



ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione
"Guglielmo Marconi" - DEI
Classe di corso L-8 R - INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea in Ingegneria Elettronica

Progettazione e validazione FPGA di un controller digitale EPC Gen2 per tag RFID

Relatore:
Prof.
Aldo Romani

Presentata da:
Alice Cavalieri

Correlatore:
Gabriele Marasca

Sessione 25 settembre 2026
Anno Accademico 2025/2026

Sommario

Per garantire l'integrità di prodotti in ambiente industriale è utile avere dispositivi capaci di monitorare gli urti a cui vengono sottoposti. In questi luoghi è presente molta energia ed è ampiamente diffuso il protocollo RFID EPC Gen2, per cui la produzione di tag RFID in un chip integrato su silicio su larga scala permette un risparmio in termine di costi e risorse. In particolare, si è voluto realizzare un tag che oltre a poter rispondere alla fase di identificazione sia anche capace di inviare al reader informazioni provenienti da sensori. Questo è stato possibile utilizzando un accelerometro, interfacciato mediante protocollo EPCTMRadio-Frequency Identity Protocols Generation-2 UHF RFID.

Indice

Introduzione	v
1 FPGA	1
1.1 Architettura della FPGA	1
1.2 Development board DE10 Lite	1
2 Tecnologia RFID	3
2.1 Sistema RFID	3
2.1.1 EPC	3
2.1.2 Tipologie di tag	4
2.1.3 Tipi di sistemi RFID	5
2.2 Protocollo UHF RFID EPC Gen2	5
2.2.1 Select	9
2.2.2 Query	9
2.2.3 ACK	12
2.2.4 ReqRN	12
2.2.5 Read	13
2.2.6 Generazione di numeri pseudo casuali (RN16)	14
2.2.7 Generazione del CRC	14
2.2.8 Stati del tag	15
3 Codice VHDL	17
3.1 DE10 Lite	17
3.1.1 MAIN_FSM	18
3.1.2 PROTOCOL_FSM	18
3.2 DECODE_FSM	19
3.3 Legame tra le FSM	20
3.4 PACKET_GENERATOR_FM0	21

3.5	ACCELEROMETRO	21
3.6	Ottimizzazione	23
3.7	Flusso di progetto per circuiti digitali	23
4	Validazione sperimentale	25
4.1	Setup	25
4.2	Fase di identificazione del tag	26
4.3	Fase di accesso alla memoria	26
	Conclusioni	29
	Bibliografia	33

Introduzione

Questa tesi tratta della programmazione VHDL di un FPGA, per l'implementazione hardware del protocollo UHF RFID EPC Gen2. La scheda di sviluppo utilizzata è la Terasic De10 Lite, sulla quale è presente il chip FPGA programmato. Per attuare il protocollo sono stati impiegati un demodulatore UHF esterno e uno switch a radio-frequenza che permette di trasmettere i dati tramite backscattering. Questa tecnica prevede di modulare un'onda continua modificando il carico dell'antenna producendo due stati: stato di adattamento nel quale l'onda continua non viene modificata (circuito chiuso), e stato di mismatch, nella quale l'onda continua viene riflessa con sfasamento di 180 gradi (circuito aperto). Il linguaggio di programmazione utilizzato per programmare l'FPGA è VHDL (Very High-speed integrated circuits Description Language). In particolare, il progetto utilizza VHDL sintetizzabile con descrizione RTL, vale a dire istanziando esplicitamente codice corrispondente a registri o elementi combinatori.

FPGA

Questo capitolo descrive una panoramica sul chip FPGA implementato sulla scheda DE10 Lite utilizzata per il progetto.

1.1 Architettura della FPGA

L’FPGA (Field-Programmable-Gate-Array) è un circuito integrato digitale costituito da blocchi programmabili, interconnessioni programmabili (e.g. linee e switch) e blocchi di I/O programmabili [1]. Questi blocchi possono essere riprogrammati più volte, permettendo al progettista di effettuare i test necessari in fase di collaudo. Un processo molto diffuso per produzioni in serie è quello di collaudare il circuito su FPGA e poi utilizzare un hardcopy compatibile, risparmiando così area e potenza.

1.2 Development board DE10 Lite

Di seguito viene descritta più nel dettaglio la scheda utilizzata per il progetto. In questa scheda di sviluppo il chip FPGA è posto in una scheda PCB, inoltre i pin del FPGA sono già connessi con altri dispositivi e portati all’esterno per rendere il chip immediatamente utilizzabile dall’utente senza ulteriori modifiche. Il chip è collegato a diversi componenti tra i quali un quarzo per avere un clock preciso, pulsanti, led, interruttori, display a 7 segmenti [2]. Gli elementi della scheda sono riportati più nel dettaglio in Fig. 1.1.

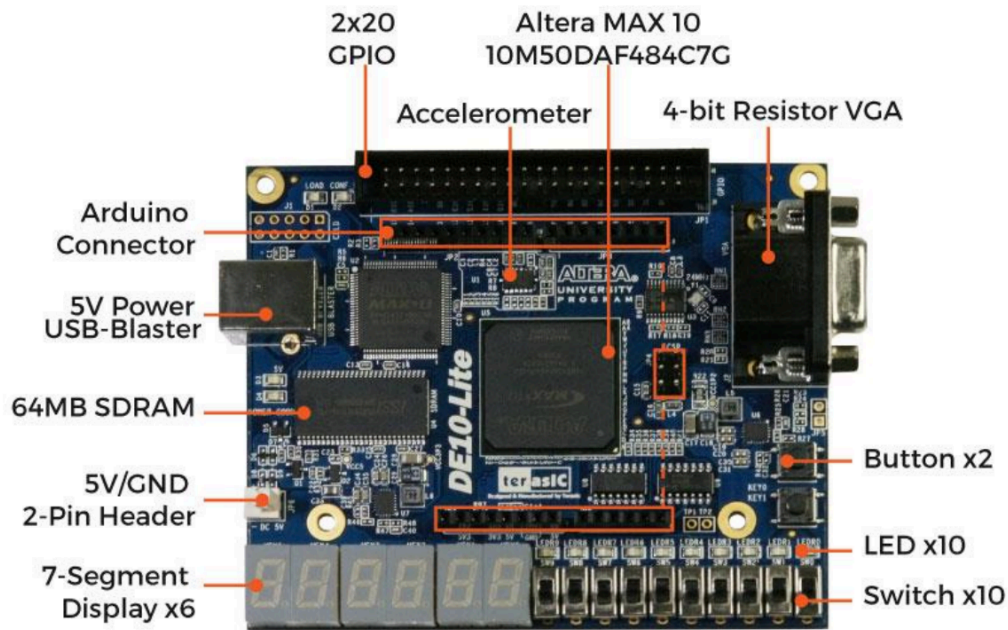


Figura 1.1: Vista dall'alto della Intel DE10 Lite dev board [2]

In questo caso tale scheda è stata programmata in VHDL. In particolare in questo progetto si sono utilizzati il segnale di clock MAX10_CLK1_50 a 50 MHz (per sincronizzare i registri), il connettore di espansione GPIO (per effettuare i collegamenti di I/O), il pulsante KEY(0) come reset, e l'accelerometro ADXL 345, e i LED. Lo schema ad alto livello è visibile in Fig. 1.2.

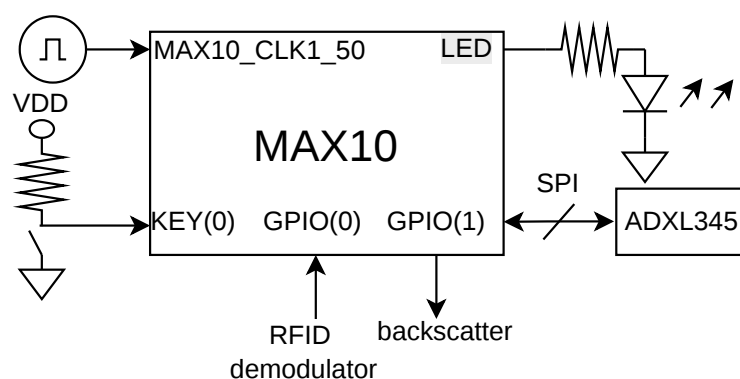


Figura 1.2: Rappresentazione ad alto livello

Tecnologia RFID

Questo capitolo descrive in cosa consiste la tecnologia RFID e il protocollo RFID EPC Gen2.

