza il cui significato dipende dallo stato di funzionamento gestito completamente dalla scheda di test.

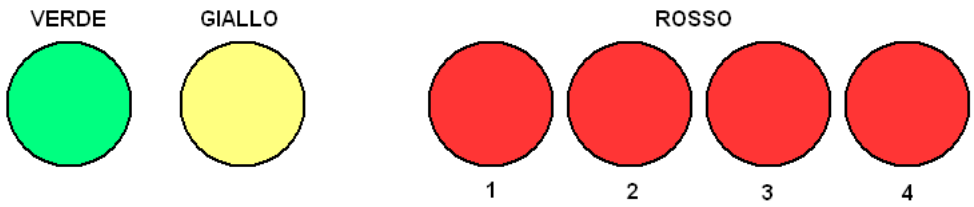


Figura 4.4: Interfaccia utente.

Il significato dei LED nello schema di Fig. 4.4 è il seguente:

- **LED verde:** Indica la presenza di alimentazione nella scheda di test, durante il funzionamento dovrà quindi essere sempre acceso;

- **LED giallo:** Indica in quale stato di funzionamento si trova la boa, se in modalità di test o in modalità normale;
- **LED rossi:** La loro interpretazione dipende dallo stato di funzionamento e possono indicare la percentuale di messaggi corretti ricevuti dal tag oppure la correttezza dei messaggi che transitano in linea.

I 2 possibili stati di funzionamento sono definiti come segue:

- **Modalità test:** Permette di verificare l'area di copertura posizionando un tag qualsiasi sotto l'antenna, mostrando sull'interfaccia utente la percentuale di riscontri positivi ed è utilizzata in fase di installazione.
- **Modalità normale:** Permette identificare il corretto funzionamento del sistema analizzando la validità dei messaggi inviati sulla rete in entrambe le direzioni ed è utilizzata in fase di funzionamento.

4.2.1 La modalità di test

In modalità di test il LED giallo lampeggia con frequenza costante di 0.5 Hz. La scheda di test in questo stato effettua continuamente una serie di interrogazioni e, se viene posto nell'area di copertura un tag qualsiasi, vengono accesi i LED rossi in modo da quantificare le risposte corrette rilevate. In questo modo è possibile effettuare una stima dell'area di copertura della boa indipendentemente dal sistema di gestione dati. Il significato dei LED rossi in questo caso è il seguente:

- **Tutti i LED rossi spenti:** Risposte corrette minori del 70%, fuori dall'area di copertura;
- **LED rosso 1 acceso, LED 2,3,4 spenti:** Risposte corrette comprese fra 70% e l'80%, comincia l'area di copertura;
- **LED rosso 1,2 accesi, LED 3,4 spenti:** Risposte corrette comprese fra il 80% e il 90%, area di copertura significativa;
- **LED rosso 1,2,3 accesi, led 4 spento:** Risposte corrette comprese fra il 90% e il 95%, area di copertura molto significativa;
- **LED rosso 1,2,3,4 accesi:** Tutti i LED rossi accesi: risposte corrette oltre il 95%, area di copertura piena.

4.2.2 La modalità normale

In modalità normale la scheda di test intercetta i messaggi inviati nella rete e verifica se vi sono errori nei dati. Il sistema realizzato è completamente trasparente ai dati in transito limitandosi solo al controllo e non influendo in nessun modo sul contenuto dei pacchetti. La gestione degli eventuali errori è a carico del sistema di elaborazione dati che controlla la boa.

Questa modalità è identificabile dallo stato del LED giallo che esegue un ciclo di 4 brevi lampeggi restando spento 5 secondi.

In questo caso i LED rossi 2 e 4 restano sempre spenti ed il significato degli altri LED è il seguente:

- **LED rosso 1 acceso per 2 s:** Transito di un messaggio corretto dal sistema elaborazioni dati verso il tag;
- **LED rosso 3 acceso per 2 s:** Transito un messaggio corretto dal tag al sistema di elaborazioni dati.

Il riconoscimento corretto di un tag prevede quindi l'accensione del LED rosso 1 per due secondi seguito dall'accensione del LED 3 per due secondi. Tutte le altre configurazioni non sono ammesse.

L'estensione del tempo di accensione dei LED rossi per 2 secondi permette all'operatore di rilevare sicuramente un evento di interrogazione da parte della boa. Un ciclo di interrogazione e risposta, dura in realtà meno di 10 millisecondi, in media; la durata effettiva dipende dalla lunghezza dei messaggi in transito e, in misura trascurabile, dalla distanza del tag dall'antenna.

In questa modalità se all'accensione del LED 1 non segue l'accensione del LED 3 si è certi che l'interrogazione del tag sia fallita, per uno o più dei seguenti motivi:

1. Il tag non è posizionato correttamente nella zona illuminata dall'antenna;
2. Il tag non funziona correttamente inviando messaggi corrotti;
3. Il tag non funziona perché la batteria è scarica.

Nel primo caso occorre spostare il tag in zona con maggior copertura e verificare la presenza di oggetti posti tra il tag e l'antenna. Nel secondo e terzo caso occorre sostituire il tag con uno funzionante.

Da notare che l'accensione singola del LED che segnala il passaggio di un messaggio corretto del tag non può mai avvenire in quanto il tag non può inviare in modo autonomo messaggi nella rete ma solo dopo essere stato interrogato.

4.2.3 Scelta della modalità di funzionamento

Quando il lettore viene alimentato tutti i LED si accendono, per consentire la verifica del corretto funzionamento dell'interfaccia visuale.

Dopo qualche secondo il LED verde resta acceso ad indicare la presenza dell'alimentazione e tutti gli altri LED si spengono mentre lo stato di funzionamento della boa sarà identificato tramite il LED giallo.

All'accensione il lettore parte sempre in modalità test e le uniche condizioni necessarie per il passaggio alla modalità normale sono le seguenti:

1. L'arrivo di un pacchetto dati dal sistema informativo in occasione di un transito;
2. L'attesa di un periodo di un ora in cui non viene rilevato la presenza di un tag.

Il passaggio alla modalità normale sarà quindi permanente e l'unico modo per ritornare alla modalità test consiste nel resettare manualmente la boa.

Durante l'installazione è quindi conveniente che il cavo dati non sia connesso onde evitare il cambio non previsto di modalità durante questa fase.

4.3 Funzionamento del tag

Il tag può essere inteso come l'unione di un certo numero di blocchi logici che realizzano diverse funzioni. In particolare l'apparato è suddiviso come nella Fig. 4.5 in:

- **Unità di controllo:** dove risiede l'intelligenza dell'apparato e da dove vengono controllate e gestite tutte le unità periferiche;
- **Interfaccia esterna:** il modulo che consente di informare l'utente in merito allo svolgersi delle operazioni, attraverso dispositivi luminosi ed acustici;
- **Modulo ricetrasmettitore radio:** che consente la gestione a livello analogico della trasmissione e della ricezione dei messaggi applicativi attraverso il canale radio;

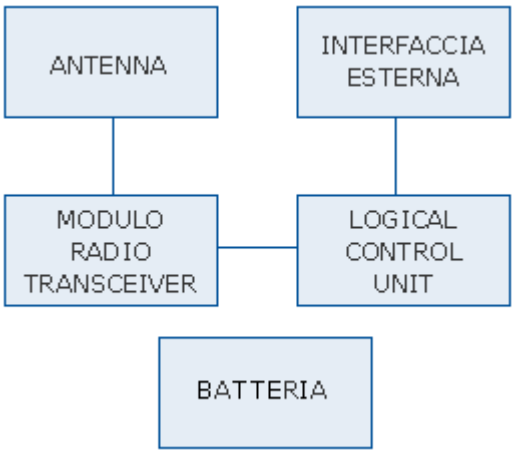


Figura 4.5: Struttura interna del tag.

- **Antenna:** che consente l'interfaccia dell'apparato col canale radio.

Possono identificarsi 3 stati principali in cui può trovarsi il tag:

- **SPENTO:** il tag si trova nello stato di SPENTO quando il lettore non riesce ad attivarlo, cioè fuori dall'area di copertura;
- **PRONTO:** il tag è nello stato di PRONTO ed in attesa di un'interrogazione dopo che il lettore è riuscito ad attivarlo;
- **RISPOSTA:** il tag si porta nello stato di RISPOSTA a seguito di un interrogazione corretta; durante il tempo di risposta del tag il lettore mantiene costante il livello di potenza trasmessa ed il tag la utilizza per trasmettere i dati in riflessione.

Il tag non può quindi attivarsi in modo autonomo ma è il sistema di gestione dati che si occupa di attivarlo rilevando il passaggio di un oggetto nella zona di identificazione.

4.4 L'architettura della rete

Nella prima parte di questo capitolo è stato analizzato il funzionamento complessivo del sistema RFID utilizzato e del sistema di test senza definire nulla sui meccanismi che permettono la trasmissione dei dati. Un analisi di basso livello sui protocolli utilizzati per l'invio dei dati è indispensabile per realizzare

il sistema di test.

Il sistema RFID utilizzato è riconducibile a una rete di telecomunicazione composta da nodi che comunicano tra di loro.

Nell'ambito delle telecomunicazioni due o più macchine possono comunicare tra loro rispettando norme che sono dette protocolli di rete. Un approccio molto utilizzato nel disegno e nello studio delle architetture di rete è quello della divisione in livelli. Ogni livello assolve solo ad alcuni compiti e fornisce servizi al livello soprastante il quale li utilizza per assolvere i propri compiti. In questo modo è possibile dividere i compiti tra i vari livelli semplificando il progetto.

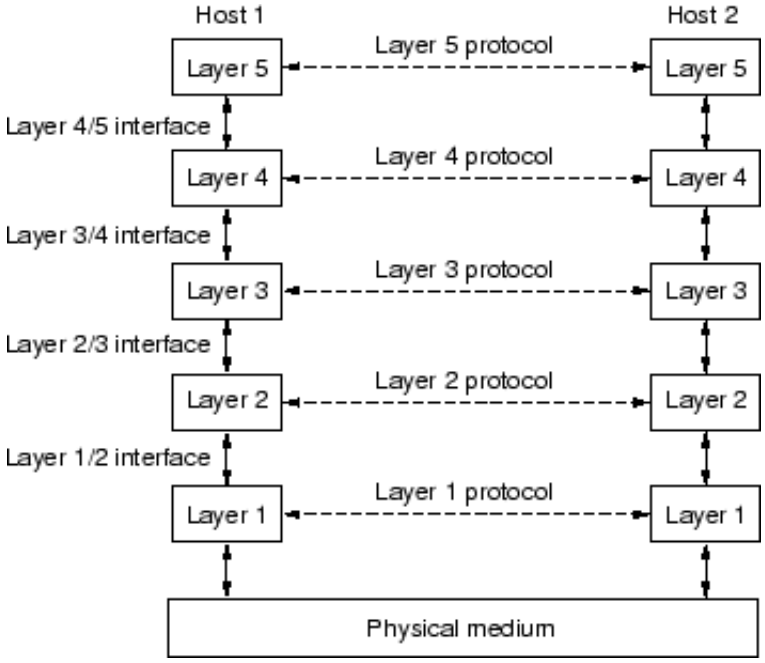


Figura 4.6: Il modello stratificato

La Fig. 4.6 mostra un esempio di modello a livelli dove la comunicazione a livello logico tra due macchine avviene tramite un livello e il suo pari sull'altra macchina anche se in realtà la comunicazione avviene al livello più basso o livello fisico. La comunicazione non è quindi diretta ma ha luogo attraversando i diversi livelli della propria macchina.

Concludendo, il livello i-esimo di una macchina comunica con il livello i-esimo dell'altra tramite il protocollo di livello i-esimo, come illustrato in Fig 4.7.

L'astrazione per livelli permette di separare un problema molto difficile, come il progetto di un intera rete, in problemi più semplici cioè il progetto di un livello.

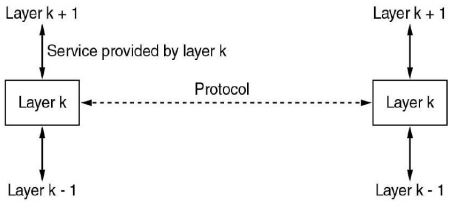


Figura 4.7: Comunicazione tra due livelli.

Non vi sono limiti al numero e alla complessità di un livello ma viene particolarmente utilizzato come riferimento il modello ISO/OSI. In particolare ISO/OSI è composto da un totale di 7 livelli che vanno dal livello fisico (quello del mezzo fisico, ossia del cavo o delle onde radio) fino al livello delle applicazioni attraverso cui si realizza la comunicazione di alto livello. Un'architettura di rete infatti non specifica i servizi e i protocolli da usare in ogni livello, ma indica solo di cosa si deve occupare un livello.

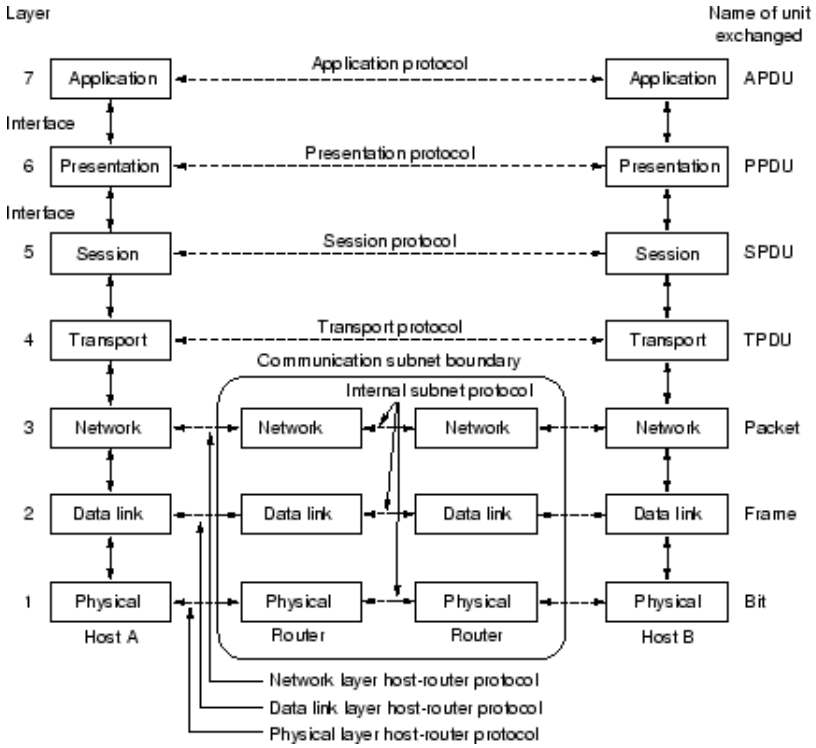


Figura 4.8: pila protocollare ISO/OSI.

La Fig. 4.8 mostra la rappresentazione del modello ISO/OSI che è analizzato più dettaglio nell'appendice B. Di seguito tratterò solo i livelli che sono stati utilizzati per la realizzazione del sistema di test.

4.5 Il livello fisico

Lo scopo di questo livello è trasmettere dati non strutturati attraverso un canale di comunicazione, traducendoli in un segnale elettrico di forma e intensità adeguate. I criteri di progettazione coinvolgono questioni meccaniche, elettriche e di temporizzazione che dipendono dal mezzo fisico attraverso cui il segnale è trasmesso.

Le informazioni digitali vengono trasmesse come variazioni di una grandezza fisica (tensione, corrente, intensità luminosa, campo elettromagnetico) nel tempo e si possono descrivere matematicamente come una funzione del tempo F(t). Nel caso del nostro sistema di test i mezzi impiegati dal livello fisico del protocollo sono l'aria ed il cavo di rame.

