Università di Pisa

Facoltà di Ingegneria

Laurea Specialistica in Ingegneria dell'Automazione

Tesi di Laurea

Sviluppo e Confronto di Strumenti di Prova per Sensori d'Immagine

Candidato:

Niccolò Capecci

Relatori:

Prof. Michele Lanzetta

Prof. Sergio Saponara

Ing. Marcello Mulé (Alkeria S.r.l.)

Sessione di Laurea 20/12/2011

Dedica

Alla mia famiglia per avermi dato la
possibilità di raggiungere questo traguardo

Agli amici e a tutti coloro che
mi sono sempre stati vicini

A Klizia che ha saputo
sostenermi in ogni momento

Sommario

Questa Tesi descrive la progettazione e lo sviluppo di un sistema d'illuminazione a LED destinato alla verifica e alla caratterizzazione di sensori d'immagine, esaminando le problematiche emerse sia in fase di progettazione, che nella fase di test successiva alla realizzazione dei prototipi. Al fine di realizzare uno strumento che garantisse un'elevata precisione ed accuratezza, è stata analizzata e parametrizzata la caratteristica dei LED al variare delle condizioni di funzionamento. Si è proceduto a sperimentare e a confrontare varie tecniche di pilotaggio; i risultati sono esposti nella sezione finale di questo lavoro, che evidenzia pregi e difetti di ogni soluzione. Il sistema inizialmente progettato era affetto da alcune criticità, prevalentemente di natura termica, che ne rendevano difficoltoso l'utilizzo. Queste problematiche sono state in buona parte superate introducendo un sistema di retroazione, ottenendo un sistema molto più semplice da utilizzare e che fornisce risultati estremamente ripetibili.

Abstract

This thesis describes the design and development of a LED lighting system aimed to the verification and characterization of image sensors, starting from design issues up to the prototype testing phase. LED characteristics have been examined under many working conditions, in order to design a precision instrument. Some different driving techniques have been exploited and compared. The results are presented in the final section of this work, comparing benefits and flaws of each solution. The system was initially affected by well-known defects, mainly due to thermal drift, unfavourably impacting on usability and long-term repeatability. Those issues have been mostly superseded using a dedicated feedback system, resulting in a new instrument, by far more accurate and easier to operate.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli algoritmi

Capitolo 1

Introduzione

L'obiettivo di questa Tesi è stato lo studio e lo sviluppo di un sistema di illuminazione artificiale per la caratterizzazione di sensori d'immagine. Lo scopo di tale strumento è quello di realizzare un'applicazione che permetta un'analisi tecnica dei sensori al fine di stabilirne proprietà, prestazioni e qualità.

L'illuminatore fa parte di un'attrezzatura denominata Alkeria Characterization Tool, in breve A.C.T. (Figura 1.1) ed è costituita da hardware e software strettamente cooperanti. La componente hardware, oltre che dal sistema di illuminazione, è costituita da una sfera di Ulbricht, caratterizzata dalla capacità di integrare il flusso radiante che si sviluppa al suo interno e proiettare una luce uniforme sulla superficie del sensore in analisi (esistono varie tipologie di sfere, di cui vengono riportati alcuni esempi in Figura 1.2). Tale sfera è stata realizzata basandosi principalmente su articoli scientifici e documenti tecnici [4]. In [1] si afferma che la sfera da loro prodotta raggiunge un'uniformità del 98,5%, inferiore a quella raggiunta dal sistema A.C.T., che risulta pari al 99.5% [5]. La sfera prodotta permette non solo di analizzare i sensori d'immagine, ma anche di misurare il flusso luminoso prodotto dai LED, come descritto in [2].

In questo lavoro sono stati realizzati e confrontati due illuminatori, che applicano metodi diversi di alimentazione per i LED, per determinare la tecnica più appropriata. Entrambi gli illuminatori sono equipaggiati con tre LED del tipo rosso, verde e blu. In Figura 1.3 è possibile vedere un prototipo dell'illuminatore. I LED possiedono una caratteristica Corrente/Flusso Luminoso molto più lineare rispetto alle tradizionali lampade ad incandescenza, sia pur con una spiccata dipendenza dalla temperatura, una vita utile circa 10 volte superiore e un rendimento assai più elevato (quindi temperature di esercizio più contenute). Questo, unitamente alla disponibilità di LED colorati (RGB), consente di realizzare illuminatori compatti e più facilmente controllabili, che, al variare del pilotaggio possono produrre l'intensità luminosa ed il colore necessari al sistema A.C.T. Un'ulteriore evidente semplificazione progettuale derivante dalla sostituzione delle lampade ad incandescenza con i

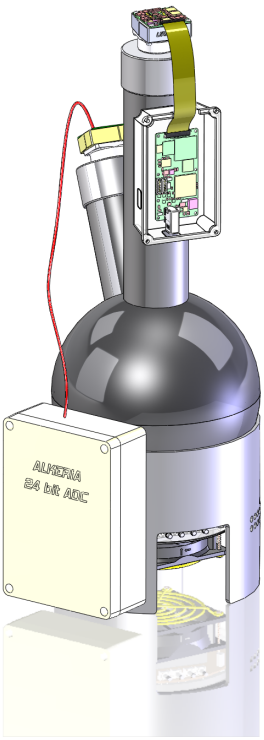


Figura 1.1: Alkeria Characterization Tool - A.C.T.

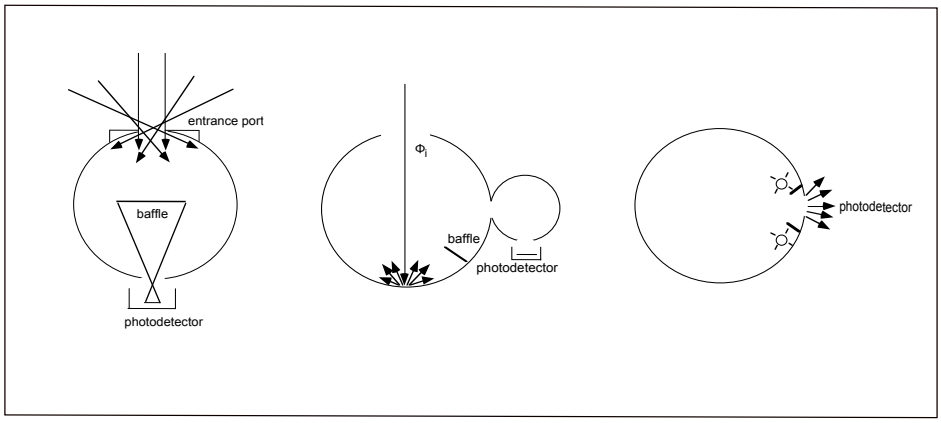


Figura 1.2: Varie tipologie di sfere integranti

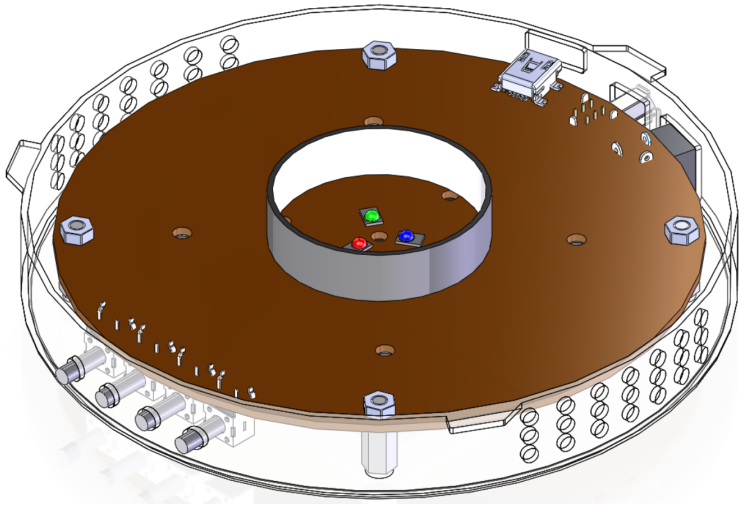


Figura 1.3: Illuminatore LED

LED, è l'eliminazione dei carousel di filtri ottici necessari a selezionare la lunghezza d'onda desiderata, essendo i LED emettitori intrinsecamente monocromatici. In letteratura esistono diversi approcci sperimentali per il pilotaggio dei LED, in corrente alternata [6] o costante [7, 8].

La prima scheda implementa un generatore di corrente variabile utilizzando un transistor MOSFET ed una catena di feedback che ne mantiene stabile l'uscita a regime. La seconda scheda pilota i LED tramite un regolatore switching, il quale effettua un controllo PWM per mantenere costante la corrente di pilotaggio. Entrambe le strategie di controllo stabilizzano a breve termine il flusso luminoso dei LED mantenendone costante la corrente, delegando la compensazione della deriva termica ad un eventuale anello di controllo esterno. Test specifici analizzeranno il comportamento delle due schede, confrontando i risultati per evidenziarne le peculiarità. Le caratteristiche ricercate sono l'accuratezza[1], la precisione[2] e la ripetibilità[3].

Sin dai primi test degli illuminatori è emerso che il comportamento del flusso emesso dai LED, non è lineare e dipende fortemente dalla temperatura d'esercizio, come analizzato in [10]. La risposta del LED ad un gradino di corrente, è costituita da un picco iniziale di luminosità, seguito da un decadimento di tipo esponenziale come illustrato in [11]. Questo ha reso necessarie alcune modifiche al progetto iniziale, mirate a realizzare un ulteriore anello di controllo che stabilizza il flusso luminoso all'interno della sfera.

L'intento è stato quello di realizzare uno strumento di misura, da utilizzare per effettuare l'analisi tecnica delle camere e standardizzarne il processo di test, consentendo di produrre misure di rumore, sensibilità, linearità e parametri analoghi

[1] La differenza tra valor medio misurato e il valore impostato

[2] La varianza dei campioni rilevati

[3] il grado di concordanza tra i campioni misurati in istanti diversi, mantenendo immutate le impostazioni e le condizioni di misura.

ripetibili e facilmente confrontabili.

Alcune delle funzionalità del sistema A.C.T.,come il dimensionamento della sfera e le lunghezze d'onda dei LED utilizzati, si ispirano ai principi dello Standard EMVA 1288 [9]. Lo standard è stato promosso da un consorzio di produttori di telecamere e nasce dall'esigenza di realizzare un protocollo per determinare in maniera univoca i parametri funzionali significativi delle camere e semplificare così la scelta del consumatore. Si evince quindi la duplice funzione del sistema A.C.T., come strumento per la creazione di datasheet comparabili con quelli degli altri produttori e come attrezzatura di verifica e sviluppo dei nuovi modelli di telecamera.

Il lavoro è stato realizzato presso l'azienda Alkeria S.r.l. di Navacchio, produttore di telecamere digitali per dispositivi industriali. Alkeria ha richiesto un'analisi critica dello standard EMVA 1288 per verificarne l'applicabilità e la creazione di uno strumento che ne implementi la metodologia di misura. Il lavoro di questa Tesi è stato condiviso con il Dott. Enrico Gastasini, che ha curato la realizzazione della parte software ed il dimensionamento della sfera di A.C.T.

Capitolo 2

Illuminatore Analogico

2.1 Introduzione

La scheda di controllo analogica è stata sviluppata presso Alkeria ed ha il compito di generare uno stimolo luminoso per i sensori d'immagine. La scheda implementa un generatore di corrente (vedi Figura 2.2) utilizzando un MOSFET ed un circuito di reazione ad amplificatore operazionale. La corrente tra il source ed il drain è controllata dalla tensione presente tra gate e source; con tensioni $V_{gs} \geq V_{th}$ il transistor è acceso e conduce, mentre per tensioni $V_{gs} < V_{th}$ il transistor commuta nello stato OFF, come descritto nella sezione 2.2.5.

Inizialmente sono state progettate due schede, una equipaggiata con tre led bianchi e l'altra con tre LED: uno rosso, uno verde ed uno blu. Successivamente, seguendo le direttive dello standard EMVA, si è deciso di utilizzare soltanto la scheda con i LED colorati (scheda RGB/analogica): lo spettro di questi ultimi è molto stretto e la lunghezza d'onda della luce generata abbastanza stabile rispetto alla temperatura di esercizio. Utilizzare lunghezze d'onda note permette di infatti determinare con precisione il numero di fotoni emessi dalla sorgente luminosa, essendo questi legati dalla formula dell'energia $E = \frac{hc}{\lambda}$ dove h è la costante di Planck, c la velocità della luce e λ la lunghezza d'onda. Non potendosi determinare con certezza la quantità di fotoni emessi dai LED bianchi, essendo questi delle sorgenti multispettrali, si è preferito quindi utilizzare sorgenti a lunghezza d'onda singola, o comunque approssimabili come tali.

Per effettuare i test sui sensori in bianco e nero, lo standard raccomanda l'utilizzo di una luce monocromatica alla lunghezza d'onda cui il sensore ottico è più sensibile (il picco della sensibilità spettrale dei sensori utilizzati in questo lavoro è intorno ai 500nm, quindi è stata utilizzata una luce verde). Per quanto riguarda le rilevazioni su sensori a colori, la sorgente luminosa deve avere più lunghezze d'onda, quindi si utilizza alternativamente il rosso, il verde ed il blu.In fase di progettazione sono stati specificati i requisiti tecnici relativi al circuito di pilotaggio dei LED:

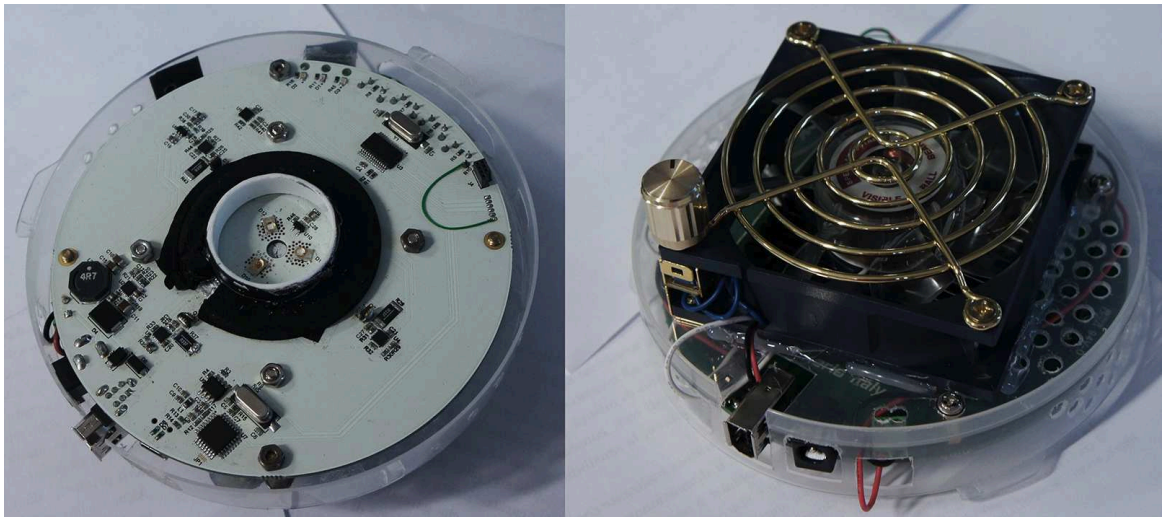


Figura 2.1: Vista superiore ed inferiore dell'illuminatore analogico

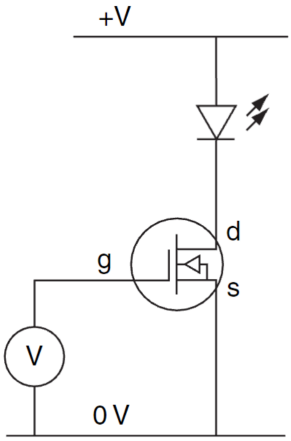


Figura 2.2: Modello semplificato del pilotaggio del LED nella scheda analogica

- Possibilità di controllare l'intensità luminosa da remoto.
- Variazione della luminosità accurata e ripetibile.
- Rilevazione della temperatura nella zona circostante i LED.

Una volta stabilite le caratteristiche del circuito, sono stati scelti i componenti della scheda che verranno di seguito illustrati.

2.2 Componenti

2.2.1 Microcontrollore PIC

L'elemento centrale nel funzionamento della scheda è il microcontrollore PIC16LF870, prodotto dalla Microchip Technology. Il dispositivo (da adesso denominato per brevità PIC) permette di eseguire una serie di funzioni che consentono il pilotaggio dei led.

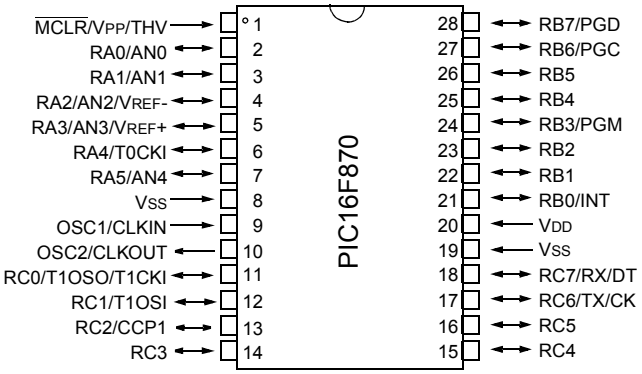


Figura 2.3: Pin-out del PIC16LF870

Le funzioni del PIC sono:

- Ricezione di un pacchetto da interfaccia seriale RS-232 contenente un valore proporzionale alla corrente da applicare ai LED.
- Comunicazione con il convertitore digitale/analogico.
- Acquisizione della temperatura, dal sensore disposto in prossimità dei LED, su richiesta remota.
- Avviso dell'avvenuta ricezione dei pacchetti.

Il diagramma di flusso che rappresenta il principio di funzionamento del PIC è visibile in Figura 2.4.

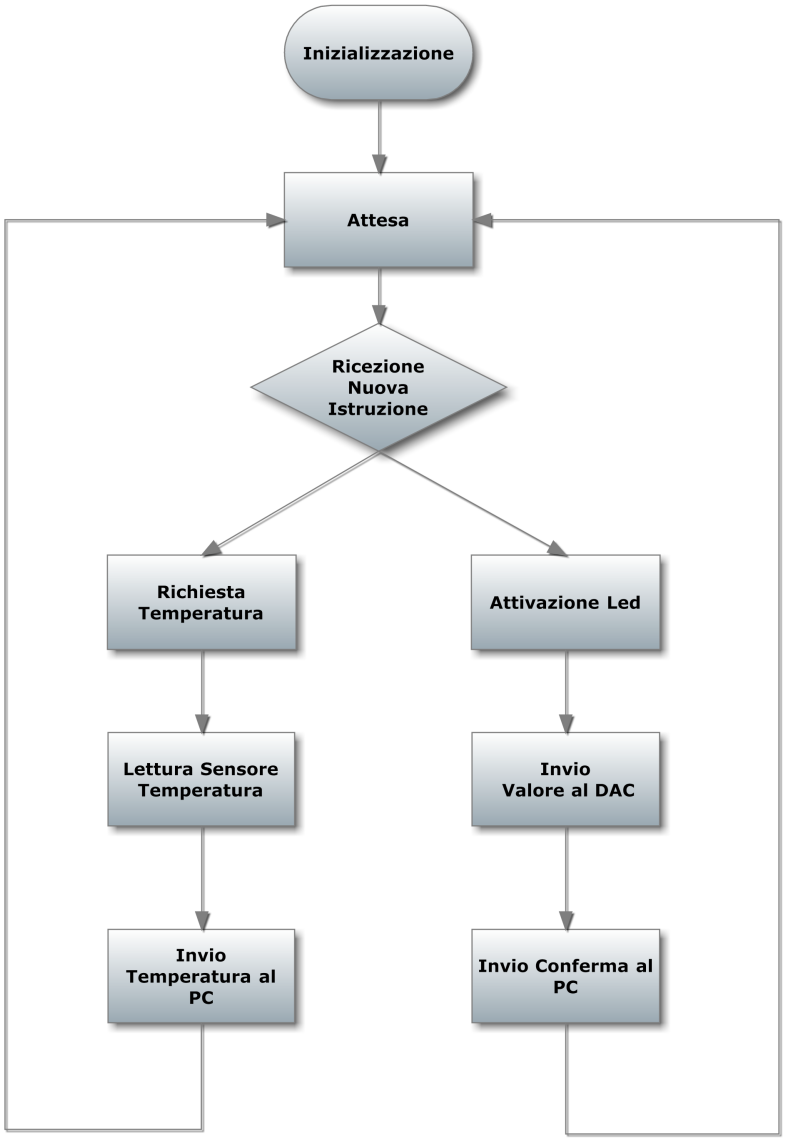


Figura 2.4: Diagramma funzionalità principali PIC

2.2.2 Convertitore Digitale Analogico

Il secondo elemento di riferimento posto nel circuito di controllo è un convertitore digitale-analogico (DAC) AD5645R della Analog Devices. Un DAC è un dispositivo elettronico in grado di generare un livello di tensione proporzionale al valore numerico che gli viene presentato in ingresso; il DAC utilizzato ha la possibilità di pilotare 3 canali indipendenti, ai quali corrispondono altrettanti pin di uscita. Nella Tabella 2.1

Parametro	Valore	Unità
Risoluzione in ingresso	14	bit
Accuratezza relativa	±2	count
Non linearità differenziale	±0.5	count
Tensione minima di uscita	0	volt
Tensione massima di uscita	2.5	volt
Risoluzione dell'uscita	152.6	µvolt

Tabella 2.1: Caratteristiche del convertitore Digitale Analogico

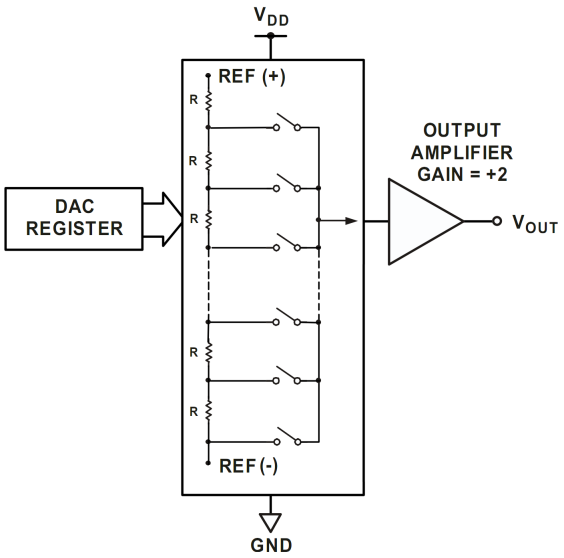


Figura 2.5: Architettura del DAC

sono riassunte le specifiche rilevanti. L'integrato utilizza una tensione di riferimento interna di 1.25 V ($Vref$) generando un'uscita full-scale di 2.5 V; il valore ideale della tensione di uscita è dato dalla formula: $V_{OUT} = 2 \times V_{REF} \times \left(\frac{count}{2^N}\right)$ dove N è la risoluzione. corrisponderanno livelli di tensione unitari da 152.6 µV (ideali).

Il convertitore al suo interno (String Ladder DAC), visibile in Figura 2.5, è composto da una stringa di più resistori della stessa dimensione (quindi di valore identico) connessi tra 2 tensioni di riferimento. I resistori formano un partitore di tensione con la Vref; tramite i bit di entrata è possibile comandare l'apertura o la chiusura degli interruttori, modificando il rapporto di partizione della tensione di riferimento. La tensione in uscita dal blocco viene moltiplicata per 2 da un successivo circuito di amplificazione e buffering. La tensione di uscita del DAC risulta quindi proporzionale alla tensione di riferimento e al numero binario in ingresso.

Il DAC è connesso al bus I2C come dispositivo slave. All'avvio, attraverso l'interfaccia di comunicazione, il PIC ne imposta il modo di funzionamento. Dopo questa operazione è possibile inviare al DAC i valori (a 14 bit) da convertire in tensione. La configurazione iniziale del modo di funzionamento avviene tramite tre pacchetti, che contengono le seguenti informazioni:

- Accensione dei tre canali A B C.
- Reset dei registri interni.
- Abilitazione della tensione di riferimento interna.

Il settaggio dei valori di uscita verrà illustrata con maggior dettaglio nella sezione 2.4.2.2.

2.2.3 Sensore di temperatura

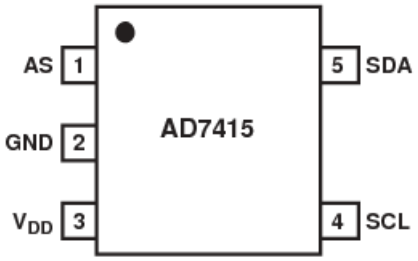


Figura 2.6: Pin-out del sensore di temperatura

Per caratterizzare la risposta dei LED in funzione della temperatura si è scelto di utilizzare il sensore integrato AD7414 della Analog Devices. Questo sensore fornisce già un'uscita digitale proporzionale alla temperatura senza che vi sia necessità di collegarlo ad un convertitore esterno. In fase di progetto si è ritenuto opportuno introdurre tale sensore per poter determinare l'influenza degli effetti termici sul decadimento del flusso luminoso nei LED. Il sensore utilizzato ha le caratteristiche elencate in Tabella 2.2.

Parametro	Valore	Unità
Risoluzione in ingresso	10	bit
Accuratezza	±0.5	gradi celsius
Temperatura Minima Rilevabile	-40	gradi celsisus
Temperatura Massima rilevabile	125	gradi celsisus
Risoluzione	0.25	gradi celsisus

Tabella 2.2: Caratteristiche del sensore di temperatura

Il dispositivo è costituito da un sensore di temperatura band gap, un convertitore ADC a 10 bit e un'interfaccia I2C.

2.2.4 Convertitore USB-Seriale

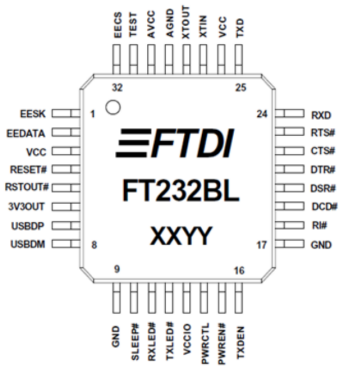


Figura 2.7: Pin-out del convertitore FTDI

È stato scelto di integrare nell'illuminatore un adattatore USB-Seriale per collegare il dispositivo al computer utilizzando un comune cavo USB. L'integrato utilizzato è il FT232BL prodotto dalla FTDI e implementa una porta seriale (COM) virtuale tramite una porta USB. Il dispositivo è stato montato in modo che fosse alimentato dall'interfaccia USB, come specificato nel datasheet. La praticità di tale dispositivo sta nella sua trasparenza: difatti sia al PIC che al PC è sufficiente predisporre una comune trasmissione seriale con parametri prefissati e il FT232BL si occupa di gestire l'intera comunicazione, trasmettendo i dati sul canale USB.

2.2.5 Circuito di alimentazione dei LED

Per regolare l'intensità luminosa è stato scelto di controllare la corrente che attraversa il LED. Questa tecnica è attuata da un circuito analogico in anello chiuso che permette di mantenere, in uscita, una corrente stabile a regime, proporzionale alla tensione in ingresso fornita dal DAC.

Il circuito in Figura 2.8, è composto essenzialmente da due amplificatori operazionali posti in retroazione e da un MOSFET, attraverso il quale passa la corrente che scorre attraverso il LED. La scheda è equipaggiata con tre canali identici di alimentazione, uno per ogni LED, i quali permettono il pilotaggio parallelo dei tre canali R, G e B. Successivamente, nella sezione 2.2.6, verranno illustrati l'analisi circuitale ed i comportamenti nel transitorio ed a regime.

Passiamo adesso ad illustrare le caratteristiche dei componenti.

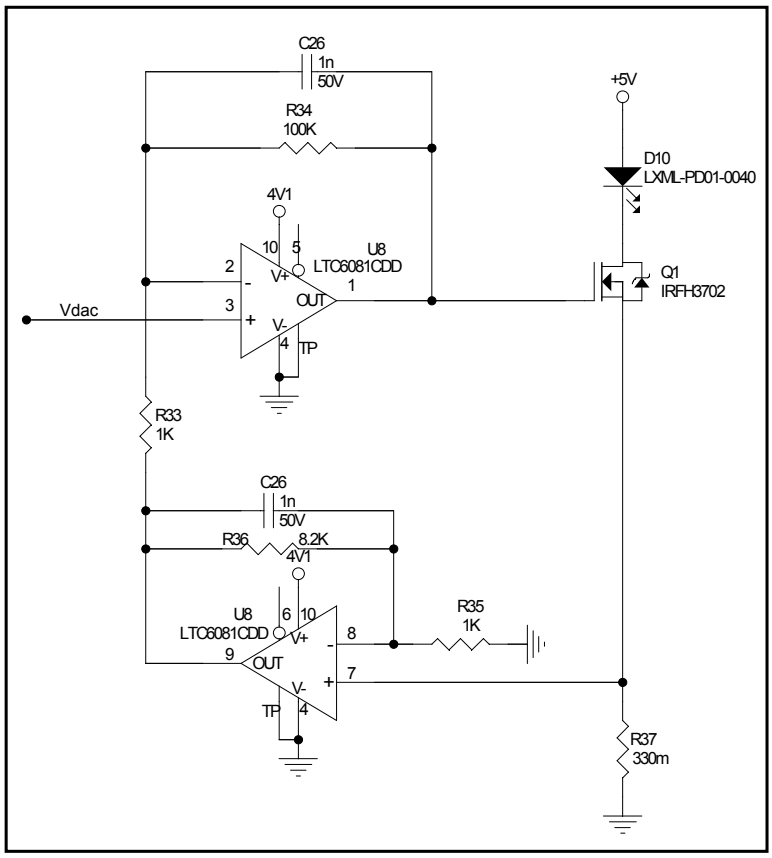


Figura 2.8: Circuito di alimentazione dei LED dell'illuminatore analogico

2.2.5.1 Amplificatore Operazionale

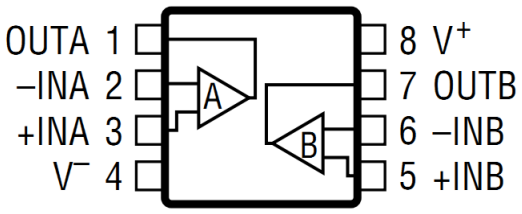


Figura 2.9: Pin-out dell'amplificatore operazionale

L'amplificatore utilizzato è un LTC6081 della Linear Technology. Le caratteristiche principali sono:

- Ingresso ed uscita Rail-to-Rail (escursione della tensione di ingresso ed uscita pari a quella di alimentazione)
- Guadagno di tensione in anello aperto di 120dB
- Prodotto guadagno-banda di 3.6 MHz
- Offset in ingresso di 70 μV

- Slew Rate di 1V/uS

Compito dell'amplificatore (sommatore) nella catena diretta è pilotare il gate del MOSFET in modo da far variare la corrente sul diodo proporzionalmente al valore della tensione in ingresso impostata tramite il DAC. L'amplificatore posto in retroazione si comporta da convertitore di corrente-tensione, fornendo in uscita una tensione proporzionale alla corrente in ingresso.

2.2.5.2 Diodo ad emissione luminosa

Figura 2.10: LED Luxeon Rebel

L'illuminatore dispone di 3 LED (light emitting diode) Luxeon Rebel prodotti da Philips, che sono caratterizzati da robustezza e affidabilità. Le principali caratteristiche sono illustrate in Tabella 2.3.

Famiglia	LXML		
Modello	PM01-0070	PB01-0023	PD01-0040
Colore	Verde	Blu	Rosso
Flusso Radiante a 700 mA	123 mW	48 mW	85 mW
Lungh. d'onda dominante	530 nm	470 nm	627 nm
Tensione di soglia a 350 mA	3.05 V	2.55 V	2.31 V
Tensione di soglia a 700 mA	3.40 V	3.40 V	3.60 V

Tabella 2.3: Caratteristiche dei LED

Come vedremo nell'analisi circuitale (sezione 2.3.4) è necessario tenere conto che il flusso radiante dei LED è funzione della temperatura di giunzione.

2.2.5.3 Mosfet

Il transistor scelto, un IRFH3702 della International Rectifier, appartiene alla categoria dei Power MOSFET e permette di sostenere alti valori di tensione e corrente con guadagno in corrente idealmente infinito ed un basso valore della $R_{DS(ON)}$. L'applicazione di una tensione al gate V_{GS} permette di controllare il passaggio di cariche

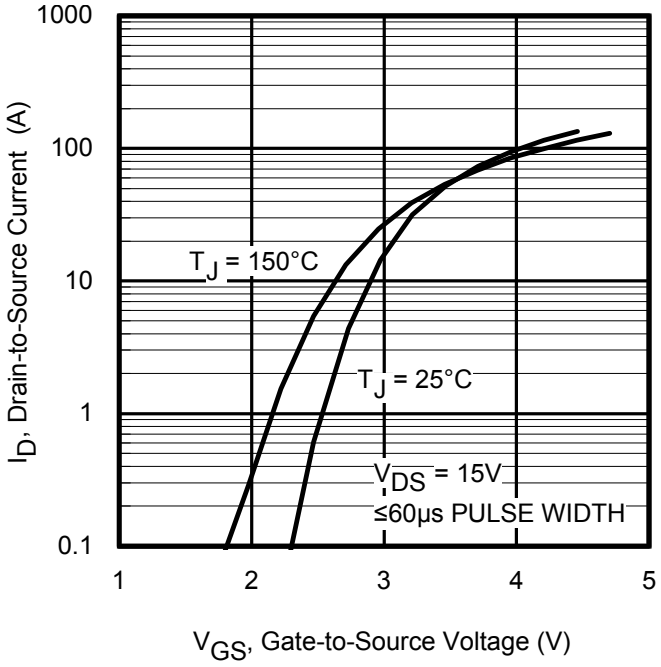


Figura 2.11: Caratteristica I_D/V_{gs} del MOSFET IRFH3702

tra il source e il drain I_D, e quindi la corrente elettrica che attraversa il dispositivo, che scorre anche nel LED.

