Politecnico di Torino

Politecnico di Torino

Ingegneria Elettronica
A.A. 2020/2021
Sessione di Laurea Luglio 2021

Progettazione di una scheda per il Monitoring and Control di una Deep Space Antenna dell'Agenzia Spaziale Europea

Relatore

Prof. Luciano LAVAGNO

Candidato

Cristian CUNSOLO

Sommario

In questa Tesi ci si è occupati della modalità di sviluppo di una scheda, chiamata FastPro, che si occupa del controllo relativo al funzionamento di un trasmettitore (Ground Station) di proprietà Agenzia Spaziale Europea (ESA®): questa è una doughter-board, in grado di supportare con efficacia il sistema precedente di controllo.

Infatti con l'utilizzo di 6 Analog to Digital Converter (ADC) comandati da una Field Programmable Gate Array (FPGA), vengono registrati dei segnali in grado di riportare i valori di tensione o di corrente necessari al funzionamento dei vari componenti e i campioni vengono salvati simultaneamente in una Synchronous Dynamic Random Access Memory (SDRAM), insieme ad altri segnali utilizzati per riportare lo stato del sistema, per poi essere inviati ad un computer in caso di un evento catastrofico che causa un guasto.

Questa funzionalità, motivo per cui è nata la necessità di progettare questo nuovo sistema, risulta utile al fine di poter ricostruire, in modo assolutamente affidabile, gli attimi precedenti all'evento bloccante ed è di aiuto nella successiva fase di diagnostica.

Inoltre la scheda va anche a sostituire, in molti casi, il precedente sistema di protezione da funzionamenti anomali: i campioni vengono messi a confronto utilizzando la FPGA con delle soglie programmabili e, nel caso di superamento di tali limiti, essa si occupa di inviare immediatamente eventuali segnali di inibizione ai componenti del trasmettitore che si trovano in uno stato di funzionamento anomalo per evitare danni irreparabili ai sotto-sistemi collegati.

Ringraziamenti

Mi è doveroso dedicare questo spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla relazione dello stesso.

In primis un ringraziamento speciale al mio relatore Prof. Luciano Lavagno, per i suoi indispensabili consigli per il percorso della stesura dell'elaborato.

Ringrazio tutto lo staff dell'azienda Microsis s.r.l.®: Roberto Antoniucci, Maurizio Bevilacqua ed Andrea Morviducci per avermi concesso la possibilità di prendere parte a questo progetto; Fabrizio Maccioni, Pietro Moldavio e Iulian Raibulet per il loro essenziale supporto durante la fase di progettazione e definizione delle specifiche; Vincenzo Padula e Antonio Russo per la loro pazienza durante il processo di saldatura della scheda; inoltre vorrei ringraziare il resto dello staff nelle persone di Claudio Serrano, Andrea Tamagnini e Paolo Valerio per l'accoglienza ricevuta per i sei mesi di stretta collaborazione.

Infine vorrei ringraziare di cuore i miei genitori e mia sorella: grazie per avermi sempre sostenuto e per avermi permesso di portare a termine gli studi universitari.