2.1 Sistema RFID

Un sistema RFID (Radio Frequency Identification) è formato da un reader (lettore) e da almeno un transponder (tag). Il reader trasmette una CW (onda continua) e la interrompe creando un fronte di discesa in uscita al demodulatore per iniziare la comunicazione. Il tag, quindi, demodula il segnale inviato dal reader e, se il comando prevede una risposta, la invia tramite modulatore backscatter della CW inviata dal reader; in particolare invia il proprio EPC (Electronic Product Code). Il tag per rispondere al reader sfrutta il backscattering, vale a dire riflette all'indietro l'onda irradiata modulandola tramite l'apertura e chiusura del carico sull'antenna. Una sua applicazione può essere quella di identificazione e localizzazione di oggetti sui cui è applicato. In particolare, in questo progetto, è stata impiegata la possibilità di utilizzare l'RFID per leggere una porzione di memoria del tag, nel quale viene salvata, dalla rete logica, una variabile relativa al rilevamento di urti. Tale scopo può essere utile in vari ambiti tra cui quello dei dispositivi medici delle produzioni industriali che per mantenere la propria integrità non devono subire elevate sollecitazioni.

2.1.1 EPC

Per l'identificazione dei tag, il protocollo utilizza l'EPC, un codice unico cablato scritto nel tag. Per il progetto si è usata la versione GID-96 (General Identifier), formato da 96 bit di cui [3]:

- i primi 8 bit sono occupati dall'header, utilizzato per capire la versione di EPC utilizzata;

- 28 bit indicano l'entità organizzativa;
- 24 bit indicano la classe dell'oggetto;
- gli ultimi 36 dal numero seriale (identificativo del tag).

La struttura è riportata più nel dettaglio in Fig. 2.1.

Scheme	GID-96			
Total Bits	96			
Logical Segment	EPC Header	General Manager Number	Object Class	Serial Number
Logical Segment Bit Count	8	28	24	36
Coding Segment	EPC Header	General Manager Number	Object Class	Serial Number
URI portion		<i>M</i>	<i>C</i>	<i>S</i>
Coding Segment Bit Count	8	28	24	36
Bit Position	$b_{95}b_{94} \dots b_{88}$	$b_{87}b_{86} \dots b_{60}$	$b_{59}b_{58} \dots b_{36}$	$b_{35}b_{34} \dots b_0$
Coding Method	00110101	Integer	Integer	Integer

Figura 2.1: Tabella di decodifica per GID-96 [3]

Per semplicità, nel codice si è riempito il campo dell'EPC ripetendo per 12 volte il byte corrispondente al valore esadecimale 0x55. In binario, questo si traduce in una sequenza di '0' e '1', che generano numerose transizioni fornendo un pattern semplice da leggere in fase di debug.

2.1.2 Tipologie di tag

A seconda di come sono alimentati i tag si possono suddividere in [4]:

- **attivi:** se hanno una loro alimentazione propria come una batteria. L'energia viene utilizzata per alimentare la circuiteria che compone il tag e per inviare la risposta al reader;
- **passivi:** Se non hanno una propria alimentazione, ma si alimentano tramite l'ambiente o la stessa onda elettromagnetica tramite cui avviene la comunicazione. Non hanno un radio trasmettitore, usano il solo segnale che gli arriva del reader e gli rispondono tramite il backscattering;

- **BAP** (Battery Assisted Passive): se usano il backscattering per la risposta, ma sono dotati di una batteria per alimentare il circuito. Spesso utilizzano forme di energy harvesting per ricaricare la batteria, garantendone il funzionamento più a lungo.

2.1.3 Tipi di sistemi RFID

I sistemi RFID si possono classificare a seconda della banda di frequenze a cui operano [5]:

- **LF** (Low Frequency) RFID tra i 125 kHz a 134 kHz, con un range di lettura dai 10 (o inferiore) ai 50 cm;
- **HF** (High Frequency) RFID a 13.5 MHz, con un range tra i 10 cm a 1 m.
- **UHF** (Ultra-High Frequency) tra i 860 a 930 MHz, range fino a 10 m.

I sistemi LF e HF lavorano in campo vicino sfruttando per la comunicazione l'accoppiamento induttivo, vale a dire il trasferimento di energia da un circuito a l'altro per mezzo di un campo magnetico condiviso; invece i UHF sfruttano la propagazione di onde elettromagnetiche per raggiungere i tag [6].

In particolare, il protocollo utilizzato opera a UHF, utilizzando il backscattering.

2.2 Protocollo UHF RFID EPC Gen2

Ci sono 3 principali operazioni che il reader utilizza per gestire la popolazione di tag:

- **Select**: permette di scegliere i tag con cui comunicare;
- **Inventory**: permette di identificare una popolazione di tag. Questa fase inizia inviando un comando Query in una delle quattro sessioni che viene specificata nello stesso. A questo possono rispondere più tag, i quali vengono identificati tramite il proprio EPC;
- **Access**: una volta identificato il tag, il reader può eseguire diverse operazioni come quella di read e write della memoria interna.

Come descritto in Fig. 2.2 si accede al tag tramite una sequenza di comandi:

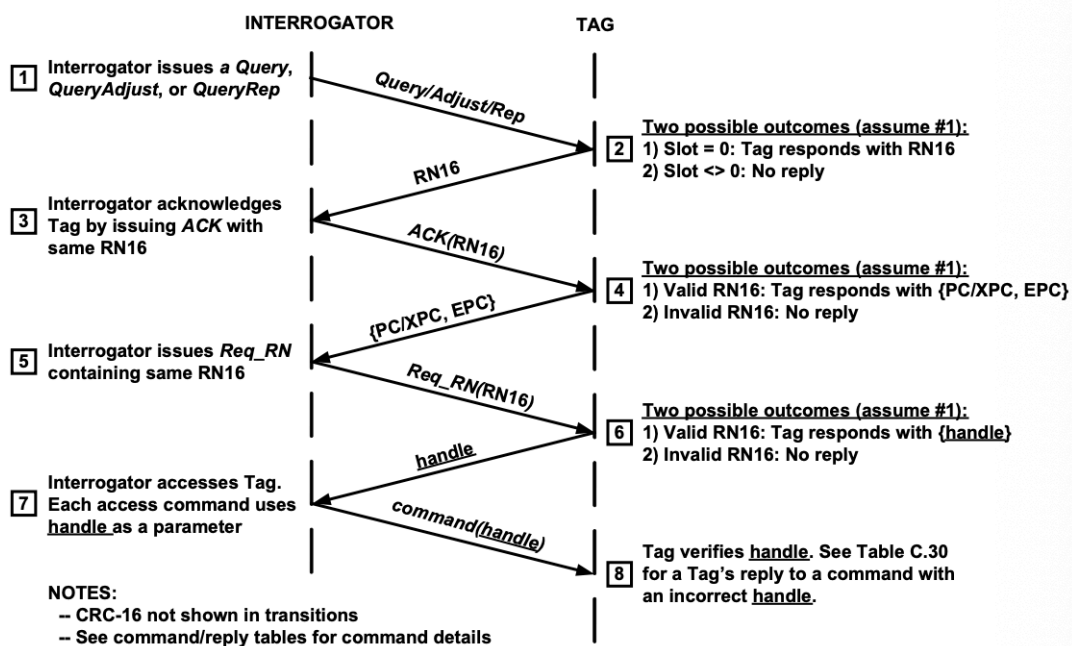


Figura 2.2: Sequenza di accesso ad un tag[7]

1. Il reader incomincia la fase di inventory inviando il comando Query per comunicare alcuni parametri e successivamente i comandi di QueryAdjust e QueryRep, che intervengono variando il numero di slot del tag che può rispondere solo quando questo è 0.
2. Quando il numero di slot del tag diventa 0 allora può procedere con l'invio del proprio random number (RN16).
3. Se il reader riceve correttamente il RN invia un ack che contiene il RN stesso.
4. Il tag lo controlla e se c'è un match col valore precedentemente inviato allora risponde col proprio EPC.
5. Il reader manda allora una richiesta di un nuovo RN, mandando ancora quello precedente.
6. Il tag lo controlla e se c'è un match col valore precedentemente inviato allora manda un nuovo RN (handle).
7. Il reader può quindi avere accesso alla memoria del tag, per esempio inviando un comando read che deve contenere il precedente handle.

8. Infine il tag verifica l'handle e risponde al comando, ad esempio inviando il dato richiesto nel caso di un comando Read.

In particolare, in questa tesi sono state implementate le operazioni di Select, Inventory e Read. Il codice è inoltre stato scritto in modo scalabile così da rendere facile, in futuro, l'aggiunta anche di altri comandi. In seguito, verranno descritti nel dettaglio i comandi implementati. La prima distinzione tra i comandi riguarda la presenza iniziale del Preamble oppure del Frame-Sync, come riportato in Fig. 2.3.