2.2.6 Analisi del comportamento del circuito di alimentazione dei LED

Prima di costruire la scheda ne è stato realizzato un modello per simulare, tramite il software LTSpice, il comportamento del circuito di pilotaggio dei LED, in modo da verificare che il circuito avesse un comportamento stabile in tutto l'intervallo di funzionamento.

Per studiare la stabilità del sistema in ciclo chiuso, è stato interrotto l'anello nel punto corrispondente l'ingresso sommatore del convertitore tensione-corrente, introducendo un generatore di tensione. In Figura 2.13 è presente il diagramma di Bode, il sistema è da considerarsi stabile ed ha le seguenti caratteristiche:

- Banda passante di 1.5KHz.
- Margine di fase di 40°
- Margine di guadagno di 20 dB
- Guadagno di 46 dB

La cuspide presente a 5 MHz può generare una risposta oscillatoria per gradini molto rapidi, tuttavia non è stata considerata dannosa per il sistema nelle frequenze in cui opera.

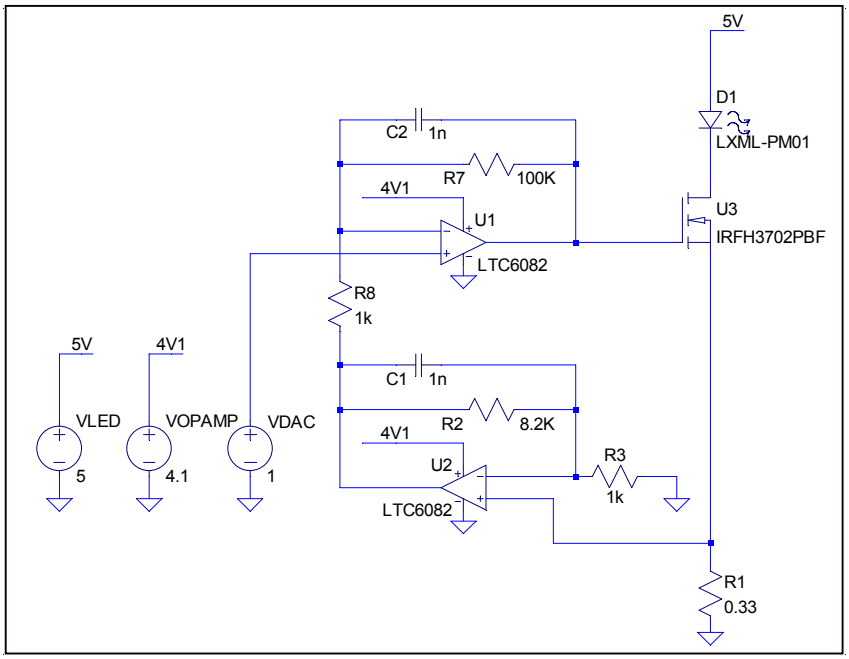


Figura 2.12: Modello LTSpice del circuito di pilotaggio LED dell'illuminatore analogico

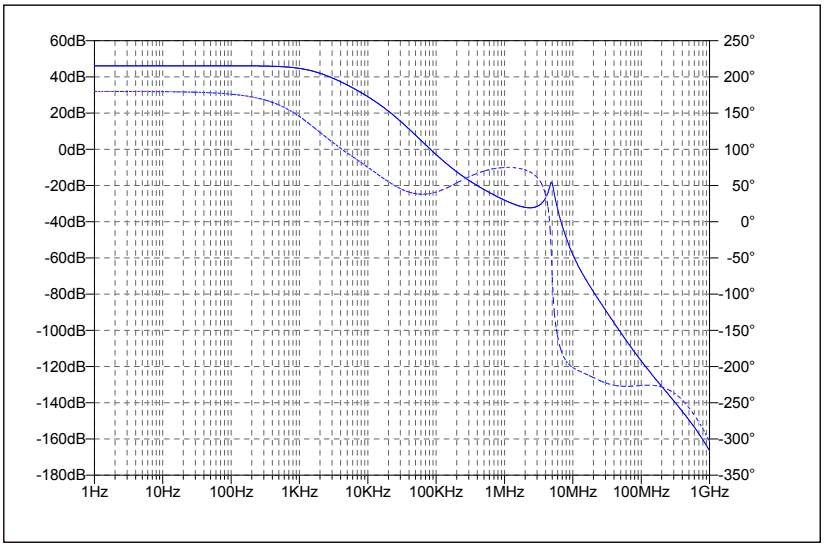


Figura 2.13: Diagramma della fase (linea tratteggiata) e del modulo (linea continua)

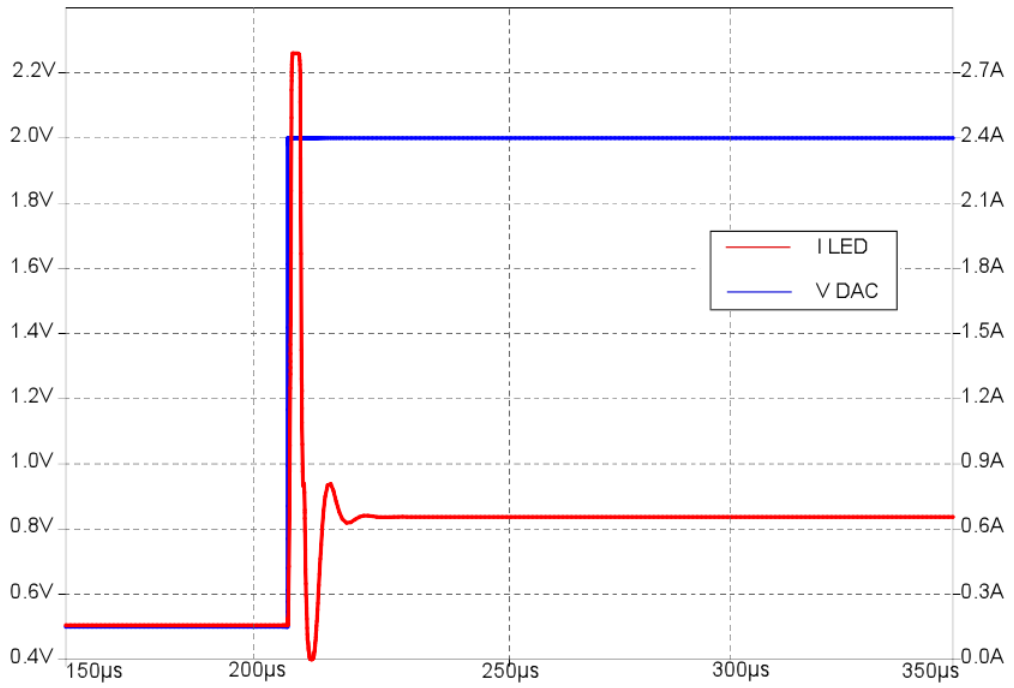


Figura 2.14: Risposta ad un gradino con $V_{DAC} = 2V$ simulata tramite LTSpice

In Figura 2.14 è riportata la risposta al gradino simulata tramite LTSpice, la *"I LED"* è stata misurata ai capi della resistenza da 0.33 ohm, mentre la *"V DAC"* è la tensione di uscita del DAC. Il sistema raggiunge il valore di regime in circa 20us; le forti sovraelongazioni sono dovute all'elevato guadagno d'anello, allo stadio di uscita asimmetrico a singolo MOSFET e alla limitata velocità di risposta[1] (slew-rate) degli amplificatori. Il gradino di tensione generato dal DAC, con tempo di salita molto piccolo, causa la saturazione dello stadio differenziale d'ingresso dell'operazionale collegato al DAC. L'elevato guadagno d'anello porta il MOSFET ad erogare una sovracorrente nel LED che, a causa del ridotto slew-rate degli operazionali, non viene immediatamente compensata. Si innesca quindi un fenomeno oscillatorio, che comunque si attenua molto velocemente (in circa 20us). Tale fenomeno non è stato ritenuto problematico per il pilotaggio dei LED; in Figura 2.15 è possibile vedere il comportamento del circuito reale. Lo stesso fenomeno oscillatorio è presente anche nei gradini di tensione verso il basso, simulati e reali.

Al fine di limitare il fenomeno della sovraelongazione della corrente è possibile rallentare le variazioni di tensione ai capi dell'amplificatore operazionale. Questo si ottiene rallentando l'uscita del DAC, ossia presentandogli una rampa di valori digitali anziché un gradino istantaneo. In Figura 2.18 è dimostrato che inviando rampe di tensione con pendenza inferiore allo slew-rate, ossia rispettando il vincolo di 1 V/us, le oscillazioni dell'uscita scompaiano.

[1]Lo slew-rate è una misura che indica la massima velocità di salita con cui il dispositivo varia la propria uscita in risposta ad una sollecitazione al gradino.

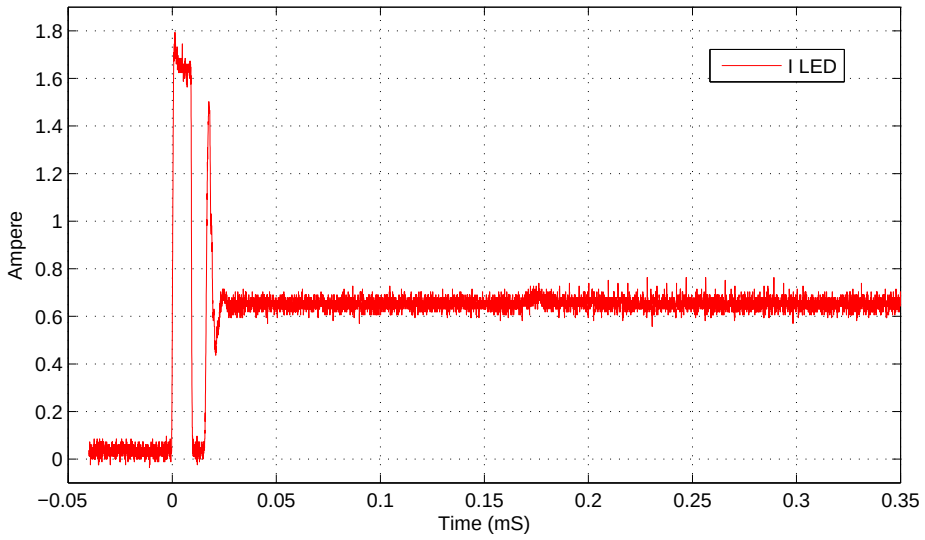


Figura 2.15: Risposta ad un gradino con $V_{DAC} = 2V$ acquisita tramite oscilloscopio

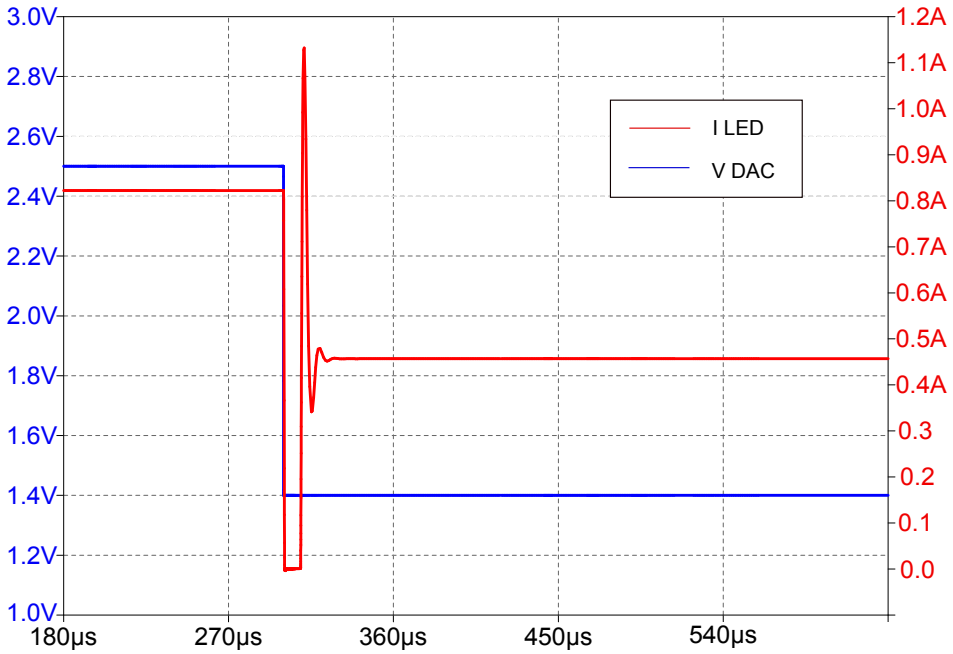


Figura 2.16: Simulazione della risposta ad un gradino verso il basso

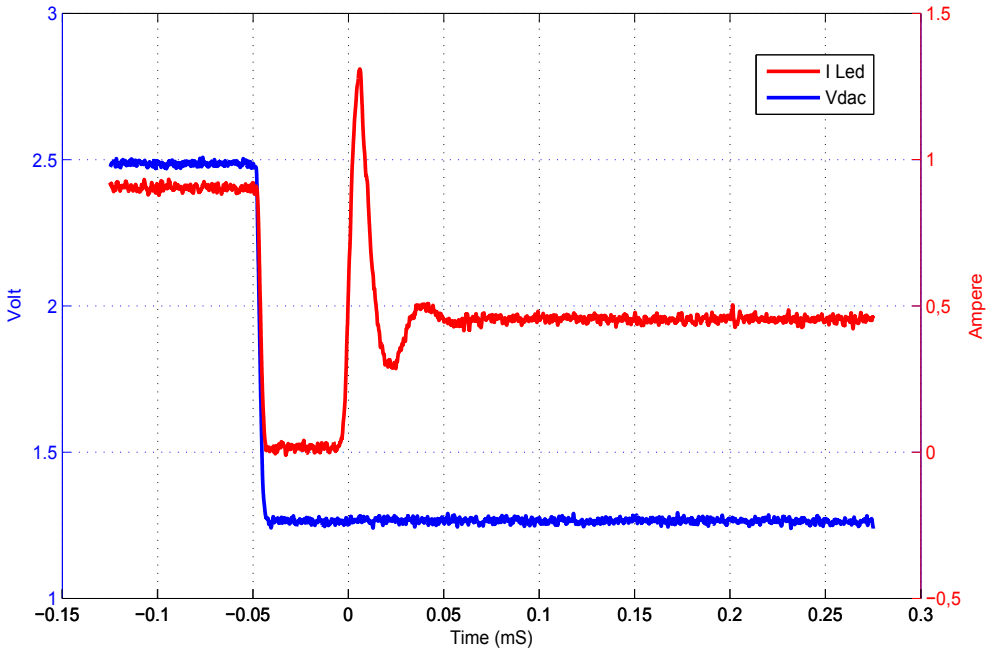


Figura 2.17: Risposta ad un gradino verso il basso acquisita tramite oscilloscopio

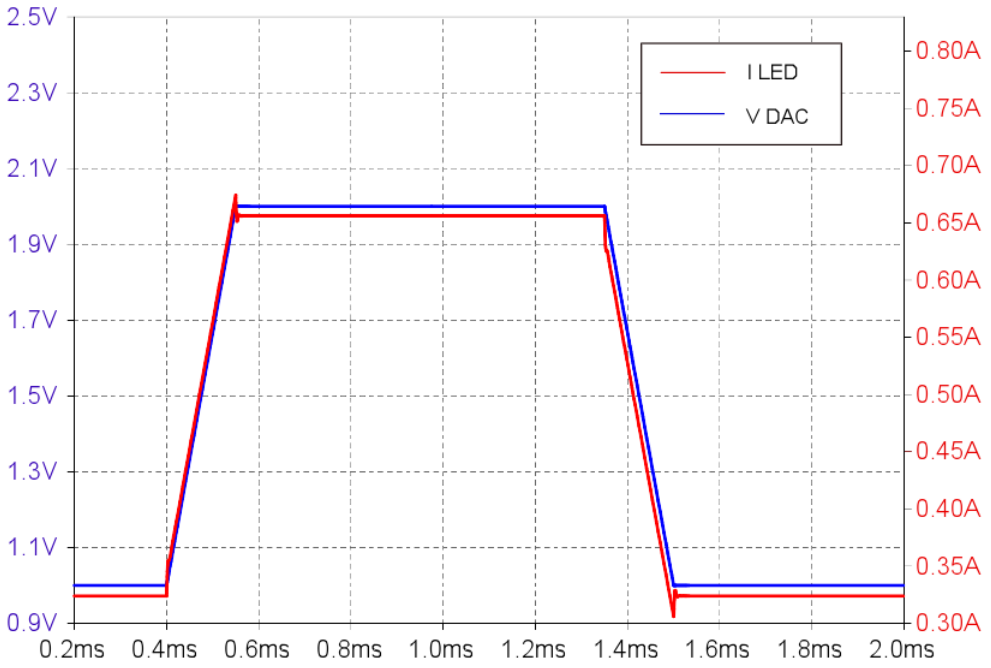


Figura 2.18: Simulazione della riduzione dell'effetto dello slew-rate

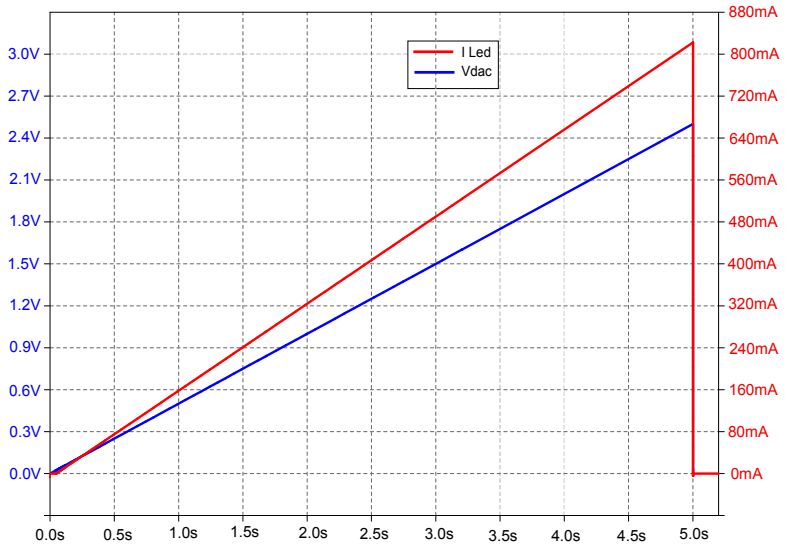


Figura 2.19: Caratteristica di uscita del circuito di pilotaggio simulato.

Per effettuare l'analisi della linearità dell'uscita del circuito e verificare il range di corrente generato, è stata inviata una rampa al circuito di amplificazione da 0V a 2.5V che corrisponde al range di uscita del DAC. Come si può vedere, i risultati del modello garantiscono una corrente massima di circa 820mA che permette di utilizzare l'intera dinamica dei LED. Confrontando la simulazione con i valori rilevati sulla scheda reale è possibile confermare che il modello è completamente verificato.

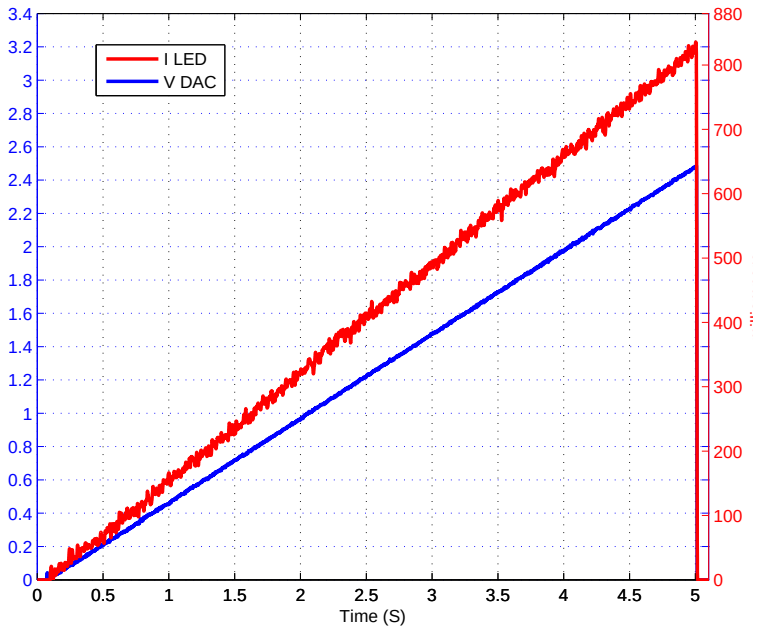


Figura 2.20: Caratteristica di uscita del circuito di pilotaggio reale

2.2.7 Altri Componenti

La scheda è stata equipaggiata inoltre con:

- Connettore USB
- Connettore di alimentazione
- Alimentatore Switching da 5 V
- Regolatore di tensione da 4.1 V
- LED di stato e pulsanti collegati al PIC

2.2.8 Schema elettrico

In seguito alla scelta dei componenti ed alla simulazione del circuito di amplificazione è stato realizzato lo schema elettrico del dispositivo visibile in Figura 2.21 ed in Figura 2.22, secondo il quale è stato disegnato il PCB e sono state realizzate le schede.

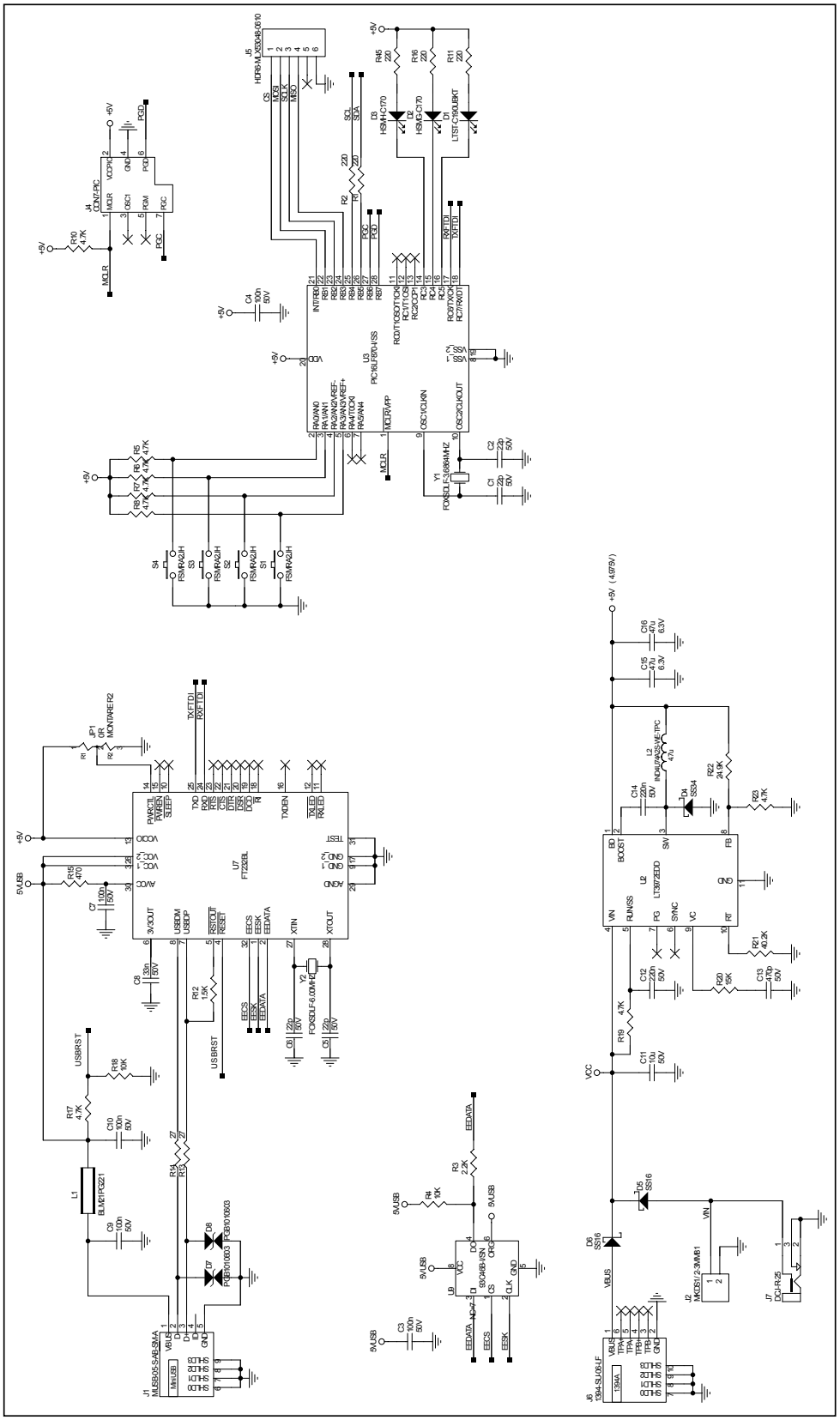


Figura 2.21: Schema elettrico dell'illuminatore analogico (parte 1)

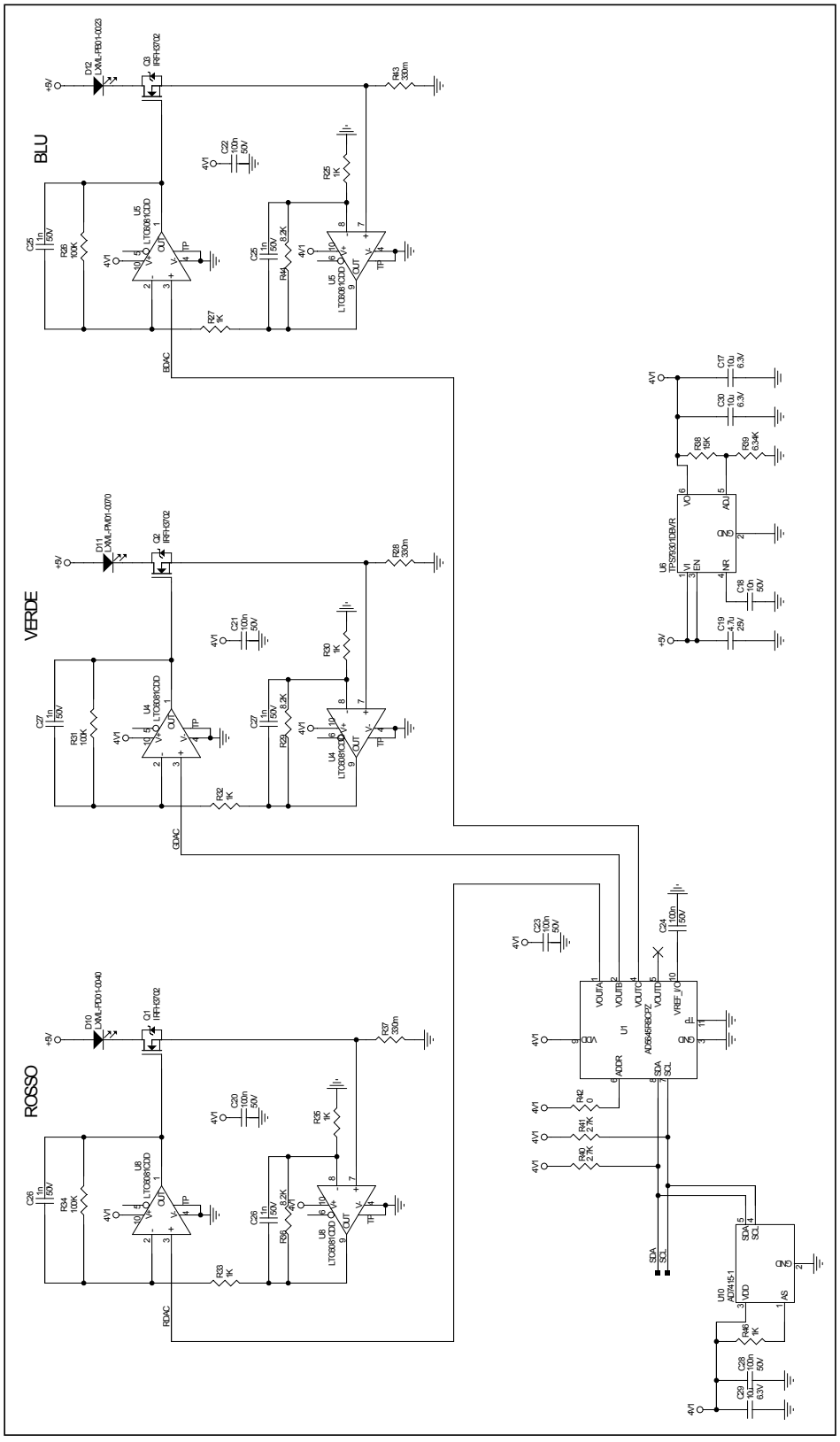


Figura 2.22: Schema elettrico dell'illuminatore analogico (parte 2)

2.3 Compensazione della luminosità

2.3.1 Strumentazione

Al fine di testare e validare il corretto comportamento della scheda prodotta è stato necessario ricorrere ad alcuni strumenti. Per la verifica e la calibrazione dell'irradianza è stato impiegato un fotodiodo, un componente che trasforma la luce incidente sulla sua superficie sensibile in corrente. Il fotodiodo utilizza infatti l'effetto fotoelettrico, il fenomeno che giustifica l'emissione di elettroni da parte di un materiale colpito da una radiazione elettromagnetica ad una certa frequenza.

Il fotodiodo, è stato inserito all'interno della sfera allo scopo di misurare la potenza incidente sull'unità di superficie (irradianza)$[W/m^2]$ dalla quale è possibile ricavare il numero di fotoni che l'hanno colpita. Poiché il fotodiodo riceve solamente fotoni ad un'unica lunghezza d'onda (nel caso sia attivato un solo led) è possibile ricavarne il numero tramite le seguenti espressioni.

Come illustrato precedentemente, l'energia associata ad un fotone $E_p = \frac{hc}{\lambda}[J]$[2] dipende dalla sua lunghezza d'onda λ. A_{px} della telecamera in un certo tempo di esposizione t_{esp}. Dal fotodiodo ricaviamo la potenza superficiale $P_{sup}\left[\frac{W}{m^2}\right]$; il numero medio di fotoni incidenti su un pixel $\mu_p = \frac{P_{sup} \cdot A_{px} \cdot t_{esp}}{E_p}$ è dato dal rapporto tra energia misurata e l'energia del singolo fotone che, sotto il profilo dell'analisi dimensionale, risulta ovviamente adimensionale. Il fotodiodo scelto è il Vishay BPW34, che presenta un intervallo di sensibilità tra i 450nm e 1050nm, quindi sufficiente a coprire le lunghezze d'onda di tutti i LED utilizzati.

2.3.2 Circuito di amplificazione del fotodiodo

Il fotodiodo è stato montato su un circuito di conversione corrente-tensione realizzato per mezzo di un amplificatore operazionale con ingressi a FET. Il modello di amplificatore utilizzato è il TLC2202CD della Texas Instruments. Come è possibile vedere dalla Tabella 2.5 questi amplificatori presentano una corrente di polarizzazione di solo 1 pA. Grazie a questo e alle bassissime correnti e tensioni di rumore è stato possibile rilevare segnali dell'ordine di 20 nA fondo scala. In Tabella 2.5 sono riassunte le caratteristiche dell'amplificatore. In Figura 2.23 è visibile il circuito caratterizzato da una resistenza di misura pari a 1MΩ.

[2]Con $h = 6.626 \cdot 10^{-34}\,[Js]$costante di Planck e $c = 2.997 \cdot 10^{8}\,\left[\frac{m}{s}\right]$velocità della luce.

Parametro	Valore	Unità
Rumore sulla corrente d'ingresso	0.6	fA√Hz
Rumore sulla tensione d'ingresso	30	nV√Hz
Tensione di offset	500	µV
Stabilità termica	0.5	µV/C°
Drift	0.005	µV/mese
Corrente di bias	1	pA

Tabella 2.5: Caratteristiche del fotodiodo

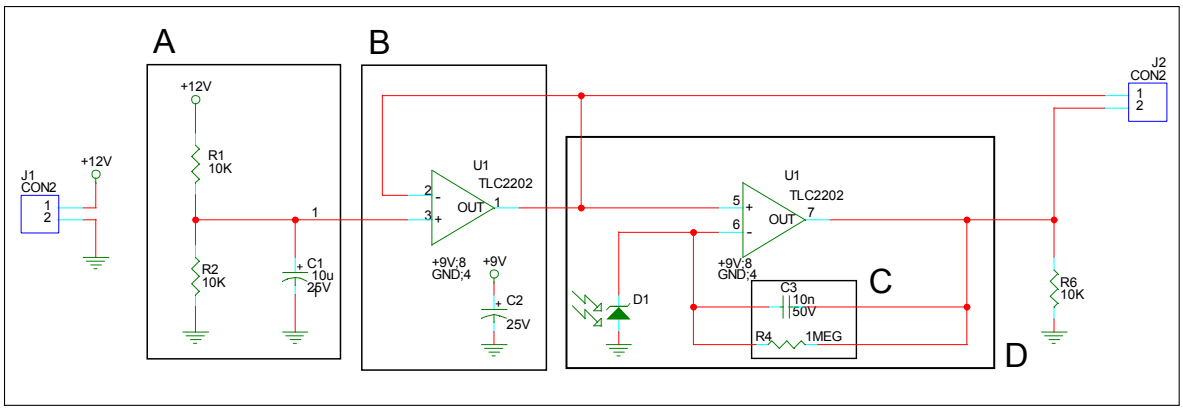


Figura 2.23: Circuito di pilotaggio del fotodiodo

In parallelo alla resistenza R4 è disposto un condensatore da 10nF (Figura 2.23C). Il circuito ha una costante di tempo pari a $\tau = RC = 1M\Omega \cdot 10nF \approx 0.01s$, quindi in circa 50ms raggiunge il valore di regime, mentre la banda del circuito è $BW = \frac{1}{2\pi RC} \approx 16Hz$. In Figura 2.23 è presente una massa virtuale con un partitore che crea un riferimento stabile a metà della tensione di alimentazione del circuito, quindi a 6V. Questo si è reso necessario poiché il sistema dispone di una sola tensione di alimentazione. È presente inoltre un buffer che diminuisce l'impedenza della massa virtuale (Figura 2.23B). La parte D della Figura 2.23 si comporta come un convertitore corrente-tensione, trasformando la corrente inversa che circola nel fotodiodo D1 in tensione secondo la dinamica appena illustrata. L'operazionale permette di rendere la rilevazione indipendente dall'impedenza d'ingresso del sistema di misura adottato.