Il lettore comunica con il tag utilizzando segnali in radiofrequenza e con il sistema di elaborazione dati utilizzando segnali elettrici.

Per realizzare il sistema di test descritto in questa tesi è stato necessario analizzare il flusso dati in ingresso o in uscita dal modulatore/demodulatore. Non è stato necessario conoscere i dettagli di funzionamento della sezione a radiofrequenza del modem e di conseguenza non si riportano qui le informazioni riguardanti le caratteristiche del segnale radio (frequenza di trasmissione, tipo di modulazione, etc.). Pensiamo al modem semplicemente come ad una "scatola nera" che invia al tag il flusso dati entrante proveniente dal sistema di elaborazione dati e viceversa invia al sistema di elaborazione i dati ricevuti dal tag.

I dati in uscita dalla boa sono quindi modulati con la frequenza richiesta e inviati all'antenna mentre i dati in ingresso alla boa sono demodulati e inviati sulla linea dati.

La codifica

Visto l'utilizzo di un canale di trasmissione non ideale è necessario cercare di ridurre il più possibile i problemi causati dal rumore.

Lo schema a blocchi nella Fig. 4.9 descrive un sistema di comunicazione digitale che è composto da tre blocchi:

- Trasmettitore;
- Canale di trasmissione;
- Ricevitore.

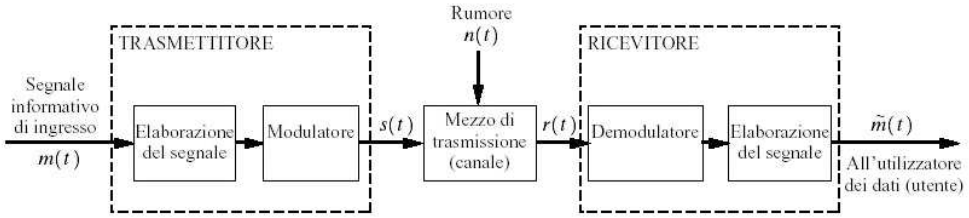


Figura 4.9: Canale di trasmissione non ideale.

Il tag e il lettore contengono entrambi sia un trasmettitore che un ricevitore e per il trasferimento dei dati utilizzano il seguente processo:

- Nel blocco trasmettitore l'informazione da trasmettere viene codificata e modulata (traslazione in frequenza del segnale);
- Il segnale modulato attraversa il canale di trasmissione il quale aggiunge inevitabilmente un segnale di disturbo (rumore);
- Il segnale viene ricevuto dal lettore che lo demodula riportandolo in banda base così da poter essere decodificato.

I simboli binari possono essere allora codificati in vari modi per cercare di ovviare ai disturbi introdotti. Risulta importante capire dove inizia e finisce un bit soprattutto nei lunghi flussi di bit 1 o 0 senza usare un clock esterno.

Trasmettere i bit utilizzando la stessa codifica binaria[1] del flusso di dati può non risultare la scelta migliore, in quanto i clock dei due dispositivi difficilmente sono sincronizzati e una lunga sequenza di 0 e di 1 potrebbe non essere interpretata correttamente.

La codifica Manchester introduce un transizione per ogni bit, un 1 logico si ottiene con un fronte di discesa mentre uno 0 logico si ottiene con un fronte di salita, in Fig. 4.10 è mostrata la differenza tra la codifica binaria e la codifica Manchester [6].

L'inserimento di una transizione a metà bit permette a chi riceve di sincronizzarsi con il clock di trasmissione ed evita di avere lunghi periodi con assenza di transizioni. L'utilizzo di 2 simboli per la rappresentazione di un bit ha come unico difetto la richiesta del doppio di banda ma ha il vantaggio di poter essere implementata utilizzando pochissime risorse di calcolo. Esistono codifiche più

[1]Detta anche NRZ, Non Return to Zero

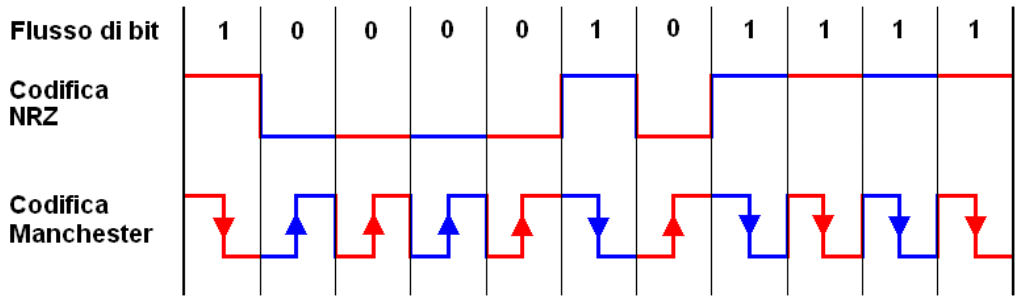


Figura 4.10: Codifica binaria e Manchester.

efficienti nel ridurre il tasso di errore di trasmissione ma il loro impiego richiede un maggior utilizzo di risorse. La codifica Manchester rappresenta un buon compromesso tra l'esigenza di semplicità del sistema e la necessità di efficacia nella protezione dagli errori di trasmissione.

Le informazioni vengono quindi trasmesse serialmente e in modo asincrono secondo la codifica Manchester dal lettore al tag con una velocità di segnalazione o *bit rate* di 478 kbps e dal tag al lettore con un *bit rate* di 728 kbps.

4.6 Il protocollo di comunicazione

Analizzeremo ora il secondo livello o strato di collegamento della pila protocollare ISO/OSI.

Principalmente il livello di collegamento permette il trasferimento affidabile dei dati attraverso il livello fisico. Invia $frame$ di dati con la necessaria sincronizzazione ed effettua un controllo degli errori e delle perdite di segnale. Tutto ciò consente di far apparire al livello superiore il mezzo fisico come una linea di trasmissione esente da errori di trasmissione. Infatti è questo strato che si occupa di trattare i pacchetti danneggiati.

In questo caso il protocollo di comunicazione usato è HDLC (High-Level Data Link Control) e si tratta di un protocollo orientato al bit e sviluppato dall'ISO. Lo standard corrente per HDLC è stabilito nella norma ISO 13239, che ha sostituito tutte le precedenti versioni.

Il formato della trama HDLC è riportato in Fig. 4.11 e risulta composta da tre parti principali: un *header*, un campo *information* a lunghezza variabile e un *trailer*.

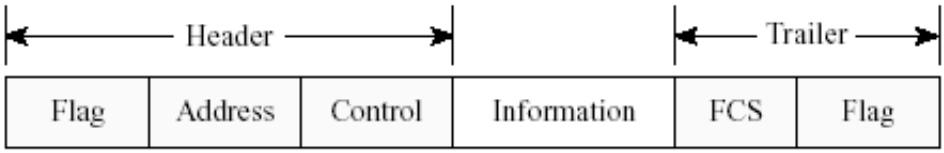


Figura 4.11: La trama HDLC.

Il significato dei vari campi, di cui è composta la trama è il seguente:

- **Flag:** identifica l'inizio e la fine del $frame$, si tratta della sequenza 01111110 oppure, in esadecimale 7E, e viene trasmesso di continuo sulle linee *idlee* punto-punto;
- **Address:** contiene l'indirizzo del terminale di destinazione, utilizzato su linee con terminali multipli per identificarli;
- **Control:** contiene i numeri di sequenza o *acknowledgements*, e determina il significato dei dati trasportati;
- **Information:** contiene qualsiasi informazione di lunghezza arbitraria;
- **FCS (*Frame Check Sequence*):** contiene il codice per il controllo del $frame$, è di dimensione fissa a 16 bit ed è calcolato sui tutti i dati trasportati esclusi i *flag*;

Il $frame$ minimo risulta composto da 32 bit con nessun bit di dati e con i *flag* esclusi.

Il protocollo HDLC contrassegna quindi l'inizio e la fine del *frame* con un *flag* che viene trasmesso anche durante tutto il tempo in cui la linea di connessione è inattiva in modo che sorgente e ricevitore possano mantenere sincronizzati i propri segnali di clock. Visto che la sequenza che identifica un $flag$ potrebbe apparire ovunque all'interno del $frame$, occorre prevedere un metodo per evitare di interpretare il dato come flag di chiusura. La tecnica usata a questo scopo si chiama ad "interposizione di bit" o *bit stuffing*.

Il *bit stuffing* in HDLC funziona in questo modo: ogni volta che la sorgente invia cinque valori 1 consecutivi all'interno del corpo del messaggio senza considerare i *flag* di apertura e chiusura, si inserisce un valore 0 prima di trasmettere

il bit successivo.

In ricezione nel caso arrivino cinque valori 1 consecutivi il ricevitore dovrà analizzare il bit successivo all'ultimo ricevuto:

- Se vi è un valore 0, significa che la sorgente ha eseguito il *bit stuffing* e deve essere eliminato;
- Se si tratta di un valore 1, allora si può trattare del *flag* di chiusura, oppure che nel flusso di bit è stato introdotto un errore.

Bisogna ancora attendere il bit successivo, per poter distinguere tra questi due possibili casi:

- Se vi è un valore 0 e, quindi gli ultimi 8 bit ricevuti sono stati 01111110, allora è stato ricevuto il *flag* di chiusura;
- Se si osserva un valore 1 e quindi gli ultimi 8 bit ricevuti sono stati 01111111, allora si è verificato un errore e l'intero *frame* viene scartato.

In questo caso il ricevitore deve attendere il successivo 01111110 prima di poter iniziare a ricevere di nuovo. É possibile che il ricevitore fallisca nella ricezione di due $frame$ consecutivi nel caso in cui invece del $flag$ di chiusura del $frame$ corrente venga riconosciuto il $flag$ di apertura del $frame$ successivo. Da notare che il ricevitore dopo aver riconosciuto un $flag$ di apertura resta in attesa di dati che siano diversi dal $flag$ stesso. Nel momento in cui arriva quindi una sequenza valida diversa dal $flag$ viene considerata come l'inizio dei dati.

Continuando con l'esempio precedente il secondo $frame$ non è stato riconosciuto in quanto il suo $flag$ di apertura è stato utilizzato al posto di quello di chiusura del $frame$ precedente. Al ricevitore arriva poi il $flag$ di chiusura del secondo $frame$ e resta in attesa di un altro $flag$ riconoscendo così correttamente il terzo $frame$.

Per fare in modo che il ricevitore resti sincronizzato è utile che in linea siano trasmessi quindi più $flag$ e in questo caso non è detto che due $frame$ debbano essere consecutivi ma risultano distanziati da un certo numero di $flag$ che aiutano a mantenere i ricevitori sincronizzati.

Naturalmente esistono ancora casi in cui errori sui $frame$ non vengano rilevati, ad esempio quando un errore genera una sequenza di fine $frame$, ma si tratta di eventi facilmente verificabili utilizzando il campo FCS contenuto nel $frame$. La probabilità che il FCS ricavato dal $frame$ troncato coincida con quello calcolato per il $frame$ completo è molto piccola.

Una caratteristica derivante dall'uso del *bit stuffing* è che la dimensione del $frame$ dipende dai dati che vengono inviati, non è infatti possibile che tutti i $frame$ abbiano la stessa dimensione perché i dati sono totalmente arbitrari. Questo comporta per lunghe sequenze di valori 1 un tempo di trasmissione maggiore delle corrispondenti sequenze con valori 0.

4.7 Verifica dei dati trasmessi

Quando si trasmettono dati usando segnali radio è inevitabile che all'interno del canale di trasmissione ci sia del rumore che può condurre ad errori in ricezione. Gli errori possono coinvolgere uno o più bit del pacchetto dati e devono essere individuati in modo che la trasmissione possa ritenersi affidabile. Ad esempio a seguito della sovrapposizione del rumore con il segnale trasmesso si può verificare che bit inviati dalla sorgente come 0 vengano ricevuti come 1 o viceversa.

Gli errori si possono suddividere a seconda del numero e della posizione dei bit coinvolti nell'errore in:

- **Single-bit:** errori su un singolo bit;
- **Multiple-bit:** errori che coinvolgono più bit ma non in sequenza;
- **Burst:** errori a raffica che coinvolgono sequenze di più bit.

Gli errori più frequenti sono quelli *single-bit* e *multiple-bit* mentre quelli di tipo *burst* sono i meno probabili. Questo accade di frequente usando la codifica binaria, mentre l'uso della codifica Manchester migliora l'immunità ai disturbi ma da sola non è in grado di garantire la correttezza dei dati ricevuti. Per tale ragione nei livelli superiori si utilizzano codici con un certo grado ridondanza che permettono la rilevazione e la correzione degli errori.

L'idea consiste nell'aggiungere a dati da inviare una certa quantità di informazioni supplementari che verranno eliminate appena verificata la correttezza dei dati.

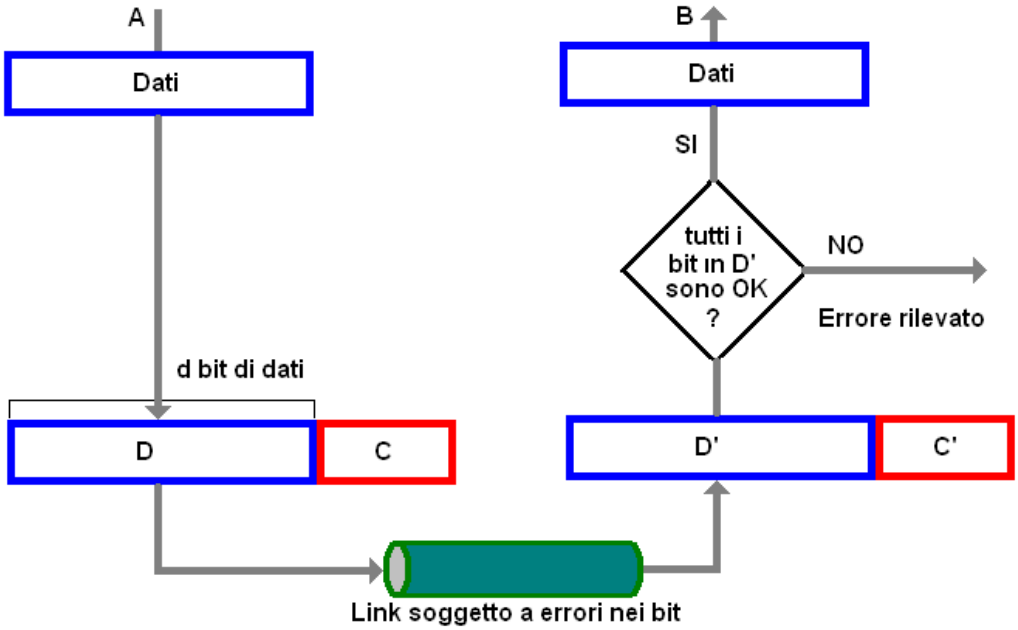


Figura 4.12: Trasmissione dati in un canale soggetto a errori.

La Fig. 4.12 rappresenta lo scenario del caso considerato: il nodo A spedisce i dati D che devono essere protetti dagli errori nei bit, aggiungendo i bit di ricerca e correzione dell'errore C. Il pacchetto D+C viaggia dunque nel canale dati e raggiunge il nodo B di destinazione. Il nodo B riceve una sequenza di bit D'+C', da notare che i dati ricevuti possono essere diversi dagli originali.

Il nodo B deve ora determinare se il dato D' è identico all'originale considerando il fatto che esso ha ricevuto solo D' e C'.

Le tecniche di ricerca e di correzione degli errori permettono al nodo B di rilevare, ma non sempre, se si sono verificati errori nei bit. Vale a dire che con l'adozione dei bit di ricerca degli errori c'è ancora la possibilità che ci siano errori dei bit non rilevati. Bisognerà quindi scegliere uno schema di ricerca degli errori che renda piccola la probabilità di queste evenienze.