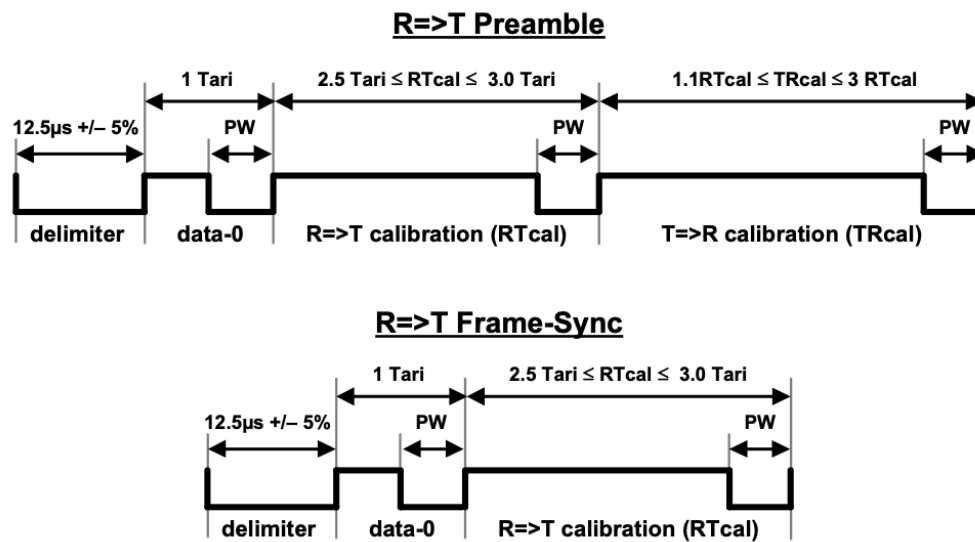


Figure 6.4: R=>T preamble and frame-sync

Figura 2.3: Sequenze di Preamble e Frame-Sync[7]

Come affermato in precedenza, il segnale demodulato del reader mentre è a riposo rimane alto, mentre la trasmissione inizia col primo fronte di discesa. A seguire si ha:

- **Delimiter:** è un impulso basso di lunghezza maggiore a tutti gli altri impulsi bassi;
- **Tari:** indica come deve essere codificato uno '0'. Per scrivere un '1', invece, si tiene la linea alta per più tempo, mentre il tempo in cui rimane bassa deve essere sempre pari a PW, vale a dire il tempo per cui l'impulso che codifica un '1' o uno '0' deve rimanere basso;
- **RTcal** (Interrogator-to-Tag calibration symbol): indica la durata complessiva di un '1' e un '0'. Per decodificare i dati si calcola il $PIVOT = \frac{RTcal}{2}$, se la distanza tra due fronti di salita è maggiore di questa variabile allora si ha un '1', altrimenti '0'.

- **TRcal** (Tag-to-Interrogator calibration symbol): è un parametro utilizzato per calcolare la frequenza a cui il tag deve rispondere ($BLF = 1/T_{pri}$). Fino a questo punto i due pacchetti iniziali sono identici per entrambi i casi, la differenza si ha nella successiva variabile che è presente solo nel Preamble.

Inoltre, lo standard indica un tempo che il tag deve attendere prima di rispondere al reader. Per questo valore vengono indicati un minimo, un massimo e un nominale. Essendoci ritardi introdotti da registri e collegamenti, si è scelto di tarare la risposta usando il valore minimo. Si ha quindi: $T_1 = \max(RT_{cal}, 10 \cdot T_{pri}) \cdot (1 - |FT|) - 2 \mu s$. T_1 è il tempo che attende il tag dopo il comando del reader prima di inviare la propria risposta, mentre $T_{pri} = 1/BLF$ è la durata di un simbolo inviato dal tag. FT indicherebbe, invece, la tolleranza dell'oscillatore, essendo il CK utilizzato pilotato da un oscillatore al quarzo, si è supposto trascurabile. Nel progetto, come frequenza di campionamento si è fissato il valore ad 1 MHz, mentre per la frequenza a cui si inviano i dati si è calcolato dinamicamente $BLF = \frac{DR}{TR_{cal}}$, dove DR (Divide ratio) è un parametro fornito nel Query, se pari a '1' vale $\frac{64}{3}$, altrimenti 8. Di seguito verranno spiegati nel dettaglio i pacchetti di domanda e risposta utilizzati e il tipo di codifica adottata.

2.2.1 Select

Nonostante l'identificazione di un tag sia possibile solo tramite l'inventary, è consentito, ma non obbligatorio, permettere al reader di gestire i tag in una fase precedente. Questo comando non prevede un pacchetto di risposta da parte del tag, la sua struttura è riportata in Fig. 2.4.

Table 6.29: *Select* command

	Command	Target	Action	MemBank	Pointer	Length	Mask	Truncate	CRC
# of bits	4	3	3	2	EBV	8	Variable	1	16
description	1010	000: Inventoried (S0) 001: Inventoried (S1) 010: Inventoried (S2) 011: Inventoried (S3) 100: SL 101: RFU 110: RFU 111: RFU	See Table 6.30	00: FileType 01: EPC 10: TID 11: File_0	Starting Mask address	Mask length (bits)	Mask value	0: Disable truncation 1: Enable truncation	CRC-16

Table 6.30: Tag response to *Action* parameter

Action	Tag Matching	Tag Not-Matching
000	assert SL or inventoried → A	deassert SL or inventoried → B
001	assert SL or inventoried → A	do nothing
010	do nothing	deassert SL or inventoried → B
011	negate SL or (A → B, B → A)	do nothing
100	deassert SL or inventoried → B	assert SL or inventoried → A
101	deassert SL or inventoried → B	do nothing
110	do nothing	assert SL or inventoried → A
111	do nothing	negate SL or (A → B, B → A)

Figura 2.4: Comando Select[7]

Non avendo allocato i 4 banchi di memoria presenti nel protocollo, non si sono considerati i parametri legati alla memoria: quali Membank (considerato con EPC), Pointer, Length. Invece si è tenuto conto di Target che indica se il comando applica una modifica al flag SL o ad una sessione, e action che ha due comportamenti diversi a seconda se ci sia un match o no tra Mask e l'EPC.

2.2.2 Query

Questo comando è obbligatorio ed imposta dei parametri che non vengono poi ripetuti dai comandi successivi come DR, TRcal e M, che sono necessari per impostare la risposta da parte del tag. In particolare, M indica il tipo di modulazione che il tag deve applicare, nel caso del progetto ci si è limitati al caso M=0 coincidente con FM0, per una questione di risparmio in termini di risorse logiche e potenza dinamica.

Si può vedere più nel dettaglio in Fig.2.5.

Table 6.32: Query command

	Command	DR	M	T _{Rext}	Sel	Session	Target	Q	CRC
# of bits	4	1	2	1	2	2	1	4	5
description	1000	0: DR=8 1: DR=64/3	00: M=1 01: M=2 10: M=4 11: M=8	0: No pilot tone 1: Use pilot tone	00: All 01: All 10: ~SL 11: SL	00: S0 01: S1 10: S2 11: S3	0: A 1: B	0-15	CRC-5

Table 6.33: Tag reply to a Query command

	Reply
# of bits	16
description	RN16

Figura 2.5: Comando Query[7]

Di seguito si descrive come è formato il pacchetto di risposta e come si implementa la codifica FM0. Innanzitutto, si ha un preambolo fisso di bit che non seguono la codifica FM0, in seguito se il parametro T_{Rext} vale 1 allora è necessario introdurre 12 '0' iniziali (pilote tone), con la funzione di sincronizzare la comunicazione. A seguito i bit successivi devono essere codificati, invertendo la linea dopo ogni simbolo, con un'inversione aggiuntiva centrale nel caso dello '0'. Infine deve essere aggiunto un '1' codificato (dummy). Si può vedere più nel dettaglio in Fig.2.6.