2.3.3 Calcolo della formula caratteristica del fotodiodo

Il fotodiodo presenta una sensibilità tra i 450nm ed i 1050nm, ma la sensitività non è costante e varia secondo il grafico in Figura 2.24. Dalla figura è possibile ricavare il coefficiente di correzione della sensibilità spettrale (il picco di sensibilità del dispositivo è nell'infrarosso). I coefficiente estratti sono riportati in Tabella 2.6.

Colore	Lunghezza d'onda	Coefficiente di correzione
Rosso	632nm	2.06
Verde	535nm	3
Blu	475nm	5.3

Tabella 2.6: Coefficiente di di correzione per i LED utilizzati

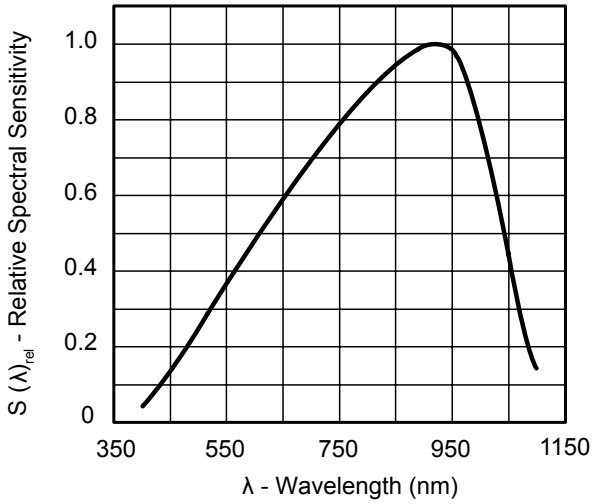


Figura 2.24: Sensibilità spettrale del fotodiodo Vishay BPW34 al variare della lunghezza d'onda

Per quanto riguarda la caratteristica di uscita, o meglio il rapporto tra l'irradianza sul piano del sensore e la corrente inversa che scorre nel fotodiodo, si rimanda alla Figura 2.25. Una volta estratti i punti noti dal grafico e ponendo $I_{inv} = I_{mis} \cdot c_{BPW34}(\lambda)$ (dove I_{mis} è la corrente misurata ai capi della resistenza del circuito di alimentazione e $c_{BPW34}(\lambda)$ è il coefficiente di correzione in funzione della lunghezza d'onda) è possibile ricavare l'equazione che risulta $I_{inv} = P_{sup} \cdot 10^{1.69897}$. La formula utilizzata per calcolare la potenza superficiale e determinare il numero di fotoni è $P_{sup} = \frac{I_{Inv}}{10^{1.69897}}$.

2.3.4 Dipendenza dell'intensità luminosa dalla temperatura

Il rilevatore a fotodiodo è stato utilizzato anche per effettuare il monitoraggio a lungo termine dell'intensità luminosa e per determinarne l'entità della dipendenza dei LED dalla temperatura.

L'illuminatore è dotato di un dissipatore metallico a contatto con la superficie sottostante i tre LED, che permette di disperdere il calore generato durante l'utilizzo. Attivando il LED alla massima potenza, mantenendo costante la corrente e misurando l'uscita del fotodiodo con un multimetro Agilent 34410A, si verifica che il flusso luminoso rilevato risulta affetto da un fenomeno di drifting che fa decrescere

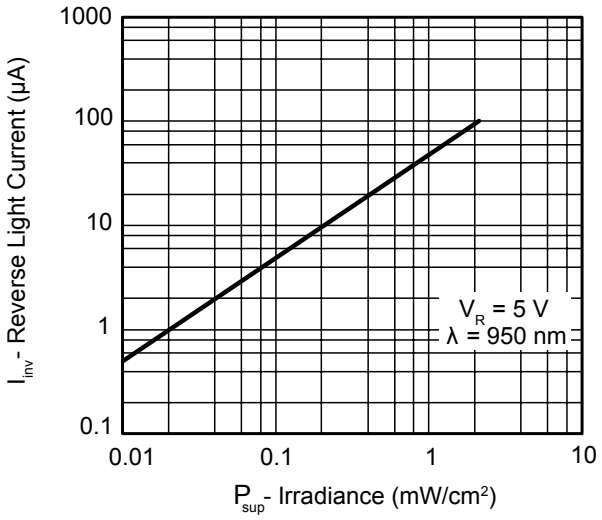


Figura 2.25: Curva caratteristica del fotodiodo Vishay BPW34

la tensione misurata ai capi del fotodiodo. L'aumento della temperatura genera, infatti, un decadimento del rendimento di conversione del LED, che produce una diminuzione dell'emissione luminosa.

Il comportamento illustrato in Figura 2.26 è suddivisibile in due sezioni. Il segmento iniziale è caratterizzato da un veloce decadimento del flusso luminoso, di circa 2 mV in 10 secondi nel caso peggiore, fino ad un valore prossimo a quello di regime. Questo fenomeno, caratterizzato da una costante di tempo τ_t, può variare in base al tipo di LED utilizzato, alla temperatura iniziale e dall'ampiezza del gradino. La sezione a regime è caratterizzata invece da un lento decadimento, con costante di tempo τ_r, di circa 30 uV ogni 10 secondi, causato dal continuo innalzamento della temperatura del LED.

Per comprendere come questi due fenomeni possano disturbare le misure effettuate sulle telecamere, prendiamo in esame due scenari di test ordinari. In un primo caso esiste la necessità di acquisire con la telecamera valori diversi di luminosità. La telecamera effettuerà un'acquisizione per ciascuno di essi. Il risultato sarà fortemente pregiudicato dalla costante di tempo τ_t , dovuta al transitorio iniziale della risposta ai gradini di tensione da parte del LED. Supponiamo invece di voler far acquisire alla telecamera una serie di valori a distanza temporale prefissata, mantenendo la luminosità costante. Anche se aspettassimo l'esaurimento del fenomeno iniziale (e questo di per sé comporterebbe l'attesa, per ogni valore misurato, di circa 10 secondi), i risultati sarebbero influenzati dalla deriva termica della sezione a regime l'errore commesso sarebbe proporzionale alla durata del test.

Questi comportamenti risultano inaccettabili, quindi è stato necessario introdur-

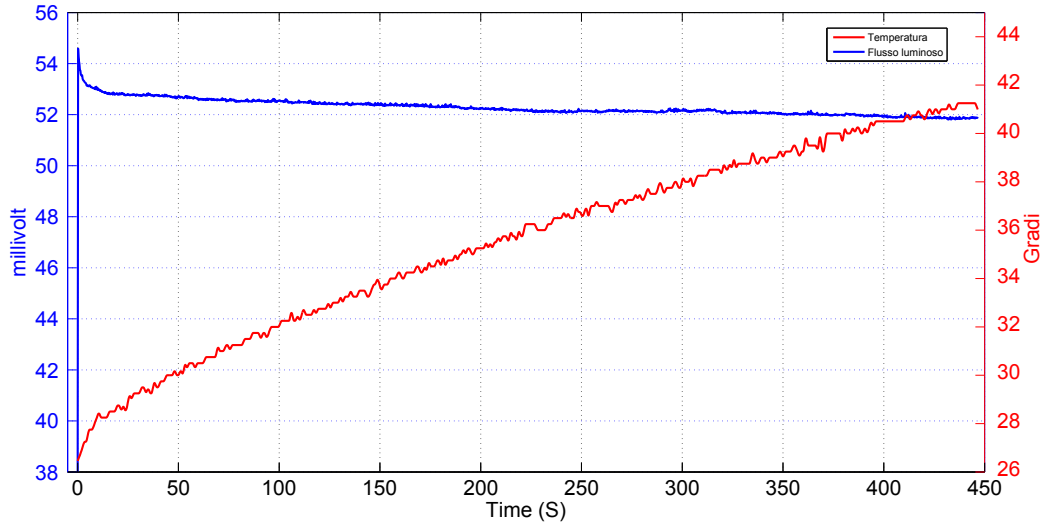


Figura 2.26: Deriva termica dell'illuminatore analogico in anello aperto

re un metodo di controllo della luminosità che garantisse una buona stabilità e tempi di assestamento accettabili dei valori di irradianza sulla porta di uscita della sfera. Si è scelto di controllare a ciclo chiuso la luminosità dei LED utilizzando la tensione ai capi del fotodiodo, letta utilizzando un ADC, come segnale di errore per la retroazione. Un primo approccio prevedeva il collegamento dell'ADC al computer, per poi interpretare i dati ad alto livello, ma il metodo è risultato inefficiente dal punto di vista della latenza introdotta dall'interfaccia seriale-USB. Si è provvisto quindi al collegamento diretto dell'ADC al microcontrollore già presente sull'illuminatore.

2.3.5 Controllo integrato del fotodiodo

Per consentire al PIC di misurare la tensione necessaria dal rivelatore a fotodiodo è stato necessario utilizzare un convertitore analogico digitale (ADC). Dato che la scheda dell'illuminatore analogico era già stata prodotta, e riprogettare una nuova scheda contenente l'elettronica per l'ADC sarebbe stato eccessivamente oneroso, è stato scelto di acquistare una scheda esterna già predisposta per l'acquisizione dati. Le due schede sono state collegate attraverso un'interfaccia di comunicazione.

È stato scelto un "Evaluation Kit" della Maxim equipaggiato dal convertitore a 24 bit MAX11210. La scheda, visibile in Figura 2.27, è fornita di alimentatori isolati dai disturbi della linea di alimentazione, e di isolatori ottici per la comunicazione con l'ADC. La scheda è fornita di un microcontrollore che permette di comunicare tramite interfaccia USB con L'ADC, ma questa configurazione è stata modificata in modo da far comunicare il PIC della scheda analogica direttamente con l'ADC. Il fotodiodo è stato collegato, tramite il proprio circuito di amplificazione, all'ingresso positivo (*AINP*) e negativo (*AINN*) del convertitore, mentre il PIC è stato connesso all'interfaccia optoisolata SPI. Una descrizione dettagliata della comunicazione e del

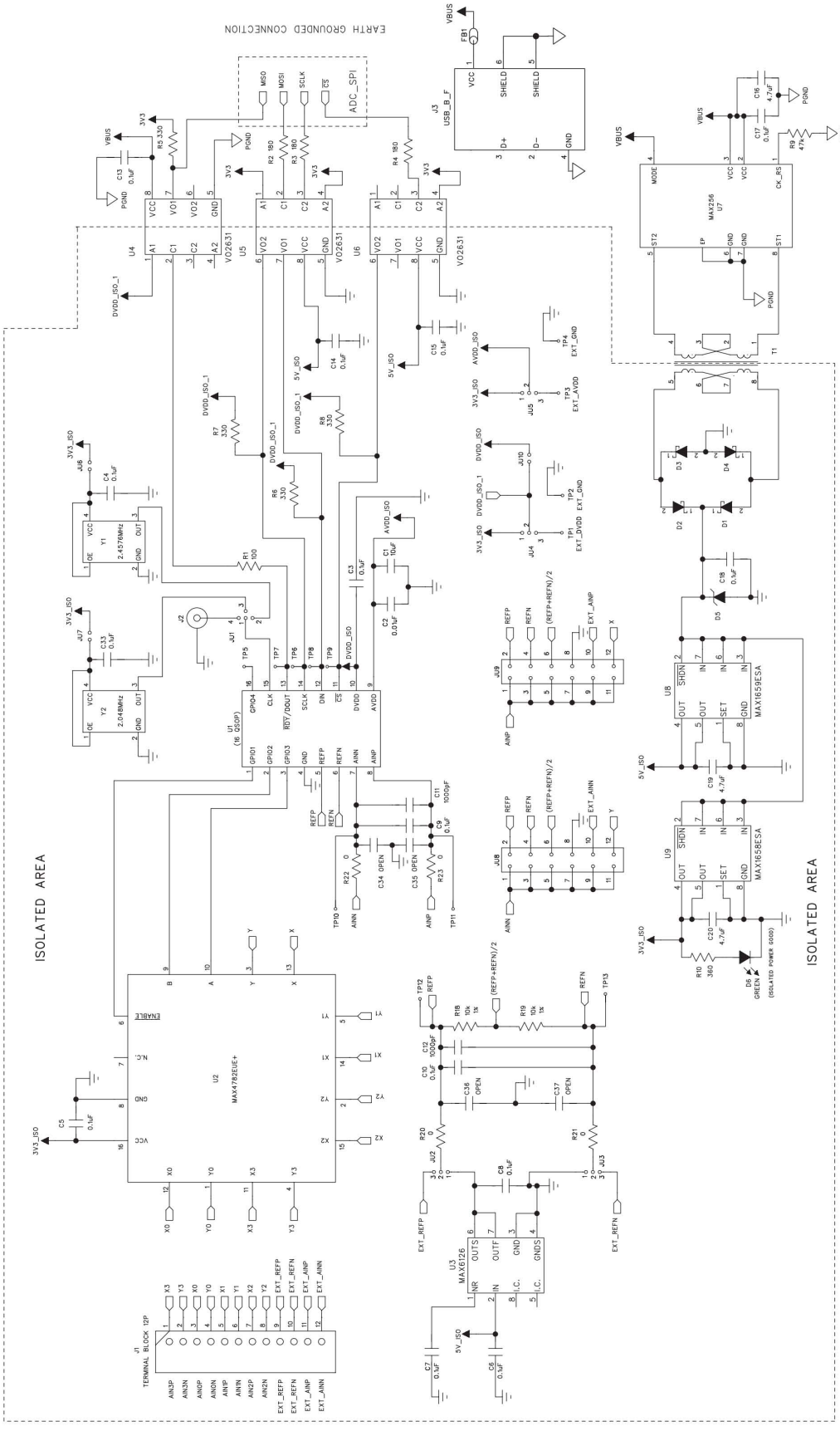


Figura 2.27: Schema elettrico della scheda ADC

pilotaggio dell'ADC è rimandata alla sezione 2.4.2.3. Questa predisposizione ha reso possibile al PIC acquisire la tensione generata dal circuito di amplificazione del fotodiodo.

2.4 Firmware Pic

2.4.1 Introduzione

Il codice si sviluppa in due parti, chiamate modalità normale e modalità in ciclo chiuso. La modalità normale permette di:

- inviare valori al DAC scegliendo opportunamente il valore in count e il LED da pilotare,
- acquisire dati dall'ADC,
- lettura della temperatura in prossimità dei LED.

La modalità in ciclo chiuso permette l'utilizzo dell'ADC e del fotodiodo come strumento di compensazione per il pilotaggio della luminosità dei LED.

2.4.2 Interfaccie di comunicazione

2.4.2.1 Interfaccia seriale

L'interfaccia seriale è stata gestita dalla "Universal Synchronous Asynchronous Receiver Transmitter", in breve USART, un modulo del PIC che gestisce l'interfaccia di comunicazione seriale. Tale interfaccia è utilizzata per la comunicazione remota con il PC.

Gestione Ricezione I pacchetti in ingresso sono acquisiti ad interruzione: la ricezione di un byte genera un'interruzione che blocca il processo in esecuzione; il sistema invoca l'Interrupt Service Routine (ISR) che contiene la funzione che acquisisce il byte dal registro *RCREG* e ritorna all'esecuzione del punto in cui il processo è stato interrotto. I pacchetti in ingresso hanno una dimensione di 6 byte come in Figura 2.29. Il pacchetto è composto da un header che segnala l'inizio del messaggio e sul quale il PIC allinea il buffer di ricezione; il dato successivo è il byte di comando che indica lo stato da eseguire e l'eventuale LED da pilotare. Il payload è composto da 3 byte e viene interpretato in base allo stato invocato. L'ultimo byte del pacchetto è il checksum, costituito dal byte meno significativo della somma dei precedenti 5 byte.

Il codice eseguito alla ricezione di un nuovo byte è specificato all'interno della funzione *void interrupt isr(void)*. Il nuovo byte ricevuto viene caricato all'interno del

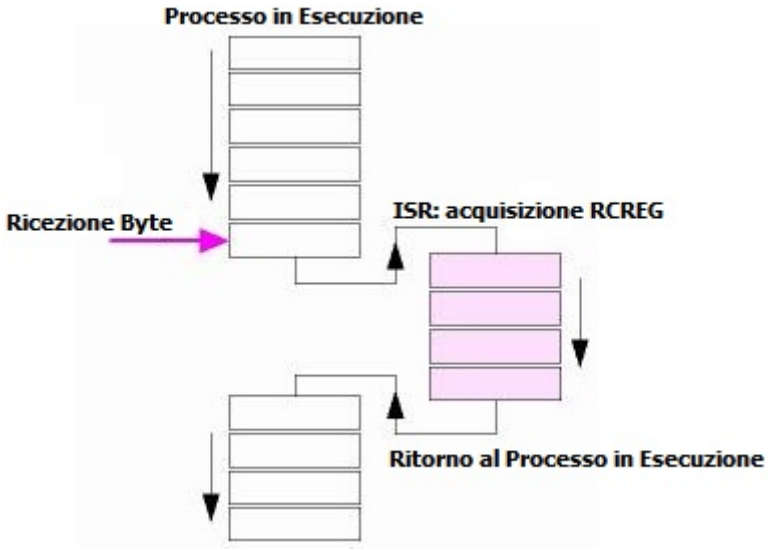


Figura 2.28: Schema concettuale avvio interruzione

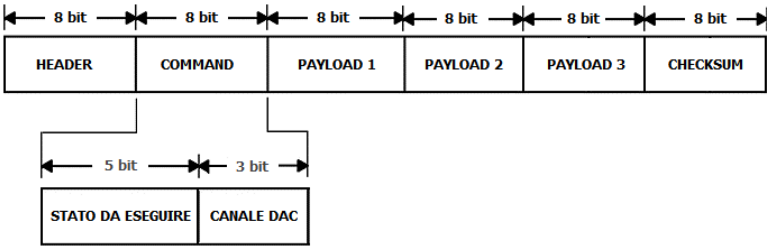


Figura 2.29: Schema del pacchetto ricevuto dal PC

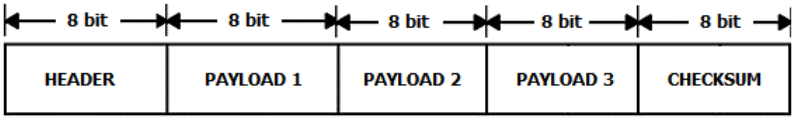


Figura 2.30: Schema del pacchetto spedito al PC

registro *RCREG* ed il bit *RCIF* viene settato ad 1, quindi si procede al salvataggio del byte in una variabile e *RCIF* viene resettato automaticamente. Si attende che di ricevere un byte corrispondente a *DATA_PACK*, alla ricezione del quale si avvia il processo di acquisizione del pacchetto. Arrivati al termine si confronta il checksum calcolato con quello ricevuto; se l'esito è positivo si provvede ad eseguire lo stato richiesto, altrimenti si comunica al PC l'errore in ricezione. Inoltre, nel caso in cui si richieda uno stato della modalità in ciclo chiuso, si setta la variabile *fbgen*, la quale indica il tipo di retroazione secondo il quale controllare il LED.

Gestione Inoltro I pacchetti inviati al PC sono rappresentati su 5 byte come in Figura 2.30. La funzione che si occupa dell'inoltro dei pacchetti è la *void Serial_Send_Packet(unsigned char, unsigned char, unsigned char, unsigned char)*(si rimanda all'appendice B per il dettaglio del suo funzionamento). L'header del pacchetto non è costante, come nel caso della ricezione, ma identifica il tipo di pacchetto spedito, ossia specifica il contenuto del payload. I valori dell'header sono:

- ACK_PACK: il payload contiene l'esito dello stato richiesto.
- TEMP_PACK: il payload contiene il valore della temperatura acquisita (in gradi).
- ADC_PACK: il payload contiene il valore campionato dall'ADC (in volt).
- FSP_PACK: il payload contiene l'ultimo valore campionato dall'ADC (in volt), segnalando il termine della findSetPoint().

Nel caso in cui si invii un pacchetto del tipo *ACK_PACK* il contenuto del payload 1 può assumere vari significati:

- STATUS_OK: il comando richiesto ha avuto esito positivo.
- STATUS_CKS_ERR: il checksum del pacchetto ricevuto dal PC è errato.
- STATUS_I2C_FAIL: comunicazione con il dispositivo su bus I2C fallita.
- STATUS_SPI_FAIL: comunicazione con il dispositivo su bus SPI fallita.
- STATUS_FSP_FAIL: mancato raggiungimento del setpoint.

Il trailer del pacchetto contiene il checksum, costituito dal byte meno significativo della somma dei primi 4 byte.

Algoritmo 2.1 ISR: Struttura della ricezione seriale

```
void interrupt isr(void)
{
   while(RCIF)
   {
      switch (pack_index)
      {
         case 0:
            if (RCREG == DATA_PACK)
            {
               inBuffer[pack_index] = RCREG;
               pack_index++;
               cks = RCREG;
            }
            else
               pack_index = 0;
         break;
         case 1:
            inBuffer[pack_index] = RCREG;
            pack_index++;
            cks += RCREG;
         break;
         case 2:
            inBuffer[pack_index] = RCREG;
            pack_index++;
            cks += RCREG;
         break;
         case 3:
            inBuffer[pack_index] = RCREG;
            pack_index++;
            cks += RCREG;
         break;
         case 4:
            inBuffer[pack_index] = RCREG;
            pack_index++;
            cks += RCREG;
         break;
         case 5:
            if (RCREG == cks) //CONFRONTO CHECKSUM
            {
               //ESTRAZIONE STATO DA ESEGUIRE
               state = (inBuffer[1]>>3)&(~0b11100000);
               if (state == STATE_FEEDBACK_GAIN) fbgen = 1;
               if (state == STATE_FEEDBACK_COMP) fbgen = 0;
               pack_index = 0;
               cks = 0;
            }
            else
            {
               //INVIO ACK: ERRORE NEL CHECKSUM
               Serial_Send_Packet(DATA_PACK, STATUS_CKS_ERR, NULL, NULL);
               pack_index = 0;
            }
         break;
      }//FINE SWITCH
      return;
   }//FINE INTERRUZIONE SERIALE

...

}//FINE ISR
```

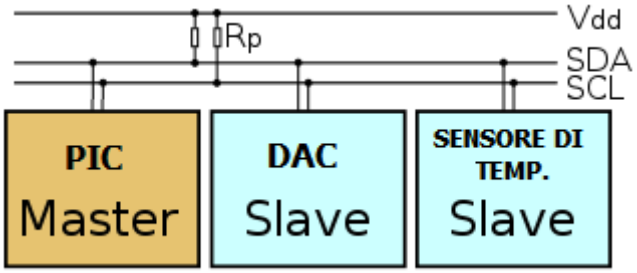


Figura 2.31: Dispositivi nel bus I2C

2.4.2.2 Interfaccia I2C

Il bus I2C è utilizzato per la comunicazione con il sensore di temperatura ed il DAC. L'I2C è un sistema di comunicazione a 2 fili (SCL e SDA) ed è composto da un master, il PIC, e da due slave, il sensore di temperatura ed il DAC. Il master invia sul bus l'indirizzo del dispositivo con cui vuole comunicare, successivamente invia o riceve i messaggi in base al comando selezionato.

Il PIC utilizzato non dispone di moduli hardware per la gestione della comunicazione I2C, quindi le primitive necessarie ad implementare la comunicazione sono state realizzate mediante firmware. Le due funzioni che permettono la scrittura e la lettura dal bus sono la *char I2C_Send_Byte(unsigned char)* e la *char I2C_Get_Byte(void)* (si rimanda in appendice B per la descrizione del codice).

Vediamo adesso come avviene la comunicazione con i due dispositivi.

Comunicazione con il DAC Due funzioni, appositamente scritte, inviano al DAC 3 byte, che riempiono un registro a scorrimento di 24 bit. La funzione I2C_Send_Packet_Init provvede all'inizializzazione del DAC accendendo i canali DAC e settando i parametri di funzionamento.

Algoritmo 2.2 I2C_Send_Packet_Init

```
/* Name: I2C_Send_Packet_Init
* Synopsis: char I2C_Send_Packet_Init(unsigned char data1, unsigned char data2)
* Arguments     :  unsigned  char : Command e DAC channel
*       unsigned  char : Command data
* Description   :  Invio dei pacchetti di inizializzazione al DAC
* Returns       :  Ritorna 0 se è avvenuto un errore in trasmissione,
*              1 se l'invio è andato a buon fine
*/
char I2C_Send_Packet_Init(unsigned char data1, unsigned char data2 )
{
   I2C_Start();
   if(!I2C_Send_Byte(DAC_I2C_ADDR))
   {
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte(data1))
   {
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte(NULL))
   {
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte(data2))
   {
      I2C_Stop();
      return(0);
   }
   I2C_Stop();
   return(1);
}
```

Algoritmo 2.3 I2C_Send_Packet

```
/*Name: I2C_Send_Packet
* Synopsis: char I2C_Send_Packet(unsigned int val, unsigned char chn)
* Arguments: unsigned  int: valore a 14 bit da convertire
*  unsigned  char: canale del DAC  (0 = A, 1 = B, 2 = C, 7 = tutti)
* Description: Invia ad uno o più canali del DAC 14 bit da convertire in tensione
* Returns: Ritorna 0 se è avvenuto un errore, 1 se l'invio è andato a buon fine
*/
char I2C_Send_Packet(unsigned int val, unsigned char chn)
{
   I2C_Start();
   if(!I2C_Send_Byte(DAC_I2C_ADDR))
   {
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte(0b00011000|chn))
   {  //|R = 0|S = 0|COMMAND = 011|DAC channel = chn|
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte((unsigned char)(val>>6)))
   {  //MSB
      I2C_Stop();
      return(0);
   }
   if(!I2C_Send_Byte((unsigned char)(val<<2)))
   {  //LSB
      I2C_Stop();
      return(0);
   }
   I2C_Stop();
   return(1);
}
```

La *I2C_Send_Packet* si occupa invece dell'invio di valori a 16 bit da convertire in tensione di uscita sul canale prescelto.

Comunicazione con il sensore di temperatura La lettura della temperatura inizia inviando l'indirizzo del sensore sul bus con la richiesta di lettura (ultimo bit dell'indirizzo settato a 1). Nei due byte successivi, il dispositivo predispone il contenuto del registro a 10 bit contenente la temperatura in count, rappresentato in complemento a due. In Figura 2.32 è possibile vedere la forma del dato estratto.

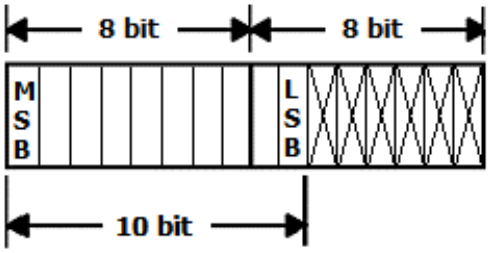


Figura 2.32: Count estratti dal sensore di temperatura

La rappresentazione della temperatura positiva è ottenibile applicando la formula $Temperatura = \frac{count}{4}$.

Algoritmo 2.4 I2C_Read_Temp

```
/*Name: I2C_Read_Temp
* Synopsis: int I2C_Read_Temp(void)
* Description: Aquisizione della temperatura
* Returns: Ritorna un intero a 16 bit
*          Nel caso sia uguale a zero è
*          avvenuto un errore nella comunicazione
*/
unsigned int I2C_Read_Temp(void)
{
   char HIbyte = 0;
   char LObyte = 0;
   I2C_Start();
   if(!I2C_Send_Byte((ST_I2C_ADDR << 1) | 0x01))
   {
      I2C_Stop();
      return(0);
   }
   HIbyte=I2C_Get_Byte();
   LObyte=I2C_Get_Byte();
   I2C_Stop();
   return(((((unsigned int)HIbyte) << 8) | LObyte) >> 6);
}
```

2.4.2.3 Interfaccia SPI

Il Serial Peripheral Interface (SPI) è un sistema di comunicazione che permette la comunicazione tra il PIC e l'ADC Maxim. Come nel bus I2C, la comunicazione avviene tra un master e più slave (in questo caso il PIC agisce da master e l'ADC da slave). Il bus SPI è di tipo seriale sincrono, caratterizzato da un clock che coordina la trasmissione e la ricezione dei singoli bit e determina la velocità di trasmissione.

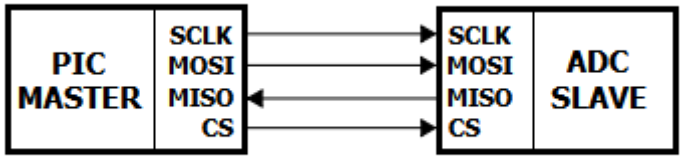


Figura 2.33: Bus Master - Slave SPI

La comunicazione può essere di tipo full-duplex, in quanto il "colloquio" può avvenire contemporaneamente in trasmissione e ricezione. La connessione fisica avviene tramite 4 fili:

- MOSI: dati dal master allo slave.
- MISO: dati dallo slave al master.
- SCLK: clock generato dal master.
- CS: chip select, attiva o disattiva il dispositivo slave.

BIT	B7	B6	B5	B4	B3	B2	B1	B0
BIT NAME	START = 1	MODE = 0	CAL1	CAL0	IMPD	RATE2	RATE1	RATE0

BIT	B7	B6	B5	B4	B3	B2	B1	B0
BIT NAME	START = 1	MODE = 1	0	RS3	RS2	RS1	RS0	$\overline{W}$/R

Figura 2.34: Contenuto del byte di comando

CTRL1 Register

BIT	B7	B6	B5	B4	B3	B2	B1	B0
BIT NAME	LINEF	U/$\overline{B}$	EXTCLK	REFBUF	SIGBUF	FORMAT	SCYCLE	UNUSED
DEFAULT	0	0	0	0	0	0	1	0

CTRL3 Register

BIT	B7	B6	B5	B4	B3	B2	B1	B0
BIT NAME	DGAIN2*	DGAIN1*	DGAIN0*	NOSYSG	NOSYSO	NOSCG	NOSCO	RESERVED
DEFAULT	0	0	0	1	1	1	1	0

Figura 2.35: Registri CTRL1 e CTRL3

Anche in questo caso, il PIC non dispone di moduli hardware per la gestione della comunicazione, quindi le primitive necessarie alla gestione del protocollo sono state implementate tramite firmware. Le due funzioni che permettono la scrittura e la lettura dal bus sono la *char SPI_Send_Byte(unsigned char)* e la *char SPI_Get_Byte(void)*, si rimanda in appendice B per la descrizione del codice. La comunicazione tra il PIC e il dispositivo avviene tramite l'utilizzo di un byte di comando, che contiene un bit che discrimina la modalità di interpretazione dei pacchetti spediti e ricevuti dal microcontrollore.

Registri CTRL1 e CTRL3 Tramite i registi *CTRL1* e *CTRL3* è possibile impostare i parametri relativi al metodo di campionamento; vediamone nel dettaglio il significato:

- CTRL1: registro abilitato in lettura e scrittura;
  - LINEF: determina la frequenza dell'oscillatore interno; settando ad 1 si applica una reiezione dei disturbi a 60Hz, settando a 0 si applica una reiezione a 50Hz.
  - U/B: configura il range di ingresso, da bipolare (±Vref) ad unipolare (0 , Vref).
  - EXTCLK: settato abilita l'utilizzo di un clock esterno.
  - REFBUF: abilita o disabilita il buffer delle tensioni in ingresso di riferimento.
  - SIGBUF: abilita o disabilita il buffer in ingresso per gli input analogici (*AINP*, *AINN*).

BIT	B7	B6	B5	B4	B3	B2	B1	B0
BIT NAME	SYSOR	RATE2	RATE1	RATE0	OR	UR	MSTAT	RDY
DEFAULT	0	0	0	0	0	0	0	0

Figura 2.36: Registro STAT1

BIT	D23	D22	D21	D20	D19	D18	D17	D16
DEFAULT	0	0	0	0	0	0	0	0

BIT	D15	D14	D13	D12	D11	D10	D9	D8
DEFAULT	0	0	0	0	0	0	0	0

BIT	D7	D6	D5	D4	D3	D2	D1	D0
DEFAULT	0	0	0	0	0	0	0	0

Figura 2.37: Registro DATA

  - FORMAT: controlla il formato digitale del registro *DATA* (complemento a due o binario con offset).
  - SCYCLE: abilità la modalità in single-shot o la modalità in continuous-read.