In genere le tecniche più sofisticate di ricerca e correzione degli errori comportano grandi ridondanze e necessitano sia di più calcoli per l'elaborazione che più tempo per la trasmissione di una maggiore quantità di dati. Normalmente sono

utilizzate varie tecniche, dai semplici controlli di parità e *checksum*, ai controlli a ridondanza ciclica (CRC o *Cyclic Redundancy Check*).

4.7.1 Controlli a ridondanza ciclica

Descriviamo alcuni semplici concetti che stanno alla base teorica e dell'implementazione dei controlli a ridondanza ciclica, necessari per calcolare il CRC contenuto nel $frame$ HDLC.

I codici CRC sono noti come codici polinomiali, poichè è possibile vedere la stringa dei bit che deve essere spedita come un polinomio i cui coefficienti sono i valori 0 e 1 nella stringa, con le operazioni sulla stringa di bit interpretate come aritmetica polinomiale.

Si basa sulla definizione dei seguenti polinomi:

- **D(x):** polinomio messaggio in cui i coefficienti della variabile x sono i valori dei bit da trasmettere;
- **G(x):** polinomio di grado r, generatore del codice noto sia al trasmettitore che al ricevitore;

Tutti i calcoli CRC sono eseguiti in aritmetica modulo 2 senza riporti in addizioni e sottrazione. Questo significa che addizione e sottrazione sono operazioni identiche e che entrambe equivalgono ad un'operazione di OR esclusivo bit a bit degli operandi.

Il trasmettitore sceglierà r bit addizionali $C(x)$ e li appenderà a $D(x)$ come nello schema di Fig. 4.12, in modo che il polinomio $D(x) + C(x)$ sia esattamente divisibile per $G(x)$ utilizzando l'aritmetica in modulo 2.

Per il calcolo del polinomio $C(x)$ da accodare ai dati $D(x)$ il trasmettitore utilizza la seguente formula:

$$\frac{D(x) \cdot x^r}{G(x)} = Q(x) + \frac{C(x)}{G(x)}$$

Dal calcolo risultano un polimonio quoziente $Q(x)$ e un polinomio resto $C(x)$, che è proprio il CRC da inviare. Da notare che la divisione in modulo 2 si riconduce ad operazioni di shift e XOR, che sono realizzabili molto semplicemente in hardware utilizzando registri a scorrimento e porte logiche XOR.

Il polinomio da trasmettere è quindi:

$$T(x) = D(x) \cdot x^r \oplus C(X)$$

Il polinomio $T(x)$ è esattamente divisibile per $G(x)$ ed ha anche la proprietà di consentire l'immediata ricostruzione di $D(x)$, dal momento che $C(x)$ ha grado r-1 e va ad occupare i bit meno significativi di $T(x)$ senza alterare i bit di $D(x)$. Il polinomio ricevuto può essere espresso come:

$$T'(x) = T(x) + E(x)$$

Qualora durante la comunicazione avvengano degli errori, $E(x)$ rappresenta il polinomio errore i cui coefficienti sono 1 in corrispondenza delle posizioni di $T(x)$ in cui si sono verificati errori.

Il ricevitore esegue la seguente operazione:

$$\frac{T'(x)}{G(x)} = \frac{T(x)}{G(x)} + \frac{E(x)}{G(x)}$$

Dove per definizione la primo addendo da resto zero e il secondo da resto zero solo se non si sono verificati errori. In realtà la costruzione del codice si riconduce a trovare un $G(x)$ tale da massimizzare la probabilità che il resto della divisione $E(x)/G(x)$ sia diverso da zero per i polinomi di errore interessanti, cioè per i casi di errore ritenuti più probabili.

Si possono dimostrare le seguenti proprietà di rivelazione degli errori:

- **Errori singoli:**$E(x) = x^j$ Rilevabili se G(x) ha il termine noto cioè il bit meno significativo diverso da 0;
- **Errori doppi:** $E(x) = x^{k+j} + x^j = x^j \cdot (x^k + 1)$ In questo caso non esistono regole generali si devono costruire per tentativi dei G(x) opportuni, ad esempio $G(x) = x^{15} + x^{14} + 1$ rileva tutti gli errori doppi fino a k = 32768;
- **Errori dispari:** $E(x) = x^j \cdot (x^i + x^k + 1)$ Rilevabili se $G(x)$ contiene fra i suoi fattori primi il binomio $(x + 1)$.
- **Errori a burst:** Rilevabili se hanno grado minore di r, cioè se la loro lunghezza è inferiore a quella di $G(x)$.

Esistono svariati tipi di polinomi CRC in particolare analizziamo le caratteristiche del polinomio utilizzato, chiamato CRC-CCITT in quanto standardizzato dal *Consultative Committee for International Telephony and Telegraphy*[2] che rappresenta l'organismo che definisce gli standard di comunicazione.

Il polinomio è il seguente: $E(x) = x^{16} + x^{12} + x^5 + 1$ e in base a quanto detto in precedenza questo polinomio consente di rilevare:

[2]Il nome della organizzazione oggi è International Telecommunication Union abbreviato in ITU

- errori singoli e doppi;
- errori di numero dispari di bit;
- errori a burst di lunghezza ≤ 16.

Supponendo che una raffica di lunghezza superiore a r+1 bit sia rilevata con probabilità di $1 - 0,7^r$ allora si possono rilevare:

- 99,997% di burst lunghi 17;
- 99,998% di burst lunghi 18.

In Fig. 4.13 è mostrato lo schema di come sia realizzabile in hardware questo CRC.

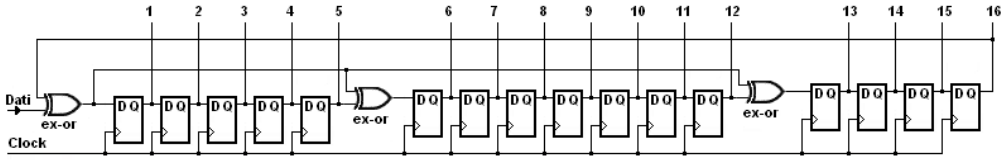


Figura 4.13: Realizzazione del CRC-CCITT con registi e porte XOR.

Esempio calcolo CRC

La divisione tra polinomi si esegue in maniera molto simile alla divisione tra numeri usando le stringhe dei coefficienti, esistono solo tre differenze:

- Per ottenere il resto si devono aggiungere alla stringa del polinomio dividendo tanti bit a 0 a destra quanti sono i bit del resto, coincidenti con il grado del polinomio CRC;
- nel corso della divisione si confronta il divisore con dividendi della stessa dimensione. Il confronto produce un 1 nella sequenza del risultato anche quando il dividendo non è strettamente maggiore del divisore: basta che siano dello stesso ordine, cioè che il bit più significativo del dividendo sia 1;
- la sottrazione per ridurre il dividendo va eseguita senza riporti, quindi si riduce ad un'operazione di OR esclusivo.

Da notare che calcolare il risultato è completamente inutile in quanto ci interessa solo il resto.

Il seguente esempio mostra la costruzione del codice CRC usando:

- Dati da inviare: D(x) = 101101110
- Polinomio CRC: G(x) = $x^3 + x + 1$ = 1011 con grado r = 3.

Visto che il polinomio CRC è di grado 3, al polinomio $D(x)$ vanno aggiunti 3 zeri in coda, poi si procede con la divisione intera come in Fig. 4.14: Il resto

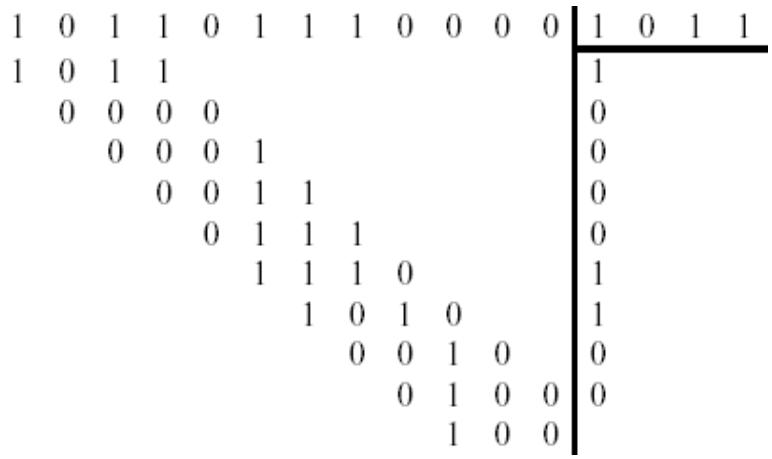


Figura 4.14: Calcolo del CRC.

dell'operazione è 100 quindi il polimonio da trasmettere è $T(x)$= 101101110 100

La Fig. 4.15 mostra la verifica sui dati ricevuti correttamente, il resto della divisione risulta essere zero.

Nella Fig. 4.16 è rappresentato un esempio nel caso si verifichi un errore in uno dei bit ricevuti. Il bit evidenziato in rosso è cambiato e il resto dell'operazione risulta essere diverso da zero, quindi l'errore è rilevato.

Nell'esempio di Fig. 4.17 è mostrato il caso in cui si verifichi un errore su un $burst$ di 3 bit. Anche in questo caso il resto risulta essere diverso da zero quindi l'errore è rilevato.

Tutte le codifiche e i protocolli analizzati in questo capitolo sono state implementate in VHDL, nel capitolo successivo è analizzato lo schema a blocchi del sistema di test realizzato.

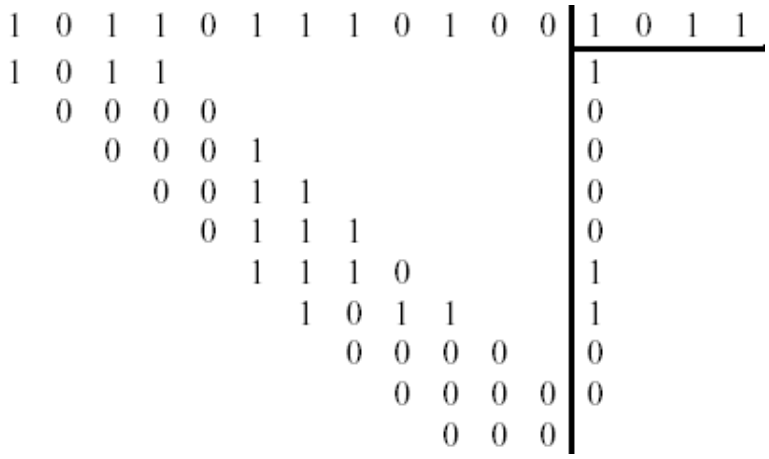


Figura 4.15: Verifica di corretta ricezione.

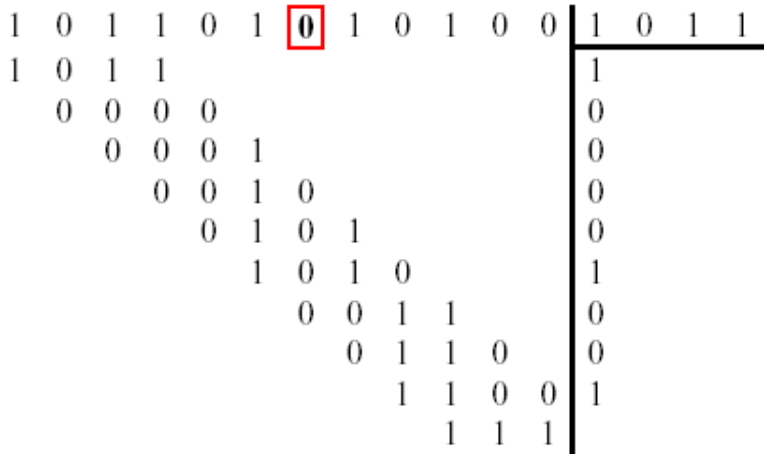


Figura 4.16: Rilevazione errore singolo.

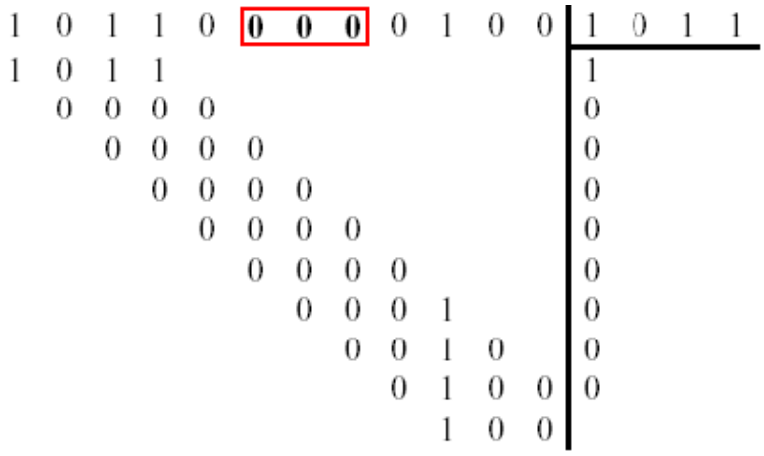


Figura 4.17: Rilevazione errore su 3 bit.

Capitolo 5

Architettura del sistema di test

In questo capitolo è affrontata tutta la descrizione del sistema digitale implementato in VHDL, con la descrizione dei singoli blocchi e delle macchine a stati che li costituiscono.

5.1 Analisi dello schema a blocchi

Il VHDL permette la divisione di un progetto complesso in più blocchi funzionali in modo da permettere il riutilizzo di blocchi che implementano funzioni comuni nelle varie parti del sistema.

Lo schema a blocchi di Fig. 5.1 è stato ottenuto tramite il software Aldec Active-HDL 7.2, dall'analisi del codice sorgente VHDL è possibile ottenere lo schematico e viceversa.
I blocchi possono essere così divisi in 5 parti principali:

- **Trasmissione:** evidenziato in verde, consente la creazione e l'invio di $frame$ HDLC;
- **Ricezione:** evidenziati in blu, consentono la decodifica dei messaggi in arrivo dal canale radio e dalla linea dati;
- **Unità di controllo:** evidenziate in rosso, gestiscono le interazioni tra i blocchi;
- **Controllo interfaccia visuale:** evidenziato in viola, è il modulo che gestisce l'interfaccia visuale semplificata;
- **Supporto:** evidenziati in giallo, eseguono semplici funzioni di supporto condivise da tutti i blocchi.

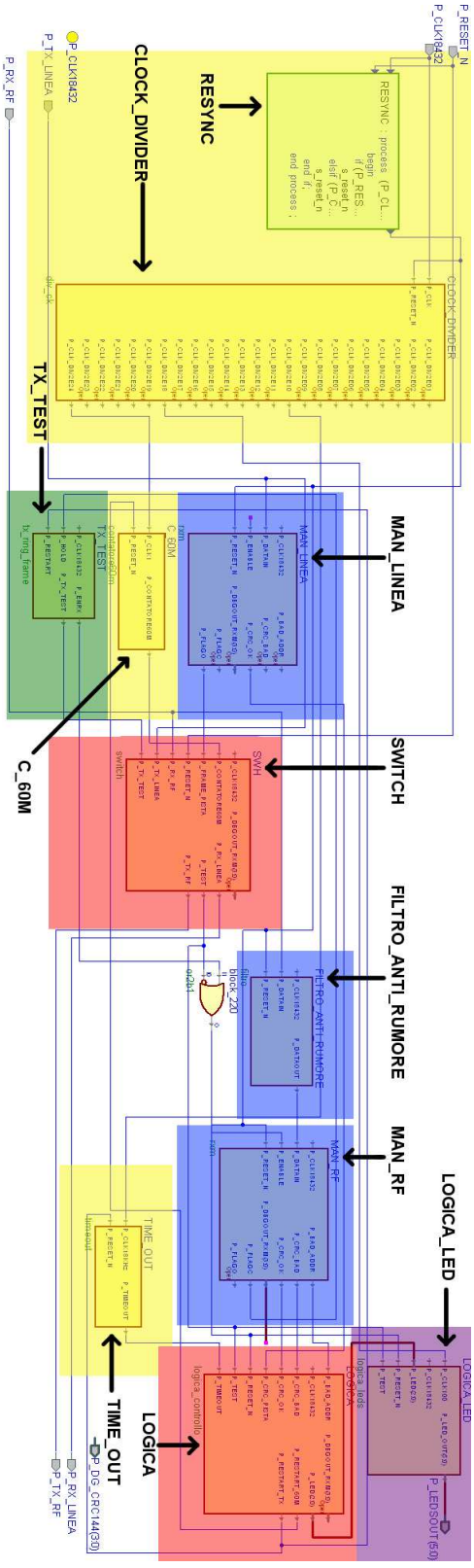


Figura 5.1: Schema a blocchi del sistema di test.