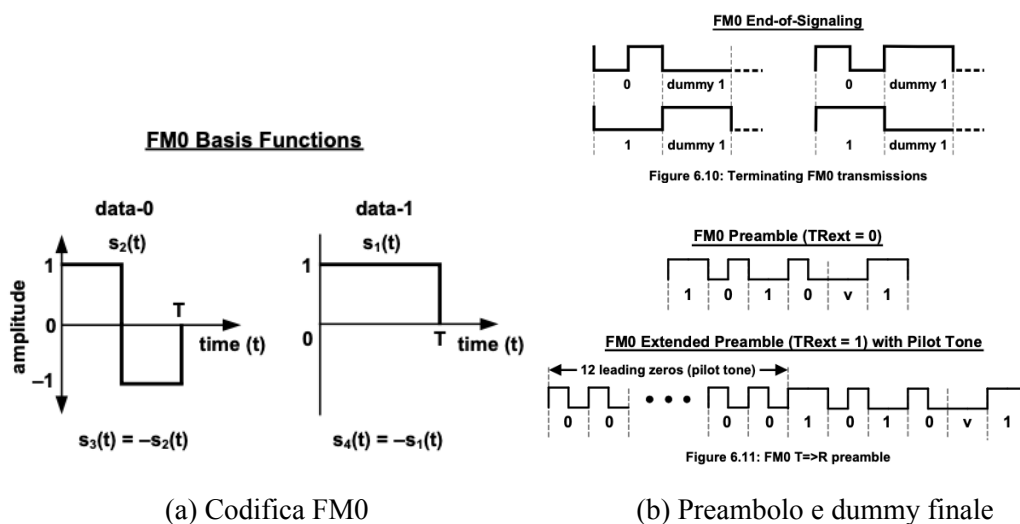


Figura 2.6: Il pacchetto nel dettaglio [7]

Successivamente, come anticipato, si attua un protocollo di anticollisione. Il parametro Q è un numero compreso tra 0 e 15. Il tag deve quindi estrarre un numero pseudo-

casuale tra 0 e $2^Q - 1$ e porlo uguale al numero di slot. Se il numero di slot è zero allora può subito inviare il proprio RN16, altrimenti deve attendere che diventi zero tramite i comandi di QueryRep e QueryAdjust. Nel caso del progetto si è generato il RN16 e si sono estratti i bit da 0 a Q-1 per generare lo slot counter. Nello specifico in caso di QueryAdjust se la sessione del tag ha un match col comando, lo esegue, lasciando il parametro Q uguale o incrementandolo o decrementandolo di uno. Invece nel caso di QueryRep, se la sessione coincide con quella del tag, lo slot counter viene decrementato di uno. La struttura dei comandi si può vedere in dettaglio in Fig. 2.7.

Table 6.34: *QueryAdjust* command

	Command	Session	UpDn
# of bits	4	2	3
description	1001	00: S0 01: S1 10: S2 11: S3	110: Q = Q + 1 000: No change to Q 011: Q = Q - 1

(a) QueryAdjust

Table 6.36: *QueryRep* command

	Command	Session
# of bits	2	2
description	00	00: S0 01: S1 10: S2 11: S3

(b) QueryRep

Figura 2.7: QueryAdjust e QueryRep nel dettaglio [7]

2.2.3 ACK

A questo punto il reader deve rispondere rimandando il RN precedentemente mandato dal tag, che esegue un confronto, il quale se risulta positivo, porta all'invio di un pacchetto contenente l'EPC. Questo pacchetto può avere lunghezza variabile a seconda di varie funzioni avanzate e tipo di EPC. Nel progetto tale pacchetto è formato da PC (Protocol Control), EPC e CRC16. In particolare, il PC, formato da 16 bit, serve ad indicare al reader la lunghezza dell'EPC nei primi 5 bit e poi altre caratteristiche avanzate. Si può vedere più nel dettaglio in Fig. 2.8.

Table 6.38: ACK command

	Command	RN
# of bits	2	16
description	01	Echoed RN16 or <u>handle</u>

Table 6.39: Tag reply to a successful ACK command

	Reply
# of bits	21 to 33,328
description	See Table 6.17

Figura 2.8: Comando ACK[7]

2.2.4 ReqRN

Dopo aver identificato il tag, il reader può accedere alla sua memoria. Il primo comando da inviare è la richiesta di un nuovo RN che dovrà essere seguito dal CRC16. Si può vedere più nel dettaglio in Fig. 2.9.

Table 6.41: *Req_RN* command

	Command	RN	CRC
# of bits	8	16	16
description	11000001	Prior RN16 or <u>handle</u>	CRC-16

Table 6.42: Tag reply to a *Req_RN* command

	RN	CRC
# of bits	16	16
description	<u>handle</u> or new RN16	CRC-16

Figura 2.9: Comando ReqRN[7]

2.2.5 Read

Per leggere un dato in memoria occorre indicare il banco a cui si vuole accedere, il puntatore alla cella di memoria e il numero di word da leggere. Come descritto in precedenza non avendo implementato i singoli banchi, si sono prefissati dei valori. Inoltre, come verifica occorre inviare il RN del comando precedente e il CRC16. La risposta del tag è invece formata da header, dati da leggere, RN e CRC16. Il nome handle viene utilizzato per indicare il RN nel caso in cui sia stato riconosciuto l'EPC e poi si debba accedere alla memoria. Si veda Fig. 2.10.

Table 6.44: *Read* command

	Command	MemBank	WordPtr	WordCount	RN	CRC
# of bits	8	2	EBV	8	16	16
description	11000010	00: Reserved 01: EPC 10: TID 11: User	Starting address pointer	Number of words to read	<u>handle</u>	CRC-16

Table 6.45: Tag reply to a successful *Read* command

	Header	Memory Words	RN	CRC
# of bits	1	Variable	16	16
description	0	Data	<u>handle</u>	CRC-16

Figura 2.10: Comando Read[7]

2.2.6 Generazione di numeri pseudo casuali (RN16)

Non essendo disponibili librerie software dedicate, la generazione di numeri casuali è stata implementata utilizzando un registro a scorrimento con feedback lineare (LFSR). Questo è formato da flip-flop D di dimensione relazionata alla dimensione del registro, dove alcuni stati, costituiscono delle prese (TAPS) che vengono retroazionate tramite l'operazione di XOR al sistema, si veda Fig. 2.11. Il termine "pseudo-casuale" deriva dal fatto che noti gli input è possibile prevedere lo stato successivo. Il circuito deve essere inizializzato con un valore diverso da quello nullo (seme), si può inoltre pensare al registro ad una sequenza di lunghezza di $2n-1$, che si ripete ad una data frequenza, associata ad un polinomio, i cui termini diversi da zero coincidono con le prese. Nel caso del RN il registro realizzato è a 16 bit, inizializzato a tutti 1 al reset, in modo da far partire subito la sequenza di numeri randomici; ad ogni clock viene calcolato un nuovo valore. Quando l'enable di richiesta viene portato a '1' allora il numero viene campionato e memorizzato nel registro contenente la variabile RN16, utilizzata nel protocollo. Il polinomio implementato è $x^{16} + x^{15} + x^{13} + x^4 + 1$. Una descrizione più dettagliata è presente in bibliografia [8].

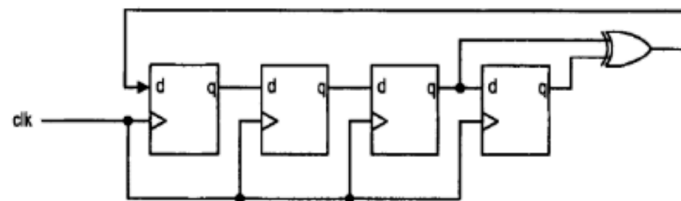


Figura 2.11: Schema del circuito per generare numeri pseudocasuali [8]

2.2.7 Generazione del CRC

Per garantire l'integrità dei dati trasmessi e ricevuti in diversi comandi viene implementato il CRC, in particolare nel progetto realizzato nella ricezione del Query (CRC5) e del Select, ReqRN e Read(CRC16), e nella trasmissione dei pacchetti in risposta a ReqRN e Read (CRC16). Per calcolare il CRC (Cyclic Redundancy Check) si sono utilizzati sempre registri a scorrimento con prese e operazione di XOR. I polinomi utilizzati, però, sono indicati rigidamente dal protocollo, così come i loro valori iniziali, sono riportati in Fig. 2.12.

Table 6.11: CRC-16 precursor

CRC-16 precursor (See also Annex F)				
CRC Type	Length	Polynomial	Preset	Residue
ISO/IEC 13239	16 bits	$x^{16} + x^{12} + x^5 + 1$	FFFF _h	1D0F _h

Table 6.12: CRC-5 definition

CRC-5 Definition (See also Annex F)				
CRC Type	Length	Polynomial	Preset	Residue
—	5 bits	$x^5 + x^3 + 1$	01001 ₂	00000 ₂

Figura 2.12: Polinomi per il calcolo del CRC [7]

2.2.8 Stati del tag

Durante il ciclo di funzionamento il tag passa attraverso vari stati. Appena riceve l'energia dal reader e si accende, il tag si trova in Ready. Nell'implementazione adottata è sempre sveglio poiché non alimentato dal reader. A seguito dell'arrivo del Query il tag passa in Arbitrate dove rimane fino a quando il suo slot counter non è zero. Si trova poi in Reply, stato in cui invia il proprio RN16. A seguito dell'ACK corretto passa poi ad Acknowledged dove invia il pacchetto contenente l'EPC. Con questo stato si conclude la fase di identificazione. Nel caso in cui si voglia leggere la memoria del tag allora il reader dovrà inviare un ReqRN che fa passare il tag allo stato Open.