- CTRL2: registro abilitato in lettura e scrittura;

  - DGAIN[2:0]: controlla il guadagno digitale.
  - NOSYSG: controlla il guadagno di sistema creato dalla calibrazione (impostato a 1 lo disabilita).
  - NOSYSO: controlla l'offset di sistema creato dalla calibrazione (impostato a 1 lo disabilita).
  - NOSCG: controlla il coefficiente del guadagno di sistema creato dalla calibrazione automatica (impostato a 1 lo disabilita).
  - NOSCO: controlla l'offset di sistema creato dalla calibrazione automatica (impostato a 1 lo disabilita).

Registro di stato STAT1 *STAT1* è il registro di stato dell'ADC, abilitato soltanto in lettura. I bit utilizzati nel codice sono:

- MSTAT: segnala l'inizio di una conversione o di una calibrazione (indica che l'ADC non è occupato)

- RDY: segnala che il dato convertito è pronto e quindi il registro può essere letto *DATA*. Nella modalità single-shot, una volta inviato il "*comando 0*" per richiedere una conversione, si controlla attivamente *RDY* aspettando che l'ADC lo setti ad 1, segnalando la fine della conversione.

Registro DATA L'ADC converte un segnale analogico in ingresso in un valore digitale a 24 bit, contenuto in *DATA*. La lettura di *DATA* varia in base alla modalità di conversione richiesta:

- Single-shot: ogni invio del "*comando 0*" genera una richiesta di conversione; prima di leggere *DATA* è necessario attendere che *RDY*, del registro *STAT1*, venga settato ad 1.
- Continuous-read: l'invio del "*comando 0*", pone l'ADC in modalità di conversione continua alla velocità selezionata; *DATA* può essere letto ogni volta che l'ADC imposta ad 1 il pin di uscita (*MISO*). Infatti tale pin, normalmente a valore logico alto, viene abbassato a 0 appena inizia la conversione. Se il registro *DATA* non viene letto entro la fine della successica conversione, l'ADC lo sostituisce appena un nuovo dato è pronto.

Comunicazione con l'ADC: Modalità 0 Inviando un byte di comando contenente MODE = 0 è possibile richiedere:

- l'inizio di una conversione (si impostano i bit *RATE[2:0]* in base alla velocità di conversione richiesta);
- una calibrazione (si imposta il bit *CAL0 = 1* per la calibrazione automatica);
- lo spegnimento dell'ADC (si imposta il bit *IMPD = 1*).

Algoritmo 2.5 Funzioni per la modalità 0

```
/*Name: SPI_modezero
* Synopsis: char SPI_modezero(unsigned char data)
* Arguments: unsigned  char:  Comando da spedire all'ADC
* Description: Invio di un pacchetto del tipo MODE = 0, è possibile richiedere una conversione o
    una calibrazione
* Returns: Ritorna 0 se è avvenuto un errore in trasmissione, 1 se l'invio è andato a buon fine
*/
char SPI_modezero(unsigned char data)
{
    for(int i = 0;;i++)
    {
        dly1u;
        adc_reg8 = SPI_readReg8(STAT1_READ);
        //CONDIZIONI DI USCITA:
        //ASPETTO MSTAT = 0, QUINDI L'ADC E' DISPONIBILE A RICEVERE UN COMANDO
        //ALTRIMENTI SE NON SI LIBERA ENTRO ADC_MSTAT_THRESHOLD RITORNO
        if ((adc_reg8 & 0x02) == 0) break;
        if (i > ADC_MSTAT_THRESHOLD1) return 0;
    }
    SPI_Start();
    SPI_Send_Byte(data);
    SPI_Stop();
    return 1;
}

/*Name: calibration
* Synopsis: char calibration (unsigned char data1, unsigned char data3)
* Arguments: unsigned  char: byte da scrivere nel registro CTRL1
*            unsigned  char: byte da scrivere nel registro CTRL3
* Description: Esegue la selfcalibration dell'ADC dopo aver
*              settato adeguatamento il funzionamento tramite i registri
* Returns: Ritorna 0 se è avvenuto un errore in trasmissione, 1 se l'invio è andato a buon fine
*/
char calibration (unsigned char data1, unsigned char data3)
{
    SPI_impd();
    dly1u;
    if(!SPI_writeReg8(data1,CTRL1_WRITE))return 0;
    if(!SPI_writeReg8(data3,CTRL3_WRITE))return 0;
    if(!SPI_modezero(SELFCALIB))return 0;
    return 1;
}

/*Name: SPI_impd
* Synopsis: void SPI_impd()
* Description: Spegnimento dell'ADC (Immediate Power Down)
*/
void SPI_impd()
{
    SPI_Start();
    SPI_Send_Byte(PWRDWN);
    SPI_Stop();
}
```

Comunicazione con l'ADC: Modalità 1 Quando MODE = 1 è possibile richiedere la lettura o la scrittura di un registro. I bit *RS[3:0]* corrispondono all'indirizzo del registro richiesto e la lettura o la scrittura sono discriminate dal bit *W/R*.

Analizziamo in dettaglio le funzioni implementate. La *SPI_readReg8(unsigned char)* permette di leggere dal DAC il contenuto di un registro ad 8 bit inviando un byte contenente "mode = 1" , l'indirizzo del registro ed il bit *W/R = 1* , e leggendo il dato inviato dal DAC. La scrittura dei registri a 8 bit è molto simile: si invia un byte di comando contenente l'indirizzo, stavolta resettando il bit *W/R* ; prima di modificare il registro si controlla che l'ADC non sia occupato verificando lo stato del bit *MSTAT* del registro *STAT1*. Sono state definite due funzioni per la lettura del registro *DATA* , una per la modalità di conversione single-shot ed una per la modalità continuous-read. In single-shot, prima di ricevere i 24 bit si controlla attivamente il bit *RDY* del registro *STAT1* , che segnala la disponibilità del nuovo dato. In modalità continuous-read, come già indicato, si controlla attivamente che *MISO* passi da 0 ad 1, dopodiché si procede a leggere i 24 bit.

Algoritmo 2.6 Lettura e scrittura di un registro ad 8 bit

```
/*Name: SPI_readReg8
* Synopsis: char SPI_readReg8(unsigned char addr)
* Arguments: unsigned  char:  Indirizzo del registro da leggere
* Description: Lettura di un registro a 8 bit dell'adc, tramite invio dell'indirizzo (lsb = 1)
* Returns: Ritorna un char contenente il valore del registro scelto
*/
unsigned char SPI_readReg8(unsigned char addr)
{
    unsigned char data = 0;
    SPI_Start();
    SPI_Send_Byte(addr);
    data = SPI_Get_Byte();
    SPI_Stop();
    return data;
}

/*Name: SPI_writeReg8
* Synopsis: char SPI_writeReg8(unsigned char data, unsigned char addr)
* Arguments: unsigned  char: Indirizzo del registro da modificare
*            unsigned  char: Byte da scrivere nel registro
* Description: Scrittura di un registro a 8 bit dell'adc, tramite invio dell'indirizzo (lsb = 1)
* Returns: Ritorna 0 se è avvenuto un errore in trasmissione, 1 se l'invio è andato a buon fine
*/
char SPI_writeReg8(unsigned char data, unsigned char addr)
{
    for(int i = 0;;i++)
    {
        dly1u;
        adc_reg8 = SPI_readReg8(STAT1_READ);
        //CONDIZIONI DI USCITA:
        //ASPETTO MSTAT=0, QUINDI L'ADC E' DISPONIBILE A RICEVERE UN COMANDO
        //ALTRIMENTI SE NON SI LIBERA ENTRO ADC_MSTAT_THRESHOLD2 RITORNO
        if ((adc_reg8 & 0x02) == 0) break;
        if (i > ADC_MSTAT_THRESHOLD2) return 0;
    }
    SPI_Start();
    SPI_Send_Byte(addr);
    SPI_Send_Byte(data);
    SPI_Stop();
    return 1;
}
```

Algoritmo 2.7 Lettura del registro DATA

```
/*Name: SPI_readReg24_S
* Synopsis: char SPI_readReg24_S(unsigned char *bufferIn)
* Arguments: unsigned char*: puntatore ad un vettore di tre byte
* Description: Acquisisce il registro DATA dell'ADC nella modalita single-shot
* Returns: Ritorna 0 se è avvenuto un errore in trasmissione, 1 se l'invio è andato a buon fine
*/
char SPI_readReg24_S(unsigned char *bufferIn)
{
    for(int i = 0;;i++)
    {
        dly1u;
        //Controllo il bit rdy
        adc_reg8 = SPI_readReg8(STAT1_READ);
        if ((adc_reg8 & 0x01) == 1) break;
        if (i > ADC_READ_THRESHOLD_S) return 0;
    }
    SPI_Start();
    SPI_Send_Byte(DATA_READ);
    bufferIn [0]=SPI_Get_Byte(); //parte alta
    bufferIn [1]=SPI_Get_Byte(); //parte centrale
    bufferIn [2]=SPI_Get_Byte(); //parte bassa
    SPI_Stop();
    return 1;
}

/*Name: SPI_readReg24_C
* Synopsis: char SPI_readReg24_S(unsigned char *bufferIn)
* Arguments: unsigned char*: puntatore ad un vettore di tre byte
* Description: Acquisisce il registro DATA dell'ADC nella modalita continuous-read
* Returns: Ritorna 0 se è avvenuto un errore in trasmissione, 1 se l'invio è andato a buon fine
*/
char SPI_readReg24_C(unsigned char *buffer)
{
    SPI_Start();
    for(int i = 0;;i++)
    {
        dly1u;
        //L'ADC segnala che il dato è pronto mettendo ad 1 l'uscita
        if (DIN == 1) break;
        if (i > ADC_READ_THRESHOLD_C)
        return 0;
    }
    SPI_Send_Byte(DATA_READ);
    buffer[0]=SPI_Get_Byte(); //parte alta
    buffer[1]=SPI_Get_Byte(); //parte centrale
    buffer[2]=SPI_Get_Byte(); //parte bassa
    SPI_Stop();
    return 1;
}
```

2.4.3 Corpo principale: main.c

All'avvio del PIC vengono effettuate le inizializzazioni delle variabili utilizzate dal sistema. Al termine dell'inizializzazione si avvia l'esecuzione della funzione *void main()* , all'inizio della quale vengono richiamate le funzioni di setup del sistema, del timer 1, del DAC e dell'interfaccia seriale.

Algoritmo 2.8 Inizializzazione variabili

```
unsigned char dac_chn_fb = 0;//CANALE DAC SU CUI ESEGUIRE IL FEEDBACK
unsigned char cks = 0;//CHECKSUM
unsigned char adc_reg8 = 0;//REGISTRO AD 8 BIT ESTRATTOO DALL'ADC
unsigned char state = STATE_IDLE;//VARIABILE DELLO SWITCH CASE
unsigned char pack_index = 0;//INDEX PACCHETTO RICEVUTO DA SERIALE
unsigned char flag_feedback = 0;//SEMAFORO PER ESECUZIONE DELLA feedback()
unsigned char flag_feedback_aux = 0;//FLAG DI SUPPORTO PER CONTROLLO ESECUZIONE feedback()
unsigned char feedback_state = 0;//MODALITA FEEDBACK ATTIVA/DISATTIVA
unsigned char inBuffer[INBUFFER_SIZE];//PACCHETTO RICEVUTO DALLA SERIALE
unsigned char adc_reg_24[3];//ULTIMO VALORE ACQUISITO DALL'ADC
unsigned int aux = 0;//VARIABILE AUSILIARIA PER LETTURA DELLA TEMPERATURA
unsigned int led_count_fb = 0;//VALORE SETTATO SUI LED NELLA MODALITA' CLOSED LOOP
unsigned int min_dac = 0;//ESTREMO SINISTRO DELL'INTERVALLO DI BISEZIONE
unsigned int max_dac = 0;//ESTREMO DESTRO DELL'INTERVALLO DI BISEZIONE
long adc_data = 0;//ULTIMO VALORE ACQUISITO DALL'ADC (LONG)
long adc_set_point = 0;//SETPOINT DA RAGGIUNGERE TRAMITE feedback()
```

Algoritmo 2.9 Avvio main

```
void main(void){
    //SETUP INIZIALI
    SYS_setup();
    TMR1_setup();
    DAC_setup();
    SRL_setup();
    //BLINKING LED ROSSO
    for(int i = 0; i <= 6; i++)
    {
        LED3 = LED3 ^ 1;
        DelayMs(100);
    }
...
```

Algoritmo 2.10 Funzioni di setup

```
/*Name: SYS_setup
* Synopsis: void SYS_setup(void)
* Description: Inizializzazione porte ed interruzioni
*/
void SYS_setup(void)
{
   ADCON1 = 0x06;//INGRESSI SULLA PORTA A DIGITALI
   TRISA = 0x0F;
   TRISB = 0X3F;
   PORTB = 0X00;
   TRISC = 0X80;
   //SPENGO I LED DI STATO
   LED1 = 1;
   LED2 = 1;
   LED3 = 1;
   //ABILITAZIONE INTERRUZIONI
   GIE = 1;
   PEIE = 1;
}

/*Name: TMR1_setup
* Synopsis: void TMR1_setup(void)
* Description: Inizializza il timer1
*/
void TMR1_setup(void)
{
   TMR1ON = 0;//DISABILITO IL TIMER
   T1CKPS1 = 1;//PRESCALER 1:8
   T1CKPS0 = 1;
   T1OSCEN = 0;//OSCILATORE DISABILITATO
   TMR1CS = 0;//CLOCK INTERNO, Fosc/4
   TMR1H = INTLAPSE >> 8;//VALORE DA CUI INIZIARE
   TMR1L = INTLAPSE & 0XFF;//AD INCREMENTARE
   TMR1IF = 0;//RESET OVERFLOW FLAG
   TMR1IE = 1;//ABILITO INTERRUPT
}

/*Name: SRL_setup
*Synopsis: void SRL_setup(void)
*Description: Inizializzazione interfaccia seriale
*/
void SRL_setup(void)
{
   SPBRG=1;//DIVISORE =>BAUD = 115200
   BRGH=1;//HIGHSPEED
   SYNC=0;//ASINCRONO
   SPEN=1;//ABILITO RC6 E RC7 COME PORTE SERIALI
   TXIE=0;//DISABILITO INTERRUZIONI IN TX
   RCIE=1;//ABILITO INTERRUZIONI IN RX
   TX9=0;//TRASMISSIONE AD 8 BIT
   RX9=0;//RICEZIONE AD 8 BIT
   CREN=0;//RESET RICEZIONE
   CREN=1;//RICEZIONE CONTINUA ATTIVA
   TXEN=0;//RESET TRASMITTER
   TXEN=1;//ABILITAZIONE TRASMISSIONE
}
```

Successivamente si avvia lo *while(1)* che continuerà a ciclare fino allo spegnimento del PIC. Il ciclo all'interno dello *while(1)* controlla la pressione del pulsante *BTN1*, che richiama l'esecuzione dello stato *STATE_OFF*; in seguito si esegue un costrutto switch-case che implementa le funzionalità di macchina a stati.

Algoritmo 2.11 Struttura completa del case switch

```
while(1)
{
    if(!BTN1)
    {
        DelayMs(80);
        if(!BTN1)
        state = STATE_OFF;
    }
    switch(state)
    {
        case STATE_IDLE:
        ... //CICLO DI ATTESA ATTIVO
        break;
        case STATE_OFF:
        ... //SPEGNIMENTO LED, DAC E FEEDBACK
        break;
        case STATE_ON:
        ... //ACCENSIONE LED IN OPEN LOOP
        break;
        case STATE_TEMP:
        ... //RICHIESTA TEMPERATURA
        break;
        case STATE_SELFCALIB_S:
        ... //SINGLESHOT CALIBRATION ADC
        break;
        case STATE_SELFCALIB_C:
        ... //CONTINUOS READ CALIBRATION ADC
        break;
        case STATE_SELFCALIB_CUSTOM:
        ... //CUSTOM CALIBRATION ADC
        break;
        case STATE_SINGLE_READ:
        ... //LETTURA DATA IN SINGLESHOT
        break;
        case STATE_CONT_READ:
        ... //LETTURA DATA IN CONTINOUOS READ
        break;
        case STATE_FEEDBACK_START:
        ... //ACCENSIONE LED IN CICLO CHIUSO (RICERCA SETPOINT)
        break;
        case STATE_FEEDBACK_RESUME:
        ... //RIATTIVA MODALITA' FEEDBACK
        break;
        case STATE_FEEDBACK_STOP:
        ... //DISABILITA LA MODALITA' FEEDBACK
        break;
    }
}
```

2.4.3.1 Modalità normale

Nella modalità di funzionamento normale il sistema rimane in attesa di ricevere un nuovo pacchetto da remoto, alla ricezione del quale interpreta il comando ed esegue il case (stato) richiesto. Al termine di ogni stato viene riassegnato *STATE_IDLE* come stato successivo. Il diagramma in Figura 2.38 schematizza il funzionamento concettuale dell'interpretazione degli stati.

Analizziamo adesso come vengono eseguiti gli stati del funzionamento normale. La *STATE_IDLE* è un ciclo di attesa attivo che attende che l'interruzione generata dalla ricezione di un pacchetto seriale cambi il contenuto di state. Lo *STATE_IDLE* contiene la chiamata alla funzione *feedback()*, che verrà illustrata successivamente durante la descrizione delle funzioni della modalità di funzionamento in anello

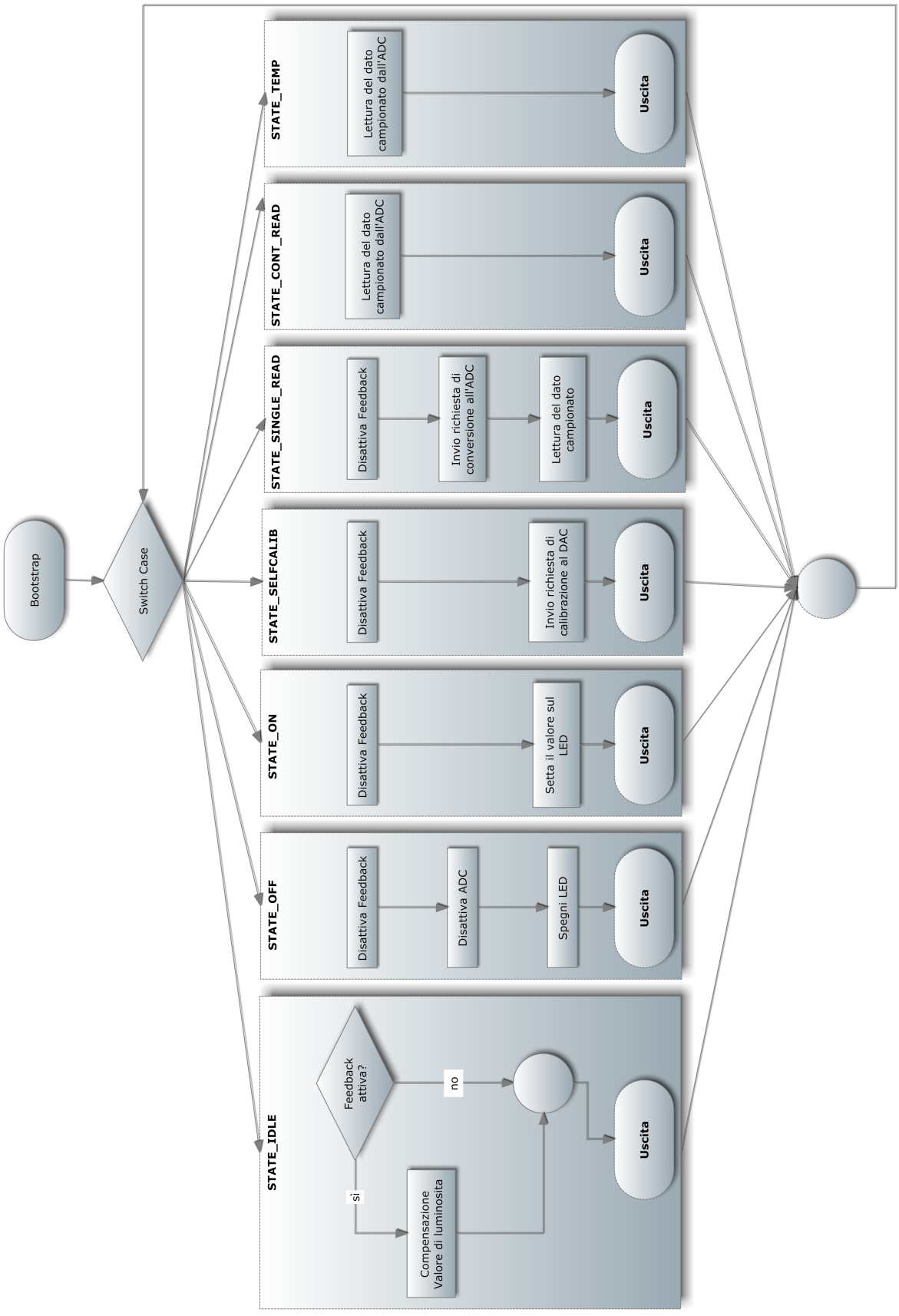


Figura 2.38: Diagramma della modalità normale

Algoritmo 2.12 Stati STATE_IDLE, STATE_OFF e STATE_ON

```
case STATE_IDLE:
  di();
  flag_feedback_aux = flag_feedback;
  ei();
  if(flag_feedback_aux)
  {
    feedback();
    di();
    flag_feedback = 0;
    ei();
  }
break;
case STATE_OFF:
  TMR1_feedback_OFF();
  SPI_impd();
  SPI_Stop();
  I2C_Send_Packet(0, DAC_ALL_CHN);
  Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
  state = STATE_IDLE;
break;
case STATE_ON:
  TMR1_feedback_OFF();
  if(I2C_Send_Packet((led_count > DAC_MAX_VAL) ? DAC_MAX_VAL : led_count, dac_chn))
    Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
  else
    Serial_Send_Packet(DATA_PACK, STATUS_I2C_FAIL, NULL, NULL);
  state = STATE_IDLE;
break;
case STATE_TEMP:
  aux = I2C_Read_Temp();
  Serial_Send_Packet(TEMP_PACK,(unsigned char)(aux>>8),(unsigned char)aux,NULL);
  state = STATE_IDLE;
break;
```

chiuso. Il case *STATE_OFF* richiama le funzioni di disattivazione del timer 1, spegnimento dell'ADC, invio del valore 0 ai LED ed invio della conferma al PC. La *STATE_ON* invia al LED selezionato il valore *led_count*, eventualmente saturato ai limiti del convertitore; la notifica al PC può essere positiva (se l'invio sul bus I2C va a buon fine) o negativa. Per acquisire la temperatura dei LED si utilizza lo stato *STATE_TEMP* che, una volta interrogato il sensore di temperatura, invia al PC la risposta.

Algoritmo 2.13 Calibrazione e lettura dell'ADC

```
case STATE_SELFCALIB_S:
    TMR1_feedback_OFF();
    if(calibration(CTRL1_SINGLE_SET, CTRL3_SET))
        Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
    else
        Serial_Send_Packet(DATA_PACK, STATUS_SPI_FAIL, NULL, NULL);
    state = STATE_IDLE;
break;
case STATE_SELFCALIB_C:
    TMR1_feedback_OFF();
    if(calibration(CTRL1_CONT_SET, CTRL3_SET))
        Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
    else
        Serial_Send_Packet(DATA_PACK, STATUS_SPI_FAIL, NULL, NULL);
    SPI_modezero(RATE_CONT);
    state = STATE_IDLE;
break;
case STATE_SELFCALIB_CUSTOM:
    TMR1_feedback_OFF();
    if(calibration(inBuffer[2], inBuffer[3]))
        Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
    else
        Serial_Send_Packet(DATA_PACK, STATUS_SPI_FAIL, NULL, NULL);
    SPI_modezero(inBuffer[4]);
    state = STATE_IDLE;
break;
case STATE_SINGLE_READ:
    TMR1_feedback_OFF();
    SPI_modezero(RATE_SINGLE);
    if(SPI_readReg24_C(adc_reg_24))
        Serial_Send_Packet(ADC_PACK, adc_reg_24[0], adc_reg_24[1], adc_reg_24[2]);
    else
        Serial_Send_Packet(DATA_PACK, STATUS_SPI_FAIL, NULL, NULL);
    state = STATE_IDLE;
break;
case STATE_CONT_READ:
    state = STATE_IDLE;
    if (feedback_state)
        Serial_Send_Packet(ADC_PACK, adc_reg_24[0], adc_reg_24[1], adc_reg_24[2]);
    else
    {
        if(SPI_readReg24_C(adc_reg_24))
            Serial_Send_Packet(ADC_PACK,adc_reg_24[0],adc_reg_24[1],adc_reg_24[2]);
        else
            Serial_Send_Packet(DATA_PACK, STATUS_SPI_FAIL, NULL, NULL);
    }
break;
```

La calibrazione dell'ADC può essere di tre tipi:

- STATE_SELFCALIB_S, per acquisizioni del tipo single-shot
- STATE_SELFCALIB_C, per acquisizioni del tipo continuous-read
- STATE_SELFCALIB_CUSTOM, per definire manualmente il contenuto dei registri *CTRL1*, *CTRL3* e del comando di tipo 0.

Ovviamente specificheremo una calibrazione *STATE_SELFCALIB_S* quando utilizzeremo *STATE_SINGLE_READ* per acquisire i dati dall'ADC; analogamente utilizzeremo *STATE_SELFCALIB_C* per effettuare letture nello stato *STATE_CONT_READ*. Gli stati per acquisire i comandi si differenziano per il fatto che quando si effettua una lettura single-read è necessario inviare ogni volta il comando 0 ed i parametri di acquisizione, mentre nella lettura continua ciò non è necessario e si può acquisire direttamente il dato.

2.4.3.2 Modalità in ciclo chiuso

Gli stati della modalità in anello chiuso permettono di controllare i LED con precisione, aggiornando continuamente il valore inviato al DAC sulla base della lettura

Algoritmo 2.14 Gestione dell'interruzione generata dal timer 1

```
void interrupt isr(void)
{

...

    if (TMR1IF)
    {
        TMR1ON = 0;
        TMR1IF = 0;
        //SEGNALAZIONE PER L'ESECUZIONE DELLA feedback()
        flag_feedback = 1;
        TMR1H=INTLAPSE>>8;
        TMR1L=INTLAPSE & 0XFF;
        TMR1ON = 1;
        return;
    }
} //FINE ISR
```

Algoritmo 2.15 Funzioni per l'attivazione e la disattivazione del timer 1

```
/*Name: TMR1_feedback_ON
* Synopsis: void TMR1_feedback_ON(void)
* Description: Attiva il timer1, ogni volta che si verifica overflow viene eseguita la feedback()
*                       Deve essere chiamata successivamente al settagio di un setPoint
*/
void TMR1_feedback_ON(void)
{
    feedback_state = 1; //FLAG, FEEDBACK ABILITATA
    TMR1ON = 0;
    TMR1H = INTLAPSE >> 8;
    TMR1L = INTLAPSE & 0XFF;
    TMR1IF = 0;
    TMR1ON = 1; //ABILITO IL TIMER1
}

/*Name: TMR1_feedback_OFF
* Synopsis: void TMR1_feedback_OFF(void)
* Description  : Disattiva il timer1, non viene eseguita la feedback.
*/
void TMR1_feedback_OFF(void)
{
    TMR1ON = 0;
    feedback_state = 0;  //FLAG, FEEDBACK DISABILITATA
}
```

dell'ADC, impostato in modalità a cadenza costante. Il diagramma in Figura 2.39 schematizza il funzionamento concettuale degli stati e delle funzioni utilizzate in questa modalità. Gli stati sono disgiunti a quelli utilizzati in modalità normale. In parallelo all'esecuzione del processo principale è definito un timer, che 20 volte al secondo setta un flag per abilitare l'esecuzione del processo di compensazione della luminosità. Tale timer viene disabilitato o abilitato all'occorrenza dagli altri stati, come visibile nei due diagrammi. L'Algoritmo 2.14 mostra la funzione eseguita alla generazione dell'interruzione del timer. Il timer conta fino a 2^{16} con una velocità data da $\frac{F_{OSC}}{4} = \frac{3686400}{4} = 921600Hz$; impostando a 8 il prescaler si ottiene $\frac{921600}{8} = 115200Hz$, quindi l'incremento del timer avviene ogni 8.7uS. Per generare una frequenza di 20Hz (con cadenza di 50mS) il timer deve quindi contare fino a 5760 . Poiché l'interruzione è generata al raggiungimento del valore 65535, è necessario far iniziare il conteggio da $65535 - 5760 = 59775$, valore contenuto nella variabile *INTLAPSE*.

Nello *STATE_ON* l'utente richiede al PIC di impostare un valore di luminosità per un LED; il primo risultato è inaffidabile, visto che non è semplice individuare una corrispondenza univoca tra il valore da impostare sul DAC e il valore di luminosità desiderata (il risultato è influenzato dalla temperatura, dalla dimensione del gradino

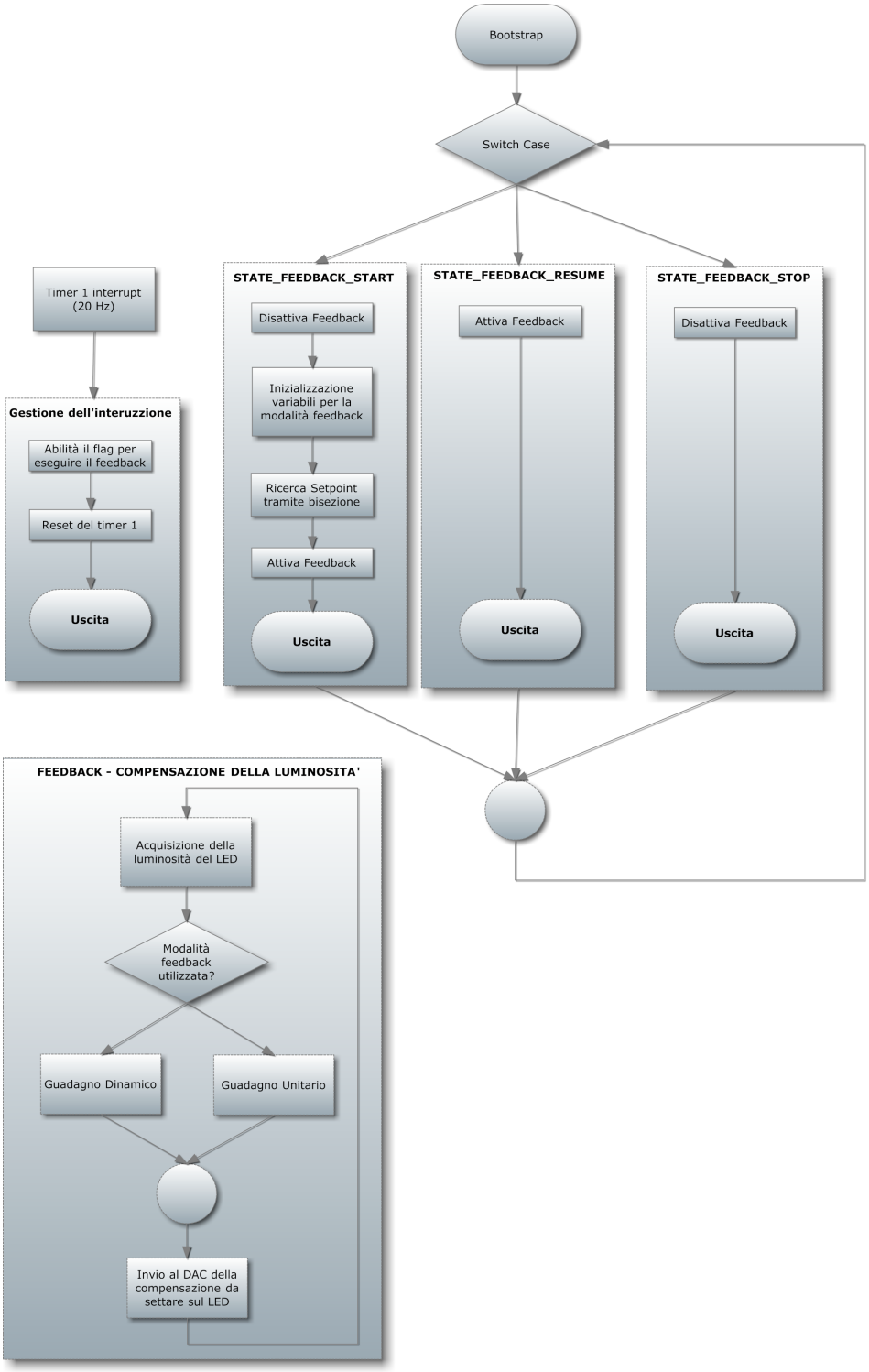


Figura 2.39: Diagramma della modalità in ciclo chiuso

e dal tipo di LED selezionato).

La *findSetPoint()* applica l'algoritmo di bisezione alla ricerca della luminosità, dato che questa è legata univocamente all'ampiezza del segnale dalla scheda del fotodiodo, acquisito mediante l'ADC. Ad ogni passo di bisezione, si controlla la lettura dell'ADC e si confronta con quella richiesta, dopo di che si riduce l'intervallo di ricerca e si invia il nuovo valore mediano al DAC. La condizione di uscita dell'algoritmo di ricerca è verificata appena si entra in un intervallo *OFFSET_DAC*[3].