Indice

Elenco delle tabelle

Elenco delle figure

Acronimi

WCC
Wired Controlled Card

ADC
Analog to Digital Converter

SDRAM
Syncronous Dynamic Random Access Memory

FPGA
Field Programmable Gate Array

DSA
Deep Space Antenna

ESOC
European Space Operation Centre

TT&C
Telemetry and Telecommand

M&C
Monitoring and Control

LEOP
Launch and Early Orbit Phase

HPA
High Power Amplifier

KPA

Klystron Power Amplifier

CS

Cooling System

UART

Universal Asynchronous Receiver-Transmitter

USB

Universal Serial Bus

I2C

Inter Integrated Circuit

SPI

Serial Peripheral Interface

I/O

Input Output

D2PAK

Double Decawatt Package

LDO

Low Dorop Out

PLL

Phase Locked Loop

EMI

Electromagnetic Interference

DDR

Double Data Rate

SDR

Single Data Rate

SDA

Serial Data Line

SCL

Serial Clock Line

EEPROM

Electrically Erasable Programmable Read Only Memory

SRAM

Static Random Access Memory

JTAG

Join Test Action Group

IP

Intellectual Property

SCLK

Serial Clock

MISO

Master In Slave Out

Avalon MM

Avalon Memory Mapped

CAS

Column Address Strobe

FIFO

First In First Out

MTBF

Mean Time Before Failure

FFT

Fast Fourier Trasform

Capitolo 1

Introduzione

In questo capitolo di introduzione viene trattato il tema dell'utilizzo, in ambito di ricerca scientifica astronomica, dei trasmettitori Deep Space Antenna (DSA) a banda X e banda S: si esaminerà come questi sistemi siano attualmente utilizzati per la telemetria e il controllo di numerose missioni condotte da ESA®, le informazioni sono state attinte da [1]

Successivamente verrà illustrato lo stato attuale del trasmettitore DSA 1 situato presso la cittadina di New Norcia, Australia: verranno definite le caratteristiche generali di performance e di funzionamento del trasmettitore e si indicheranno quali missioni vengono attualmente supportate, questa parte è stata redatta grazie a [2]

Successivamente si entrerà nel dettaglio del funzionamento della DSA 1: verranno analizzati i componenti e si spiegherà come questi vengano mantenuti entro un regime di funzionamento secondo le specifiche dei vari sottosistemi ([3]) , infine verrà trattato come si è voluto migliorare il sistema di controllo per una seconda generazione DSA.

A seguito di questa prima fase di trattazione si introdurrà il progetto della nuova DSA voluta da ESA®che verrà installata sempre nel sito di New Norcia, per la quale è stata progettata la scheda oggetto di questa Tesi: atta a fornire su di essa informazioni di carattere generale, in modo da rendere più chiara e comprensibile la lettura dei capitoli successivi, per quest'ultima parte di trattazione si è fatto affidamento alle informazioni contenute in [4].

1.1 Utilizzo delle Deep Space Antenna per le missioni nello spazio profondo

Durante le missioni nello spazio profondo, vi è la necessità di dover scambiare una grande quantità di informazioni: di conseguenza ESA®ha sviluppato un sistema

di antenne per le funzioni di Telemetry and Telecommand (TT&C) controllate dal European Space Operation Centre (ESOC) situato in Germania.

Come si può vedere dalla Figura 1.1 una di queste antenne è situata nella cittadina australiana di New Norcia ed è uno dei più grandi trasmettitori attualmente presenti nel mondo, utilizzati nelle comunicazioni relative alla TT&C.

La DSA risulta composta da un'antenna parabolica dal diametro di 35 m a cui viene fornito un segnale attraverso un sistema con una guida d'onda a fascio, il piatto viene poggiato su un sistema di movimento che è in grado di orientare l'antenna in tutte le possibili direzioni: l'altezza complessiva della struttura è di circa 40 m e il suo peso è, approssimativamente, 630 t.

Questo sistema di trasmissione vanta degli amplificatori a basso rumore per la banda X e la banda S, insieme ad altri amplificatori di potenza a 20 kW. Questo è servito e continua a servire numerose missioni nello spazio profondo tra cui si possono citare quelle più significative: Rosetta, Mars Express e Venus Express.

Infatti la comunicazione con questi veicoli spaziali necessita di requisiti molto rigorosi per le stazioni di trasmissione di terra, in quanto le limitazioni di peso da portare in orbita e la necessità di consumare poca potenza per gli spacecraft rendono le prestazioni dei loro sistemi di comunicazione non molto elevate soprattutto in relazione alle grandi distanze a cui avvengono le trasmissioni: ad esempio, le comunicazioni con la sonda Rosetta avvenivano su un tragitto lungo 900×10^6 km, ovvero 7 volte la distanza tra la Terra e il Sole.

Di conseguenza, questi sistemi di comunicazione necessitano di antenne con un diametro molto grande e un'ampiezza del cono di trasmissione molto piccola: quest'ultima specifica causa e porta all'ulteriore necessità di avere un sistema di puntamento estremamente preciso.