A questo punto inviando un comando Read il reader può leggere la memoria. In Fig. 2.13 viene riassunto lo schema proposto.

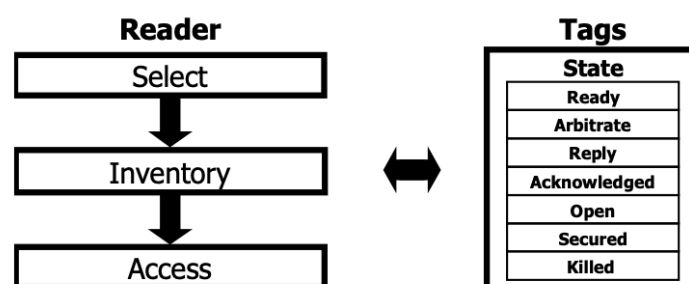


Figure 6.22: Interrogator/Tag operations and Tag state

Figura 2.13: Stati del tag [7]

Il protocollo è descritto in dettaglio nello standard EPC Gen2 [7].

Codice VHDL

Il progetto realizzato si compone di una top-level "DE10_LITE.vhd" dove vengono coordinati tutti i componenti e gestite le macchine a stati. Oltre a questa si sono realizzati due componenti: "PACKET_GENERATOR_FM0" che ha il compito è quello di costruire il pacchetto codificato da inviare al reader e "ACCELEROMETRO.vhd" che si occupa di gestire l'accelerometro ADXL 345 integrato sulla scheda DE10 lite. Lo schema RTL della top-level è visibile nel dettaglio in Fig. 3.1, i riquadri in giallo indicano le FSM.

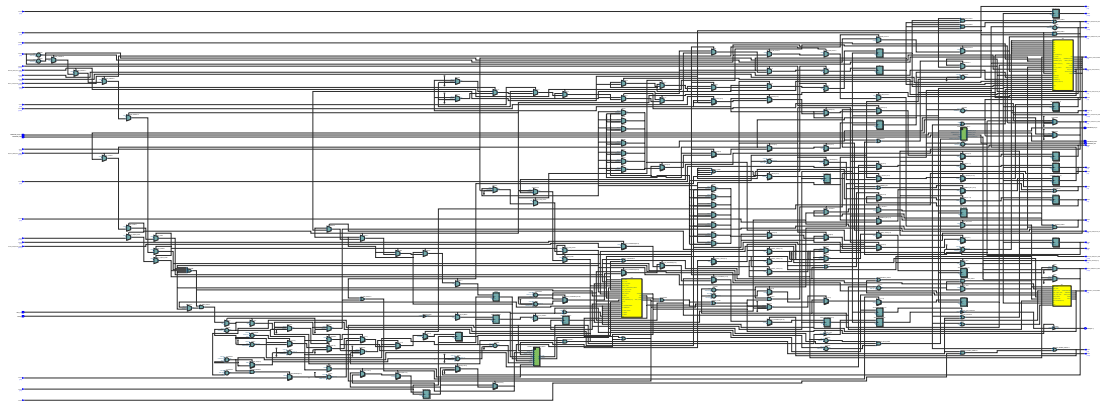


Figura 3.1: RTL viewer della top level

3.1 DE10 Lite

Per garantire la modularità del codice, si è deciso di suddividere cosa accade a livello di ricezione e trasmissione a basso livello, da ciò che accade a livello di protocollo. Nella top_level sono presenti altre strutture come quelle dedicate al calcolo dello Slot counter, T_{pri} (TIME_PIN), T_1 (TIME_reader).

3.1.1 MAIN_FSM

La "MAIN_FSM" gestisce il tag a basso livello, indicando quando è a riposo (IDLE), quando deve aspettare per allinearsi al reader (WAIT_STATE), quando deve trasmettere (SEND) o ricevere/decodificare (DECODE). In particolare:

- **IDLE**: è lo stato iniziale del tag, aspetta il primo fronte di discesa per iniziare la comunicazione;
- **DECODE**: indica che il tag sta campionando i dati e delega il processo di decodifica a DECODE_FSM. Il flag ERROR_FLAG indica un qualsiasi errore rilevato e se presente obbliga il tag a tornare allo stato di IDLE;
- **WAIT_STATE**: se il segnale demodulato è corretto e richiede la trasmissione si aspetta in questo stato un tempo pari a T_1 . Inoltre all'ultimo ciclo di WAIT prepara i dati da trasmettere a "PACKET_GENERATOR_FM0", che provvede a fare la decodifica;
- **SEND**: si occupa della trasmissione del dato, settando il pin per il tempo richiesto (T_{pri}). La macchina a stati precedentemente descritta è illustrata in Fig. 3.2.

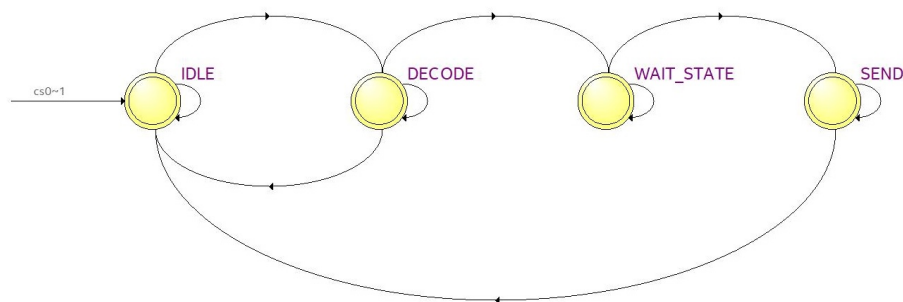


Figura 3.2: MAIN_FSM

3.1.2 PROTOCOL_FSM

Questa FSM ricalca gli stati del tag (si veda 2.2.8), riproducendo più ad alto livello cosa accade. La macchina a stati viene anche riportata in Fig. 3.3.

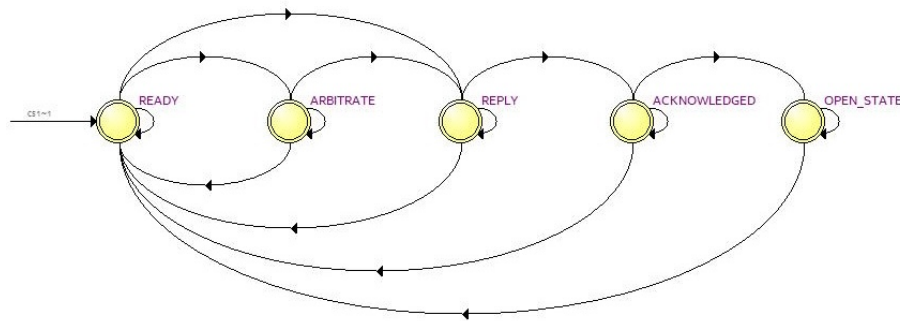


Figura 3.3: PROTOCOL_FSM

3.2 DECODE_FSM

Per comprendere meglio il funzionamento del processo di decodifica si è suddiviso il flusso in stati. Come spiegato in precedenza per decodificare un bit è necessario calcolare il tempo che intercorre tra due fronti di salita. Il contatore utilizzato per calcolare il tempo viene attivato sul primo fronte di discesa, quando la MAIN_FSM passa da IDLE a DECODE. La struttura della FSM può quindi essere riassunta in tre punti principali:

- **SYNCH:** in questo stato vengono calcolati, al termine dei primi tre fronti di salita, rispettivamente: delimiter, TARI e RTcal (quindi il PIVOT). Il PIVOT viene poi utilizzato per calcolare ad ogni fronte di salita il bit decodificato, come già affrontato. Al terzo fronte si distinguono il caso Frame-Sync dal caso preamble. Nel primo caso si viene indirizzati verso COMMAND_STATE, nel secondo verso QUERY. In questo stato vengono anche inizializzati i valori per il calcolo del CRC, per mantenere un allineamento con i dati ricevuti.
- **COMMAND_STATE:** questo comando raccoglie i 4 o 8 bit necessari al riconoscimento del comando trasmesso, reindirizzando poi lo stato futuro della FSM verso lo stato corrispondente.
- **stati dedicati ai comandi:** a seconda del tipo di comando vengono raccolti i parametri ricevuti, effettuano eventuali check del CRC o altri parametri, producendo dei flag utilizzati dalla PROTOCOL_FSM per proseguire nel protocollo. Al termine dell'acquisizione o in caso di errore si ritorna in SYNCH.