Le porte di ingresso al blocco sono rappresentate sulla parte sinistra mentre le porte di uscita sulla parte destra del blocco. Come si può vedere non è detto che tutte le porte di ingresso o di uscita ad un blocco siano state effettivamente utilizzate, questo perché ogni blocco può implementare svariate funzioni logiche e può essere riutilizzato.

L'entità rappresenta quindi un modello che può essere istanziato quante volte si vuole e con diversi parametri di funzionamento.

Certi blocchi quindi sono sviluppati per uso generale ed è possibile utilizzare i diversi segnali di cui si ha bisogno in quella specifica parte del circuito trascurando quelli non necessari, che sullo schema sono identificati con il termine *open*. É compito del sintetizzatore rilevare parti di codice non utilizzate ed ottimizzarle se possibile.

Osservando lo schema si può notare che sui blocchi sono presenti due nomenclature:

1. Quella in alto rappresenta il nome dell'istanziazione di quel tipo di entità;
2. Quella in basso rappresenta il nome dell'entità utilizzata che in genere coincide con il nome del file.

Ad esempio nella sezione di ricezione i blocchi MAN_LINEA e MAN_RF rappresentano due istanziazioni della stessa entità richiamata con parametri diversi per permetterne l'adattamento ai diversi *bit rate* di trasmissione e ricezione. Già in piccoli progetti il riutilizzo del codice porta a una riduzione del tempo impiegato. Nei paragrafi seguenti sono analizzate le caratteristiche di ogni singolo blocco descrivendone le funzioni svolte e se possibile una analisi della macchina a stati che lo costituisce.

5.2 La sezione di trasmissione

La sezione di trasmissione è composta dal solo blocco trasmettitore TX_TEST, la cui funzione è quella di trasmettere il messaggio di test per interrogare il tag.

La trasmissione può essere divisa principalmente nei seguenti passaggi:

1. Invio del segnale di risveglio del tag;
2. Invio seriale in codifica Manchester del $frame$ HDLC;

3. Mantenimento della potenza alta in uscita dall'antenna della boa fino alla ricezione di una risposta corretta dal tag.

La procedura per l'invio di un $frame$ HDLC è schematizzata in Fig. 5.2 tramite la descrizione del diagramma degli stati della macchina sequenziale che realizza i tre passaggi precedenti.

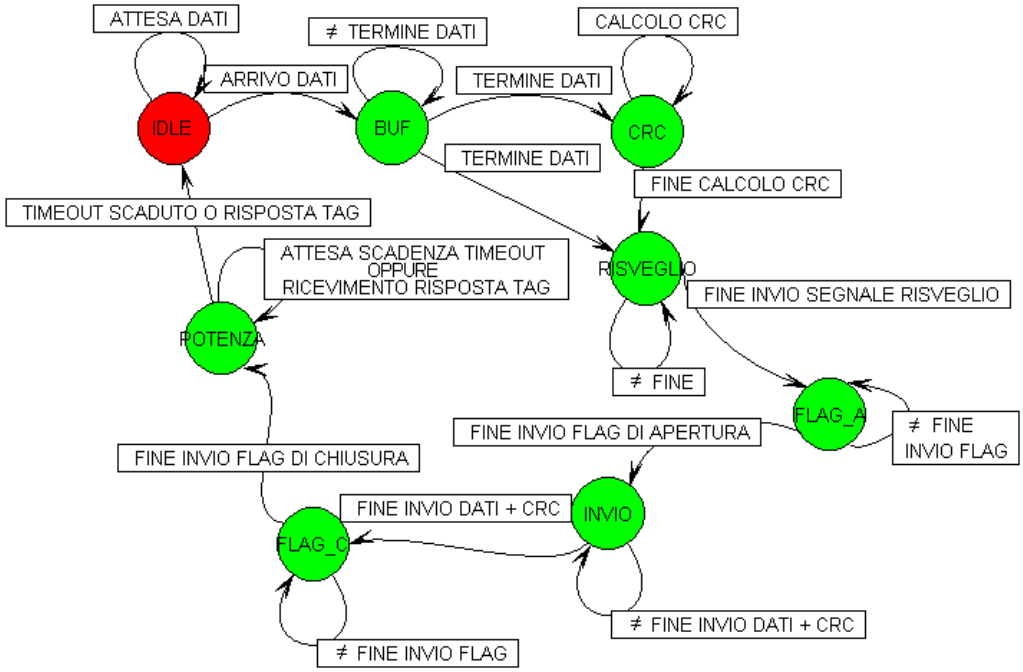


Figura 5.2: Diagramma degli stati della macchina che genera un frame HDLC.

I dati con il messaggio di interrogazione ricevuti dallo strato superiore devono venire incapsulati in un $frame$ HDLC valido. Nello stato IDLE si resta in attesa dell'arrivo dei dati, passando nello stato BUF nel momento in cui arrivano i dati. Lo stato BUF si occupa d'inserire i dati nel buffer di trasmissione, portandosi al termine dell'operazione negli stati successivi.

Al termine dell'arrivo dei dati nel buffer di trasmissione iniziano sia il calcolo del CRC sia la trasmissione del segnale di risveglio. La durata del calcolo del CRC è infatti di 3 ordini di grandezza inferiore alla durata del segnale di risveglio. Il CRC è calcolato durante l'invio del segnale di risveglio in modo da avere già a disposizione tutti i dati da inviare nella fase successiva. Il blocco di generazione del CRC calcola il CRC sui dati in ingresso e il risultato dell'operazione è inserito dopo i dati in coda al buffer di trasmissione.

Lo stato denominato RISVEGLIO invia al modulatore il segnale di risveglio del tag. Terminato l'invio del segnale di risveglio si può procedere con la trasmissione al codificatore Manchester del $frame$ HDLC al *bit rate* richiesto.

Gli stati FLAG_A e FLAG_C si occupano di creare il $flag$ di apertura e chiusura del $frame$, mentre il blocco INVIO si occupa di inviare i dati contenuti nel buffer di trasmissione, eseguendo il *bit stuffing*. Questa operazione modifica nei dati le sequenze uguali ai $flag$ che delimitano il $frame$ HDLC permettendo al ricevitore di determinare correttamente l'inizio e la fine di un $frame$.

Terminato l'invio dell'ultimo $flag$ si mantiene un 1 logico in trasmissione che serve per continuare a inviare potenza in uscita dall'antenna della boa. Questa potenza è necessaria al tag per poter inviare il messaggio di risposta.

Per il calcolo del CRC si è utilizzato il sistema con registro a scorrimento e porte XOR per ogni termine del polinomio generatore, visto nel capitolo precedente. Questa procedura si rivela molto veloce tanto da impiegare al massimo un ciclo di clock per ogni bit da calcolare. Considerando che il clock utilizzato nel sistema ha una frequenza di 18.4 MHz, il calcolo del CRC su un pacchetto dati di 128 kbyte impiega circa 56 ms cioè l'equivalente di $128 \cdot 2^{13}$ cicli di clock.

Per semplificare il blocco di trasmissione del sistema di test il dato da trasmettere al tag è statico e implementato direttamente nella descrizione in VHDL del blocco. Non è prevista al momento la necessità di modificare il dato diagnostico sotto il controllo del sistema di elaborazione dati.

5.3 La sezione di ricezione

La sezione di ricezione è composta dai blocchi che trattano i dati in arrivo dalla linea dati e dal demodulatore di radiofrequenza. Risulta composta da un filtro antirumore e da 2 decodificatori Manchester.

Blocco filtro antirumore

La funzione di questo blocco è quella di ridurre i disturbi dovuti al rumore sul segnale ricevuto dal demodulatore. Il rumore è l'insieme di segnali in tensione o corrente elettrica indesiderati che si sovrappongono al segnale utile. Il rumore può venire da sorgenti esterne, da componenti interne al modulatore e anche da riflessioni della stessa onda trasmessa.

I disturbi in entrata all'antenna vengono amplificati dagli stadi amplificatori e inviati al demodulatore. Il rumore è dovuto all'utilizzo di componenti non ideali nel demodulatore di frequenza che portano ad avere in uscita sul decisore a soglia di Fig. 5.3 un segnale digitale casuale in assenza di trasmissioni.

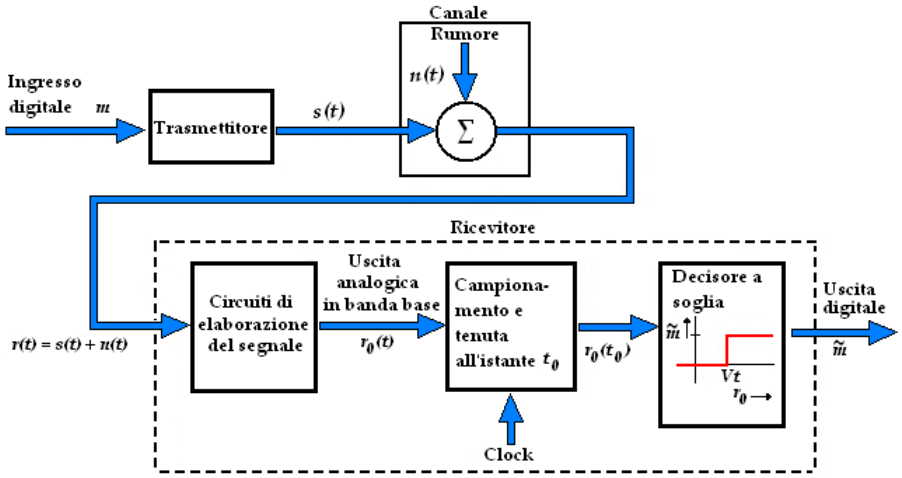


Figura 5.3: Schema generale di un sistema di comunicazione digitale.

Il problema si presenta soprattutto nelle fasi di attesa tra un ciclo di trasmissione ed il successivo; infatti durante le fasi di trasmissione il livello del segnale copre sicuramente quello del rumore. Negli attimi prima o dopo la trasmissione il rumore potrebbe indurre il ricevitore ad anticipare l'inserimento dei dati nel buffer di ricezione rilevando nel rumore un $flag$ di apertura e chiusura inesistenti.

Il filtro implementato lavora sui segnali digitali in ingresso e non considera tutte le variazioni che durano meno di un certo periodo di tempo costante δt che sono considerate rumore e quindi soppresse. Il valore della costante può essere scelta a piacere prestando attenzione però a non superare il tempo per cui un bit resta alto, perché altrimenti verrebbe eliminato.

I dati in ingresso al filtro sono restituiti in uscita filtrati, eliminando un possibile disturbo impulsivo o *spike* sul segnale digitale. In Fig. 5.4 è rappresentato un esempio dei segnali in ingresso(a) ed in uscita(b) al blocco.

Il segnale in ingresso presenta un parte iniziale di rumore evidenziata in verde, e la codifica Manchester di due bit con valore 0, dei quali il primo bit risulta affetto da rumore mentre il secondo no. Il segnale che si presenta in usci-

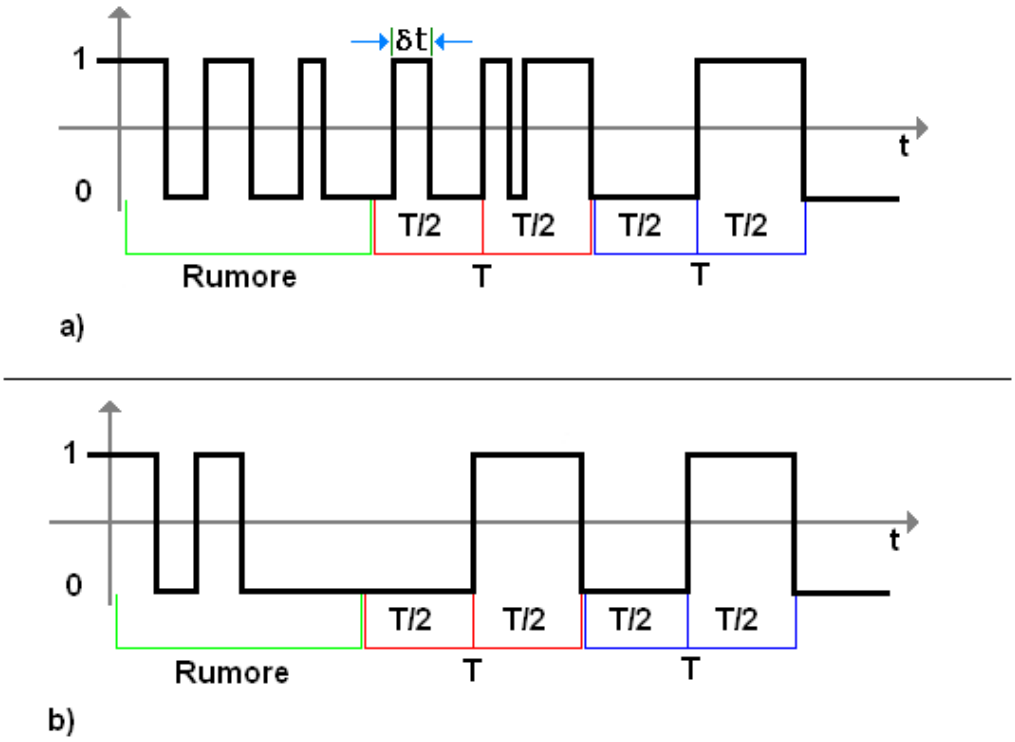


Figura 5.4: Esempio funzionamento filtro.

ta risulta filtrato da tutte le variazioni che durano meno del valore di soglia scelto.

Visto che la metà del periodo di un bit è circa $1.43 \cdot 10^{-6} s$ secondi il tempo di filtro dovrà essere inferiore a tale valore e in particolare dopo varie prove è stato scelto il valore di circa 0.7 μs.

Da notare come il rumore dei segnali in arrivo dalla linea dati in rame sia quasi completamente trascurabile.

I blocchi di decodifica e riconoscimento del frame

Il blocco è istanziato 2 volte dallo stesso codice con parametri diversi per permetterne l'adattamento ai diversi *bit rate* di trasmissione e ricezione. La funzione di questo blocco è quella di decodificare i dati in Manchester e inserirli nel buffer di ricezione dove si verifica se il $frame$ ricevuto è valido o meno.
Il blocco è formato da due sottosistemi:

- Il decodificatore Manchester;
- Il controllore sul formato del $frame$, il *destuffing* e la verifica del CRC.

Il decodificatore Manchester

Il decodificatore Manchester deve estrarre dal flusso di dati in arrivo serialmente il clock e i dati. Per riconoscere il dato il segnale di ingresso deve essere campionato più volte per diminuire la probabilità di errore nel caso in cui il campionamento del segnale venga effettuato vicino a un fronte di salita o discesa.