La funzione *feedback()* che si occupa di agire sulla luminosità del LED viene chiamata quando il processo esegue la *STATE_IDLE*; una volta terminata si resetta il flag di esecuzione e si attende che il timer generi una nuova interruzione, che setterà nuovamente il flag. Tale funzione è costituita da due modi di funzionamento, una con guadagno unitario e l'altra con guadagno dinamico.

La modalità con guadagno unitario legge il valore di tensione dall'ADC e lo confronta con il setpoint richiesto, aumentando o diminuendo all'occorrenza di 1 il valore da spedire al DAC. La modalità a guadagno dinamico conteggia il numero di volte consecutive che si è incrementato o decrementato, quindi individua la direzione in cui la luminosità sta derivando; superato un dato numero di avanzamenti consecutivi nella stessa direzione, raddoppia il passo (guadagno) con cui avviene la compensazione del LED. Ogni volta che si inverte direzione, il guadagno viene dimezzato.

Gli stati *STATE_FEEDBACK_GAIN* e *STATE_FEEDBACK_COMP* effettuano un'inizializzazione delle variabili utilizzate nella modalità feedback; procedono poi alla ricerca del setpoint tramite la *findSetPoint()*, all'uscita della quale segnalano all'utente il valore raggiunto e abilitano il timer controllerà il meccanismo di compensazione della luminosità. Nel caso in cui la *findSetPoint()* fallisca, superando il limite di iterazioni massime, non si attiverà il meccanismo di interruzione e verrà inviata una segnalazione al PC. Gli stati *STATE_FEEDBACK_RESUME* e *STATE_FEEDBACK_STOP* permettono di abilitare o bloccare il timer e devono essere invocati soltanto dopo aver eseguito almeno una volta uno degli stati *STATE_FEEDBACK_GAIN* o *STATE_FEEDBACK_COMP*.

[3]Il valore di default è 2

Algoritmo 2.16 findSetPoint() e feedback()

```
/*Name: findSetPoint
* Synopsis: void findSetPoint(void)
* Description: Esegue un algoritmo di bisezione alla ricerca del setpoint precedentemente settato
*/
char findSetPoint(void)
{
    for(int i = 0; i < BISECTION_IT_THR; i++)
    {
        //VALORE DA SPEDIRE AL DAC
        led_count_fb = (min_dac + max_dac) >> 1;
        I2C_Send_Packet(led_count_fb, dac_chn_fb);
        //ATTESA ASSESTAMENTO DEL LED
        DelayMs(100);
        //ACQUISIZIONE ADC
        SPI_readReg24_C(adc_reg_24);
        adc_data = cast(adc_reg_24);
        //SCELTA DEL NUOVO INTERVALLO DI BISEZIONE
        if(adc_data > adc_set_point) max_dac = led_count_fb;
        if(adc_data < adc_set_point) min_dac = led_count_fb;
        //CONDIZIONE DI USCITA DAL CICLO
        if(max_dac - min_dac <= OFFSET_DAC) return 0;
    }
    return 1;
}

/*Name: feedback
* Synopsis: void feedback(void)
* Description: Pilotaggio del led tramite compensazione a guadagno dinamico o unitario
*/
void feedback(void)
{
    //LETTURA DEL DAC
    SPI_readReg24_C(adc_reg_24);
    adc_data = cast(adc_reg_24);

    //RETROAZIONE CON GUADAGNO DINAMICO
    if (fbgen==1)
    {
        if (adc_data < (adc_set_point - FB_HYST))
        {
            direction = UP;
            led_count_fb += dac_inc;
        }
        if (adc_data > (adc_set_point + FB_HYST))
        {
            direction = DOWN;
            led_count_fb -= dac_inc;
        }
        if (direction == old_direction)
        {
            direction_counter += 1;
            if (direction_counter > 5)
            {
                dac_inc = (dac_inc >= 512)? 512 : (dac_inc << 1);
                direction_counter = 0;
            }
        }
        else
        {
            direction_counter = 0;
            dac_inc = (dac_inc == 1)? 1 : dac_inc >> 1;
            dac_inc = 1;
        }
        old_direction = direction;
    }

    //RETROAZIONE CON GUADAGNO UNITARIO
    if (fbgen == 0)
    {
        if (adc_data > adc_set_point) led_count_fb -= 1;
        if (adc_data < adc_set_point) led_count_fb += 1;
    }

    //SATURAZIONE ED INVIO AL DAC
    led_count_fb=(led_count_fb>DAC_MAX_VAL)?DAC_MAX_VAL:
        ((led_count_fb<DAC_MIN_VAL)?DAC_MIN_VAL:led_count_fb);
    I2C_Send_Packet(led_count_fb, dac_chn_fb);
}
```

Algoritmo 2.17 Stati della modalità in ciclo chiuso

```
case STATE_FEEDBACK_GAIN:
case STATE_FEEDBACK_COMP:
    //RICERCA SETPOINT E CONTROLLO A COMPENSAZIONE
    TMR1_feedback_OFF();
    //INIZIALIZZAZIONE VARIABILI
    dac_chn_fb = dac_chn;
    adc_set_point = cast(&inBuffer[2]);
    led_count_fb = 0;
    min_dac = DAC_MIN_VAL;
    max_dac = DAC_MAX_VAL;
    //AVVIO ALGORITMO DI BISEZIONE
    if(findSetPoint())
    {
        //L'ALGORITMO HA SUPERATO LA SOGLIA DI CONVERGENZA
        Serial_Send_Packet(DATA_PACK, STATUS_FSP_FAIL, NULL, NULL);
        //ESCO SENZA ATTIVARE LA feedback()
    }
    else
    {
        //NOTIFICA TRAMITE SERIALE DEL TERMINE DELLA BISEZIONE
        Serial_Send_Packet(FSP_PACK, adc_reg_24[0], adc_reg_24[1],adc_reg_24[2]);
        TMR1_feedback_ON();
    }
    state = STATE_IDLE;
    //AVVIO COMPENSAZIONE AD INTERRUZIONE
break;
case STATE_FEEDBACK_RESUME:
    //RIATTIVA MODALITA' FEEDBACK
    TMR1_feedback_ON();
    Serial_Send_Packet(DATA_PACK, STATUS_OK, NULL, NULL);
    state = STATE_IDLE;
break;
case STATE_FEEDBACK_STOP:
    //DISABILITA LA MODALITA' FEEDBACK
    TMR1_feedback_OFF();
    state = STATE_IDLE;
    Serial_Send_Packet(ADC_PACK, adc_reg_24[0], adc_reg_24[1],  adc_reg_24[2]);
break;
```

2.4.4 Confronto dei controlli a guadagno unitario e dinamico

Al fine di limitare gli effetti della deriva termica a regime si è deciso di introdurre una compensazione in ciclo chiuso. La strategia più semplice è aumentare o diminuire di 1 il valore inviato al DAC che controlla il LED sulla base della tensione misurata dall'ADC. Questo tipo di controllo è risultato però inadeguato, poiché la deriva iniziale cresce molto rapidamente e un guadagno unitario richiede un tempo di inseguimento troppo elevato. L'utilizzo del controllo a guadagno dinamico rende più veloce l'avvicinamento al valore di regime; lo strumento diviene così sufficientemente reattivo per un utilizzo reale.

In Figura 2.40 è possibile notare, che per piccoli stimoli, entrambi i controlli raggiungano il valore di regime contemporaneamente. Aumentando il gradino impostato (Figura 2.41), inizia a differenziarsi la tecnica di pilotaggio, difatti il valore di regime, in modalità di controllo a guadagno dinamico, viene raggiunto dopo 1.8 secondi circa, mentre il controllo a guadagno unitario impiega 4.5 secondi.

In Figura 2.42 è possibile vedere come, all'uscita dell'algoritmo di bisezione, il controllo a guadagno dinamico riesca a raggiungere il valore di regime in circa 2.5 secondi, mentre il controllo a guadagno unitario raggiunge il regime dopo 21 secondi.

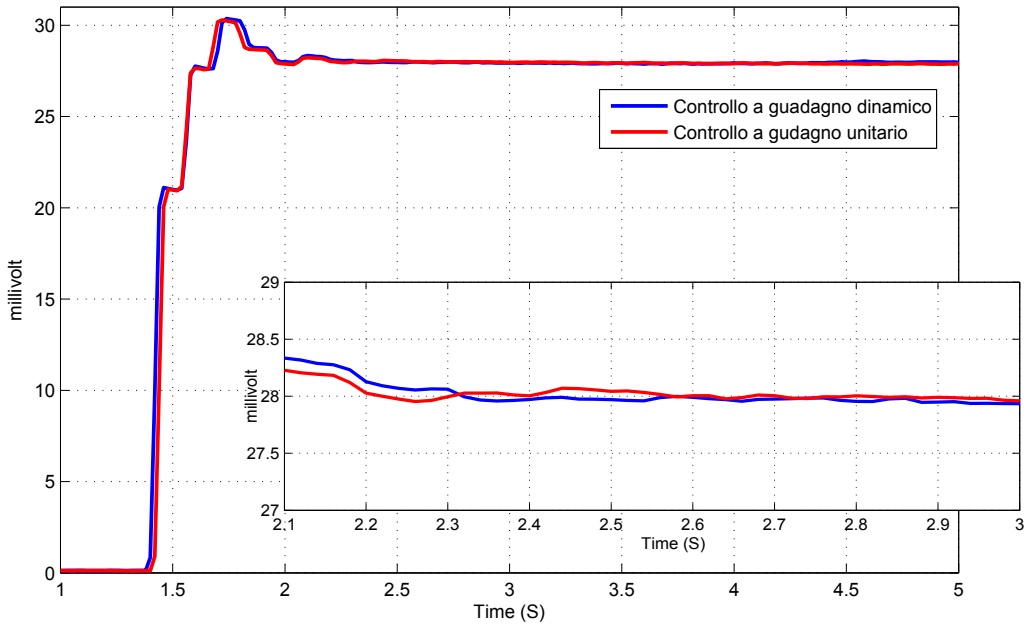


Figura 2.40: Confronto tra controllo a guadagno unitario e dinamico con gradino di 28mV sul LED verde

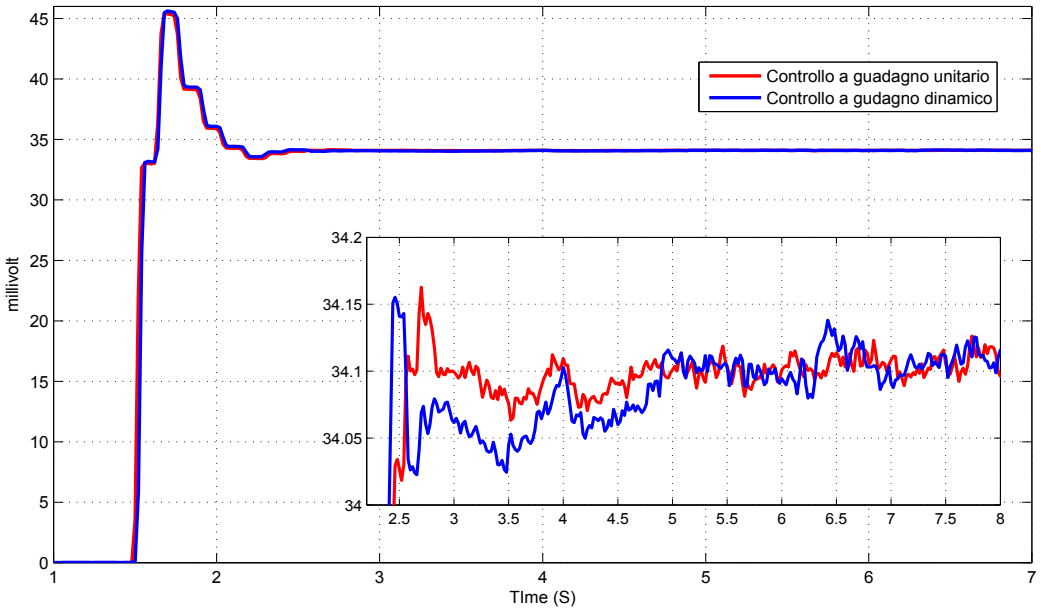


Figura 2.41: Confronto tra controllo a guadagno unitario e dinamico con gradino di 34mV sul LED blu

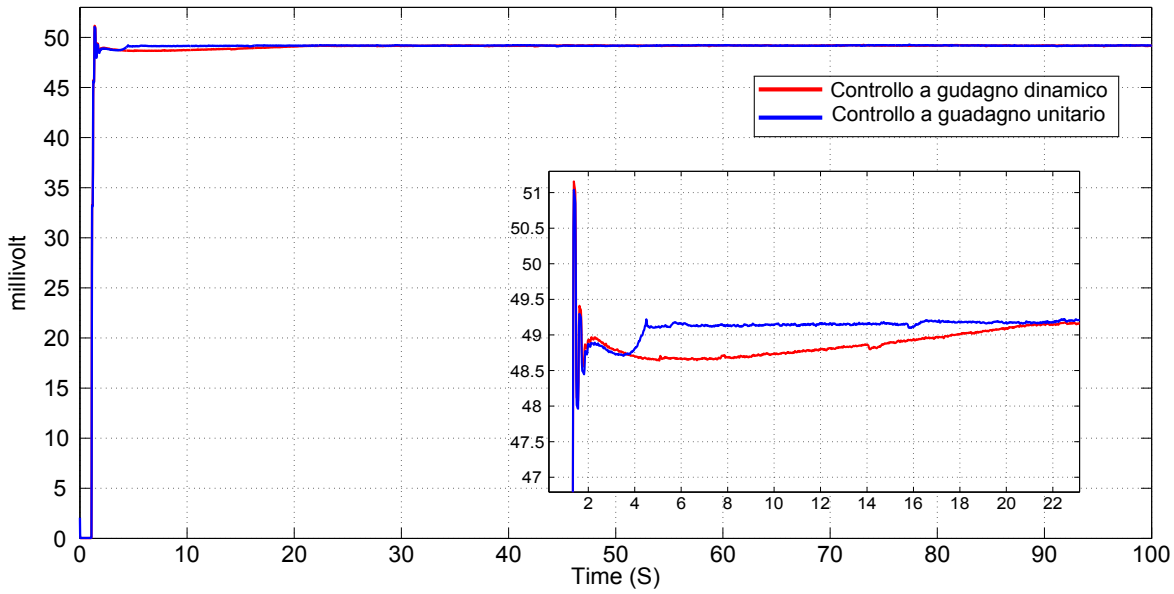


Figura 2.42: Confronto tra controllo a guadagno unitario e dinamico con gradino di 49mV sul LED blu

Capitolo 3

Illuminatore PWM

3.1 Introduzione

La scheda in questione è stata realizzata parallelamente all'illuminatore analogico, con la quale condivide la maggior parte dei componenti ma differisce nel pilotaggio dei LED. Difatti, a differenza della precedente, questa scheda utilizza una tecnica detta a "modulazione di larghezza di impulso" in breve PWM (Pulse-Width Modulation). Tale tipo di modulazione digitale permette di ottenere una corrente a media variabile dipendente dal rapporto tra durata dell'impulso positivo e l'intero periodo.

La scheda PWM utilizza lo stesso layout della scheda analogica e ritroviamo la maggior parte degli integrati precedentemente illustrati, ad eccezione del PIC sostituito con un CPLD e del circuito di alimentazione dei LED. L'obiettivo di questa implementazione è quello di realizzare un'illuminatore meno accurato e preciso ma che comunque mantenga una soddisfacente ripetibilità nel tempo. Ci si aspetta inoltre un miglior comportamento termico per via della maggiore efficienza di

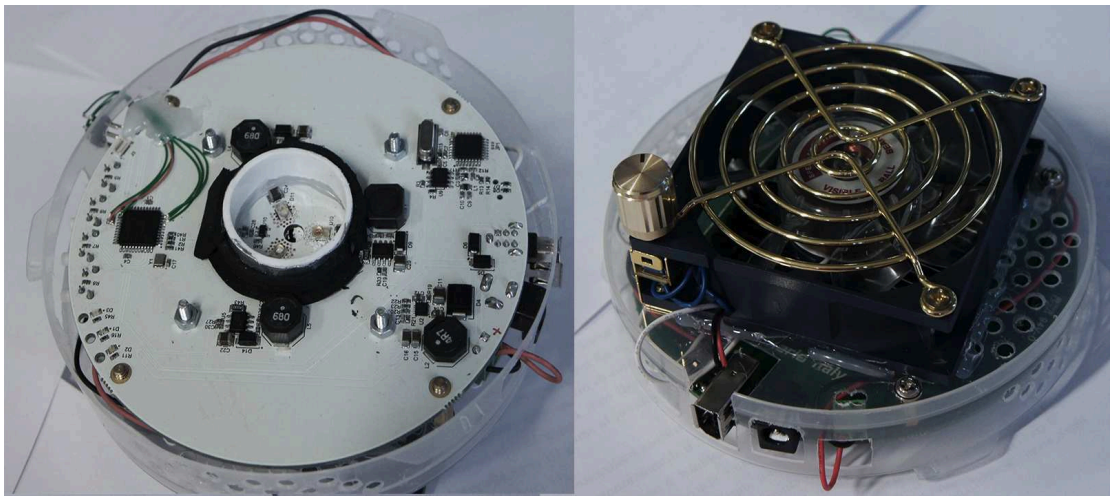


Figura 3.1: Vista superiore ed inferiore dell'illuminatore PWM

rendimento dovuta al pilotaggio non lineare.

Il controllo dei LED viene fatto in anello aperto quindi il sistema necessita di una taratura iniziale tramite strumentazione dedicata, una volta ricostruita la caratteristica dei LED è possibile pilotarli tramite software ad alto livello. Un'altra motivazione per la quale è stato realizzato questo illuminatore è stata l'opportunità di produrre la scheda senza gravare sui costi, visto che accorpare più circuiti su un unico pannello non comporta un significativo incremento dei costi. Quindi è stata sfruttata questa possibilità per effettuare test sul comportamento delle telecamere stimolate da una luce pulsante, visto che su questa seconda scheda viene controllato il valor medio della corrente considerando che quest'ultima presenta un consistente ripple.

3.2 Componenti

3.2.1 Complex Programmable Logic Device

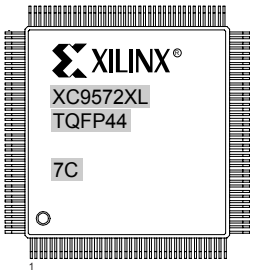


Figura 3.2: CPLD Xilinx XC9572XL

L'elemento che si occupa del pilotaggio della scheda è il XC9572XL della Xilinx. Il dispositivo è un Complex Programmable Logic Device, in breve CPLD. Un CPLD è un integrato composto da più elementi logici programmabili che permettono di realizzare un circuito digitale tramite reti combinatorie e sequenziali. Il CPLD è stato preferito al PIC per la capacità di operare a velocità elevate, necessarie al fine di generare il PWM per il controllo dei LED. La progettazione del circuito è stata effettuata utilizzando il software ISE sempre fornito dalla Xilinx e tramite il linguaggio VHDL. Il VHDL è un linguaggio che descrive il funzionamento e la struttura dei componenti del circuito digitale che andranno a funzionare in maniera concorrenziale. La scelta delle operazioni svolte dal CPLD sono state limitate dalla quantità di 72 macrocelle/flip-flop a disposizione, quindi sono state ridotte a due:

- Settaggio di un PWM sul led comandato da remoto.
- Invio della temperatura su richiesta.

Nella sezione 3.4 vedremo nello specifico come sono state implementate. Il funzionamento concettuale è simile a quello descritto inizialmente per il PIC in Figura 2.4.

3.2.2 Alimentatore Switching

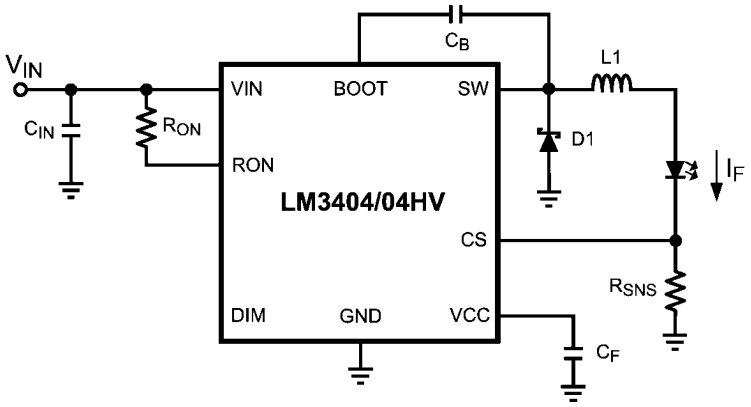


Figura 3.3: Regolatore switching LM3404

Per l'alimentazione dei LED è stato scelto il regolatore switching LM3404 prodotto dalla National Semiconductor. Il dispositivo pilota il LED con un controllo PWM ad alta frequenza sulla corrente, il cui valor medio è determinato dalla resistenza R_{on} ed accetta sull'ingresso *DIM* un segnale logico generato dal CPLD. Operando un ulteriore modulazione PWM su questo segnale, è possibile parzializzare la corrente che andrà a scorrere nel LED. La forma d'onda del pilotaggio del pin *DIM* è visibile in Figura 3.4. Il rapporto tra T_{on} ed il periodo $(T_{on} + T_{off})$ prende il nome di Duty Cycle.

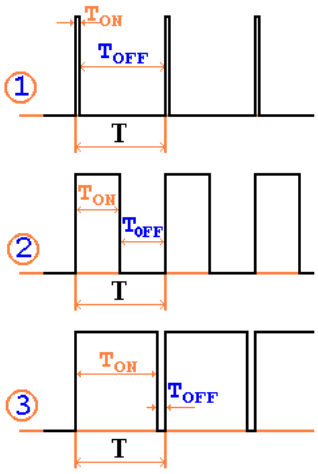


Figura 3.4: Forme d'onda per il pilotaggio del regolatore switching

In Figura 3.5 è possibile visualizzare come un PWM, generato dal CPLD, in ingresso all'alimentatore (in rosso) venga trasformato in un PWM per la carica

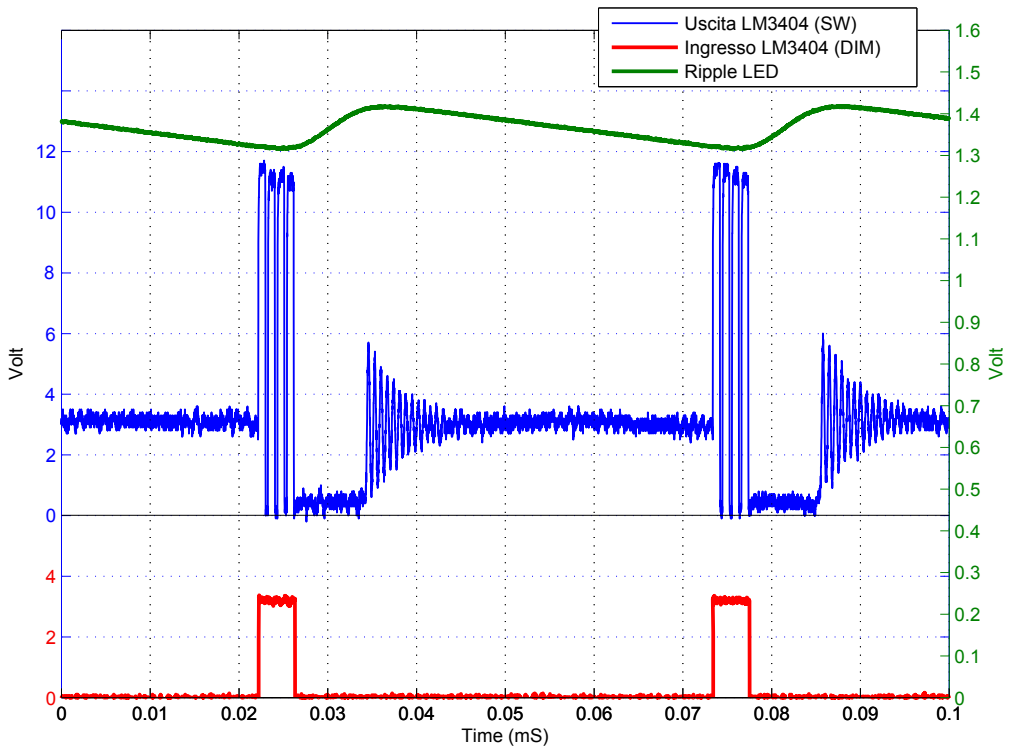


Figura 3.5: Forme d'onda del circuito di alimentazione

dell'induttanza. Il principio di funzionamento è del tipo step-down con controllo in retroazione della corrente misurata ai capi di un resistenza di sensing R_{sns}. Ai capi della R_{sns} è misurata, tramite il pin *CS*, la caduta di tensione V_{sns} generata dalla I_{led}. La V_{sns} è confrontata con una tensione di riferimento interna di 200 mV, quando risulta minore viene messo in conduzione un MOSFET, collegato al pin *SW,* per la durata di tempo $t_{on-LM3404} = 1.34 \cdot 10^{-10} \cdot \frac{R_{on}}{V_{in}}$. Al termine del $t_{on-LM3404}$ il regolatore disattiva il MOSFET per la durata di 300ns, trascorso questo tempo si riattiva il processo di comparazione della V_{sns}. Questo procedimento avviene nella fase T_{on} del PWM della CPLD mantenendo ad 1 il pin *DIM*. Nella fase T_{off} il pin *DIM* viene settato a 0 ed il MOSFET viene mantenuto spento. Il fenomeno, visibile sull'uscita (blu), oscillatorio smorzato è dato dall'interdizione del diodo schottky, che trasforma il circuito in un filtro composto da due condensatori ed un induttanza.

Nella Figura 3.5 è visibile l'andamento (in verde) della luminosità, che cresce nella fase di carica dell'induttanza e decresce nella fase di scarica, dando origine al fenomeno del ripple[1]. In Figura 3.6 è possibile vedere il circuito definitivo per il pilotaggio dei 3 LED. Inoltre è presente un sensore di temperatura come quello descritto nella sezione 2.2.3.

[1]Tale misura è stata effettuata utilizzando come fotorilevatore un LED ausiliario posto in prossimità del LED emettitore. I fotoni emessi dal fascio luminoso inducono una corrente inversa sul fotorilevatore che viene misurata facendola richiudere su una resistenza da 1MΩ con un'oscilloscopio.

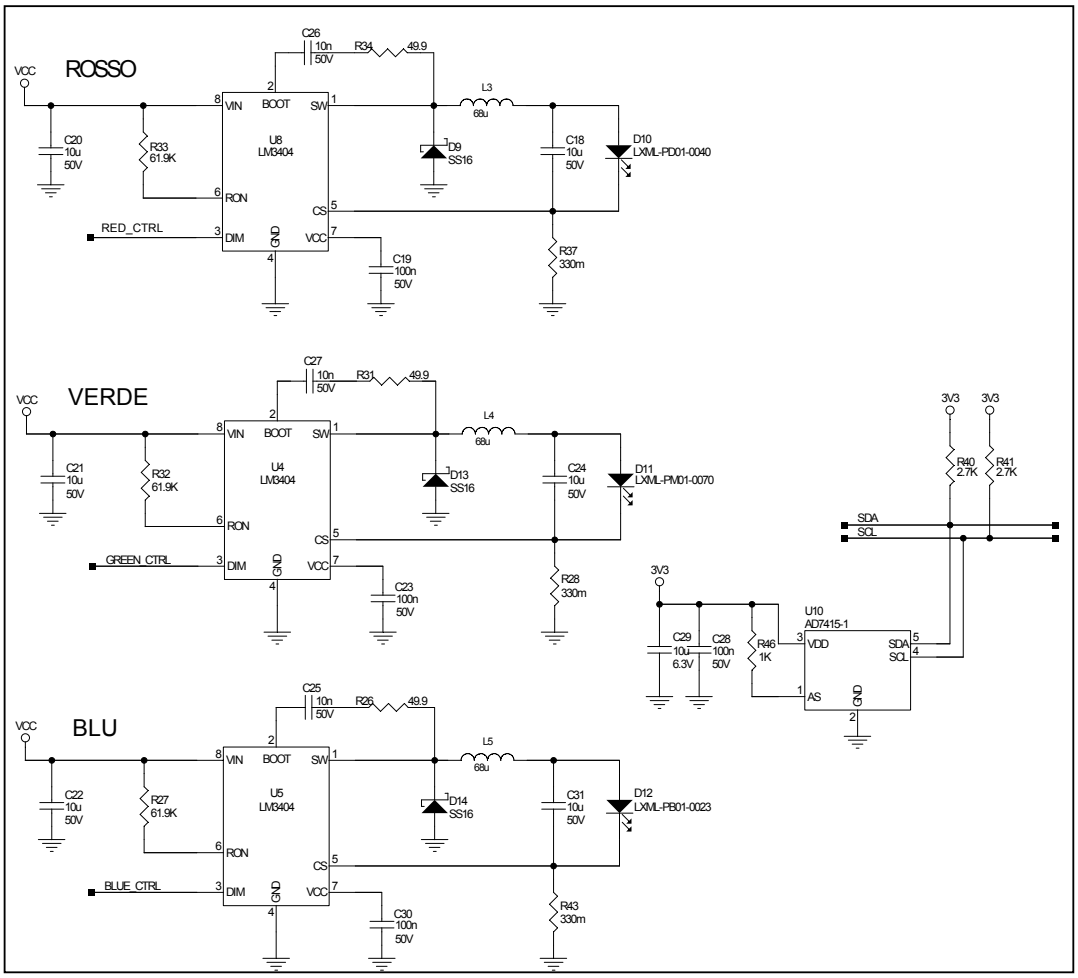


Figura 3.6: Circuito di alimentazione LED con nell'illuminatore PWM

3.2.3 Schema elettrico

In seguito alla scelta dei componenti ed alla progettazione del circuito step-down è stato realizzato lo schema elettrico del dispositivo visibile in Figura 3.7 ed il circuito in Figura 3.6. Successivamente si è proceduto a realizzare il PCB e le schede.

3.3 Pilotaggio LED

3.3.1 Interfaccia di comunicazione

L'interfaccia di comunicazione che permette di inviare e ricevere dati dalla CPLD al PC è gestita tramite protocollo seriale, utilizzando, come precedentemente descritto nella sezione 2.2.4, un convertitore USB - Seriale.

I pacchetti inviati al CPLD da remoto sono composti da 3 byte come in Figura 3.8. Il primo byte contiene i 5 bit meno significativi del valore sul quale viene calcolato il duty-cycle mentre nel secondo byte è contenuta la parte più significativa del valore e 2 bit che discriminano il LED da pilotare. In Figura 3.9 è rappresentato il pacchetto utilizzato per la richiesta della temperatura costituita dall'indirizzo del sensore di temperatura. Il contenuto del trailer non ha significato, come informazione contenuta, ma fornisce un riferimento per la comunicazione I2C.

La risposta alla richiesta della temperatura è costituita da 2 byte ed ha la medesima forma descritta nella sezione 2.4.2.2. La CPLD genera un pacchetto di risposta anche nel caso in cui sia inviato un pacchetto contenente il duty-cycle da settare su un LED, il contenuto del pacchetto di risposta è 0xFFFF ed ha la funzione di acknowledgement.