In caso di errore ERROR_FLAG viene portato a '1'. Inoltre per ottimizzare le risorse si sono accorpati la ricezione di RN16 e CRC16 presente sia in ReqRn che in Read. In Fig. 3.4 si può vedere più nel dettaglio il flusso di transizioni.

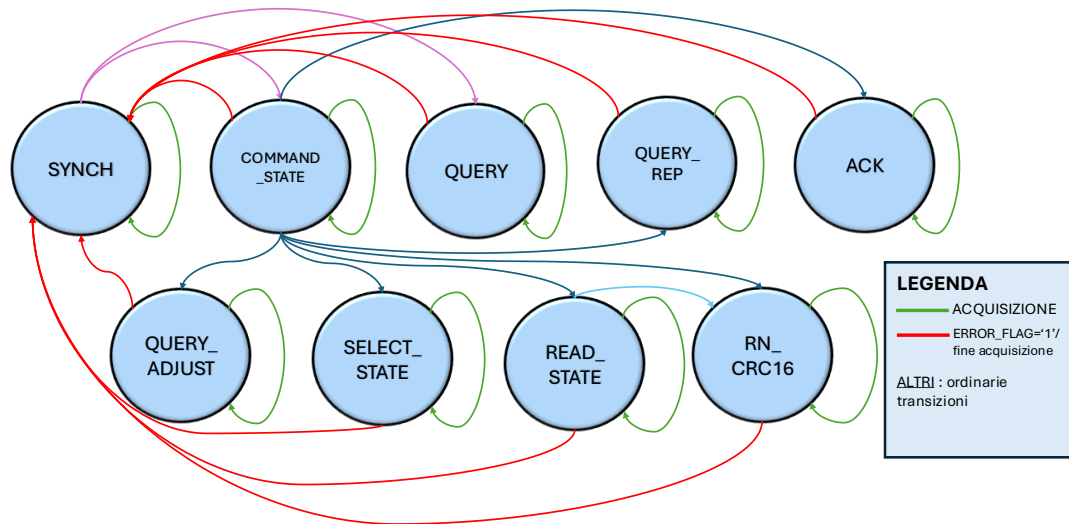


Figura 3.4: DECODE_FSM

3.3 Legame tra le FSM

In Fig. 3.5 si può vedere come le macchine a stati interagiscono tra loro. In particolare, DECODE_FSM pilota le altre, ma non è guidata direttamente da nessuna. Comunicano tra loro impostando dei flag e leggendoli. I nomi riportati nella legenda identificano un insieme di flag che svolgono quella funzione, dunque sono nomi fittizi che non compaiono nel codice ma sono solo a titolo esplicativo.

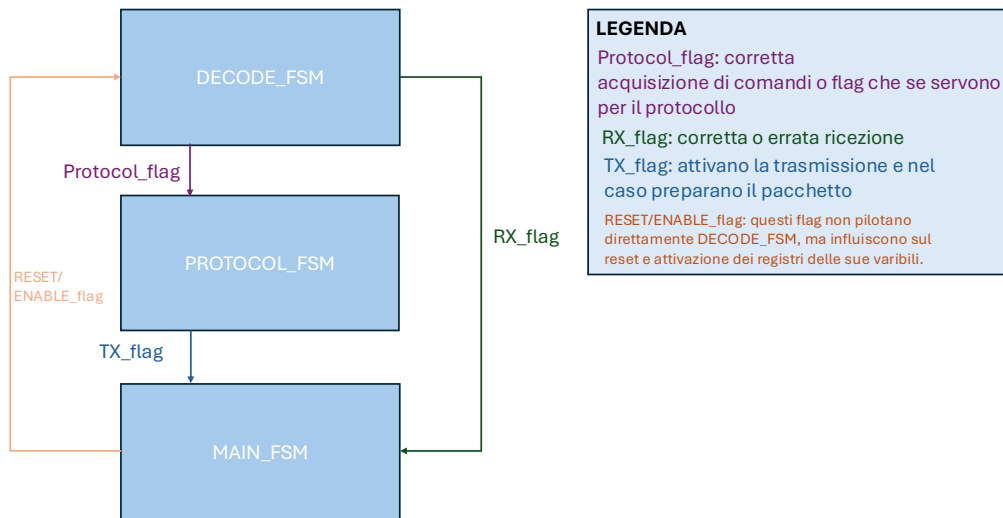


Figura 3.5: Legame tra FSM

3.4 PACKET_GENERATOR_FM0

Questo componente fornisce bit per bit il pacchetto da trasmettere. Per prima cosa viene costruito un pacchetto non codificato, che viene realizzato valutando T_{Rext} e la tipologia del pacchetto a seconda della fase del protocollo; inoltre viene precalcolata anche la lunghezza complessiva di quello finale. Questo componente coopera con lo stato SEND della MAIN_FSM, che trasmette il valore che gli viene passato e lo mantiene sul pin per un tempo pari a $T_{pri}/2$. Questo accade poiché la codifica FM0 è strutturata sull'inversione a metà simbolo, per cui i valori trasmessi corrispondono a mezzi simboli. Calcolato l'ultimo mezzo simbolo porta il flag di fine trasmissione TX_DONE a '1'.

3.5 ACCELEROMETRO

Questo componente si occupa della gestione dell'accelerometro, in particolare fornisce alla top_level la variabile che indica il numero di urti subiti dalla scheda. Per essere configurato l'ADXL 345 necessita della scrittura di registri specifici, per approfondire l'argomento si veda [9]. Tra i protocolli di comunicazione supportati dal dispositivo si è scelto di impiegare l'SPI, in particolare si è selezionata la modalità a 4 fili. In fase di test si è calibrato il dispositivo al fine di rilevare gli urti, impostando la soglia ad

un valore di 0.25g. Il codice attiva un FLAG di interrupt che diventa '1' nel momento in cui viene superata la soglia impostata. Il funzionamento del dispositivo è stato suddiviso in due macchine a stati, rispettivamente LIFE_FSM e SPI_FSM. La prima è formata da 4 stati: INIT per formare il pacchetto da scrivere nei registri, ACQUISITION che legge il flag di attività GSENSOR_INT(1) finchè non diventa true, infine RESET che prepara il reset dei flag che avviene tramite lettura del registro INT_SOURCE. La seconda ha il compito di gestire l'SPI ed è composta da due stati: IDLE che indica uno stato di riposo e SEND che si occupa di inviare i dati per configurare i registri. Per realizzare la sincronizzazione ad 1 MHz, ovvero GSENSOR_SCLK, come clock è stato impiegato un contatore. Questo permette di controllare e di implementare facilmente i fronti di salita e discesa previsti dal' SPI. Il pacchetto da inviare è formato da 2 byte, uno per indicare il comando di scrittura o lettura, il numero di byte trasmessi/ricevuti e l'indirizzo, l'altro per indicare il dato da scrivere o leggere. Il flusso dei dati si può vedere più nel dettaglio in Fig. 3.6.

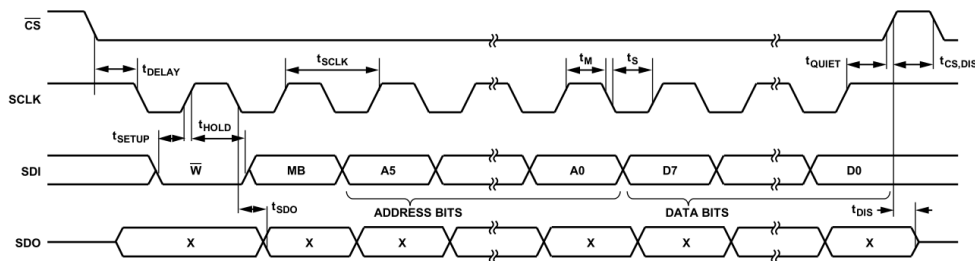


Figure 37. SPI 4-Wire Write

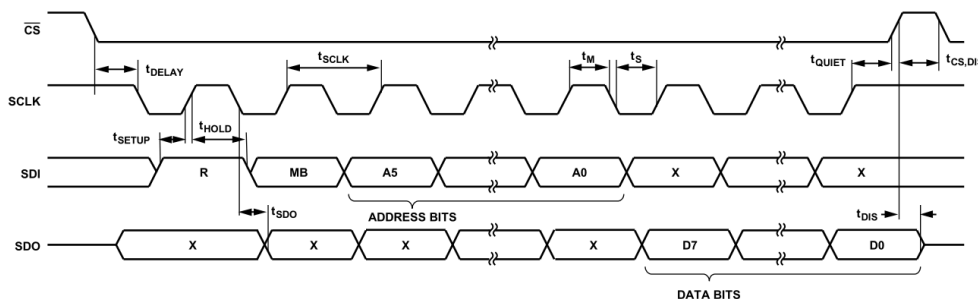


Figure 38. SPI 4-Wire Read

Figura 3.6: SPI 4 fili[9]

3.6 Ottimizzazione

Il progetto è stato inizialmente validato con una frequenza di clock pari a 50 MHz. Per diminuire i consumi, si è deciso di valutare il funzionamento del sistema anche per frequenze di clock inferiori impiegando il PLL fornito da Quartus al fine di simulare il funzionamento ad 1 MHz. Grazie al funzionamento modulare del codice, è stato sufficiente eliminare il contatore impiegato per campionare l'enable corrispondente e aggiornare i cicli di clock utilizzati per il calcolo di alcune variabili, quali T_{pri} e T_1 . Per mantenere la struttura del codice precedentemente implementato si è deciso di implementare l'SPI a 250 kHz. Inoltre, per rendere il codice più robusto sono stati aggiunti due stati nella LIFE_FSM, WAIT_STATE che aggiunge un ciclo di clock dopo aver settato ogni registro per permettere al chip select di tornare alto stabilmente e WAIT_RESET che aspetta che il flag di attività torni a '0' prima di riprendere l'acquisizione.