Un bit rate di 348 kbit/s corrisponde ad un tempo di bit di 1/348 kHz pari a 2.87 μs.

In Manchester ogni bit risulta composto da metà periodo di tempo in cui il segnale resta basso e metà per cui resta alto corrispondente alla durata di un mezzo bit, circa $1.43 \cdot 10^{-6} s$. Ogni bit codificato contiene una transizione a metà del periodo di bit. Nella prima metà del tempo di bit viene trasmesso il valore vero del bit e nella seconda metà il valore complementato. La direzione della transizione determina se il bit è uno 0 o un 1, come si vede in Fig. 5.5.

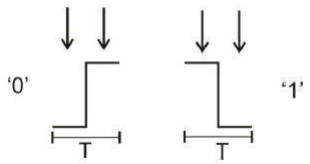


Figura 5.5: Periodo di un bit in codificato in Manchester.

Nel capitolo precedente si è affermato che la codifica Manchester permette la sincronizzazione del ricevitore con il trasmettitore. Può verificarsi però un errore in lettura di mezzo periodo di bit, il che porterebbe ad una decodifica invertita dal lato del ricevitore con conseguente rilevazione errata dei dati ricevuti. A questo problema pone rimedio la codifica stessa. L'errore di sincronizzazione infatti porterebbe a rilevare sequenze di dati specifiche che causano la violazione della codifica. Quando il ricevitore trova che il segnale ha lo stesso valore in due mezzi tempi di bit consecutivi si accorge dell'errore e si risincronizza sulla corretta interpretazione dei dati.

Il decodificatore deve fare i conti anche con un altro problema delle telecomunicazioni: il *jitter* cioè la fluttuazione di un parametro del segnale ricevuto, tipicamente la durata o la fase dei segnali ricevuti rispetto al clock di ricezione ricostruito.

Questo fenomeno si presenta spesso sul bordo dell'area illuminata dall'antenna. Per questo la verifica della durata del mezzo periodo in cui il bit è alto sia uguale alla seconda metà rappresenta una condizione troppo restrittiva. Invece di attendere l'arrivo della transizione esattamente a metà periodo di bit bisognerà aspettarsi che questa possa variare nel zona intorno alla metà periodo con una certa tolleranza.

In Fig. 5.6 è mostrato un esempio di segnale affetto da *jitter* e si può notare che il contenuto informativo dei dati ricevuti non risulta danneggiato ed è quindi possibile recuperare i dati che presentano un *jitter* limitato. In particolare sono recuperabili anche le informazioni contenute nei bit affetti da *jitter* in posizione 2,3,5,6,7,8.

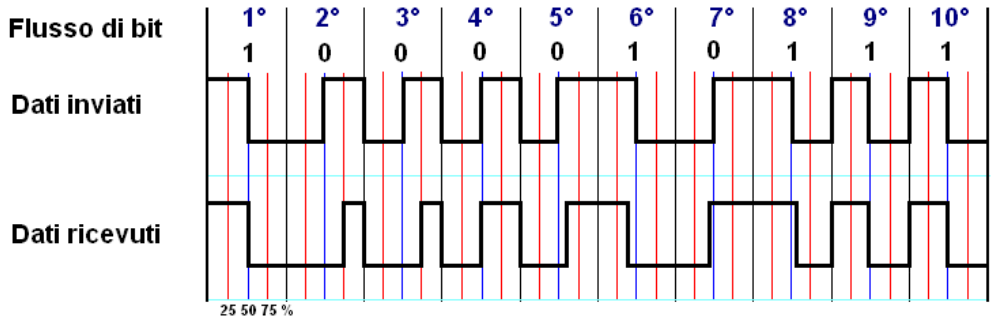


Figura 5.6: Esempio di dati affetti da jitter.

La tolleranza sulla variazione dovuta dal jitter può essere considerata come il 25 % sul tempo di mezzo bit T/2.

Ogni bit riconosciuto viene inserito nel buffer di ricezione dove sarà analizzato per verificare la correttezza del $frame$.

Riconoscimento del frame HDLC

Il sottosistema di controllo si occupa di riconoscere se un frame HDLC è stato ricevuto correttamente o se si sono verificati degli errori in ricezione.

Il sottosistema di controllo attende l'arrivo del $flag$ di apertura nel buffer che indica l'inizio del $frame$ e inizia a calcolare il CRC sui dati in arrivo togliendo il *bit stuffing* quando necessario.

Per la descrizione del blocco è utile analizzare la descrizione semplificata della macchina a stati che lo organizza. In Fig. 5.7 è mostrato il relativo diagramma di transizione degli stati.

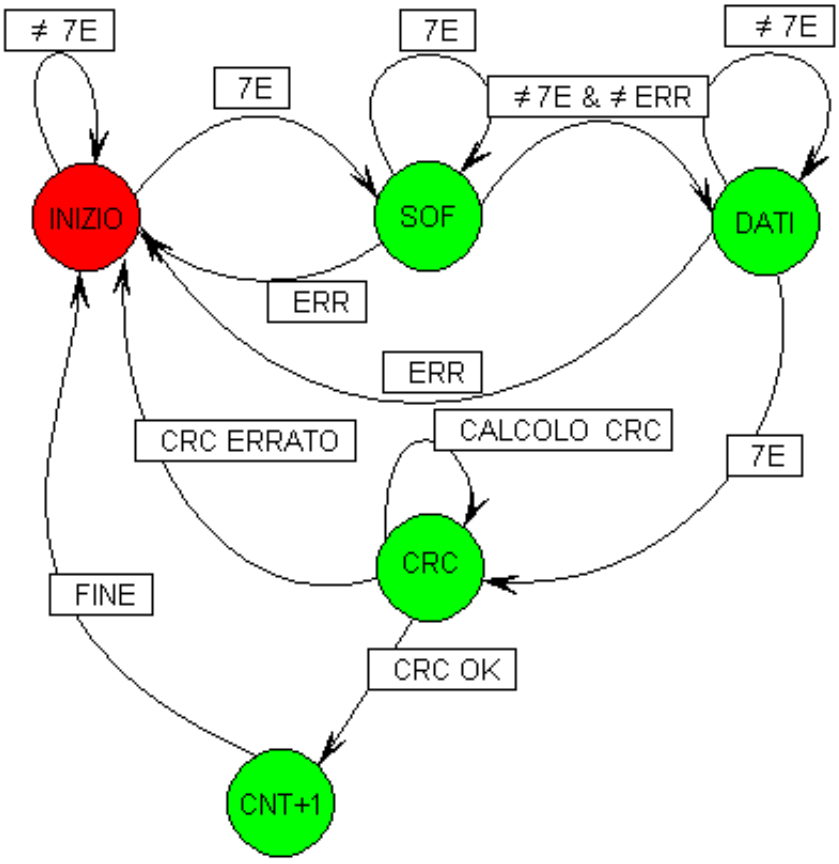


Figura 5.7: Diagramma degli stati della macchina che riconosce un frame HDLC.

I dati sono inseriti nel buffer di ricezione dal decodificatore Manchester e la macchina nello stato denominato INIZIO attende che nel buffer si presenti il $flag$ di inizio $frame$. L'operazione viene eseguita tutte le volte che entra un bit nel buffer di ricezione.

Rilevato un $flag$ si passa nello stato SOF (*Start Of Frame*), dove si resta in attesa nel caso si presenti ancora un $flag$ mentre si cambia stato per qualunque sequenza di 8 bit diversa dal $flag$ stesso e dalle possibili condizioni di errore. Nel caso arrivino $frame$ troncati, visto che i $flag$ di apertura e chiusura sono identici, esiste la possibilità che il $flag$ di chiusura del $frame$ precedente sia scambiato con il $flag$ di apertura del $frame$ successivo. Lo stato SOF consente sempre di rilevare correttamente l'inizio del $frame$, risolvendo il problema precedente.

Quando è rilevato l'inizio dei dati si passa nello stato DATI, che copia i dati dal buffer di ricezione, di dimensione limitata, nel registro dati effettuando il

$destuffing$ quando necessario. Lo stato DATI si occupa anche di rilevare la fine del $frame$ cercando il $flag$ di chiusura.

L'arrivo del $flag$ di chiusura coincide con il termine del $frame$ e si passa quindi nello stato CRC che controlla il processo di calcolo del CRC sui dati ricevuti. Si può quindi verificare il valore contenuto nel registro usato per il calcolo del CRC per determinare se il pacchetto ricevuto era affetto da una condizione di errore.

In caso di esito positivo si incrementa il contatore dei pacchetti corretti andando nello stato CNT+1. In tutti gli altri casi si ritorna nello stato di partenza in attesa di un nuovo $flag$.

5.4 L'unità di controllo e di scambio

La funzione di questo blocco è quella di calcolare le statistiche sui pacchetti ricevuti in base ai riscontri positivi ottenuti dal blocco di ricezione dei dati in radiofrequenza.

Questo blocco presenta 2 stati di funzionamento, a seconda che sia attiva la modalità test o quella normale.

Funzionamento in modalità test: in questa modalità il blocco si occupa di attivare l'invio del pacchetto di interrogazione e attendere l'esito dal blocco di decodifica. Se allo scadere del tempo di 10 ms non è ancora stato ancora ricevuto nessun riscontro il pacchetto viene considerato perso e si procede con un altro ciclo. Al termine dell'invio di 100 pacchetti si calcola la media dei pacchetti corretti ricevuti e la si codifica in uscita al blocco di interfaccia visuale utente.

Per ogni pacchetto ricevuto correttamente si resetta il contatore dei 60 minuti e il sistema resta così in modalità test per tutto il tempo in cui un tag è posizionato correttamente nell'area di copertura.

Funzionamento in modalità normale: In questa modalità il blocco rileva gli eventi da entrambi i blocchi di decodifica dati e invia in uscita la relativa codifica quando è rilevato un pacchetto corretto dal tag e quando è rilevato un pacchetto corretto dalla linea dati.

Il blocco di scambio

La funzione di questo blocco è quella di eseguire lo scambio tra la modalità test e quella normale.

Visto che il sistema parte sempre in modalità test il blocco attende uno dei seguenti eventi per eseguire lo scambio:

- Scade il contatore dei 60 minuti;
- Viene rilevato l'arrivo di un pacchetto dalla linea.

L'arrivo di un pacchetto dalla linea coincide con il primo transito di un oggetto rilevato vicino all'antenna. Il blocco di scambio invia in uscita un segnale che indica lo stato del sistema e che viene utilizzato da tutti i blocchi che hanno 2 distinte modalità di funzionamento.

5.5 I blocchi di supporto

Si tratta di blocchi che generano funzioni molto semplici, come sono il divisore di frequenza, i timer e il blocco di sincronizzazione.

I contatori

Si tratta di semplici timer che una volta avviati iniziano a contare e scaduto il tempo impostato inviano un segnale in uscita.

Anche in questo caso i blocchi sono istanziazione dello stesso codice richiamato con parametri diversi.

Il contatore dei 10 ms utilizza un clock con frequenza di circa 18 kHz che consente di avere una tolleranza di circa 55 μs; le interrogazioni vengono quindi a distanza di 10 $ms \pm$ 55 μs.

Nel contatore dei 60 minuti è stato utilizzato invece un clock con frequenza di circa 1 Hz che consente di ottenere una tolleranza di $\pm$ 1 s, quindi un errore massimo di 1 s su 60 minuti.

L'utilizzo di tali clock consente di ottenere una tolleranza trascurabile rispetto al totale dei tempi contati e di diminuire il numero di flip-flop utilizzati nei registri di conteggio semplificandone la logica.

Il divisore di frequenza

Questo modulo è utilizzato per generare tutti i segnali di temporizzazione utilizzati nel test e la cui frequenza è inferiore a quella del clock principale in entrata alla FPGA di test.

Il divisore è ottenuto implementando 24 flip-flop collegati in cascata. In ingresso al blocco è collegato il segnale di clock di riferimento con frequenza di 18 Mhz e in uscita al blocco sono generati tutti i clock con divisori da 2^1 a 2^{24}.

Il sincronizzatore

Il blocco di sincronizzazione che si occupa di campionare, filtrare e ridistribuire il segnale di reset.

Controllo interfaccia visuale

Il blocco si occupa delle decodifica dei segnali di comando e delle temporizzazioni dei LED.

Alla partenza tutti i LED restano accesi per 2 secondi per consentire la verifica del loro funzionamento, dopodichè si spengono per 2 secondi e il loro significato dipenderà dalla modalità di funzionamento attuale come spiegato nel capitolo 4. Il blocco si occupa di decodificare i dati in arrivo dalla logica di controllo ricavando quali LED accendere. I contatori interni si occupano di temporizzare le accensioni dei LED in base alle specifiche richieste dalla modalità di funzionamento corrente.

5.6 I livelli del sistema

Lo schema a blocchi visto in precedenza in Fig. 5.1 può essere considerato a livello superiore come un unico blocco che realizza il sistema di test, come è evidenziato in rosso in Fig. 5.8.

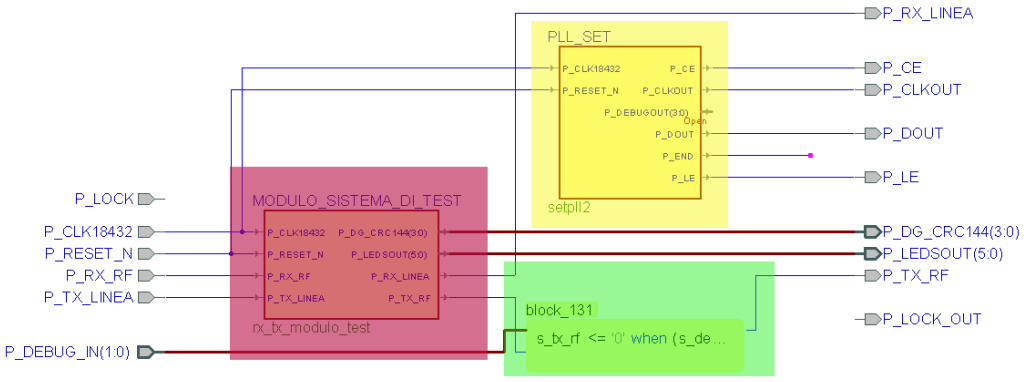


Figura 5.8: Il diagramma del blocco a livello superiore.

Il blocco definito come *top* racchiude l'istanziazione del sistema di test, del blocco di controllo del PLL[1] evidenziato in giallo e il blocco di configurazione evidenziato in verde.

L'impostazione del PLL

Il PLL è un circuito elettronico progettato per generare un'onda di una specifica frequenza, sincronizzata con un'onda di frequenza diversa fornita in ingresso. Il blocco che si occupa di impostare la frequenza del PLL utilizzato per la modulazione del segnale non fa altro che inviare nella corretta sequenza le istruzioni necessarie per programmare il PLL esterno alla FPGA. Il PLL è impostato per funzionare a 2,4 GHz.

Il blocco di configurazione

La scheda dispone di un *dip switch* che consente la possibilità di impostare alcune modalità utili per la diagnostica del dispositivo. In particolare permette di disabilitare la trasmissione dei dati forzando la trasmissione di segnale dall'antenna per consentire una rapida verifica della massima potenza trasmessa.

Nel seguente capitolo sono trattate le caratteristiche dell'ambiente di sviluppo e del *setup* di test.