3.3.2 Generazione del PWM

Il duty-cycle è generato per mezzo di un contatore, che alla ricezione del valore da remoto viene inizializzato con la durata della "fase on". La logica descritta in 3.3 ha il compito di pilotare il pin di uscita del LED prescelto, il contatore a 10 bit che viene istanziato raggiunge un massimo di 1024 incrementando alla velocità di un count ogni 50ns. La CPLD ha una velocità di 20 MHZ quindi il periodo risulta 19531 HZ. Al fine di valutare l'uscita della luminosità è stata generata una rampa incrementale di duty-cycle dal 0% al 100%. In Figura 3.11 è evidenziata l'uscita discretizzata del contatore a 10 bit. Tale evento è dovuto alla soglia T_{off} interna dell'alimentatore switching, che ammonta a 300 ns. Entro questo limite non si può agire sulla regolazione della carica dell'induttanza. Per rendere la caratteristica più omogenea è possibile rallentare il processo di generazione del PWM e quindi ridurre la risoluzione temporale. È stato modificato il contatore moltiplicandolo per 4, quindi mantenendo la stessa risoluzione di 1024 valori, ma con velocità 4 volte

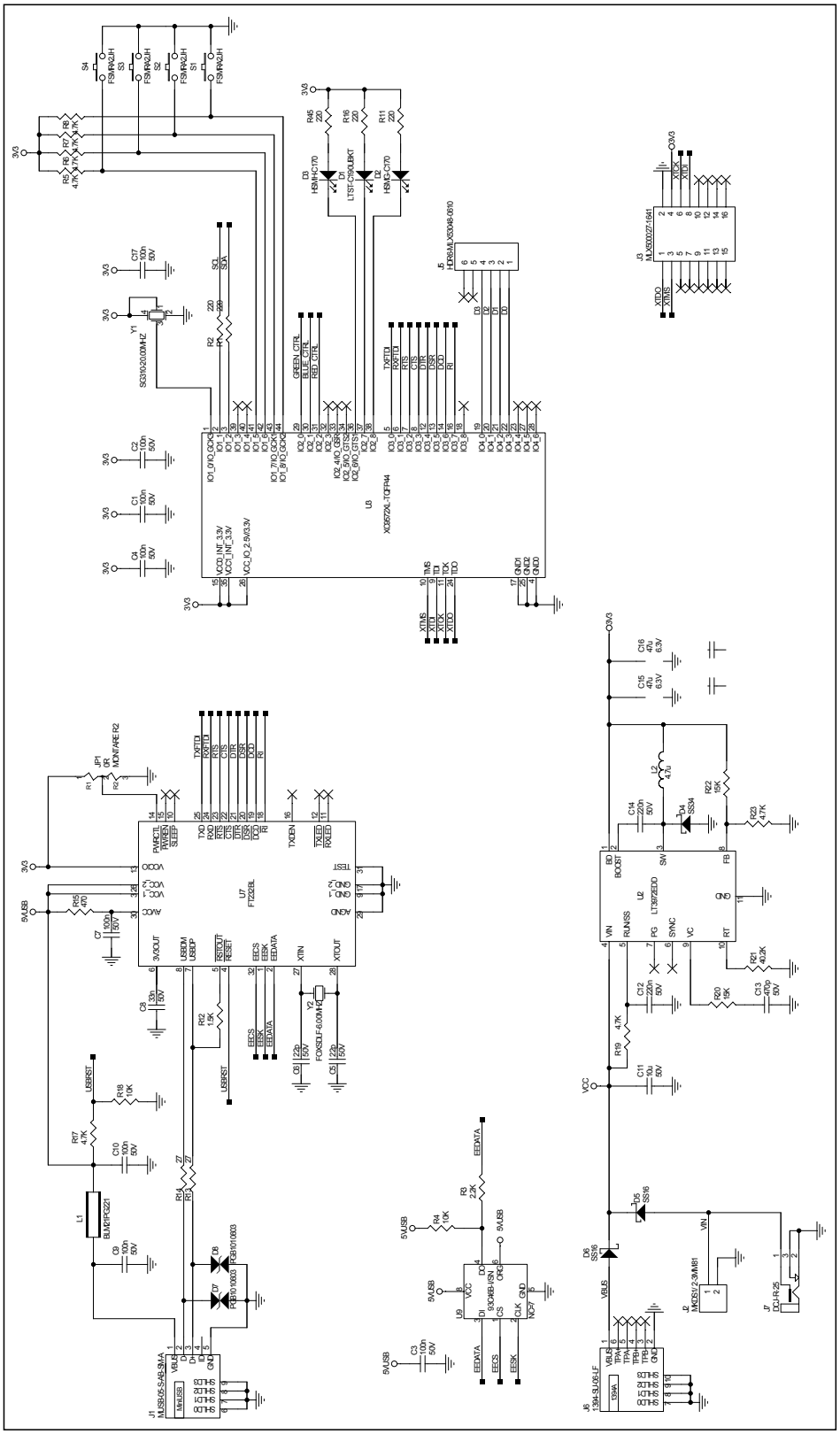


Figura 3.7: Circuito elettrico dell'illuminatore PWM, parte di controllo

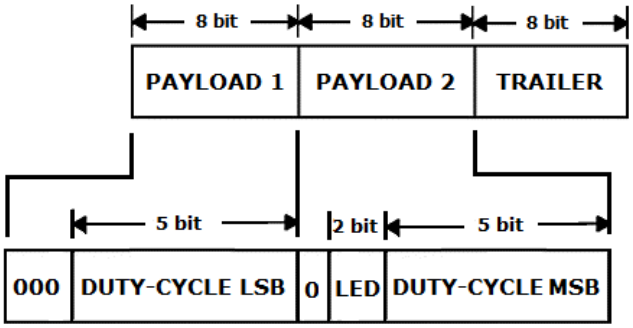


Figura 3.8: Schema del pacchetto spedito al CPLD per il settaggio del PWM

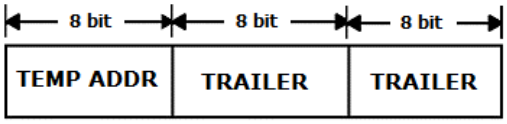


Figura 3.9: Schema del pacchetto spedito al CPLD per la richiesta della temperatura

più lenta; il periodo ottenuto è di 4883 Hz ma così facendo si rischia di incorrere in problemi di pulsazione visibile sull'uscita della telecamera.

3.3.3 Dipendenza dell'intesità luminosa dalla temperatura

Una delle caratteristiche peculiari di questa scheda è la ridotta dissipazione di potenza rispetto all'illuminatore analogico, difatti il problema viene risolto utilizzando il regolatore a commutazione che permette un pilotaggio più efficiente, riducendo così il problema dello smaltimento del calore. In Figura 3.12 è possibile vedere come la temperatura raggiunta rispetto all'illuminatore analogico sia inferiore di circa 8 gradi. Questo influisce riducendo la deriva termica a regime, portando la perdita di luminosità a 15uV ogni 10 secondi.

3.4 Firmware CPLD

Il circuito è composto da 2 reti che si occupano della comunicazione I2C/Seriale e della creazione del PWM. Gli ingressi del CPLD sono:

- *srlrec*: ingresso interfaccia seriale (RX).
- *srlsend*: uscita interfaccia seriale (TX).

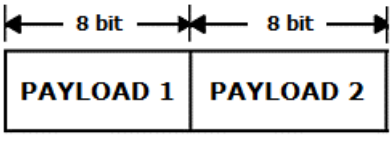


Figura 3.10: Pacchetto di risposta della CPLD

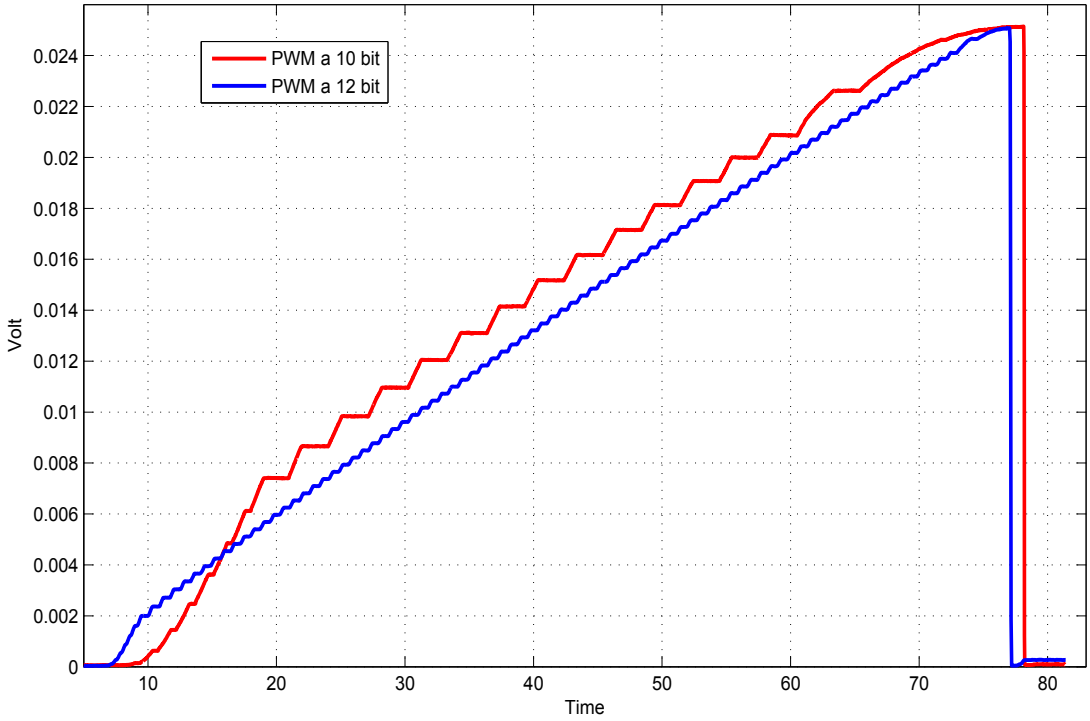


Figura 3.11: Caratteristiche di uscita dell'illuminatore PWM a confronto

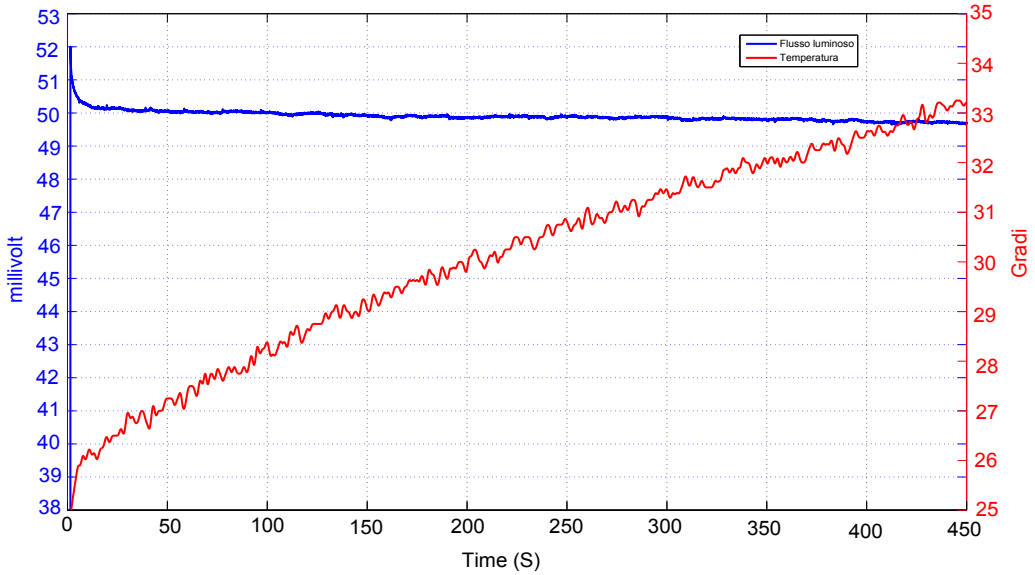


Figura 3.12: Deriva termica dell'illuminatore PWM

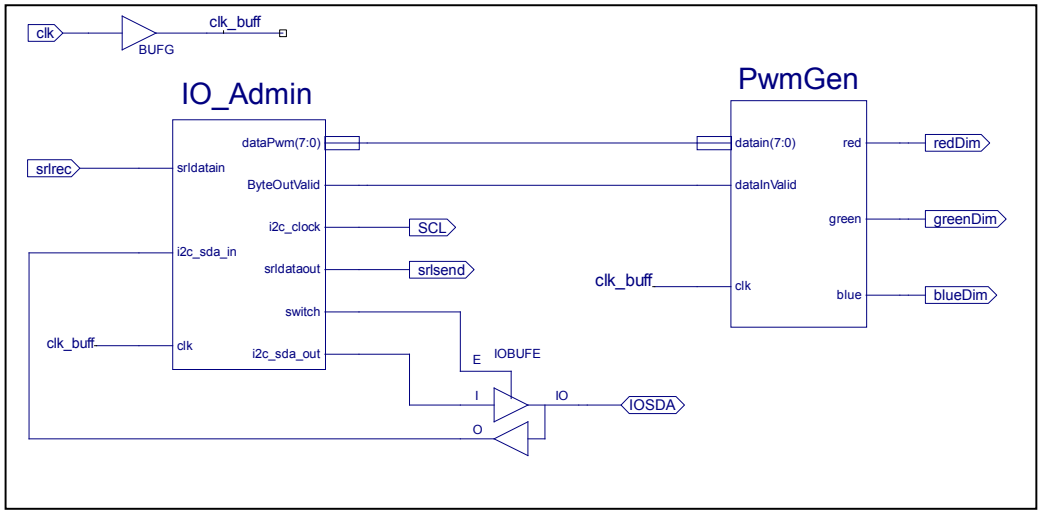


Figura 3.13: CPLD Top

- *SCL*: clock interfaccia I2C.
- *IOSDA*: linea dati input/output.
- *redDim*: pin destinato al PWM del led rosso.
- *greenDim*: pin destinato al PWM del led verde.
- *blueDim*: pin destinato al PWM del led blue.

Da notare che la porta *IOSDA* è gestita da un buffer bidirezionale pilotato tramite uno switch che permette di gestire separatamente l'ingresso e l'uscita alla rete *IO_Admin*.

3.4.1 IO_Admin

La rete *IO_Admin* si occupa della gestione dei pacchetti della comunicazione seriale, la lettura del sensore di temperatura e l'attivazione della rete *PwmGen*. L'algoritmo attende la condizione di start della seriale (generata dal PC) che consiste nel mettere a 0 logico il filo *srlrec*, in seguito si inizializza i contatori *bitsReceived* e *bytesReceived* i quali discrimineranno lo stato di ricezione del pacchetto. Ogni volta che un byte è stato ricevuto completamente si segnala tramite *ByteOutValid*, alla rete *PwmGen*, che il dato è pronto per essere prelevato.

Visto che le risorse del CPLD erano limitate l'interfaccia I2C è stata pilotata sfruttando le temporizzazioni dell'interfaccia seriale, quindi le due comunicazioni avvengono in parallelo. Difatti alla ricezione del primo byte da *srlrec*, questo viene inviato direttamente su *SDA*. Se il primo byte contiene l'indirizzo del sensore di temperatura il dispositivo risponde su *SDA* tramite due byte che sono rispediti su *srlsend*, cortocircuitando le connessioni. Altrimenti, se il pacchetto contiene un

Algoritmo 3.1 Parte 1 della rete IO_Admin

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity IO_Admin is
    Port(    clk : in  STD_LOGIC;
    srldatain : in  STD_LOGIC;
    i2c_sda_in : in  STD_LOGIC;
    srldataout : out STD_LOGIC ;
    i2c_clock : out STD_LOGIC ;
    i2c_sda_out : out STD_LOGIC;
    ByteOutValid : out STD_LOGIC := '0';
    dataPwm : out STD_LOGIC_VECTOR(7 DOWNTO 0):= (others => '0');
    switch : out STD_LOGIC := '1');
end IO_Admin;

architecture Behavioral of IO_Admin is
signal baudcounter : STD_LOGIC_VECTOR(7 DOWNTO 0):= (others => '0');
signal bitsReceived: STD_LOGIC_VECTOR(3 DOWNTO 0):= (others => '0');
signal idataPwm : STD_LOGIC_VECTOR(7 DOWNTO 0):= (others => '0');
signal din: STD_LOGIC; signal dout: STD_LOGIC:='1'; signal iSCL : STD_LOGIC := '1';
signal SDA_OUT: STD_LOGIC := '1';
signal bytesReceived: STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
signal switch_p1: STD_LOGIC:= '1'; signal switch_p2: STD_LOGIC:= '1';
signal endOfCount_0: STD_LOGIC;
signal endOfCount_1: STD_LOGIC;
--INIT MACCHINA A STATI
type stateType is ( stop, start, tx);
signal state: stateType := stop;

begin
dataPwm <= idataPwm;
i2c_clock <= iSCL;
i2c_sda_out <= SDA_OUT;
srldataout <= dout;

--GENERAZIONE DEI SEGNALI PER LA GESTIONE
--DEL CLOCK DELLA SERIALE E DELL'I2C
process(clk)
begin
    --OGNI FRONTE IN SALITA DEL CLOCK
    if clk'event and clk = '1' then
        din <= srldatain;
        if baudcounter = 78 then
            endOfCount_0 <= '1' ;
        else
            endOfCount_0 <= '0';
        end if;
        if baudcounter = 173 then
            endOfCount_1 <= '1';
        else
            endOfCount_1 <= '0';
        end if;
    end if;
end process;

--RICEZIONE DEL PACCHETTO DALLA SERIALE
process(clk)
begin
    if clk'event and clk = '1' then
        --ATTENDO LO START BIT DEL NUOVO BYTE RICEVUTO DALLA SERIALE
        if bitsReceived = "0000"  and srldatain = '1' then
            baudcounter <= (others => '0');
        else
            --INIZIO L'AQUISIZIONE DEL NUOVO BYTE
            if (bitsReceived = 0 and endOfCount_0 = '1') or endOfCount_1 = '1' then
                baudcounter <= (others => '0');
                idataPwm <= din & idataPwm(7 downto 1);
                bitsReceived <= bitsReceived + 1;
                if bitsReceived = 9  then
                    bitsReceived <= "0000";
                    ByteOutValid <= '0';
                    bytesReceived <= bytesReceived +1;
                end if;
            else
                --FINE RICEZIONE DEL BYTE, SEGNALO ALLA PwmGen CHE
                --IL DATO E' VALIDO
                baudcounter <= baudcounter + 1;
                if bitsReceived = 9 and endOfCount_0 = '1' then
                    ByteOutValid <= '1';
                end if;
            end if;
        end if;
        --FINE DELLA RICEZIONE DEL PACCHETTO
        if bytesReceived = 3 then
            bytesReceived <= "00";
        end if;
    end if;
end process;
```

Algoritmo 3.2 Parte 2 della rete IO_Admin

```
--GESTIONE DELLA COMUNICAZIONE I2C
process(clk)
begin
    if clk'event and clk = '1' then
        --INVIO DEL PRIMO BYTE, INDIRIZZO DEL SENSORE DI TEMPERATURA
        if bytesReceived = 0 then
            SDA_OUT <= din;
        else
            SDA_OUT <= '0';
        end if;
        --CLOCK SULL'USCITA SCL
        if bitsReceived = 0 then
            iSCL <= '1';
        else
            if endOfCount_0 = '1' or endOfCount_1 = '1' then
                iSCL <= not iSCL; --clock su scl
            end if;
        end if;
        --GESTIONE DELLA PORTA TRI-STATE
        --COLLEGATA AD SDA
        if bytesReceived = 0 then
            if endOfCount_0 = '1' then
                if bitsReceived = 9 then
                    switch_p2 <= '0';
                end if;
            end if;
        else
            if endOfCount_0 = '1' then
                if bitsReceived = 9 then
                    switch_p2 <= '1'; --ouput
                elsif bitsReceived = 1 then
                    switch_p2 <= '0';
                end if;
            end if;
        end if;
        --RITARDO SULLO SWITCH DELLA TRISTATE
        switch_p1 <= switch_p2;
        switch <= switch_p1;
    end if;
end process;

--MACCHINA A STATI PER LA GESTIONE DELL'INVIO
--AL PC DELLA TEMPERATURA
process (clk)
begin
    if clk'event and clk = '1' then
        case state is
            when stop =>
                if bytesReceived /= 0 and srldatain = '0' and bitsReceived = 0 then
                    state <= start;
                end if;
                dout <= '1';
            when start =>
                if endofcount_0 = '1' and bitsReceived /= 0 then
                    state <= tx;
                end if;
                dout <= '0';
            when tx =>
                if endofcount_0 = '1' and bitsReceived = 9 then
                    state <= stop;
                end if;
                --SDA VIENE COLLEGATO A TX
                dout <= i2c_sda_in;
        end case;
    end if;
end process;
end Behavioral;
```

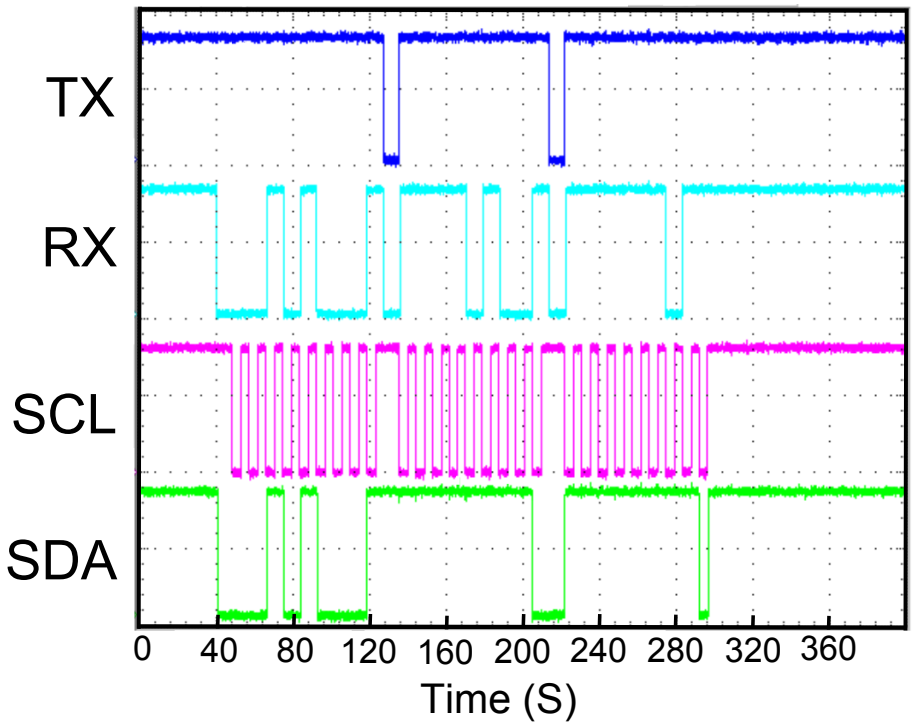


Figura 3.14: Pacchetto contenente n duty-cycle

duty-cycle, il sensore di temperatura non risponde e i due byte di risposta su *srlsend* contengono una serie di 1, che corrisponde al valore alto del bus I2C. Questo meccanismo è permesso dal valore dell'indirizzo del sensore che inizia per "1", difatti nella rete *PwmGen* vengono acquisiti soltanto i byte con cifra più significativa pari a 0.

In Figura 3.14 è visibile la gestione di un pacchetto contenente un duty-cycle. In *RX* abbiamo i tre byte in arrivo, contenenti il dato, parallelamente si invia il primo byte su I2C ma il sensore non risponde, quindi si procede all'invio su *TX* dell'ack di ricezione per il PC, che equivale ad una serie di 1 (i due 0 corrispondono allo startbyte). Il primo byte è spedito sul bus I2C (pin SDA) al fine di risparmiare risorse, inoltre viene inviato l'ack di fine byte e la copia dello *startbyte* della seriale. Questi segnali non pregiudicano il comportamento del sensore di temperatura.

In Figura 3.15 è visibile la richiesta e la lettura della temperatura. In *RX* abbiamo i tre byte in arrivo, dove il primo contiene l'indirizzo del sensore di temperatura e gli altri due hanno funzione di riferimento nella la rete *IO_Admin* per allineare i clock interni. In questo caso il sensore risponde su *SDA* con due byte contenenti la temperatura, pertanto si procede ad unire *SDA* con *TX*.[2]

[2]Da notare che la rappresentazione dei byte tra seriale e I2C risulta invertita quindi l'indirizzo del sensore di temperatura deve essere spedito al rovescio e la temperatura ricevuta da remoto deve essere invertita da destra verso sinistra.

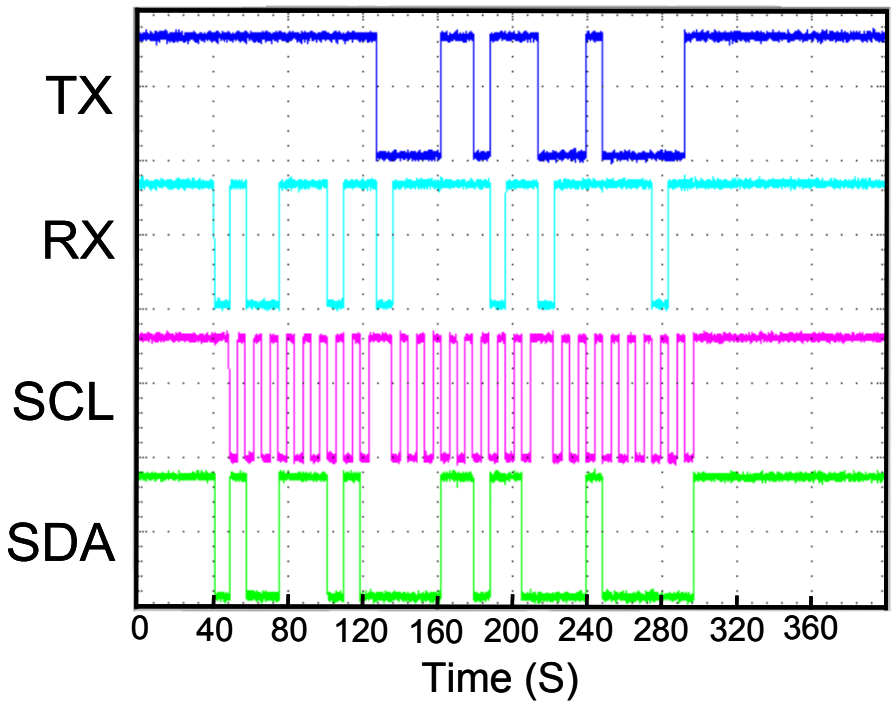


Figura 3.15: Pacchetto contenente la temperatura

3.4.2 PwmGen

Il blocco *PwmGen* si occupa della creazione del PWM per il pilotaggio dei LED. Ogni volta che la rete IO_Admin segnala un nuovo byte la rete PwmGen controlla il contenuto dei primi tre bit che andranno ad identificare il contenuto del byte:

- 000: Parte bassa del duty-cycle
- 001: Parte alta del duty-cycle e led rosso selezionato
- 010: Parte alta del duty-cycle e led verde selezionato
- 011: Parte alta del duty-cycle e led blu selezionato

Non è possibile pilotare i tre LED contemporaneamente per via della struttura del codice che prevede un solo contatore. Il duty-cycle ricevuto viene istanziato in *cntRGB* che specifica la durata della parte alta del PWM. Ad ogni istante di clock la rete incrementa il valore di *counter* e successivamente confrontato con il contenuto di *cntRGB* per stabilire il valore del pin di uscita. Raggiunta la soglia dei 1024 valori il registro *counter* si resetta.

Algoritmo 3.3 Blocco PwmGen

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
entity PwmGen is
    Port (
            clk : in  STD_LOGIC;
            datain: in STD_LOGIC_VECTOR (7 DOWNTO 0);
            dataInValid: in STD_LOGIC;
            red: out STD_LOGIC;
            green: out STD_LOGIC;
            blue: out STD_LOGIC);
end PwmGen;

architecture Behavioral of PwmGen is
signal cntRGB: STD_LOGIC_VECTOR(9 DOWNTO 0):= (others => '0');
signal cntRGB_LO: STD_LOGIC_VECTOR(4 DOWNTO 0):= (others => '0');
signal dataInValidRisEd: STD_LOGIC;
signal counter: STD_LOGIC_VECTOR(9 DOWNTO 0):= (others => '0');
signal dataInValid_d : STD_LOGIC; signal PWMout: STD_LOGIC;
signal ledswitch: STD_LOGIC_VECTOR(1 DOWNTO 0);

begin
red <= PWMOut when ledswitch = "01" else '0';
green <= PWMOut when ledswitch = "10" else '0';
blue <= PWMOut when ledswitch = "11" else '0';
dataInValidRisEd <= '1' when (dataInValid = '1' and dataInValid_d = '0') else '0';

--FLIP FLOP SULL'INGRESSO
process (clk)
begin
    if clk'event and clk = '1' then
        dataInValid_d <= dataInValid;
    end if;
end process;

--ACQUISIZIONE DEL VALORE PER LA
--GENERAZIONE DEL PWM
process (clk)
begin
    if clk'event and clk = '1' then
        if dataInValidRisEd = '1' then
            if (datain(7 DOWNTO 5) = "000") then
                cntRGB_LO <= datain(4 DOWNTO 0);
            elsif (datain(7 DOWNTO 5) = "001") or (datain(7 DOWNTO 5) = "010") or (datain(7
                DOWNTO 5) = "011") then
                cntRGB <= datain(4 DOWNTO 0) & cntRGB_LO;
                ledswitch <= datain(6 DOWNTO 5);
            end if;
        end if;
    end if;
end process;

--GENERAZIONE PWM
process(clk)
begin
    if clk'event and clk = '1'  then
        counter <= counter + 1;
        if counter(9 downto 0) < cntRGB then
            PWMOut <= '1';
        else
            PWMOut <= '0';
        end if;
    end if;
end process;
end Behavioral;
```

Capitolo 4

Illuminatori a confronto

4.1 Introduzione

Dopo aver definito il funzionamento e le caratteristiche costruttive degli illuminatori, passiamo adesso al confronto diretto dei due. Il fine di questa analisi comparativa è quello di determinare come la differente modalità di pilotaggio dei LED sia in grado di influenzare l'analisi dei sensori d'immagine.

4.2 Risposta al gradino

Questo test è stato predisposto per valutare la risposta al gradino delle due schede e con questo apprezzare eventuali differenze nel transitorio.Per effettuare questa comparazione è stata cercata una corrispondenza tra count del DAC (scheda analogica) e duty cycle del PWM (scheda PWM) in modo tale che per entrambi i dispositivi risultasse uguale la tensione letta a regime ai capi di un fotodiodo di misura posto sulla porta di uscita della sfera. Successivamente si è provveduto ad inviare la corrispondente eccitazione ai due illuminatori, campionando l'andamento della tensione durante transitorio. È stato ottenuto un comportamento pressoché identico per le due configurazioni. Il test è stato ripetuto per più valori ed i risultati hanno mantenuto coerenza. Questo è dovuto al fatto che il decadimento dell'efficienza del LED su un transitorio di breve durata, è sostanzialmente indipendente dalla modalità di pilotaggio ed è invece totalmente correlabile alla corrente impostata.

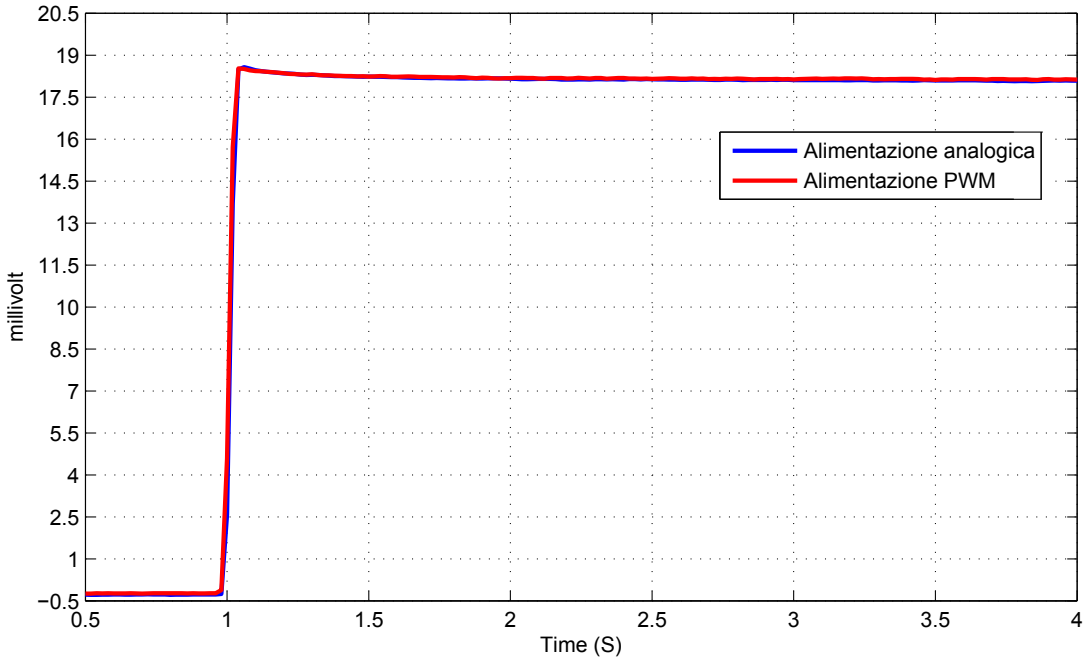


Figura 4.1: Confronto risposta al gradino per il LED verde a 1600 fotoni

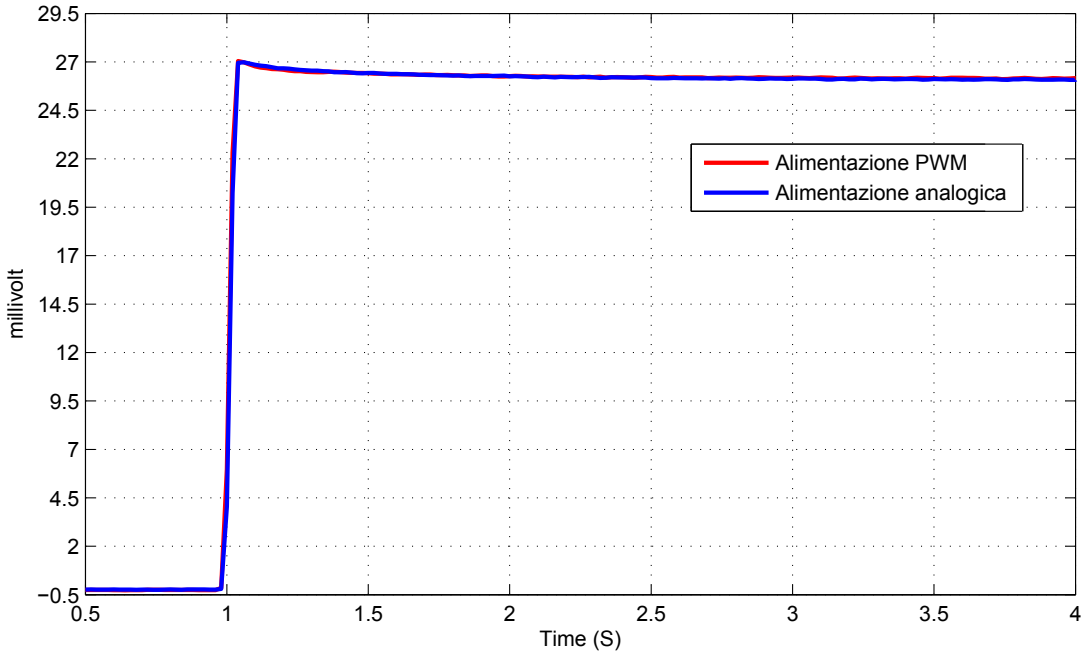


Figura 4.2: Confronto risposta al gradino per il LED verde a 2300 fotoni

4.3 Deriva termica

Al fine di determinare quanto l'intensità luminosa fosse dipendente dalla temperatura, il segnale generato dal rilevatore a fotodiodo è stata monitorato su tempi lunghi. Il valore di tensione rilevato è stato convertito in fotoni su unita di superficie al fine

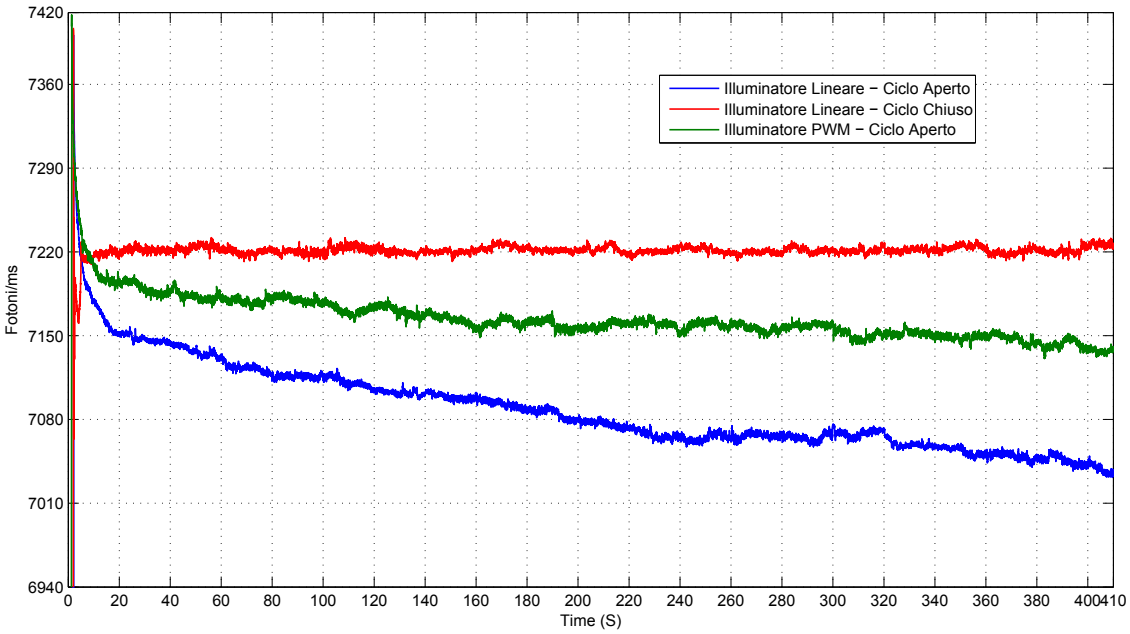


Figura 4.3: Confronto del drift termico senza ventola a regime

Illuminatore	Pendenza della deriva termica a regime	Rumore della misura
Analogico - Ciclo Aperto	13 fotoni ogni 10s	±3 fotoni
Analogico - Ciclo Chiuso	-	±5 fotoni
PWM - Ciclo Aperto	7 fotoni ogni 10s	±6 fotoni

Tabella 4.1: Dati relativi al drift termico senza ventola

di poter meglio comprendere quanto questo fenomeno potesse inficiare i test sulle telecamere.