3.7 Flusso di progetto per circuiti digitali

Durante questo progetto si sono seguite le tipiche fasi di una progettazione hardware. Dopo la descrizione VHDL, il codice è stato sintetizzato, la sintesi logica evidenzia che sono stati utilizzati 1459 su 49760 disponibili, inoltre si sono utilizzati 551 registri. In Fig. 3.7 si può vedere il Chip Planner, dove i blocchi in blu scuro indicano le risorse logiche utilizzate.

Il codice è stato poi simulato facendo uso del tool Questa -Intel FPGA Edition di Intel Quartus Prime Lite Edition 21.1. Per mezzo dell'utilizzo di testbench, si è analizzato il comportamento del codice ponendo in input (GPIO(0)) un file. Quest'ultimo è stato ricavato dalla simulazione di un microcontrollore STM32 con firmware programmato per rispondere al protocollo nella fase di inventory. Questo metodo ha permesso di analizzare nel dettaglio i valori dei registri. Si è anche salvato la forma d'onda dell'output della scheda GPIO(1) in un file. Questo ha permesso di avere un confronto tra le due forme d'onda su matlab. Successivamente si è eseguita la validazione sperimentale, che verrà discussa nel prossimo capitolo.

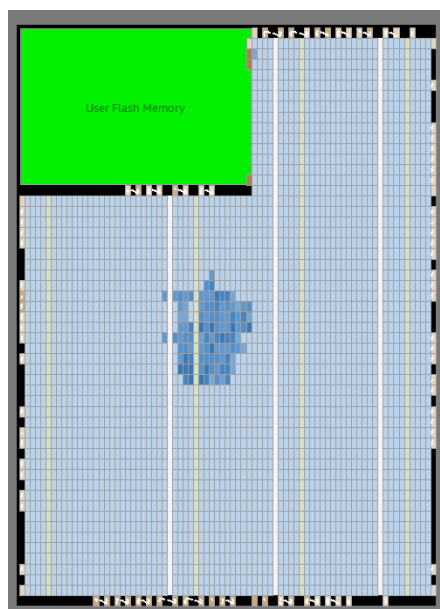


Figura 3.7: Chip Planner

Validazione sperimentale

In questo capitolo verranno descritte le simulazioni eseguite per verificare il funzionamento del codice e il setup utilizzato.

4.1 Setup

La DE10 Lite, di cui si sono usati i pin di GPIO(0) come input del segnale demodulato del reader e GPIO(1) per trasmettere la risposta del tag, è stata utilizzata insieme ad un circuito integrato. Allo switch che pilotava il carico dell'antenna si era collegato GPIO(1) e all'uscita del demodulatore GPIO(0). In Fig. 4.1 viene rappresentato lo schema più nel dettaglio.

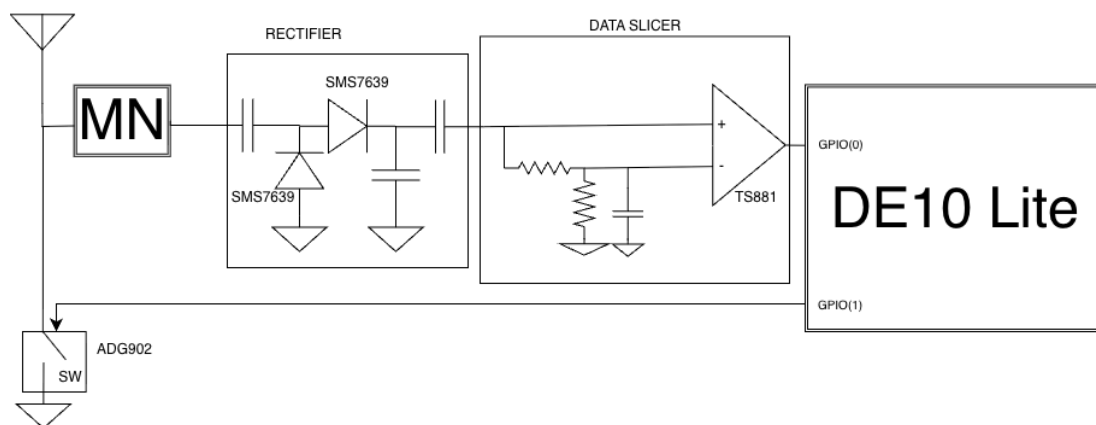


Figura 4.1: Schema a blocchi del setup utilizzato

Per la validazione sperimentale è stato utilizzato il reader Impinj Speedway Revolution R420, collegato ad una antenna patch quadrata RAIN RFID. Il reader è stato controllato tramite la versione: 1.4.1.0. del software Impinj ItemTest, da cui era possibile configurare alcuni parametri. L'antenna era posizionata fino a 2 metri dal tag, e trasmetteva con una potenza di 33dBm, data anche dal guadagno dell'antenna. I parametri del reader

sono configurabili dall'interfaccia. In particolare, è stata impiegata la modalità 1000, che prevede massima velocità di lettura e codifica FM0, abbinata con velocità di trasmissione dati massima, ma meno robusta alle interferenze. Si è inoltre scelta come sessione la Session 0 e come banco di memoria l'EPC.

4.2 Fase di identificazione del tag

Per testare la fase di identificazione del tag, si è utilizzata la sessione dedicata del software. In Fig. 4.2 viene mostrato il grafico dei segnali del reader presente su GPIO(0) e del tag su GPIO(1). Sono stati ottenuti tramite oscilloscopio e poi rielaborati tramite matlab. Per semplicità si sono indicati i vari comandi con un nome esplicativo, senza specificare la struttura.

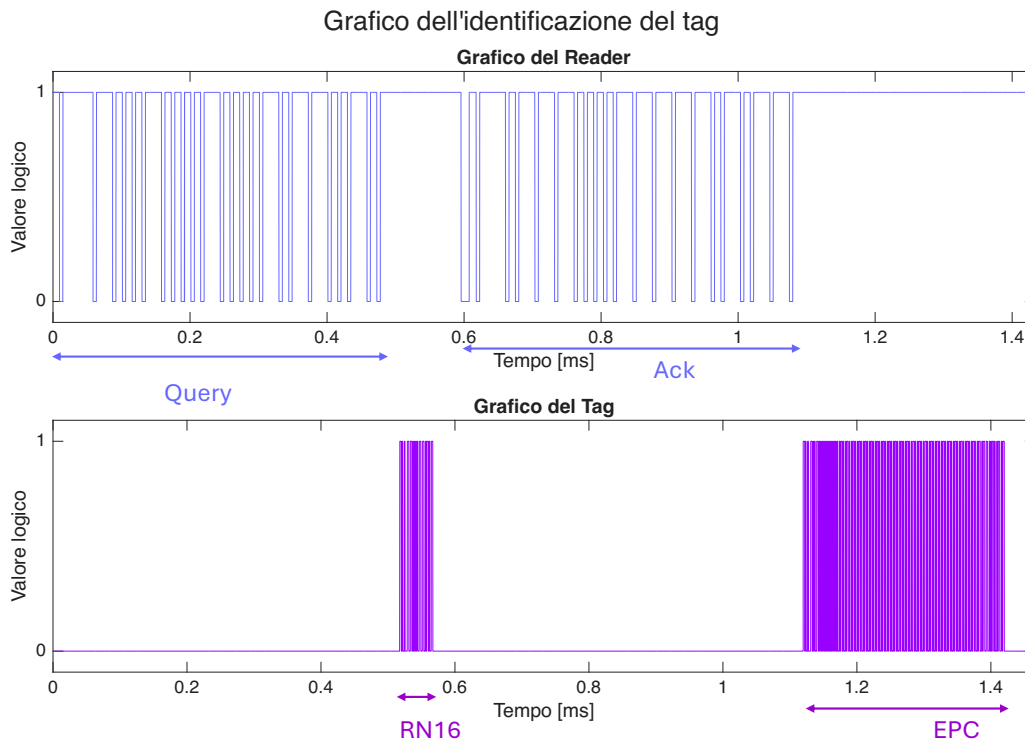


Figura 4.2: Identificazione del tag

4.3 Fase di accesso alla memoria

Una volta identificato il tag, l'interfaccia permette di accedere alla sua memoria. Per questo scopo è presente un pulsante dedicato "Read". Premendo il pulsante si

ripete tutto il processo di identificazione del tag, descritto nella sezione precedente. Una volta ricevuto correttamente l'EPC, il reader invia ReqRn, il comando Read a seguito dell'handle. In Fig. 4.3 viene riportato il grafico.