Di notevole importanza è anche il garantire un sistema di movimento del piatto dell'antenna molto agevole, ma allo stesso tempo capace di resistere alle condizioni atmosferiche del luogo di installazione.

Questo risulta di grande importanza, in quanto, ai disturbi elettrici di trasmissione dovuti a variazioni delle condizioni meteorologiche, si aggiungono eventuali disturbi meccanici, causati dalla forza esercitata dal vento sull'antenna parabolica che, a causa delle sue notevoli dimensioni, crea un imprevedibile effetto vela, rischiando di avere oscillazioni con un'ampiezza tale da far perdere il puntamento dello spacecraft, qualora non ci siano dei sistemi robusti abbastanza da essere immuni e da opporre resistenza ai disturbi atmosferici di questo tipo: In Figura 1.2 è presente un breve schema dell'architettura delle DSA.

A questo punto della trattazione è fondamentale, per capire l'importanza di questi sistemi, proporre una lista di alcune missioni spaziali che sono state rese possibili grazie al sistema di DSA:

Figura 1.1: DSA 1 New Norcia, Australia. (Courtesy of ESA®)

Rosetta La sonda Rosetta è partita nel 2 Marzo 2004 e dopo essere entrata nell'orbita della cometa Churyumov-Gerasimenko nel 2014, ha rilasciato un lander sulla sua superficie: gli obbiettivi della missione sono stati quelli di studiare l'origine delle comete, la relazione tra il materiale interstellare e quello delle comete e le sue implicazioni sulla nascita del sistema solare

Mars Express Questa è stata, in assoluto, la prima missione su Marte pianificata da ESA®con l'obbiettivo specifico di sondare l'atmosfera, la struttura e la composizione geologica del pianeta rosso alla ricerca di indizi sulla possibilità che Marte ospiti delle riserve idriche sotterranee: da notare che i dati della ricerca riescono ad

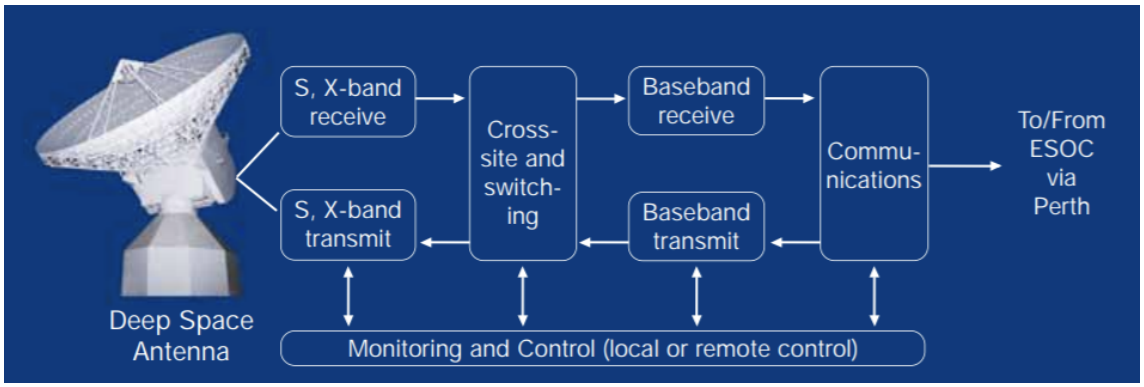


Figura 1.2: Schema Generale DSA. (Courtesy of ESA®)

essere trasmessi ad una distanza di 450×10^6 km.

Venus Express Partita nel Novembre 2005, Venus Express è stato in assoluto il primo spacecraft a tentare di realizzare una ricerca omnicomprensiva dell'atmosfera di Venere: è risultata di notevole utilità per capire quali siano le caratteristiche dell'atmosfera di quel pianeta, come avvengano e quali siano i moti di circolazione dell'atmosfera e come questa interagisca con le radiazioni provenienti dal vento solare.