[1] Phase Locked Loop

Capitolo 6

Ambiente di sviluppo e test di verifica

Il questo capitolo sono descritti l'ambiente di sviluppo e tutto il software utilizzato in questo progetto. É riportata inoltre la descrizione degli strumenti utilizzati e della scheda prototipale. Infine sono riportate una breve analisi dei risultati del collaudo eseguito sul sistema finale, un'analisi delle probabilità di errore rilevate ed un'analisi dei costi e dei consumi del sistema realizzato.

6.1 Software utilizzato

L'ambiente di sviluppo integrato (*Integrated Development Environment*, IDE) utilizzato per la realizzazione del progetto VHDL è Actel Libero IDE versione 8.

L'ambiente di sviluppo è messo a disposizione dal produttore del circuito logico programmabile e in abbinamento alla licenza Libero Gold è possibile utilizzarlo gratuitamente per un anno.

Actel non sviluppa direttamente tutti i software inclusi nell'IDE ma si occupa di integrare software prodotti da aziende leader nel settore, permettendo la scelta di prodotti diversi da quelli inclusi. In particolare ho utilizzato:

1. Il sintetizzatore Synplify Pro 8.8AE prodotto da Synplicity;
2. Il simulatore ModelSim 6.2g AE prodotto da Mentor Graphics.

Libero IDE è modulare e fornisce la possibilità di utilizzare software diversi in base alla necessità dello sviluppatore. É possibile ad esempio sostituire il sintetizzatore con quelli prodotti da altre aziende.

L'interfaccia utente mostrata in Fig. 6.1 si presenta semplice e intuitiva riproponendo il percorso del flusso di progettazione visto in precedenza nel capitolo 1.

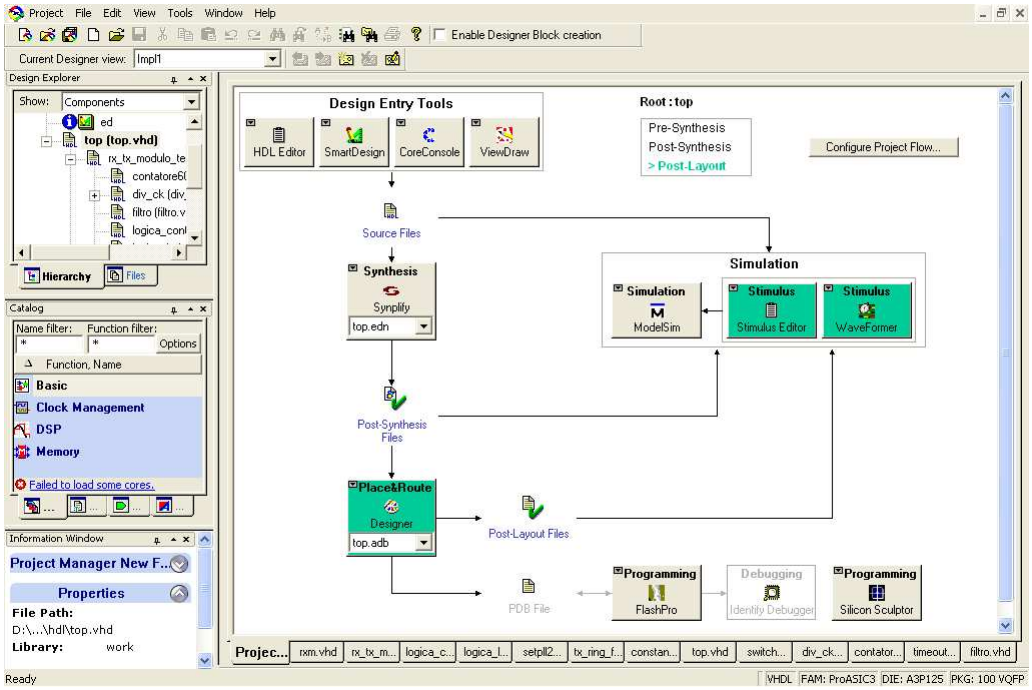


Figura 6.1: Interfaccia ambiente di sviluppo.

Ogni blocco descritto nel capitolo 5 è costituito da almeno un file VHDL. Al livello più alto vediamo solo il file top.vhd, che è composto dall'istanziazione di tutti gli altri file o blocchi.

6.2 Il sintetizzatore

La sintesi è il risultato della traduzione del codice VHDL in una struttura in forma di porte logiche chiamata anche $netlist$. Tra le varie opzioni consente di decidere il grado di ottimizzazione della rete, di porre vincoli sul $fanout$, sulla frequenza di funzionamento e sul tipo di codifica delle macchine a stati.

É possibile visualizzare la rete logica sintetizzata in forma RTL(*Register Transfer Level*), in forma dipendente dalla logica utilizzata e in forma di macchina a stati finiti. Il sintetizzatore si occupa di verificare anche la sintassi controllando non solo la correttezza formale delle descrizioni ma anche la loro sintetizzabilità.

Il risultato della sintesi è costituito da un file .edn che costituisce la netlist delle porte logiche del progetto. Inserendo il modello della logica designata è possibile stimare le frequenze massime di lavoro per ogni blocco, anche se la massima frequenza di lavoro effettiva sarà nota solo dopo il *fitting* finale sul dispositivo. Il sintetizzatore consente comunque di ottenere un'ottima stima dell'area utilizzata e delle frequenze massime di funzionamento.

6.3 Il simulatore

Il simulatore si occupa di simulare il comportamento del modello a seguito di stimoli di tutti i possibili ingressi.

La simulazione è fatta attraverso un file di $testbench$ che genera le forme d'onda d'ingresso per poter verificare se il comportamento del sistema realizzato è quello voluto.

Vista la notevole quantità di dati da introdurre in ingresso è stato scritto, sempre in VHDL, un blocco che simula il funzionamento del tag e il funzionamento della linea dati in ingresso alla boa.

Il simulatore permette di analizzare i 3 livelli di simulazione descritti nel capitolo 1.

Un'analisi del funzionamento del divisore di frequenza può mostrare le differenze tra le varie modalità di simulazione. Il periodo di tempo di funzionamento considerato è quello relativo al periodo dall'avvio del blocco al rilascio del reset sincrono. Nell'istante successivo all'arrivo del primo fronte di clock da dividere, i segnali in uscita dal divisore di frequenza si devono portare da bassi ad alti.

In Fig. 6.2 è raffigurato il risultato della simulazione pre-sintesi, eseguita sul codice in VHDL. Il clock di riferimento è il segnale in rosso, il segnale di reset è evidenziato in verde, mentre tutti i segnali di clock con divisori da 2^1 a 2^{24} sono in blu in ordine dall'alto verso il basso. Si può notare che i ritardi non vengono considerati, infatti tutte le uscite commutano contemporaneamente all'arrivo del primo fronte di salita del clock, subito dopo il reset.

La simulazione post-sintesi è eseguita usando il file .edn risultato dalla sintesi, è quindi eseguita sulla rete logica sintetizzata e considera i ritardi delle porte logiche. In Fig. 6.3 è raffigurato il risultato di questa simulazione e si può vedere come dal ricevimento del primo fronte di clock le uscite commutino con ritardo costante di circa 0.4 ns. In particolare l'uscita del segnale con divisore 2 commuta con un ritardo di circa 0.1 ns rispetto al clock di riferimento,

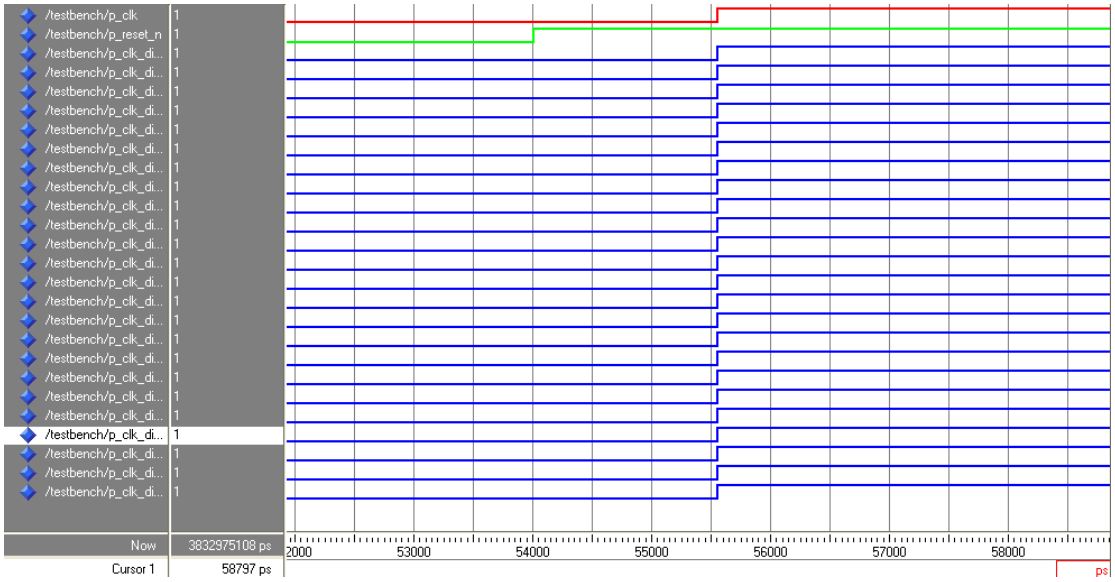


Figura 6.2: Simulazione pre-sintesi, risoluzione 1 ns.

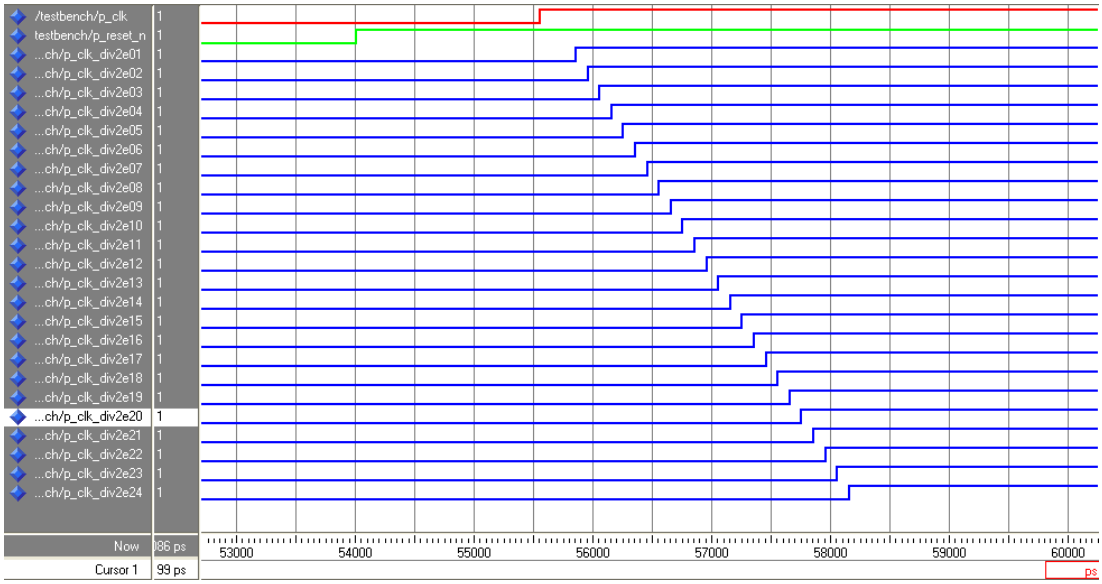


Figura 6.3: Simulazione post-sintesi, risoluzione 1 ns.

mentre quella con divisore 2^{24} commuta con un ritardo di 2 ns rispetto alla prima.

L'ultima simulazione di Fig. 6.4 considera anche i risultati del processo di place and route, dipendenti dal dispositivo utilizzato. Rispetto a prima certe uscite commutano con ritardi variabili da 1 ns fino a 4 ns. In particolare l'uscita del se-

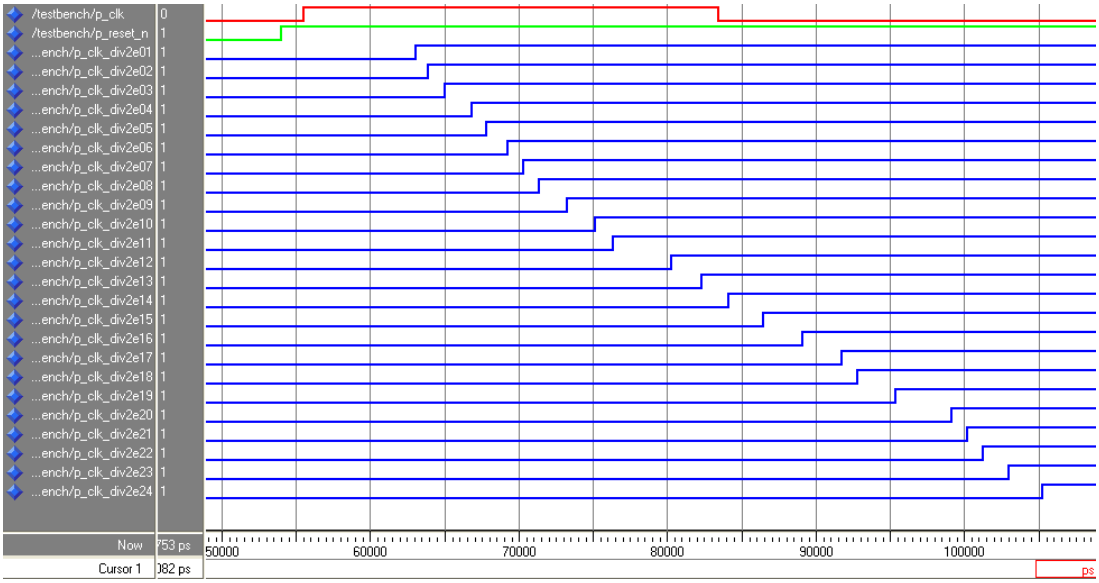


Figura 6.4: Simulazione post place and route, risoluzione 10 ns.

gnale con divisore 2 commuta con un ritardo di circa 8 ns rispetto al clock di riferimento, mentre quella con divisore 2^{24} commuta con un ritardo di 40 ns rispetto alla prima.

Considerando che il periodo di clock è di 54 ns, l'ultima uscita commuta prima del ciclo di clock successivo. Se il clock di riferimento avesse un periodo inferiore al tempo di commutazione dell'ultimo segnale si perderebbero dei cicli di clock con conseguenti ritardi su tutti i blocchi che utilizzo quel segnale.

É anche possibile che lo strumento di place and route non possa minimizzare i percorsi critici. Questo può succedere in particolare quando l'area della FPGA utilizzata dal progetto è vicina al 100%.

Questa simulazione è molto precisa ma anche molto lenta, il computer in dotazione con processore a 2 GHz ha impiegato più di 24 ore per la simulazione di 1 secondo dell'intero sistema finale.

6.4 Il Place and route

La fase di place and route consiste nell'adattare la rete logica, creata in precedenza con la sintesi, al dispositivo reale su cui andrà implementata.

Il software utilizzato è il Designer versione 8 prodotto dall'Actel stessa, in quanto questi software sono solitamente sviluppati dal produttore della FPGA

perché è richiesta una dettagliata conoscenza delle caratteristiche costruttive della FPGA.

La Fig. 6.5 mostra l'interfaccia grafica del software riproponendo il percorso con tutte le possibili fasi.