Applicando le formule descritte precedentemente nelle sezioni 2.3.3 e 2.3.1 otteniamo il numero medio di fotoni al millisecondo che raggiungono ogni pixel:

- $\frac{\mu_p[AD]}{t_{esp}[ms]} = 50.34 \cdot A_{px}\,[\mu m^2] \cdot \lambda\,[\mu m] \cdot P_{sup}\left[\frac{\mu W}{cm^2}\right]$.

Nel caso specifico della telecamera utilizzata, la Lira 424BW descritta in Appendice, dispone di un'area per pixel pari a $A_{px} = 7.4 \cdot 7.4 = 54.76 \mu m^2$. Il LED blu utilizzato è caratterizzato da una lunghezza d'onda di $\lambda = 475nm$ e di un coefficiente di correzione pari a $c_{BPW34} = 5.3$. Nel caso di raffreddamento passivo del dissipatore, otteniamo i valori visibili in Tabella 4.1.

La pendenza è stata calcolata come media tra 20 e 200 secondi, mentre il rumore è stato stimato in un intervallo di 2 secondi. Per tempi maggiori rispetto alla simulazione è possibile osservare come la temperatura raggiunga un asintoto e pertanto tenda a diminuire la pendenza della deriva. Al fine di velocizzare il transitorio termico del sistema e limitare la temperatura di lavoro, usurante sia per i LED che per l'elettronica, è stata introdotta una ventola sul dissipatore. In questo caso la temperatura tende a assestarsi più velocemente su una soglia asintotica più bassa della precedente come visibile in Figura 4.4.

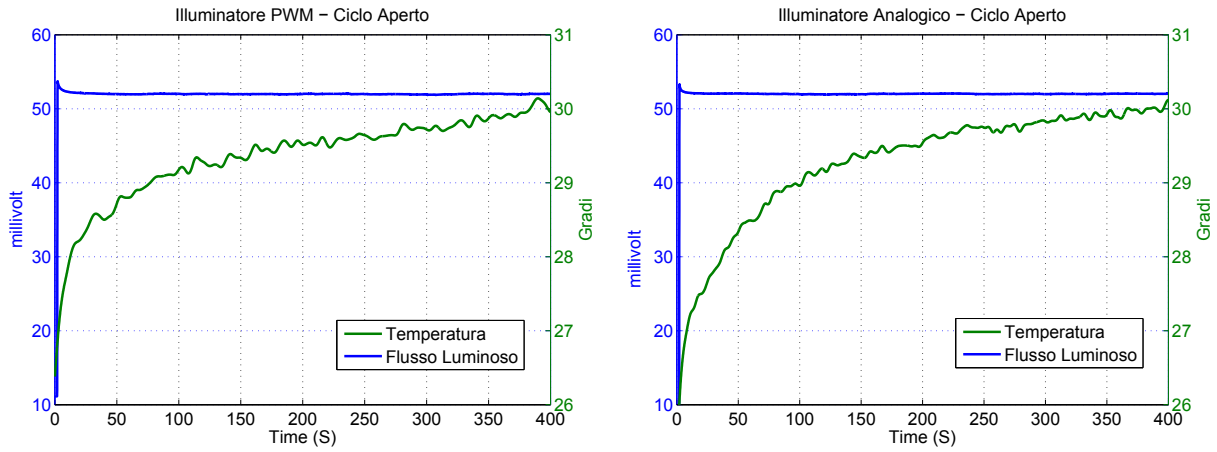


Figura 4.4: Andamento della temperatura con dissipazione attiva

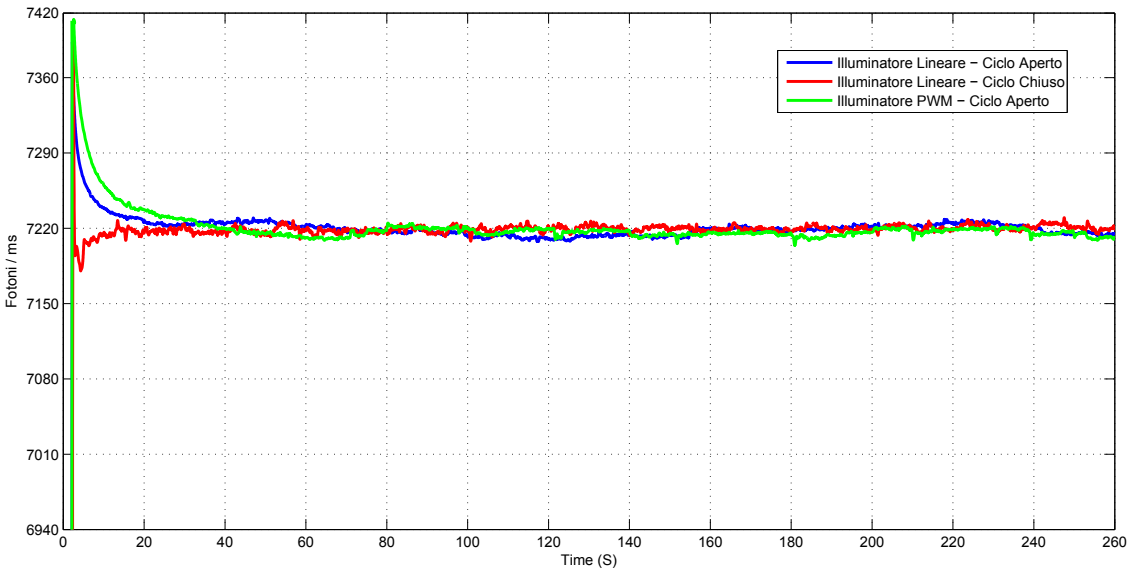


Figura 4.5: Confronto del drift termico con ventola a regime

In Figura 4.5 è possibile osservare come l'introduzione della ventola comporti il raggiungimento di un valore di regime accettabilmente stabile entro i 30 secondi, anche per i pilotaggi in ciclo aperto.

4.4 Test EMVA

Infine sono stati effettuati alcuni test seguendo lo standard EMVA [9]. I test eseguiti sono descritti in modo approfondito in [5]. La piattaforma di test è stata realizzata tramite la sfera prodotta presso Alkeria S.r.l., sfruttando gli algoritmi del software A.C.T., e alternando l'utilizzo dell'illuminatore analogico e PWM.

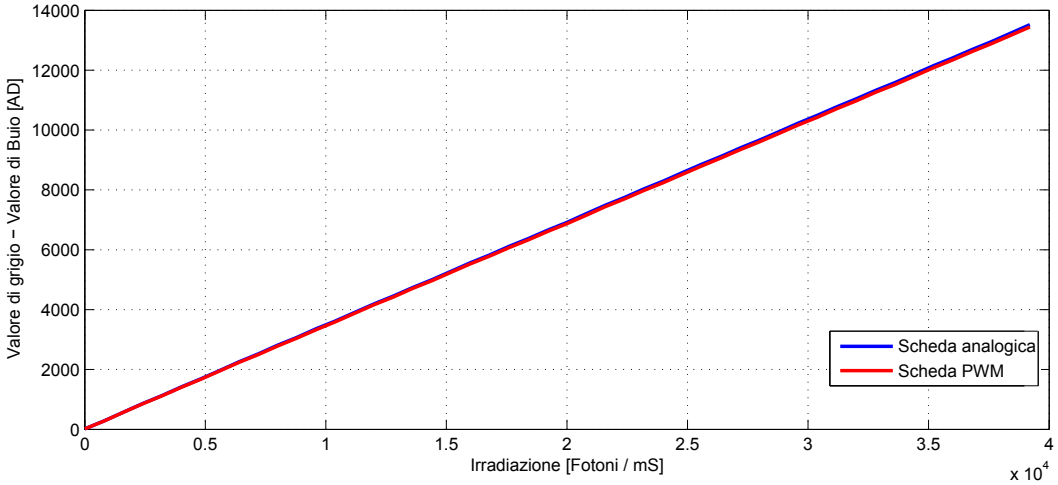


Figura 4.6: Linearità

Il test di linearità evidenzia, al variare del tempo di esposizione e mantenendo l'illuminazione ad intensità costante, come le due schede abbiamo un comportamento ininfluente sul risultato finale.

La deviazione dalla linearità è calcolata come il discostamento percentuale dal valore medio calcolato dalla regressione dei valori in Figura 4.6. Il comportamento a dente di sega è dovuto al seguente fenomeno. I tempi di esposizione stabiliti ad alto livello sono espressi secondo multipli di 100µs ma non corrispondono all'effettivo tempo di esposizione della telecamera: la quantizzazione del tempo di esposizione effettivo viene fatta in base alla durata del tempo necessario per acquisire una riga di pixel del sensore (t_{riga}). Quando ad alto livello si imposta un certo tempo di esposizione (t_{esp}), il sensore rimane effettivamente esposto per il numero (q_{riga}) di multipli del t_{riga} che più si avvicina al tempo di esposizione impostato t_{esp}. Questo fa si che, ad esempio, impostando una quantità doppia di tempo di esposizione, non si ritrovi la stessa proporzione nel q_{riga}.

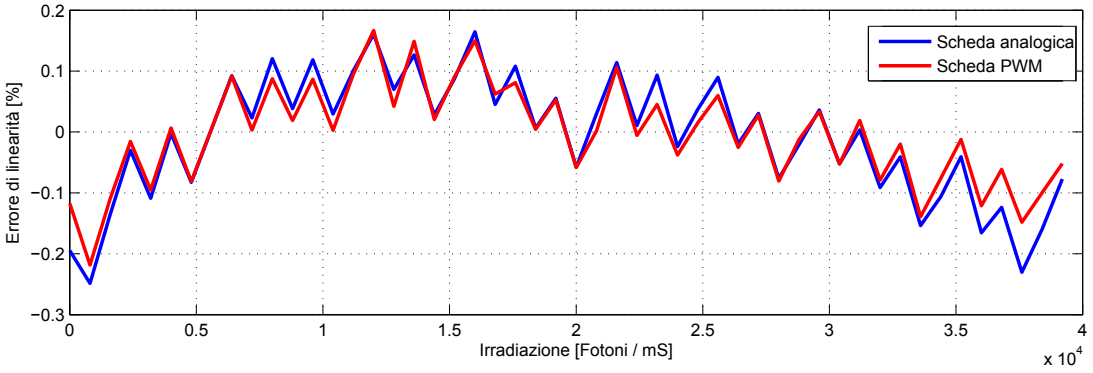


Figura 4.7: Deviazione dalla linearità

Il signal-to-noise ratio (SNR) è calcolato mediante la formula

- $SNR\left(\mu_p\right) = \frac{\eta\mu_p}{\sqrt{\left(\frac{\sigma_q^2}{K^2}\right)+\sigma_d^2+\eta\mu_p}}$

dove μ_pè il numero dei fotoni medi, η l'efficienza quantica, σ_q^2 è il rumore di quantizzazione, σ_d^2 rappresenta i disturbi provenienti dalla lettura del sensore d'immagine e dall'amplificazione, K il guadagno totale.

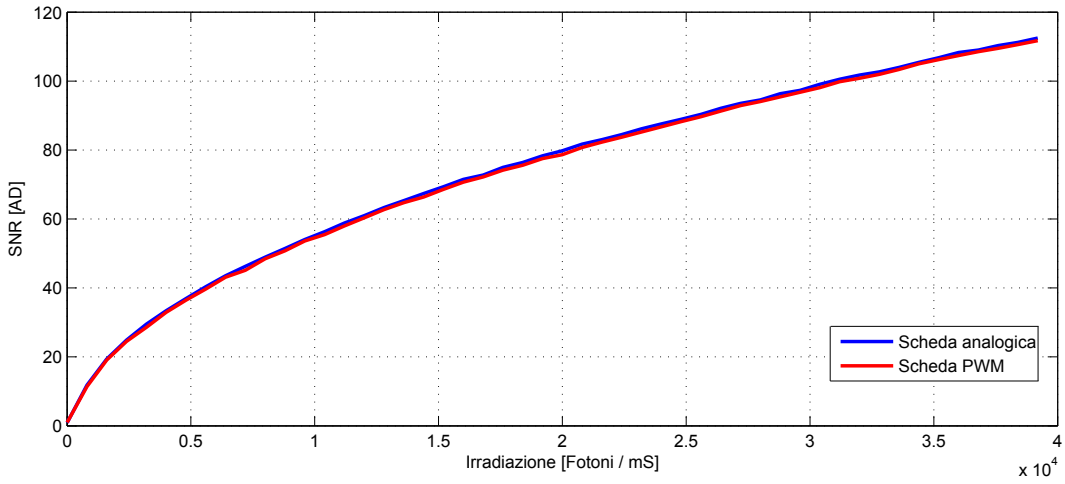


Figura 4.8: SNR

L'ultimo test prevede il calcolo della varianza temporale, al crescere del tempo di esposizione. La formula applicata risulta:

- $\sigma_y^2 = \frac{1}{2MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(y^A\left[m\right]\left[n\right] - y^B\left[m\right]\left[n\right]\right)^2$

dove y^A e y^Bsono due immagini campionate a distanza temporale prefissata di dimensione $M \times N$. Tramite il test della Figura 4.9 è possibile evidenziare come la varianza temporale della scheda PWM risulti leggermente più alta rispetto a quella della scheda analogica. Questo è dovuto al fenomeno del ripple precedentemente

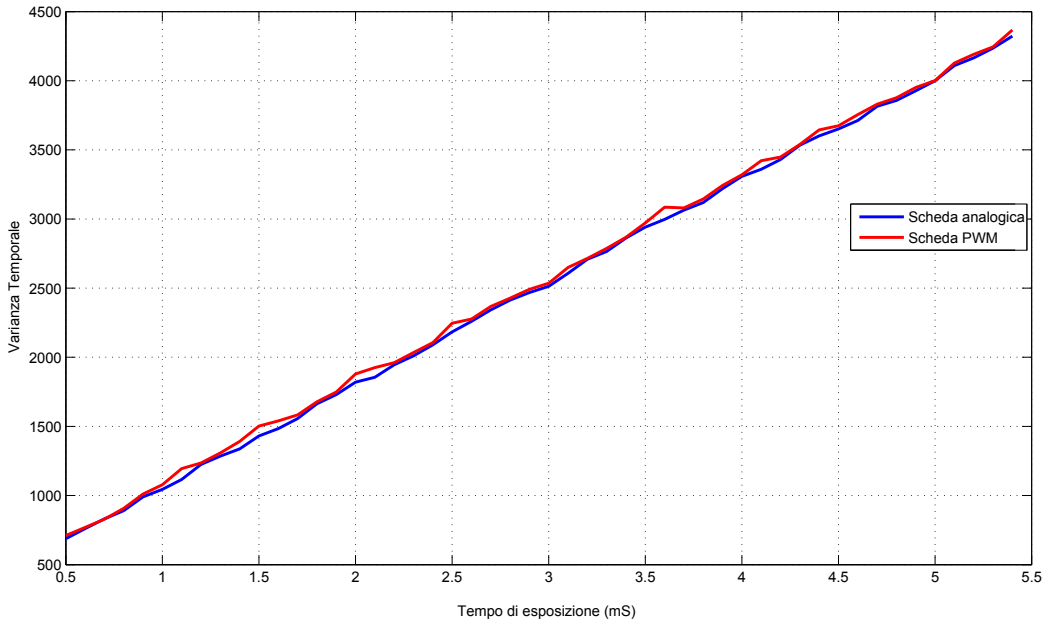


Figura 4.9: Varianza Temporale

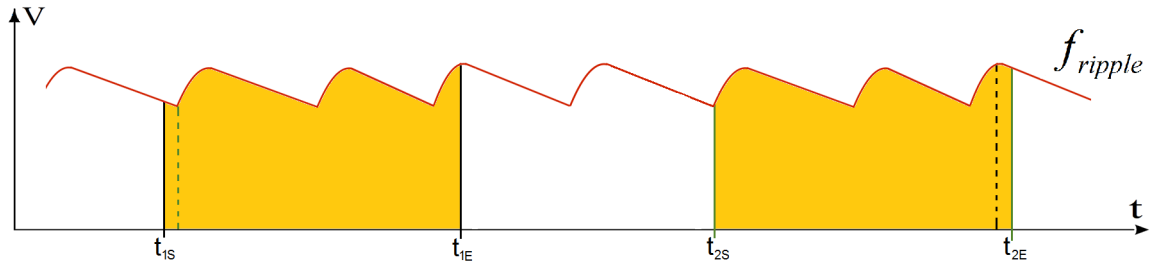


Figura 4.10: Ripple acquisito durante il tempo di esposizione

spiegato, difatti durante il tempo di esposizione si integreranno n onde ognuna della durata di 51.2 us[1].

Il problema sorge al campionamento della prima onda e dell'ultima, la varianza tra le due immagini cresce quando il tempo di esposizione non è un multiplo perfetto del periodo del ripple e quindi la probabilità che si campioni la stessa quantità di luce varia in base a quando la telecamera inizia e finisce l'esposizione. Dalla Figura 4.10 è evidente che, per due immagini acquisite ad una certa distanza temporale, mantenendo costante il duty-cycle, si ottenga

- $\int_{t_{1S}}^{t_{1E}} f_{ripple} dt < \int_{t_{2S}}^{t_{2E}} f_{ripple} dt$.

Ovviamente tale fenomeno compare maggiormente nei tempi di esposizione più piccoli mentre risulta praticamente irrilevante per esposizioni superiori a 20 ms.

[1]Con frequenza del PWM di 19531 Hz

Capitolo 5

Conclusioni

Gli obiettivi di questa Tesi erano la creazione di un sistema di misura per effettuare confronti con telecamere di terze parti, avere un sistema per effettuare test di qualità nella linea di produzione, avere un sistema di misura relativo da utilizzare durante lo sviluppo di nuove telecamere per determinare come certi parametri influissero sul comportamento della telecamera.

Durante il lavoro presso Alkeria S.r.l. sono stati realizzati vari prototipi di sfera al fine di studiarne il comportamento e migliorarne l'efficienza ottimizzando il processo produttivo. Congiuntamente alla creazione delle sfere sono state prodotte 2 schede per l'illuminazione analogica e 2 schede per l'illuminazione digitale. Tali sfere dispongono di firmware dedicati e di una libreria unificata in C# che permette la comunicazione da PC, documentata nell'Appendice E. Inoltre è stato realizzato un sistema di retroazione della luminosità dell'uscita della sfera che permette di controllare attivamente il pilotaggio dei LED.

È stato così possibile confrontare tre sistemi diversi:

1. illuminatore analogico in ciclo aperto;
2. illuminatore PWM in ciclo aperto;
3. illuminatore analogico in ciclo chiuso.

L'illuminatore analogico in ciclo aperto dispone di un interfaccia semplice di comando e un'uscita limitatamente rumorosa. Il circuito di alimentazione richiede una retroazione della corrente al fine di mantenere stabile la corrente a regime. Tra i difetti troviamo le elevate temperature generate per mezzo dalla tecnica di pilotaggio applicata. L'aumento della temperatura comporta un decadimento del rendimento di conversione che produce una diminuzione dell'emissione luminosa sul LED attivato.

L'illuminatore PWM risulta più efficiente dal punto di vista termico grazie all'utilizzo di componenti digitali che permettono una ridotta dissipazione di potenza. Il sistema ha una componentistica più semplice rispetto alla scheda analogica e può

essere controllato facilmente da remoto. I difetti di tale modello sono la ridotta risoluzione e il ripple presente sulla corrente che attraversa i LED.

L'illuminatore analogico in ciclo chiuso permette di risolvere il problema del decadimento della luminosità, compensando il fenomeno tramite retroazione dell'irradianza in uscita dalla sfera. Tale metodo non riduce la dissipazione di potenza, che comunque persiste, e obbliga ad utilizzare una ventola per migliorare lo smaltimento del calore. L'illuminatore è composto da molti componenti e il pilotaggio risulta più complesso rispetto ai precedenti. La velocità di raggiungimento del regime è da ritenersi considerevolmente migliorabile, anche se non può essere considerato un parametro limitante per il set di misure attualmente effettuate.

Negli sviluppi futuri potrà essere realizzato un nuovo illuminatore caratterizzato da un nuovo circuito comprendente l'ADC, che adesso risiede su scheda esterna, al fine di limitare le connessioni ed i disturbi. Nel corso di una eventuale revisione progettuale dovrà necessariamente essere valutata l'opportunità di utilizzare un microcontrollore più evoluto che permetta di rendere più veloci gli algoritmi di conversione e compensazione. Uno dei fattori limitanti all'estensione di ulteriori funzionalità del sistema risulta essere l'eccessiva semplicità del microcontrollore e dei relativi tool di sviluppo impiegati su questo progetto. Si ritiene opportuno porre sotto considerazione inoltre, che un più potente e articolato sistema di sviluppo, proprio di microprocessori di fascia più alta, potrebbe permettere l'implementazione e l'agevole verifica di algoritmi e funzionalità complesse.

L'evoluzione naturale A.C.T. potrà essere la realizzazione di un sistema di test automatizzato per la verifica delle telecamere della linea di produzione, tale sistema potrà generare una reportistica dei dati acquisiti insieme ad una validazione delle telecamere del tipo pass/fail.

Figura 5.1: Due generazioni di sfere a confronto

Appendice A

Modelli Spice

Per poter compiere le simulazioni con LTSpice in modo accurato è stata reperito online un modello precostruito del MOSFET composto dai file:

- *LTC\LTspiceIV\lib\sym\irfh3702pbf.asy*
- *LTC\LTspiceIV\lib\sub\irfh3702pbf.lib.*

Algorithm A.1 Disegno del modello del MOSFET *irfh3702pbf.asy*

```
Version 4
SymbolType CELL
LINE Normal 48 48 48 96
LINE Normal 16 80 48 80
LINE Normal 40 48 48 48
LINE Normal 16 48 40 44
LINE Normal 16 48 40 52
LINE Normal 40 44 40 52
LINE Normal 16 8 16 24
LINE Normal 16 40 16 56
LINE Normal 16 72 16 88
LINE Normal 0 80 8 80
LINE Normal 8 16 8 80
LINE Normal 48 16 16 16
LINE Normal 48 0 48 16
WINDOW 0 56 32 Left 0
WINDOW 3 56 72 Left 0
SYMATTR Prefix X
SYMATTR Value irfh3702pbf
SYMATTR ModelFile irfh3702pbf.lib
SYMATTR SpiceModel irfh3702pbf
SYMATTR Description N-Channel MOSFET transistor
PIN 48 0 NONE 0
PINATTR PinName D
PINATTR SpiceOrder 1
PIN 0 80 NONE 0
PINATTR PinName G
PINATTR SpiceOrder 2
PIN 48 96 NONE 0
PINATTR PinName S
PINATTR SpiceOrder 3
```

Inoltre è stato creato un modello ad hoc dei LED Rebel Lumilux per svolgere le simulazioni con LTSpice in modo accurato. Il file in cui aggiungere il modello è: *LTC\LTspiceIV\lib\cmp\standard.dio.*

Algorithm A.2 Modello della caratteristica del MOSFET *irfh3702pbf.lib*

```
.SUBCKT irfh3702pbf 1 2 3
* SPICE3 MODEL WITH THERMAL RC NETWORK
**************************************
*   Model Generated by MODPEX    *
*Copyright(c) Symmetry Design Systems*
*      All Rights Reserved       *
*   UNPUBLISHED LICENSED SOFTWARE   *
* Contains Proprietary Information    *
*    Which is The Property of       *
*   SYMMETRY OR ITS LICENSORS    *
*Commercial Use or Resale Restricted *
* by Symmetry License Agreement    *
**************************************
* Model generated on Aug 4, 09
* MODEL FORMAT: SPICE3
* Symmetry POWER MOS Model (Version 1.0)
* External Node Designations
* Node 1 -> Drain
* Node 2 -> Gate
* Node 3 -> Source
M1 9 7 8 8 MM L=100u W=100u
.MODEL MM NMOS LEVEL=1 IS=1e-32
+VTO=2.16903 LAMBDA=0 KP=42.6639
+CGSO=1.41418e-05 CGDO=1.84247e-07
RS 8 3 0.0001
D1 3 1 MD
.MODEL MD D IS=4.78582e-17 RS=0.00464443 N=0.701834 BV=30
+IBV=0.00025 EG=1 XTI=1 TT=1e-07
+CJO=5.77035e-10 VJ=2.00056 M=0.509092 FC=0.5
RDS 3 1 5e+07
RD 9 1 0.0001
RG 2 7 6
D2 4 5 MD1
* Default values used in MD1:
* RS=0 EG=1.11 XTI=3.0 TT=0
* BV=infinite IBV=1mA
.MODEL MD1 D IS=1e-32 N=50
+CJO=1.64615e-10 VJ=2.67208 M=0.3 FC=1e-08
D3 0 5 MD2
* Default values used in MD2:
* EG=1.11 XTI=3.0 TT=0 CJO=0
* BV=infinite IBV=1mA
.MODEL MD2 D IS=1e-10 N=0.415902 RS=3e-06
RL 5 10 1
FI2 7 9 VFI2 -1
VFI2 4 0 0
EV16 10 0 9 7 1
CAP 11 10 2.71531e-10
FI1 7 9 VFI1 -1
VFI1 11 6 0
RCAP 6 10 1
D4 0 6 MD3
* Default values used in MD3:
* EG=1.11 XTI=3.0 TT=0 CJO=0
* RS=0 BV=infinite IBV=1mA
.MODEL MD3 D IS=1e-10 N=0.415902
.ENDS irfh3702pbf
*SPICE Thermal Model Subcircuit
.SUBCKT irfh3702pbft 5 1
R_RTHERM1 5 4 18.384153
R_RTHERM2 4 3 16.184774
R_RTHERM3 3 2 8.0194450
R_RTHERM4 2 1 2.3361100
C_CTHERM1 5 1 0.0315339
C_CTHERM2 4 1 0.2410105
C_CTHERM3 3 1 0.0064503
C_CTHERM4 2 1 0.0006244
.ENDS irfh3702pbft
```

Algoritmo A.3 Modello del LED Rebel Lumilux

```
.model LXML-PM01 D
(
    Is=2.52n //Body diode saturation current
    Rs=0.1 //Source ohmic
    N=1.752 //Body diode emission coefficient
    Cjo=100p //Zero-bias body diode junction capacitance
    M=.4 //Body diode junction potential
    tt=20n //Body diode transit time
    Iave=450m //Average maximum current
    Vpk=75 //Peak voltage
    mfg=Luxeon
    type=silicon
)
```

Appendice B

Funzioni secondarie del PIC

Sono illustrate di seguito le funzioni utilizzate nel firmware del microcontrollore PIC e non menzionate precedentemente.