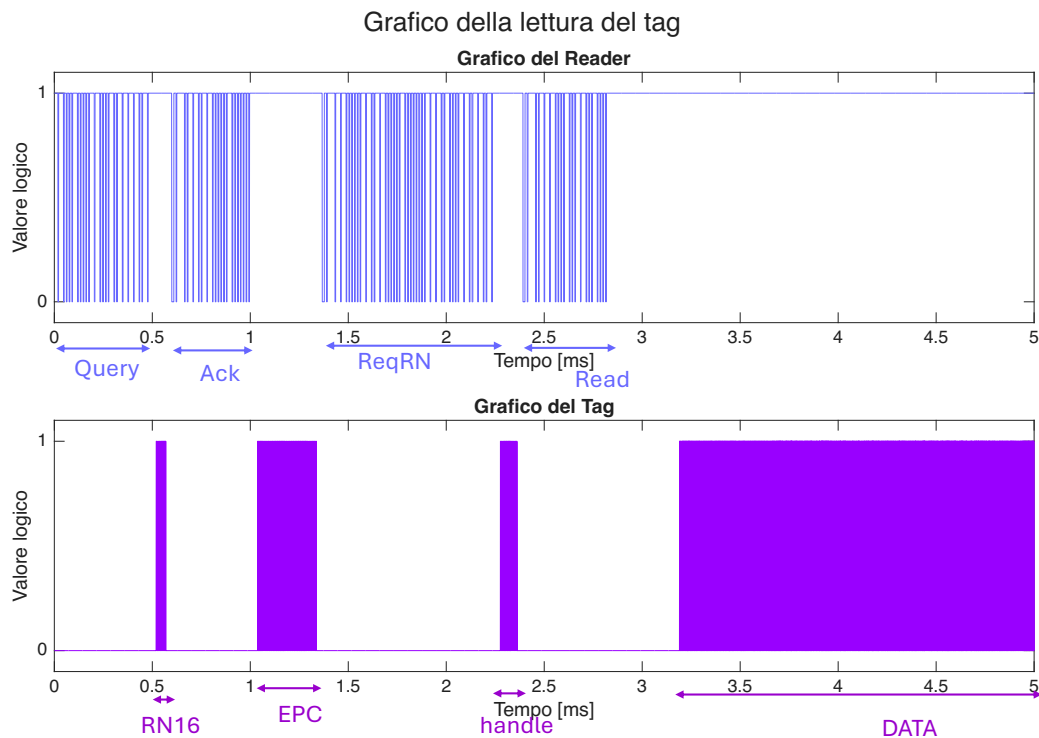


Figura 4.3: Identificazione del tag e lettura della memoria

A seguire in Fig. 4.4 viene riportato anche il messaggio che si vede dall'interfaccia, che indica quante volte il tag ha subito vibrazioni prima di essere letto dal reader, una volta inviato il messaggio il conteggio viene azzerato. L'interfaccia permette di visualizzare il messaggio tradotto con la codifica ASCII. Come si può leggere il tag è stato mosso 4 volte, "4 shocks detected". Come descritto precedentemente, tale stringa non è allocata in un vero banco di memoria, ma in una variabile. In aggiunta, il numero di urti era visibile in binario a seconda dei led accesi, questo è stato utile durante la fase di debug per valutare la stabilità e il funzionamento dell'accelerometro. Inoltre, sono anche visibili i parametri impostati per la comunicazione.

Tag Memory Access

Unknown Tag: Turn on Fast ID to identify tag model

EPC: 5555-5555-5555-5555-5555-5555 EPC Bit Length: 96

Select Memory Bank: User Use Reader: 28-229

Word Count: 9 Access Password:

Word Offset: 0

	Tag Memory								ASCII
0	3420	7368	6F63	6B73	2064	6574	6563	7465	4 shocks detected
8	6420								
16									
24									

SUCCESS! 9 word(s) read.

Read Write

Figura 4.4: Messaggio decifrato

Conclusioni

Con questo progetto di tesi si è implementato un controller digitale per tag RFID. Il codice sviluppato è modulare, rende quindi semplice modificare il codice e ampliarlo in futuro. Per esempio, si potrebbero implementare i restanti comandi del protocollo e allocare i 4 banchi di memoria. Inoltre, si potrebbe adattare il codice scritto per una particolare di tipo di applicazione industriale, tarando l'accelerometro secondo le esigenze e aggiungendo altri sensori a seconda delle necessità. La suddivisione da basso ad alto livello delle macchine a stati, differenzia i compiti rendono più facile il debug, la sua lettura e comprensione. In aggiunta, il codice sviluppato può essere anche utilizzato con altre tecnologie, per esempio potrebbe essere mappato su silicio. Le poche risorse logiche utilizzate dimostrano la compattezza dell'architettura. Inoltre, l'ottimizzazione fatta, che permette di far lavorare i registri a 1 MHz, implica un costo ridotto in termini di consumi. In questo progetto si sono quindi seguite le tipiche fasi del flusso di progettazione digitale: descrizione RTL, sintesi logica, simulazione e infine collaudo su scheda. L'attività svolta ha quindi permesso di comprendere e affrontare i problemi legati alla simulazione e validazione sperimentale, permettendo di acquisire la metodologia necessaria per affrontare questo tipo di progettazione.

Ringraziamenti

Vorrei ringraziare in ambito accademico il Prof. Aldo Romani per avermi seguito e permesso di fare questa tesi, e il correlatore Gabriele Marasca per avermi prestato aiuto nel progetto e per la stesura della tesi. Per quanto riguarda invece le persone a me vicine, voglio ringraziare coloro che mi hanno prestato supporto morale e mi sono state vicine durante questo percorso, in particolare voglio ringraziare mia mamma, che ha sempre creduto in me e mi ha cresciuto insegnandomi che con l'impegno si può raggiungere qualunque traguardo.

Bibliografia

- [1] J. Rose, A. El-Gamal, A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays", Proc. IEEE, vol. 81, n. 7, July 1993, pp. 1013.
- [2] Terasic Inc., "DE10-Lite User Manual", June 2020. [Online]. Available: https://www.mouser.com/datasheet/2/598/DE10-Lite_User_Manual-1100361.pdf?srsltid=AfmB0ooM0vEQHLiXgkuhk0jlDNIKNGsqhRmySsFbKTMqGhtevw5yKu8Y
- [3] GS1, "EPC Data Standard" (Version 1.9), November 2014. [Online]. Available: https://gs1go2.azureedge.net/sites/gs1/files/docs/epc/TDS_1_9_Standard.pdf
- [4] «What is the difference between passive and active tags?» [Online]. Available: <https://support.gs1.org/support/solutions/articles/43000733398-what-is-the-difference-between-passive-and-active-tags->
- [5] «What is the difference between low-, high-, and ultra-high frequencies?» [Online]. Available: <https://support.gs1.org/support/solutions/articles/43000734278-what-is-the-difference-between-low-high-and-ultra-high-frequencies->
- [6] «Inductive and Backscatter Coupling» [Online]. Available: <https://rfid4u.com/inductive-and-backscatter-coupling/>
- [7] GS1, "EPCTMRadio-Frequency Identity Protocols Generation-2 UHF RFID Specification for RFID Air Interface Protocol for Communications at 860 MHz – 960 MHz Version 2.0.1, Aprile 2015. [Online]. Available: https://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf

- [8] ENGINEERS GARAGE, "Linear-feedback shift register (LFSR) design in vhdl", Maggio 2019. [Online]. Available: <https://www.engineersgarage.com/feedback-register-in-vhdl/>

- [9] ANALOG DEVICES, Datasheet "ADXL345 (REV. G)". [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.PDF>