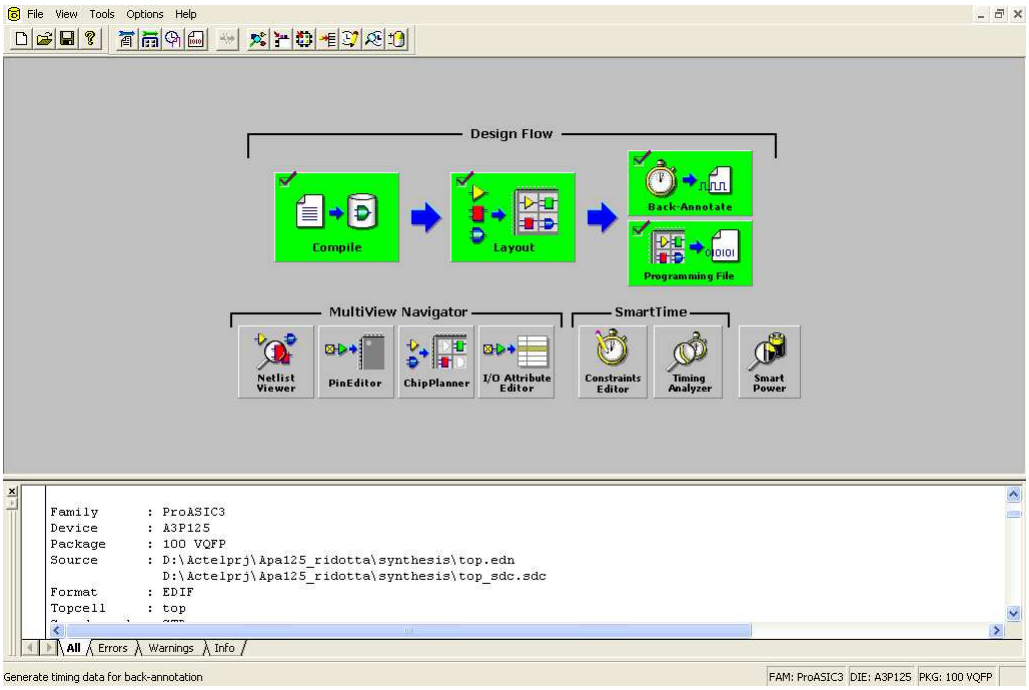


Figura 6.5: Interfaccia grafica Designer.

Questo software permette di selezionare il tipo e la velocità della FPGA, il package utilizzato, la tensione di alimentazione, i livelli delle tensioni di I/O e il range di temperatura ammesso dalle giunzioni. Si tratta quindi di inserire tutte le caratteristiche fisiche ed elettriche della logica finale utilizzata.
Le caratteristiche della FPGA utilizzata sono:

1. Die: A3P125
2. Package: vq100
3. Velocità: standard
4. Tensione alimentazione die: 1.5 Volt
5. Tensione I/O: CMOS 3.3 Volt
6. Range di Temperatura : -40° +85° C°.

Dopo aver selezionato il dispositivo e importato la $netlist$ si procede compilando il progetto. La compilazione controlla che non vi siano errori nella $netlist$ o problemi di *fan out*, rimuove la logica non utilizzata e combina le funzioni logiche per ridurre la superficie utilizzata. Il $layout\ designer$ si occupa di calcolare il place and route ottimale decidendo dove posizionare i singoli blocchi logici e le relative connessioni. Questo strumento permette di riparare al mancato raggiungimento dei vincoli temporali ed ottimizzarli per ottenere la massima frequenza di funzionamento. Al termine viene mostrato il *report*, con tutte le informazioni riguardante la superficie utilizzata della FPGA.

La maggior parte delle decisioni è presa in automatico dal software ma è possibile specificare, tramite il programma *chip planner*, la disposizione manuale dei blocchi logici.

Altre operazioni utili sono la possibilità di visualizzare la rete logica sintetizzata, l'impostazione del *pin out*, l'analisi dei vincoli temporali e dei consumi.

L'ultimo passaggio consiste nel realizzare il file binario che verrà programmato nella FPGA. É anche possibile eseguire il *back annotate*, cioè creare un file con tutte le informazioni sul place and route attuale che utilizzato insieme al simulatore permette di eseguire un'accurata simulazione sul funzionamento finale senza disporre della FPGA.

Il file binario così realizzato è pronto per essere trasferito tramite il programmatore della Actel e il software FlashPro sulla FPGA.

6.5 L'ambiente di test

Al termine dello sviluppo si è utilizzata una scheda dimostrativa (*demo board*) dell'Actel per verificare il funzionamento del progetto in attesa dell'arrivo del prototipo della prima scheda.

L'utilizzo della *demo board* ha permesso di testare e verificare il progetto con una logica simile a quella utilizzata sulla scheda finale, consentendo di eseguire una prima fase di verifica e debug.

Il prototipo della scheda finale del presente progetto ha poi permesso di completare tutta l'analisi sui segnali in ingresso ed uscita alla FPGA.
L'ambiente di test finale è pertanto composto da:

1. Vari tag di prova;
2. Un antenna e i relativi cavi di collegamento;

3. Una boa;
4. La scheda di test;
5. La scheda di interfaccia visuale;
6. Un computer per simulare il sistema di gestione dati.

La strumentazione di misura utilizzata è la seguente:

1. Vari oscilloscopi in particolare il modello Tektronix TDS 2024;
2. Un multimetro digitale;
3. Un analizzatore di spettro (*spectrum analyzer*) Hewlett Packard 8593E.

La boa

In Fig. 6.6(a) è mostrata la boa prima della modifica mentre in Fig. 6.6(b) e mostrata la boa dopo la modifica per l'inserimento del sistema di test.

In particolare è visibile la finestra trasparente sotto la quale è posizionata l'interfaccia grafica semplificata.

(a) Prima della modifica. (b) Dopo la modifica.

Figura 6.6: La boa

Le schede del sistema di test

La scheda prototipale è stata realizzata in due versioni per permettere l'applicazione su due diversi modelli di boa. In Fig. 6.7 è raffigurata la prima versione che dispone di un unico connettore che si collegherà allo slot di espansione presente

sulla scheda madre della boa. In questa scheda possiamo distinguere i seguenti componenti:

1. Il connettore d'interfaccia I/O per la linea dati, il modulatore di radiofrequenza e la scheda LED;
2. i regolatori di alimentazione della scheda;
3. un regolatore di tensione per alimentare i LED dell'interfaccia visuale;
4. un oscillatore da 18 Mhz collegato al pin 13 riservato della FPGA;
5. la FPGA;
6. un regolatore di tensione per in dati in ingresso e uscita dalla FPGA;
7. l'interfaccia di programmazione;
8. un *dip switch* per il debug.

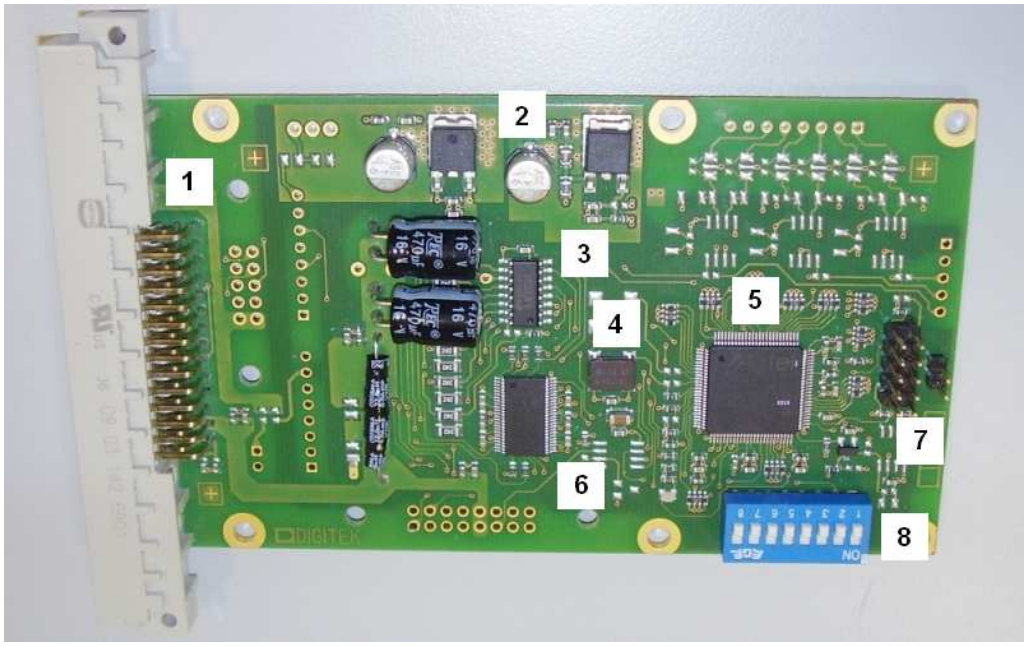


Figura 6.7: Scheda di test 1° versione.

La seconda versione raffigurata in Fig. 6.8 differisce solamente per il tipo di connettori ed alcuni regolatori di tensioni. In particolare è stato rimosso il connettore bianco e sono stati aggiunti:

1. L'interfaccia di I/O della linea dati;
2. il connettore per l'interfaccia visuale;
3. i regolatori di tensione per i segnali della FPGA;
4. il connettore per l'alimentazione;
5. l'interfaccia di connessione con il modulatore di radiofrequenza.

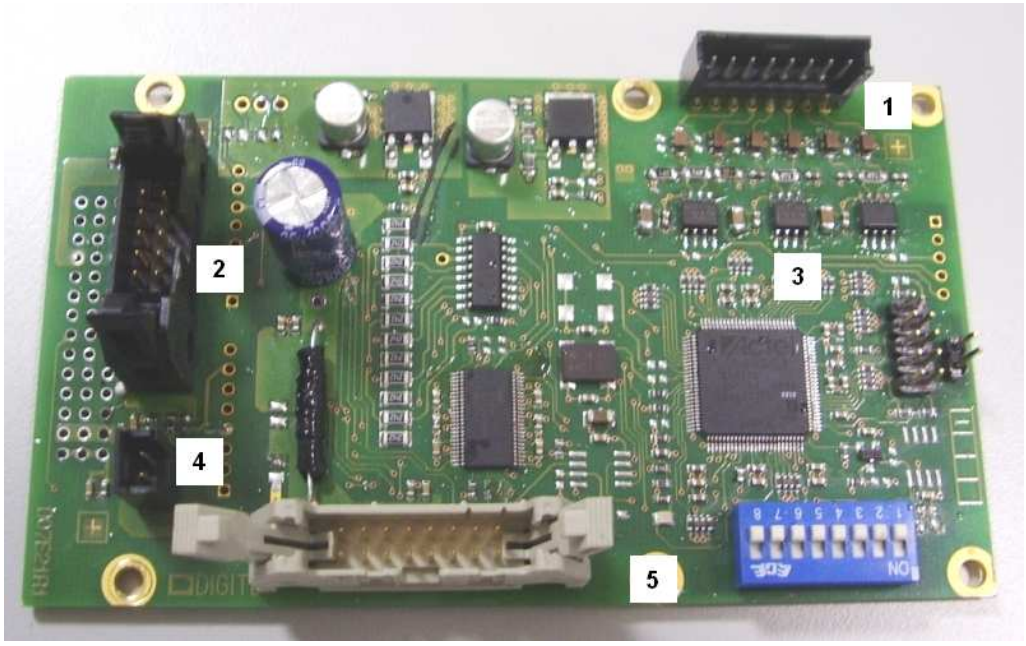


Figura 6.8: Scheda di test 2° versione.

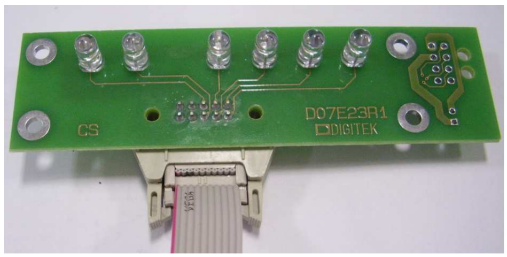


Figura 6.9: La scheda dell'interfaccia grafica semplificata.

La scheda dell'interfaccia visuale si presenta come in Fig. 6.9 ed è composta da 6 LED ad alta efficenza. All'avvio del sistema, durante la fase diagnostica dell'interfaccia grafica tutti i LED risultano accesi come in Fig. 6.10.

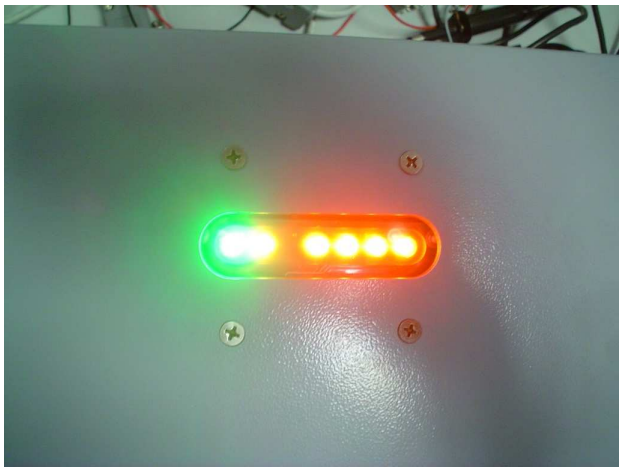


Figura 6.10: Diagnostica interfaccia grafica.

Test di funzionamento

Per verificare il funzionamento del sistema di test sono state eseguite varie prove comparative con il sistema di test software esistente.

Con il sistema di test software di Fig. 6.11 si può analizzare la percentuale d'efficienza di trasmissione dei pacchetti.

I tempi dei segnali codificati in Manchester e i relativi *bit-rate* dei pacchetti generati dal sistema di test sono stati analizzati tramite l'utilizzo dell'oscilloscopio, come si vede in Fig. 6.12.

La frequenza di modulazione dell'antenna è stata verificata tramite l'utilizzo di uno *spectrum analyzer* per verificare la corretta programmazione del PLL interno al modulatore di frequenza che deve essere impostato esattamente a 2.4 GHz, come visibile in Fig. 6.13. Risulta importante verificare che la frequenza di modulazione sia quella attesa, perché in caso contrario c'è il rischio di interferire con altre bande di frequenza non permesse dalle normative.

Test climatici

Visto che il sistema in questione sarà installato in un ambiente esterno, sono state eseguite alcune prove per verificarne il funzionamento nelle peggiori condizioni

```
        Test canali RF  -  versione 02.04_test 27/01/1998

Risultati Test

 CH0
 Trasmissioni:      10198
 Ricevuti:          10197
 RicevutiOK:        10197

 Allarmi BOA:           0
 Tipo allarme:      cmp:0
                    pre:7
                    dat:7
EFFICIENZA
 Ricevuti   =       99.50%
 RicevutiOK =       99.50%
-
```

Figura 6.11: L'interfaccia del test software.

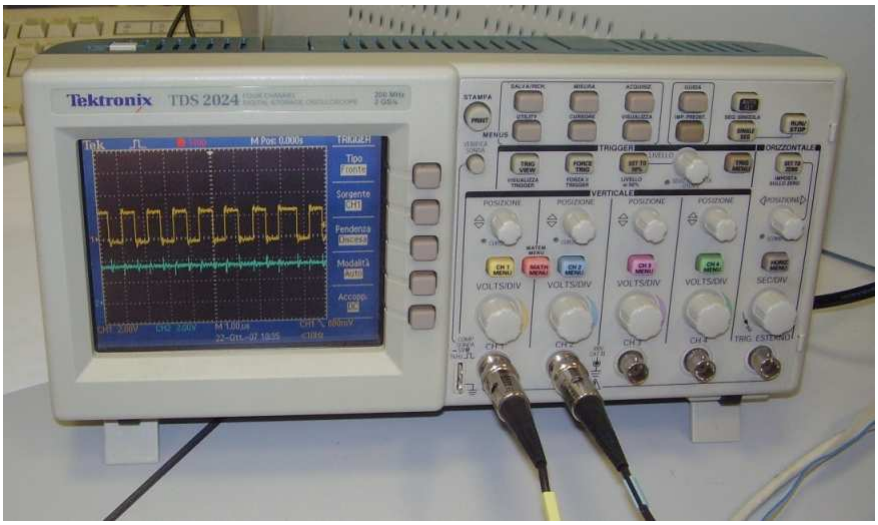


Figura 6.12: Controllo del bit-rate.

in cui si può trovare il sistema comprensivo della boa e della scheda di test.