Algorithm B.1 Serial_Send_Packet

```
/* Name: Serial_Send_Packet
 * Synopsis: void Serial_Send_Packet(unsigned char start, unsigned char hibyte,
                        unsigned char midbyte,unsigned char lobyte)
 * Arguments: unsigned  char: Byte di intestazione
 *            unsigned  char: Parte alta payload
 *            unsigned  char: Parte centrale payload
 *            unsigned  char: Parte bassa payload
 * Description: Invio tramite interfaccia seriale di un pacchetto al pc
 */
void Serial_Send_Packet(unsigned char start, unsigned char hibyte,
                        unsigned char midbyte,unsigned char lobyte)
{
  //TRMT == 1 -> TSR VUOTO
  //TRMT == 0 -> TSR PIENO
  //CALCOLO DEL CHECKSUM
  unsigned char cks = (unsigned char)(start + hibyte + midbyte + lobyte) ;
  while (!TRMT);
  TXREG= start;
  while (!TRMT);
  TXREG= hibyte;
  while (!TRMT);
  TXREG= midbyte;
  while (!TRMT);
  TXREG= lobyte;
  while (!TRMT);
  TXREG= cks;
}
```

Algoritmo B.2 I2C_Send_Byte e I2C_Get_Byte

```
/* Name: I2C_Send_Byte
 * Synopsis: char I2C_Send_Byte(unsigned char data)
 * Arguments: unsigned  char :  byte da inviare
 * Description: Invia un byte sul bus I2C
 * Returns: Ritorna 0 se è avvenuto un errore, 1 se l'invio è andato a buon fine
 */
char I2C_Send_Byte(unsigned char data)
{
  char count;
  PORTB = 0;
  dly1u;
  for(count=8;count>0;count--)
  {
    if( (data & 0x80)== 0)
    {   //INIZIA IL MSB
        SDA = I2CLOW;
      SDA_TRIS=OUTPUT; //BIT == 0
    }
    else
        SDA_TRIS=INPUT; //BIT == 1
    dly1u;
    SCL_TRIS=INPUT; //CLOCK SU SCL
    dly1u;
    SCL_TRIS=OUTPUT;
    dly1u;
    data=data<<1; //PROSSIMO BIT
  }
  SDA_TRIS=INPUT; //RICEZIONE ACK
  SCL_TRIS=INPUT; //9' VALORE ALTO SU SCL
  dly1u;
  if(SDA) //CONTROLLO SE HO RICEVUTO UN ACK
    return(0); //NO ACK
  dly1u;
  SCL_TRIS=OUTPUT;
  dly1u;
  return(1);
}

/* Name: I2C_Get_Byte
 * Synopsis: char I2C_Get_Byte(void)
 *Description: Preleva un byte dal bus I2C
 *Returns: Ritorna il byte prelevato dal dispositivo I2C
 */
char I2C_Get_Byte(void)
{
  char count,Byte=0;
  SDA=I2CLOW;
  SCL=I2CLOW;
  dly1u;
  for(count=8;count>0;count--)
  {
    Byte=Byte <<1;
    SDA_TRIS=INPUT;
    if(I2C_Read_Bit())
      Byte +=1;
  }
  SDA=I2CLOW; //INVIO ACK
  SDA_TRIS=OUTPUT;
  dly1u;
  SCL_TRIS=INPUT;
  dly1u;
  SCL_TRIS=OUTPUT;
  dly1u;
  SDA_TRIS=INPUT;
  dly1u;
  return(Byte);
}
```

Algorithm B.3 I2C_Start, I2C_Stop e I2C_Read_Bit

```
/* Name: I2C_Start
 * Synopsis: void I2C_Start(void)
 * Description: Condizione di start, bus I2C
 */
void I2C_Start(void)
{
  PORTB = 0;
  SDA_TRIS=INPUT;
  SCL_TRIS=INPUT;
  dly1u;
  SDA_TRIS=OUTPUT;
  dly1u;
  SCL_TRIS=OUTPUT;
}

/* Name: I2C_Stop
 * Synopsis: void I2C_Stop(void)
 * Description: Condizione di stop, bus I2C
 */
void I2C_Stop(void)
{
  SDA_TRIS=OUTPUT;
  dly1u;
  SCL_TRIS=INPUT;
  dly1u;
  SDA_TRIS=INPUT;
}

/* Name: I2C_Read_Bit
 * Synopsis: char I2C_Read_Bit(void)
 * Description: Lettura del singolo bit
 * Returns: Ritorna il bit letto su SDA
 */
char I2C_Read_Bit(void)
{
  char Data=0;
  dly1u;
  SCL_TRIS=INPUT;
  dly1u;
  if(SDA !=0 ) Data=1;
  dly1u;
  SCL_TRIS=OUTPUT;
  dly1u;
  return(Data); // 0 : 1
}
```

Algorithm B.4 SPI_Send_Byte e SPI_Get_Byte

```
/* Name: SPI_Send_Byte
 * Synopsis: char SPI_Send_Byte(unsigned char data)
 * Arguments: unsigned  char : byte da inviare
 * Description: Preleva un byte dal bus SPI
 * Returns: Ritorna 0 se è avvenuto un errore, 1 se l'invio è andato a
    buon fine
 */
char SPI_Send_Byte(unsigned char data)
{
  char count;
  DOUT = 0;
  CLK = 0;
  CLK_TRIS = OUTPUT;
  DOUT_TRIS = OUTPUT;
  dly1u;
  for(count = 8; count > 0; count--)
  {
    if( (data & 0x80)== 0)
    {   //INIZIA IL MSB
      DOUT = 0;
        DOUT_TRIS = OUTPUT; // BIT == 0
    }
    else
         DOUT_TRIS=INPUT; //BIT == 1
    dly1u;
    CLK_TRIS = INPUT;
    dly1u;
    CLK_TRIS = OUTPUT;
    dly1u;
    data=data<<1; //PROSSIMO BIT
  }
  return(1);
}

/* Name: SPI_Get_Byte
 * Synopsis: char SPI_Get_Byte(void)
 * Description: Preleva un byte dal bus SPI
 * Returns: Ritorna il byte prelevato dal dispositivo SPI
 */
unsigned char SPI_Get_Byte(void)
{
  char count,Byte=0;
  CLK = 0;
  CLK_TRIS = OUTPUT;
  for(count=8;count>0;count--)
  {
    Byte=Byte <<1;
    dly1u;
    CLK_TRIS = INPUT;
    if(DIN) Byte +=1;
    dly1u;
    CLK_TRIS = OUTPUT;
  }
  return (Byte);
}
```

Algorithm B.5 SPI_Start e SPI_Stop

```
/* Name: SPI_Start
 * Synopsis: void SPI_Start(void)
 * Description: Condizione di start, bus SPI
 */
void SPI_Start(void)
{
  CLK = 0;
  DOUT = 0;
  CS = 0;
  CLK_TRIS = OUTPUT;
  DOUT_TRIS = OUTPUT;
  CS_TRIS = OUTPUT;
}

/* Name:  SPI_Stop
 * Synopsis:  void SPI_Stop(void)
 * Description :  Condizione di stop, bus SPI
 */
void SPI_Stop(void)
{
  CLK_TRIS = INPUT;
  DOUT_TRIS = INPUT;
  CS_TRIS = INPUT;
}
```

Appendice C

Dimensionamento della Sfera

Al fine di effettuare misurazioni accurate è necessaria una fonte di illuminazione il più uniforme possibile, così che ogni pixel del sensore sia sottoposto alla stessa intensità di luce. Se ad esempio la luce non fosse abbastanza uniforme, non potrebbero essere eseguiti correttamente i test sull'uniformità spaziale del sensore. Nel caso standard EMVA 1288 non viene imposto un sistema di illuminazione particolare, ma consiglia l'utilizzo di una Sfera Integrante (o di Ulbricht) dato che questo tipo di dispositivi garantisce una luce uniforme al 99%. Per tale motivo è stata realizzata la costruzione di una sfera di questo tipo, che ha la capacità di integrare il flusso luminoso che si sviluppa al suo interno. Data la natura di prototipo del sistema di misura costruito, è stato optato di mantenere ridotti i costi di sviluppo e perciò procedere all'autocostruzione della sfera. Per la costruzione ci si è basati principalmente sui documenti tecnici dei produttori di tali dispositivi e dei documenti scientifici trattanti tale argomento. La prima generazione di sfera è composta da due porte, di tipo simmetrico, ovvero con l'illuminatore posto in linea retta rispetto alla porta di uscita, utilizzando un deflettore nel mezzo. Per la scelta del tipo di Sfera da costruire devono essere considerati i seguenti punti:

- selezione del giusto diametro in base al numero e alla dimensione delle porte di uscita della luce e dei dispositivi periferici;
- scelta del rivestimento interno della Sfera in base allo spettro del segnale desiderato in uscita;
- utilizzo di deflettori (baffle) per convogliare la luce all'interno della Sfera.

La porta di uscita, ha condizionato il dimensionamento della Sfera per i seguenti motivi.

- I test devono essere condotti con la meccanica di attacco dell'ottica del sensore.

- Si deve poter generare in uscita una quantità di luce rilevabile da uno strumento che misuri l'irradiamento e che riesca a saturare la telecamera sotto test.
- Lo standard EMVA 1288 impone una distanza porta di uscita-sensore ben precisa, ovvero 8 volte il diametro della porta.
- Il produttore di Sfere consiglia di non utilizzare più del 5% della superficie della Sfera per le porte di ingresso/uscita.

I primi due punti creano un vincolo sulla dimensione minima della porta, infatti l'attacco dell'ottica del sensore è di tipo C, questo tipo di attacco ha un diametro di 25mm. È stato scelto di mantenere al di sopra di questo valore, il diametro della porta di uscita per poter illuminare tutto l'attacco. Inoltre, a parità di luce generata dall'illuminatore, un foro di uscita più piccolo convoglia meno luce verso il sensore e gli strumenti radiometrici. Gli ultimi due punti invece creano un vincolo sulla dimensione massima: avere un diametro della porta troppo elevato significa porre il sensore ad una distanza maggiore e parallelamente non garantire più l'uniformità della luce in quanto è presente una minore superficie su cui effettuare l'integrazione spaziale. Per questi motivi si è scelto di creare delle porte che complessivamente rappresentassero l'1% della superficie della Sfera, mantenendosi così abbondantemente sotto la soglia massima. Il diametro scelto per la porta di ingresso/uscita è stato di 35mm. Quindi la superficie totale della aperture è di $S_{porte} = 2 \cdot (\pi \cdot 1.75^2) = 19.62cm^2$. Imponendo una superficie 100 volte maggiore per la Sfera, si può ricavare il raggio $r_{sfera} = \sqrt{\frac{S_{porte} \cdot 100}{4\pi}} \approx 12.5cm$. Quindi il diametro scelto per la Sfera è di 25cm. Sono stati quindi reperiti due profilati plastici semisferici di tale dimensione su cui sono stati effettuati fori di 35mm di diametro.

La scelta del rivestimento interno della Sfera è basata sul fatto che era necessaria un'attenuazione ridotta sulle lunghezze d'onda utilizzate per i test, la superficie quindi deve avere una riflettanza maggiore possibile . Quindi si è proceduto a pitturare l'interno della sfera con una vernice di colore bianco. Esistono vernici particolari al solfato di bario che garantiscono una risposta piatta su tutto lo spettro visibile, arrivando a livelli di riflettanza superiori al 99%. Per questo prototipo, è stato scelto l'utilizzo di una comune vernice bianca, per poi valutarne l'efficienza ed eventualmente passare ad una vernice più tecnica.

All'interno della sfera è stato posto un baffle. I deflettori o baffle servono per convogliare la luce su specifiche traiettorie. Avendo scelto un design della Sfera con ingresso ed uscita simmetrici, è stato necessario introdurre un deflettore per evitare che la telecamera/sensore fosse esposto alla luce diretta del led, vanificando di fatto lo scopo della Sfera Integrante. In questo modo si è permesso alla luce di effettuare il primo rimbalzo e convogliarla quindi sulle pareti della Sfera. Il deflettore deve

essere rivestito dello stesso materiale dell'interno della Sfera. Inoltre deve essere di una dimensione tale da permettere alla telecamera di non inquadrare i bordi della sfera oppure gli illuminatori, ma solo il deflettore stesso. Maggiore è il diametro del deflettore, più luce viene parzializzata e quindi una minore quantità di fotoni attraverserà la porta di uscita. Il deflettore, è stato posto ad un'altezza di 45mm più in basso rispetto alla superficie che separa l'emisfero superiore ed inferiore della Sfera. Il diametro del disco deflettore è di circa 100mm.

Successivamente è stata introdotta una nuova generazione di sfere di dimensioni ridotte del 40% rispetto alla precedente e quindi più manegevoli. Tale sfera è caratterizzata da due uscite, una per il collegamento dei sensori d'immagine e l'altra per permettere il fissaggio del fotodiodo. Tale sfera permette una irradianza d'uscita maggiore rispetto alla precedente grazie all'ottimizzazione del processo produttivo. La lunghezza del tubo ed il diametro del foro d'uscita della porta dedicata al fotodiodo è stata dimensionata in modo che permettesse di uguagliare la luminosità sulle due uscite.

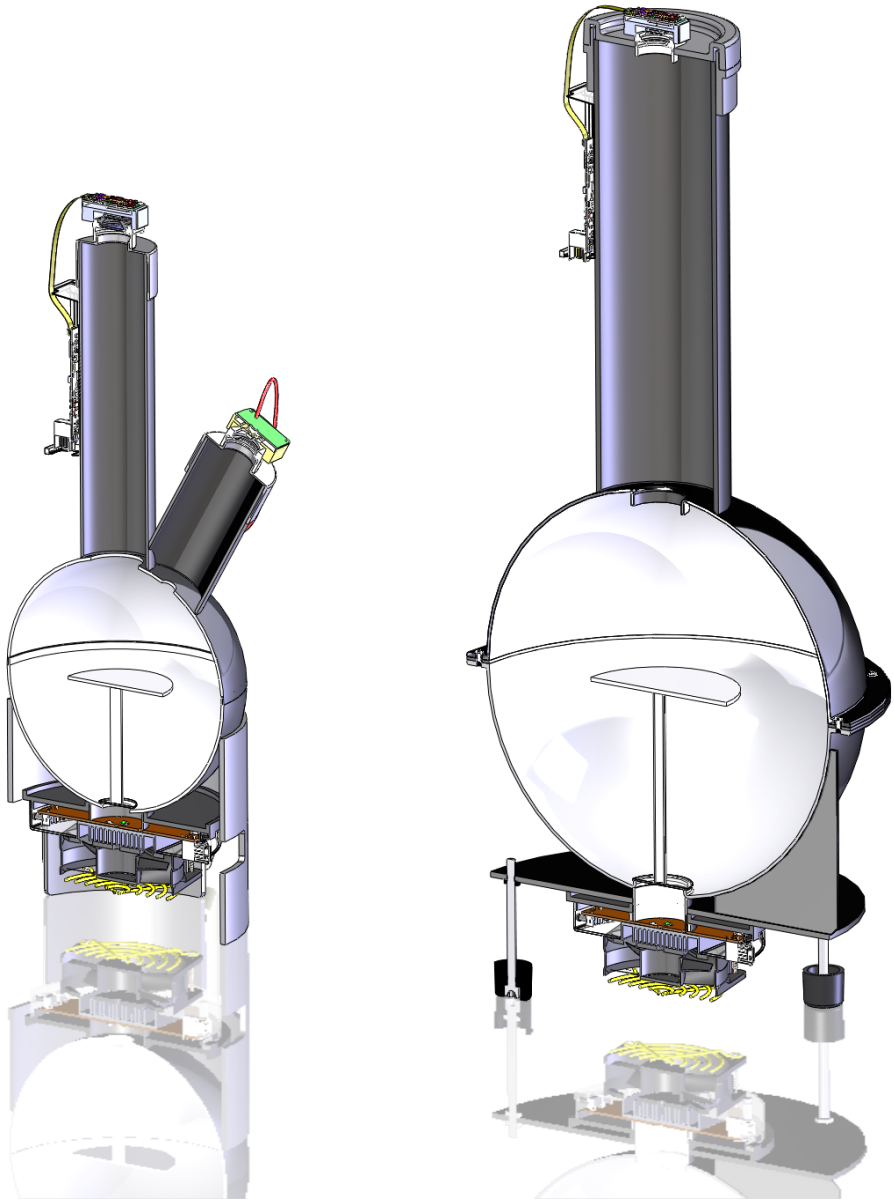


Figura C.1: Vista in sezione delle due generazioni di sfere

Appendice D

Telecamera utilizzata nei test EMVA

La telecamera che è stata usata come base dei test è un modello prodotto da Alkeria, la Lira 424BW. Questa telecamera in bianco e nero monta un sensore CCD Sony ICX424 e un convertitore a 14bit, nella Tabella D.1 compaiono le principali caratteristiche. La Lira 424 si presenta in due parti distinte unite da un cavo di collegamento piatto: la prima parte contiene il sensore, l'ADC e il supporto per il montaggio della lente; sul PCB della seconda parte sono presenti le interfacce di collegamento, tra cui la firewire.

Modello	Lira 424BW
Tipo	Bianco e nero
Sensore	Sony ICX 424
Dimensione sensore	1/3"
Dimensione pixel	7.4µm x 7.4µm
Dimensione immagine	652 x 490 pixel
Interfaccia	IEEE 1394a 400Mbit/s
ADC	14bit per pixel
Modalità video	Mono8, Mono16
FPS massimi	70 @ Mono8 e massima risoluzione
Buffer interno	64MB
Controlli	Luminosità, contrasto, esposizione, guadagno,
Adattatore lenti	C-Mount
Alimentazione	8-35V

Tabella D.1: Caratteristiche Lira 424BW

Appendice E

Libreria per la comunicazione con gli illuminatori

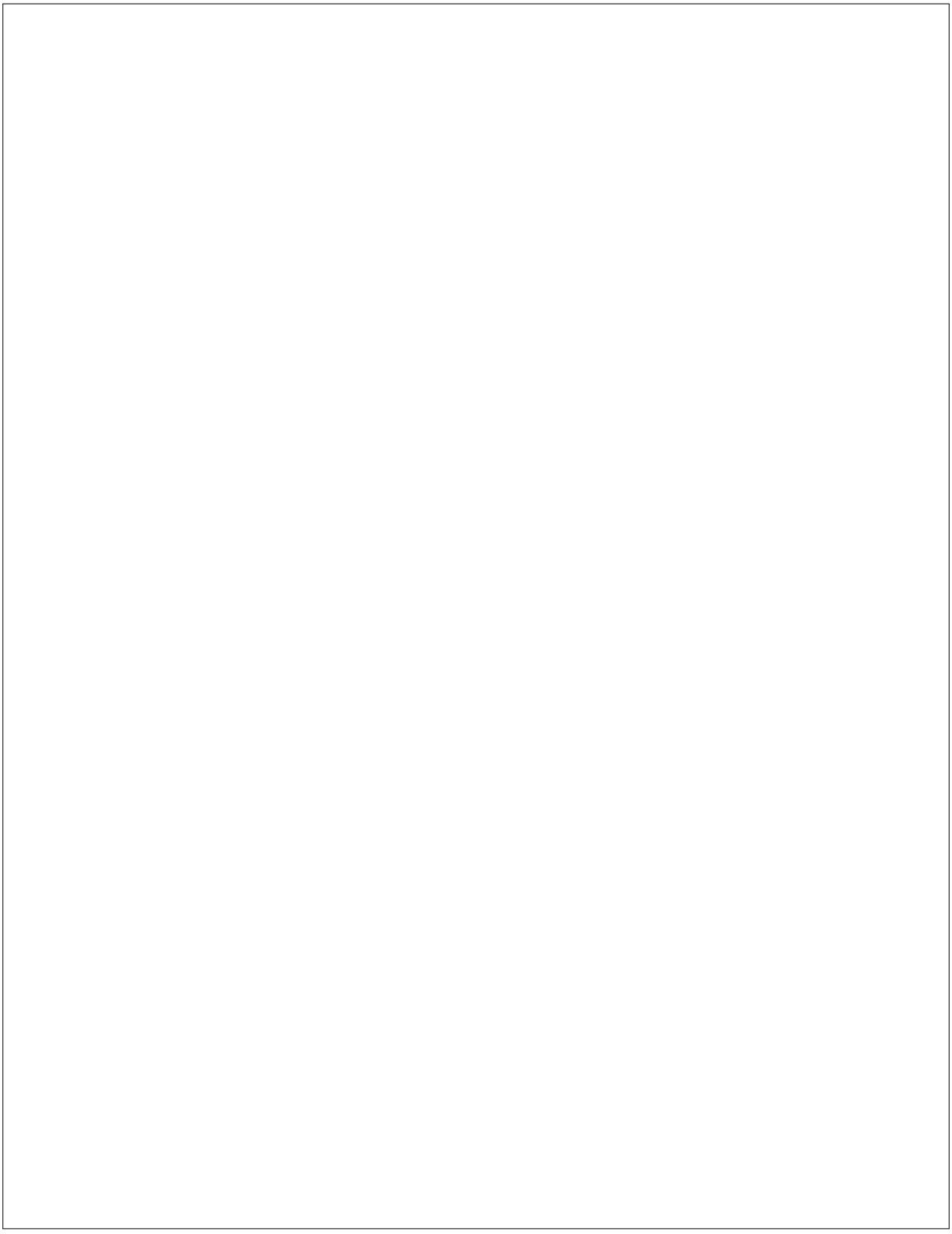
Per rendere possibile la comunicazione con i protocolli degli illuminatori è stata realizzata una libreria unificata in C#. Tale libreria permette di comunicare con la l'illuminatore lineare e l'illuminatore PWM. Le funzioni implementate permettono ad un utilizzatore finale del sistema A.C.T. di poter pilotare le schede sfruttando le loro potenzialità. Sarà sufficiente creare un'interfaccia nella quale collegare la dll fornita e seguire la documentazione delle funzioni implementate. Di seguito la documentazione riguardo la libreria.

SerCom

2

Generated by Doxygen 1.7.6.1

Wed Dec 14 2011 03:47:02



Contents

Chapter 1

Class Documentation

1.1 SerialCommunication.SerCom Class Reference

Classe per la comunicazione seriale con illuminatore analogico e PWM.

Public Types

- enum ol_param { LEDAON = 0x08, LEDBON = 0X09, LEDCON = 0X0A, STATE_OFF = 0X10, LEDABCON = 0x0F }

  Parametri per il pilotaggio dei led in anello aperto.
- enum fb_param { LEDAFB1 = 0x28, LEDBFB1 = 0X29, LEDCFB1 = 0X2A, LEDAFBD = 0X48, LEDBFBD = 0X49, LEDCFBD = 0X4A }

  Parametri per il pilotaggio dei led in anello chiuso.

Public Member Functions

- bool SetPortNameValues (object obj)

  Ritorna la lista delle com collegate.
- void SetParityValues (object obj)

  Ritorna i metodi di parità selezionabili.
- void SetStopBitValues (object obj)

  Ritorna i valori per la dimensione della singola unita del pacchetto.
- SerCom ()

  Costruttore di default.
- SerCom (string name)

  Costruttore con valori di default: baud = 115200, parity = none, stopbits = 1, databits = "8".
- SerCom (string name, string baud)

  Costruttore con valori di default: parity = none, stopbits = 1, databits = "8".
- SerCom (string name, string baud, string par, string sBits, string dBits)

Costruttore che permette l'inserimento di ogni parametro.

- bool OpenPort ()

  Apertura della porta seriale, precedentemente inizializzata dal costruttore.
- void ClosePort ()

  Chiusa della porta seriale.
- byte communicate (uint msg, ol_param setting)

  Funzione per l'invio di pacchetti al PIC Accendere un led al valore richiesto (STATE_ON),.
- byte communicate (ref double data)

  Lettura della temperatura della scheda analogica.
- byte adc_init_single ()

  Richiesta di autocalibrazione adc per modalità single-shot.
- byte adc_init_cont ()

  Richiesta di autocalibrazione per modalità continua.
- byte adc_readData_S (ref double data)

  Lettura dell'adc in modalita single-shot.
- int adc_readData_C (ref double data)

  Lettura dell'adc in modalita continua.
- byte adc_FeedBack_Start (double msg, fb_param led, ref double data)

  Avvia modalità feedback Richiede che sia stata chiamata la adc_init_cont() precedentemente.
- byte adc_FeedBack_Resume ()

  Riavvia la modalità feedback.
- byte adc_FeedBack_Stop (ref double data)

  Termina la modalità feedback.
- int pwm_communicate (uint msg, ol_param led)

  Invia un pwm su 10 bit alla CPLD.
- int pwm_communicate (ref double pwm_temp)

  Lettura della temperatura della scheda PWM La temperatura è limitata tra 0 e 50°
- void Pause (int t)

  Pausa effettuata tramite timer di sistema TickCount.
- UInt32 TimeGTC ()

  Ritorna il valore del timer di sistema in millisecondi.
- byte Reverse (byte b)

  Inverte un byte.

Public Attributes

- const int MAXLIGHT = 16383

  Valore massimo impostabile sul DAC (16383)
- const int LIGHTOFF = 0

  Valore minimo settabile sul DAC (0)

Properties

- bool isopen `[get]`

  Ritorna 1 se la porta seriale è aperta.
- string PortName `[get, set]`

  Nome della porta seriale.
- string BaudRate `[get, set]`

  Baudrate scelto per la comunicazione.
- string Parity `[get, set]`

  Bit di parità
- string StopBits `[get, set]`

  Durata dello stop bit.
- string DataBits `[get, set]`

  dimensione del singola unita del pacchetto spedito (8 bit)

1.1.1 Detailed Description

Classe per la comunicazione seriale con illuminatore analogico e PWM.

1.1.2 Member Enumeration Documentation

1.1.2.1 enum SerialCommunication::SerCom::fb_param

Parametri per il pilotaggio dei led in anello chiuso.

Enumerator:

LEDAFB1 Il count scelto viene inviato al canale A (rosso) e mantenuto in ciclo chiuso con gain unitario.

LEDBFB1 Il count scelto viene inviato al canale B (verde) e mantenuto in ciclo chiuso con gain unitario.

LEDCFB1 Il count scelto viene inviato al canale C (blu) e mantenuto in ciclo chiuso con gain unitario.

LEDAFBD Il count scelto viene inviato al canale C (blu) e mantenuto in ciclo chiuso con gain dinamico.

LEDBFBD Il count scelto viene inviato al canale C (blu) e mantenuto in ciclo chiuso con gain dinamico.

LEDCFBD Il count scelto viene inviato al canale C (blu) e mantenuto in ciclo chiuso con gain dinamico.

1.1.2.2 enum SerialCommunication::SerCom::ol_param

Parametri per il pilotaggio dei led in anello aperto.

Enumerator:

LEDAON Invia al canale A del DAC (rosso) il count scelto.

LEDBON Invia al canale B del DAC (verde) il count scelto.

LEDCON Invia al canale C del DAC (blu) il count scelto.

STATE_OFF Si richiede lo stato STATE_OFF, spegnimento di tutti i led, disattivazione dell'ADC.

LEDABCON Il count scelto viene spedito ai tre canali del LED.

1.1.3 Constructor & Destructor Documentation

1.1.3.1 SerialCommunication.SerCom.SerCom ( )

Costruttore di default.

1.1.3.2 SerialCommunication.SerCom.SerCom (string *name*)

Costruttore con valori di default: baud = 115200, parity = none, stopbits = 1, databits = "8".

Parameters

name	Inserire il nome della porta com, es. "COM1"

1.1.3.3 SerialCommunication.SerCom.SerCom (string *name,* string *baud*)

Costruttore con valori di default: parity = none, stopbits = 1, databits = "8".

Parameters

name	Inserire il nome della porta com, es. "COM1"
baud	Baud Rate

1.1.3.4 SerialCommunication.SerCom.SerCom (string *name,* string *baud,* string *par,* string *sBits,* string *dBits*)

Costruttore che permette l'inserimento di ogni parametro.

Parameters

name	Enter COM Port Name, es. "COM1"
baud	Baud Rate
par	Parity
sBits	Stop Bits
dBits	Data Bits

1.1.4 Member Function Documentation

1.1.4.1 byte SerialCommunication.SerCom.adc_FeedBack_Resume ( )

Riavvia la modalità feedback.

Returns

0 se la comunicazione è andata a buon fine

1.1.4.2 byte SerialCommunication.SerCom.adc_FeedBack_Start (double *msg*, fb_param *led*, ref double *data*)

Avvia modalità feedback Richiede che sia stata chiamata la adc_init_cont() precedentemente.

Parameters

msg	valore in volt da raggiungere
led	led da accendere in modalità feedback
data	valore in volt raggiunto

Returns

0 se la comunicazione è andata a buon fine

1.1.4.3 byte SerialCommunication.SerCom.adc_FeedBack_Stop (ref double *data*)

Termina la modalità feedback.

Parameters

data	valore double di ritorno in V

Returns

0 se la comunicazione è andata a buon fine

1.1.4.4 byte SerialCommunication.SerCom.adc_init_cont ( )

Richiesta di autocalibrazione per modalità continua.

Returns

0 se la comunicazione è andata a buon fine

1.1.4.5 byte SerialCommunication.SerCom.adc_init_single ( )

Richiesta di autocalibrazione adc per modalità single-shot.

Returns

0 se la comunicazione è andata a buon fine

1.1.4.6 int SerialCommunication.SerCom.adc_readData_C (ref double *data*)

Lettura dell'adc in modalita continua.

Parameters

data	valore double di ritorno in V

Returns

0 se la comunicazione è andata a buon fine

1.1.4.7 byte SerialCommunication.SerCom.adc_readData_S (ref double *data*)

Lettura dell'adc in modalita single-shot.

Parameters

data	valore double di ritorno in V

Returns

0 se la comunicazione è andata a buon fine

1.1.4.8 void SerialCommunication.SerCom.ClosePort ( )

Chiusa della porta seriale.

1.1.4.9 byte SerialCommunication.SerCom.communicate (uint *msg,* ol_param *setting*)

Funzione per l'invio di pacchetti al PIC Accendere un led al valore richiesto (STATE_ON),.

inviare a tutti i led lo stesso valore (LEDABCON), spegnere tutti i led e l'adc (STATE_OFF).

Parameters

msg	Count a 14 bit da inviare al DAC
setting	Comando

Returns

0 se la comunicazione è andata a buon fine

1.1.4.10 byte SerialCommunication.SerCom.communicate (ref double *data*)

Lettura della temperatura della scheda analogica.

Parameters

data	Riferimento a double dove scrivere la temperatura

Returns

0 se la comunicazione è andata a buon fine

1.1.4.11 bool SerialCommunication.SerCom.OpenPort ( )

Apertura della porta seriale, precedentemente inizializzata dal costruttore.

Returns

1.1.4.12 void SerialCommunication.SerCom.Pause (int *t*)

Pausa effettuata tramite timer di sistema TickCount.

Parameters

t	Time in Milliseconds

1.1.4.13 int SerialCommunication.SerCom.pwm_communicate (uint *msg*, ol_param *led*)

Invia un pwm su 10 bit alla CPLD.

Parameters

msg	Unsigned int, valore massimo 1024 (10 bit)
led	Led da accendere

1.1.4.14 int SerialCommunication.SerCom.pwm_communicate (ref double *pwm_temp*)

Lettura della temperatura della scheda PWM La temperatura è limitata tra 0 e 50°

Parameters

pwm_temp	Riferimento a double dove scrivere la temperatura

Returns

0 se la comunicazione è andata a buon fine

1.1.4.15 byte SerialCommunication.SerCom.Reverse (byte *b*)

Inverte un byte.

Parameters

b	byte da invertire

Returns

1.1.4.16 void SerialCommunication.SerCom.SetParityValues (object *obj*)

Ritorna i metodi di parità selezionabili.

Parameters

obj	ComboBox

1.1.4.17 bool SerialCommunication.SerCom.SetPortNameValues (object *obj*)

Ritorna la lista delle com collegate.

Parameters

obj	ComboBox

Returns

1.1.4.18 void SerialCommunication.SerCom.SetStopBitValues (object *obj*)

Ritorna i valori per la dimensione della singola unita del pacchetto.

Parameters

obj	ComboBox

1.1.4.19 UInt32 SerialCommunication.SerCom.TimeGTC ( )

Ritorna il valore del timer di sistema in millisecondi.

Returns

1.1.5 Member Data Documentation

1.1.5.1 const int SerialCommunication.SerCom.LIGHTOFF = 0

Valore minimo settabile sul DAC (0)

1.1.5.2 const int SerialCommunication.SerCom.MAXLIGHT = 16383

Valore massimo impostabile sul DAC (16383)

1.1.6 Property Documentation

1.1.6.1 string SerialCommunication.SerCom.BaudRate [get, set]

Baudrate scelto per la comunicazione.

1.1.6.2 string SerialCommunication.SerCom.DataBits [get, set]

dimensione del singola unita del pacchetto spedito (8 bit)

1.1.6.3 bool SerialCommunication.SerCom.isopen [get]

Ritorna 1 se la porta seriale è aperta.

1.1.6.4 string SerialCommunication.SerCom.Parity [get, set]

Bit di parità

1.1.6.5 string SerialCommunication.SerCom.PortName [get, set]

Nome della porta seriale.

1.1.6.6 string SerialCommunication.SerCom.StopBits [get, set]

Durata dello stop bit.

The documentation for this class was generated from the following file:

- SerCom.cs

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Lanzetta per il continuo supporto ricevuto durante la tesi ed i preziosi insegnamenti ricevuti.
Inoltre ringrazio l'Ing. Festa e l'Ing. Sottile per le numerose ore dedicate alla mia tesi insieme agli altri colleghi dell'azienda Alkeria S.r.l.
Intendo poi ringraziare i colleghi nonché amici Enrico e Michele per i continui consigli ricevuti.

Bibliografia

[1] Ducharme, A.; Daniels, A.; Grann, E.; Boreman, G.; , "Design of an integrating sphere as a uniform illumination source ," Education, IEEE Transactions on , vol.40, no.2, pp.131-134, May 1997 doi: 10.1109/13.572326

[2] Poikonen, Tuomas; Manninen, Pasi; Karha, Petri; Ikonen, Erkki; , "Multifunctional integrating sphere setup for luminous flux measurements of light emitting diodes," Review of Scientific Instruments , vol.81, no.2, pp.023102-023102-7, Feb 2010 doi: 10.1063/1.3285263

[3] Jiaming Lin, Xin Zhang, Zhe Zhang, Dingguo Sha, Illumination technique based on integrating sphere with an unequal radius, hyper-uniform luminance and large-field angle, Optik - International Journal for Light and Electron Optics, Volume 119, Issue 9, 10 July 2008, Pages 446-450, ISSN 0030-4026, 10.1016/j.ijleo.2007.02.006.

[4] Labsphere, A guide to integrating sphere theory and applications.

[5] Gastasini Enrico, Sviluppo e realizzazione di un'attrezzatura sperimentale per la caratterizzazione di un sensore d'immagine, Università di Pisa, Facolta di Ingegneria, Tesi di Laurea Specialistica in Ingegneria dell'Automazione.

[6] Weifeng Feng; Yongzhi He; Shi, F.G.; , "Investigation of LED Light Output Performance Characteristics Under Different Alternating Current Regulation Modes," Selected Topics in Quantum Electronics, IEEE Journal of , vol.17, no.3, pp.720-723, May-June 2011 doi: 10.1109/JSTQE.2011.2105859

[7] In-Hwan Oh; , "An analysis of current accuracies in peak and hysteretic current controlled power LED drivers," Applied Power Electronics Conference and Exposition, 2008. APEC 2008. Twenty-Third Annual IEEE , vol., no., pp.572-577, 24-28 Feb. 2008 doi: 10.1109/APEC.2008.4522778

[8] Tse-Ju Liao; Chern-Lin Chen; , "Robust LED backlight driver with low output voltage drop and hig output current accuracy," Sustainable Energy Technologies, 2008. ICSET 2008. IEEE International Conference on , vol., no., pp.63-66, 24-27 Nov. 2008 doi: 10.1109/ICSET.2008.4746973

[9] EMVA, Standard for Characterization of Image Sensors and Cameras, Release 3.0, http://www.emva.org.

[10] Xiaohui Qu; Siu-Chung Wong; Tse, C.K.; , "Temperature Measurement Technique for Stabilizing the Light Output of RGB LED Lamps," Instrumentation and Measurement, IEEE Transactions on , vol.59, no.3, pp.661-670, March 2010 doi: 10.1109/TIM.2009.2025983

[11] Elger, G.; Lauterbach, R.; Dankwart, K.; Zilkens, C.; , "Inline thermal transient testing of high power LED modules for solder joint quality control," Electronic Components and Technology Conference (ECTC), 2011 IEEE 61st , vol., no., pp.1649-1656, May 31 2011-June 3 2011 doi: 10.1109/ECTC.2011.5898733