1.2 La Deep Space Antenna 1 di New Norcia

Il sito dove si trova la DSA 1 è a 10 km da New Norcia e a 150 km da Perth dove avvengono alcune delle operazioni di controllo e monitoraggio locale: le sue strutture tecniche comprendono un sistema di uplink e uno di downlink per le comunicazioni a banda X e banda S, uno di timing, un altro di frequenza, uno successivo di Monitoring and Control (M&C) e ultimo, in enumerazione, un sistema di comunicazione con le stazioni di controllo situate a Perth e ESOC.

La costruzione del sito è terminata nel 2002 e i primi test di puntamento sono stati completati con successo con la missione NASA® Stardust e la stazione ha servito la prima missione ESA®nel Marzo 2003.

Infatti il sistema di questa antenna è composto da una guida d'onda a fascio, da uno specchio che ha la funzionalità di filtro dicroico, da degli amplificatori a basso rumore a radiofrequenza per le bande X e S i quali vengono mantenuti a temperature criogeniche e da due trasmettitori a 20 kW, anche essi funzionanti per le bande X e S.

Pertanto il sito di New Norcia ha supportato e supporta le operazioni di routine necessarie per le missioni dello spazio profondo condotte da ESA®: ad esempio, Mars

Express, Gaia e Rosetta, in quanto è anche in grado di comunicare con i veicoli, durante la fase di Launch and Early Orbit Phase (LEOP).

La struttura meccanica, in grado di permettere l'orientamento dell'antenna, può muoversi con una velocità pari a 1 grado al secondo su entrambi gli assi Azimut e altezza: il Servo Control System garantisce una delle migliori precisioni di puntamento, in quanto tiene anche conto delle condizioni atmosferiche del luogo.

L'antenna di New Norcia è riconosciuta come una delle più grandi al mondo per i processi di TT&C e si è inoltre dimostrata essenziale per le comunicazioni per cui servono altissime prestazioni come quelle con spacecraft che avvengono a milioni di chilometri di distanza.

Per avvalorare questa affermazione viene portato ad esempio come la DSA 1 si è rivelata di vitale importanza nelle missioni Rosetta e Mars Express: per quanto riguarda la sonda Rosetta è stato possibile ricevere, in misura assolutamente affidabile i dati dalla navicella, anche a milioni di chilometri di distanza.

È necessario puntualizzare che la posizione di questo trasmettitore è stata accuratamente selezionata in rapporto alle condizioni atmosferiche favorevoli, come la poca attenuazione delle comunicazioni dovute alla pioggia e il vento non troppo veloce, le poche interferenze sulle frequenze coinvolte nelle bande X e S e infine per ottimizzare i costi di costruzione e mantenimento.

Questo sistema è stato anche dotato delle migliori tecnologie disponibili al tempo della costruzione, in relazione al tracciamento delle navicelle e a quello geodetico che invia i dati a ESOC. Inoltre la sede di New Norcia, in quanto ospita degli spazi per gli studiosi in grado di analizzare sul posto i dati ricevuti dalle sonde e per condurre esperimenti sulle onde radio, risulta utile per i dati che vengono analizzati.

La DSA 1 situata a New Norcia è composta da un sistema di trasmissione di 20 kW alle frequenze di $7145-7235$ MHz, basato su un klystron High Power Amplifier (HPA) e raffreddato a liquido, sviluppato nella sede di Roma di Rheinmetall s.p.a.®Lo schema generale del HPA è schematizzato in Figura 1.3.

Ogni Klystron Power Amplfier (KPA) è controllato internamente dal sistema M&C che gestisce le operazioni dell'amplificatore e permette il funzionamento anche in presenza di condizioni anomale.

Inoltre questo sistema è costantemente in grado di registrare le condizioni di funzionamento e poi di salvarle in un file di log utile alla manutenzione, in quanto genera e mantiene un'interfaccia di comunicazione tra il klystron e il sistema M&C remoto che opera attraverso un computer centrale, sotto il sistema operativo Linux: uno screenshot del pannello di M&C è riportato in figura 1.4, mentre le specifiche tecniche sono elencate nella tabella 1.1.