La boa è stata tenuta accesa per 8 ore sotto il sole per verificare l'area di copertura in condizioni simili a quelle effettive di funzionamento. Il tag è stato identificato correttamente nell'area di circa 1,8 x 4,5 m; l'area di copertura rilevata con il dispositivo di test oggetto di questa tesi è risultata paragonabile con le

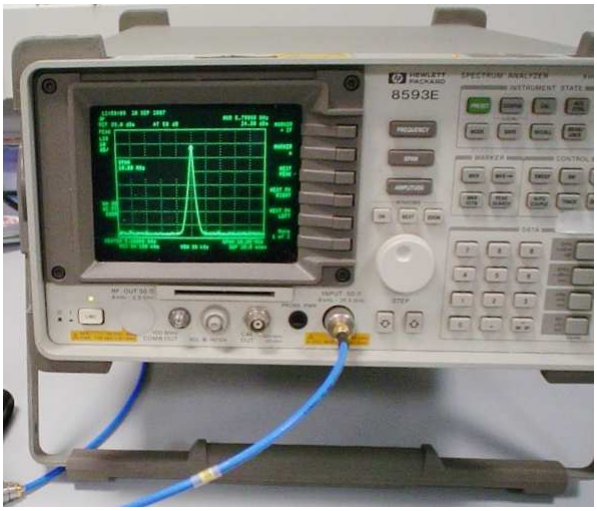


Figura 6.13: Il controllo eseguito con l'analizzatore di spettro.

misure effettuate con il software di test.

Oltre a questa prova per raggiungere le condizioni peggiori sono state utilizzate le celle climatiche disponibili nei laboratori, visibili in Fig. 6.14.

I test effettuati sono i seguenti:

1. Funzionamento 24 ore a temperatura di +55 C°;
2. Funzionamento 6 ore a temperatura di -25 C°;
3. Prova avviamento a freddo dopo lo spegnimento per almeno 15minuti alla temperatura di -25 C°.

Il sistema si è rilevato stabile in tutto il range di temperatura ammissibile.

Altre prove sono state effettuate in condizioni di disturbo cioè utilizzando il sistema all'interno dei laboratori dove sono presenti superfici riflettenti vicino la zona di lettura del tag. In questi casi i risultati non hanno mostrato notevoli cambiamenti, se non in caso di interferenze dovute alla presenza di più tag nella stessa area di copertura, che porta a non riconoscere nessuno dei tag presenti. Questo può essere o meno un problema a seconda della situazione in cui si utilizza il sistema. Nel presente progetto non è fonte di problemi.

6.6 Analisi

Si tratta di alcune brevi considerazioni riguardo i test di funzionamento eseguiti, i costi, i consumi e l'affidabilità dell'intero sistema della scheda di test.

Figura 6.14: Cella climatica.

Analisi dei consumi

Risulta interessante una veloce analisi dei consumi complessivi di tutto il sistema di test.

Il consumo di potenza della scheda finita è risultato essere circa 1 W mentre il solo consumo dei LED ad alta efficienza è stimabile in 0.2 W.

Il consumo dell'intero sistema risulta quindi molto ridotto.

Analisi dei costi

La scheda finale risulta avere un costo totale di 30 euro, considerando solo i materiali di cui è composta ed i relativi cablaggi e tralasciando i costi di assemblaggio. Su questo prezzo hanno influito maggiormente i seguenti componenti:

1. Il circuito stampato: circa 15 €;
2. La FPGA: circa 7 €;
3. Il generatore di clock a 18 MHz: circa 4 €.

Il maggior costo del sistema di test è quindi da imputarsi maggiormente al circuito stampato che influisce sulla metà del costo finale dei componenti. I bassi volumi di produzione previsti in qualche centinaio di pezzi non permettono di ammortizzare il costo di questo componente.

Le probabilità di errore

Quando il sistema è in funzione in modalità test può permettere di identificare il funzionamento corretto di un tag. Un tag viene ritenuto funzionante se risponde quindi ad almeno il 70% dei messaggi di interrogazione.

Si possono distinguere due casi in cui il tag viene ritenuto non funzionante e quindi scartato:

1. Falso positivo: si calcola la probabilità che un tag non funzionante sia rilevato idoneo. Se un tag non funziona correttamente risulta altamente improbabile che su 100 test eseguiti il 70% dei messaggi di risposta risulti corretto.

2. Falso negativo: si calcola la probabilità che un tag funzionante sia ritenuto non idoneo. Se un tag funziona con un'efficenza inferiore al 70% sui 100 test eseguiti, il tag viene ritenuto non funzionante.

Nel prossimo capitolo sono analizzate le considerazioni finali ed i progetti per il futuro.

Capitolo 7

Conclusioni

In questa tesi è stato progettato e realizzato un sistema di test per dispositivi di identificazione a radiofrequenza. Il sistema di test permette di migliorare la precisione d'installazione dell'antenna del lettore e verificare il funzionamento dell'intero sistema RFID tramite l'uso di un'efficace interfaccia visuale. L'utilizzo del linguaggio VHDL si è rilevato fondamentale per lo sviluppo e la realizzazione del progetto, consentendo la realizzazione del modello del sistema di test che, insieme al simulatore, ha permesso di analizzare il funzionamento del sistema in ogni sua parte. L'uso della FPGA ha poi permesso di realizzare fisicamente il sistema di test consentendo la creazione dei prototipi in tempi brevi.

In particolare sono stati realizzati in VHDL un codificatore per la creazione di pacchetti HDLC e un decodificatore per l'analisi dei pacchetti HDLC in transito nella rete.

Alcuni problemi tipici che affliggono le trasmissioni digitali come il rumore e il $jitter$ sono stati ridotti realizzando un filtro digitale per eliminare i possibili *spike* prodotti dal rumore e un decodificatore Manchester capace di interpretare i dati soggetti a un *jitter* limitato.

La scheda di test e l'interfaccia visuale sono stati inseriti all'interno del contenitore esistente della boa utilizzando un apposito slot di espansione.

Dopo aver verificato il funzionamento del sistema di test e ottenuta la certificazione dei laboratori di analisi CEE è stata avviata la produzione della boa e della scheda di test.

Una espansione del progetto è prevista in futuro e consiste nella possibilità di fornire la scheda di test di una porta seriale per permetterne una veloce riconfigurazione attraverso il collegamento con un computer.

Le modifiche principali dovrebbero riguardare la possibilità di variare il mes-

saggio di interrogazione e analizzare esattamente le percentuali dei riscontri rilevando le possibili cause per cui un pacchetto di dati non è stato ritenuto corretto.

Appendice A

Dettagli revisioni VHDL

Le varie revisioni del linguaggio VHDL approvate dall'IEEE, mostrano come questo linguaggio sia in continua evoluzione, dovuta alla volontà di arrivare a documentare e simulare il maggior numero di dispositivi elettronici.

Nel 1987 VHDL è riconosciuto come uno standard HDL, dall'Institute of Electrical and Electronics Engineers (IEEE Standard 1076) e dall'United States Department of Defense (MIL-STD-454L). Questa prima release ufficiale del linguaggio è nota come VHDL-87, e include parecchi tipi di dato, per rappresentare numeri interi e reali, stati logici, tempo, caratteri e stringhe.

Nel 1993 lo standard è stato revisionato dalla IEEE (IEEE standard 1164) giungendo alla versione nota come VHDL-93, dove sono stati risolti i problemi riguardo alla simulazione dei segnali che possono assumere più valori logici, come quelli necessari per la modellazione di sistemi complessi come i bus. E' stata aggiunta così la logica a 9 valori molto più adatta della logica a 2 valori utilizzata prima. Cambiamenti minori sono stati apportati nel 2000 e nel 2002 per aggiungere il concetto di tipo protetto (simile al concetto di classe nei linguaggi ad oggetti) e rimuovere alcune restrizioni nel port map.

In aggiunta allo standard IEEE 1164, sono stati aggiunti altri standard per estendere le funzionalità del linguaggio:
IEEE standard 1076.2 aggiunge una migliore gestione dei numeri reali e complessi.
IEEE standard 1076.3 introduce i tipi signed and unsigned per facilitare le operazioni con i vettori.
IEEE standard 1076.1 (conosciuto come VHDL-AMS) estensione per i segnali analogici.

Nel giugno del 2006 il VHDL Technical Committee of Accellera è stato delegato per lavorare al prossimo aggiornamento dello standard VHDL-2006. Questa versione deve mantenere la piena compatibilità con la prima versione e propone numerose estensioni per scrivere codice più semplicemente, come una sintassi più flessibile per gli statement "case" e "generate", inclusione di VHPI (interfaccia per linguaggi C/C++). Questi cambiamenti dovrebbero migliorare la qualità di codice sintetizzabile, la creazione di *testbenches* più flessibili e un più vasto numero di sistemi descrivibili da VHDL.

Appendice B

Il modello ISO OSI

L'*Open Systems Interconnection* (OSI) è uno standard definito nel 1978 dall'*International Organization for Standardization* (ISO) il principale ente di standardizzazione internazionale che produsse una serie di standard per le reti di calcolatori. Il progetto OSI definì un modello standard di riferimento per l'interconnessione di sistemi aperti. Il documento che illustra tale attività è il *Basic Reference Model* di OSI, noto come standard ISO 7498.

Il modello ISO/OSI è costituito da una pila di protocolli attraverso i quali viene ridotta la complessità implementativa di un sistema di reti di comunicazioni. In particolare ISO/OSI è costituito da strati o livelli che racchiudono uno o più aspetti fra loro correlati della comunicazione fra due nodi di una rete. I livelli sono in totale 7 e vanno dal livello fisico fino al livello delle applicazioni attraverso cui si realizza la comunicazione di alto livello, come rappresentato in Fig. B.1 .

Livello 1: fisico L'obiettivo del livello fisico è trasmettere dati non strutturati attraverso un canale di comunicazione, traducendoli in un segnale elettrico di forma e intensità adeguate. I criteri di progettazione coinvolgono questioni meccaniche, elettriche e di temporizzazione che dipendono dal mezzo fisico attraverso cui il segnale è trasmesso.

In questo livello si decidono:

- Le tensioni scelte per rappresentare i valori logici 0 e 1;
- La durata in microsecondi del segnale elettrico che identifica un bit;
- L'eventuale trasmissione simultanea in due direzioni;

- La forma e la meccanica dei connettori usati per collegare l'hardware al mezzo trasmissivo.

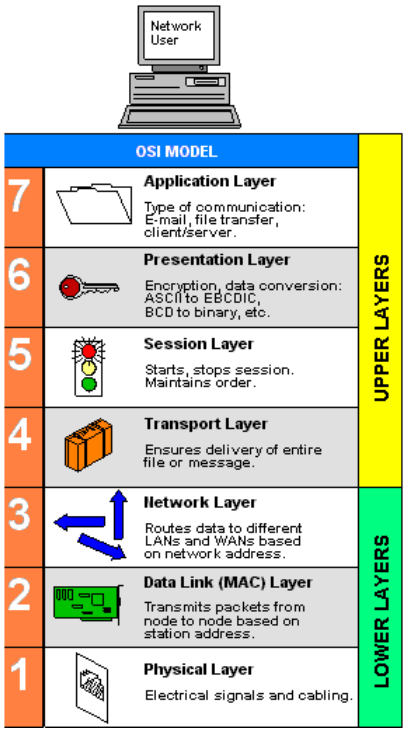


Figura B.1: La pila protocollare ISO/OSI.

Livello 2: datalink Lo scopo del livello 2 è permettere il trasferimento affidabile di dati attraverso il livello fisico. Invia $frame$ di dati con la necessaria sincronizzazione ed effettua un controllo degli errori e delle perdite di segnale. Tutto ciò consente di far apparire al livello superiore il mezzo fisico come una linea di trasmissione esente da errori di trasmissione.

Questo livello si occupa anche di controllare il flusso di dati, rallentando l'invio dei dati della macchina più veloce e minimizzando così le perdite dovute a sovraccarico di chi riceve i dati .

Livello 3: rete L'obiettivo è rendere i livelli superiori indipendenti dai meccanismi e dalle tecnologie di trasmissione usate per la connessione. Si occupa di stabilire, mantenere e terminare una connessione garantendo il corretto e ottimale funzionamento della sottorete di comunicazione. Questo livello si occupa di:

- indirizzamento e $routing$: della scelta ottimale del percorso da utilizzare per garantire la consegna delle informazioni;
- gestione della congestione: evitare che troppi pacchetti arrivino allo stesso $router$ contemporaneamente
- conversione dei dati nel passaggio fra una rete ed un'altra con diverse caratteristiche.

Livello 4: trasporto Il livello di trasporto permette il trasferimento di dati affidabile (implementando anche un controllo degli errori e delle perdite) tra due macchine.

A differenza dei livelli precedenti che si occupano di connessioni tra nodi contigui di una rete, il livello di trasporto si occupa solo del punto di partenza e di quello di arrivo.

Inoltre, ottimizza l'uso delle risorse di rete e usa strategie per prevenire la congestione.

Livello 5: sessione L'obiettivo del livello 5 è controllare la comunicazione tra applicazioni. Stabilire, mantenere e terminare connessioni (sessioni) tra applicazioni cooperanti.

Esso consente di aggiungere ai servizi forniti dal livello di trasporto servizi più avanzati, quali la gestione del dialogo (mono o bidirezionale), la gestione del $token$ (per effettuare mutua esclusione) o la sincronizzazione (inserendo dei $checkpoint$ in modo da ridurre la quantità di dati da ritrasmettere in caso di gravi malfunzionamenti).

Si occupa anche di inserire dei punti di controllo nel flusso dati: in caso di errori nell'invio dei pacchetti, la comunicazione riprende dall'ultimo punto di controllo andato a buon fine.

Livello 6: presentazione Si occupa di trasformare i dati forniti dalle applicazioni in un formato standardizzato e offrire servizi di comunicazione comuni, come la crittografia, la compressione del testo e la riformattazione.

Esso consente di gestire la sintassi dell'informazione da trasferire. Sono previste tre diverse sintassi:

- astratta: la definizione formale dei dati che gli applicativi si scambiano;
- concreta locale: cioè come i dati sono rappresentati localmente;

- di trasferimento: come i dati sono codificati durante il trasferimento.

Livello 7: applicazione L'obiettivo del livello di applicazione è interfacciare l'utente con la macchina.

Fornisce un insieme di protocolli che operano a stretto contatto con le applicazioni. È errato identificare un'applicazione utente come parte del livello applicazione.

I protocolli delle applicazioni tipiche di questo livello realizzano operazioni come:

- Trasferimento di file
- Terminale virtuale
- Posta elettronica

Bibliografia

[1] Peter J. Ashenden, *The Designer's guide to VHDL*, Morgan Kaufmann, San Francisco - CA, 2002

[2] David A. Hodges, Horace G. Jackson, Resve Saleh, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill Professional, 2003

[3] Clive Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*, Elsevier, 2004

[4] Dal sito del produttore Actel: *http://www.actel.com/products/pa3/ProASIC3.htm*

[5] Klaus Finkenzeller, *RFID Handbook*, Wiley, 2003

[6] Andrew S. Tanenbaum, *Computer networks*, Upper Saddle River: Pearson education international, 2003

[7] Peterson Larry L., Davie Bruce S., *Reti di calcolatori*, Apogeo, 2004