Il KPA è il principale sottosistema del HPA, l'amplificatore di potenza è basato su un amplificatore a klystron CPI-VA876J in grado di operare a 20 kW, il klystron e il rispettivo alimentatore sono raffreddati a liquido al fine di mantenere le temperature entro i limiti prefissati di specifica.

Amplifiers Parameter	Value
Output Power (kW)	20
Instantaneous Bandwidth (MHz)	7145-7235
ALC Output Power (dB)	±0.25
Gain Variation in Band (dB pk-pk)	1
G.D Variation in Band (ns pk-pk)	12
In Band Spurious (dBc)	<-60
IM3 Products of Saturation (dBc)	12
Allan Deviation (at 1000 sec)	$5 \cdot 10^{-17}$

Protections
Output WG Arc
Klystron parameters
Inhibit and Interlock
Coolant temp. and flow

Tabella 1.1: Tabella delle specifiche di HPA (Courtesy of Rheinmetall s.p.a.®)

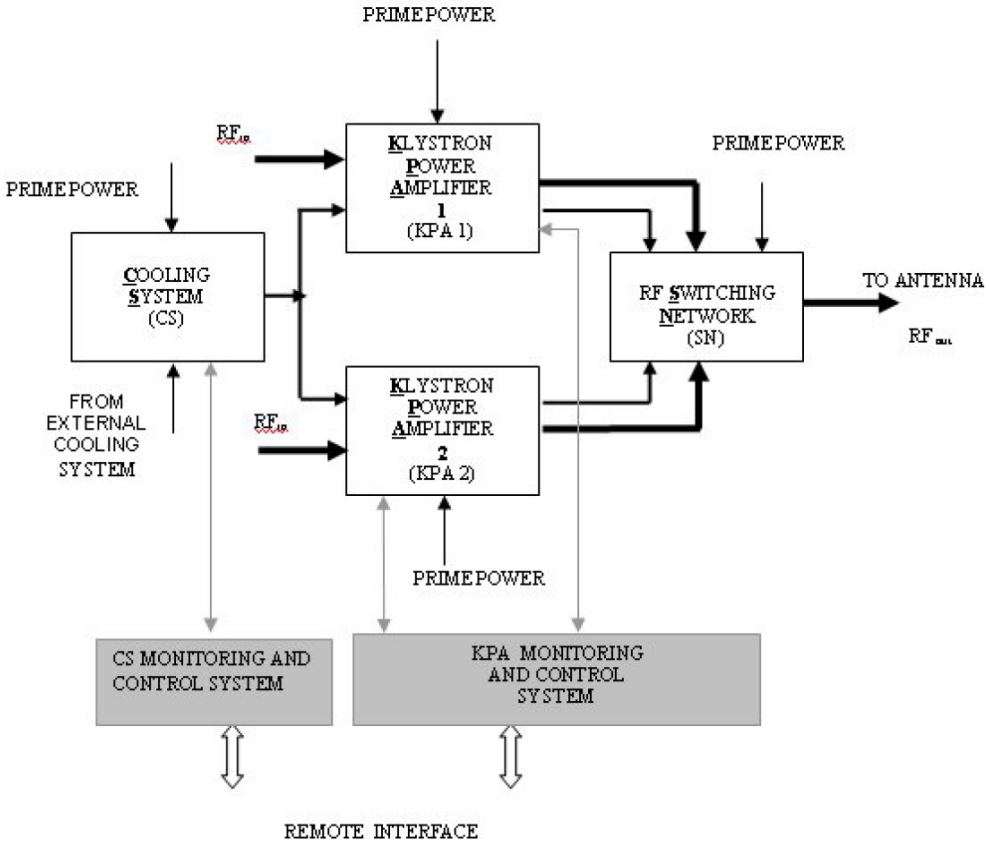


Figura 1.3: Schema a blocchi HPA (Courtesy of Rheinmetall s.p.a.®)

Questo sistema di raffreddamento interno è alimentato dal sistema principale di raffreddamento chiamato Cooling System (CS) che esercita un controllo termico ad alta precisione (0,1 °C) : mantiene nei valori ottimali la temperatura del body e della

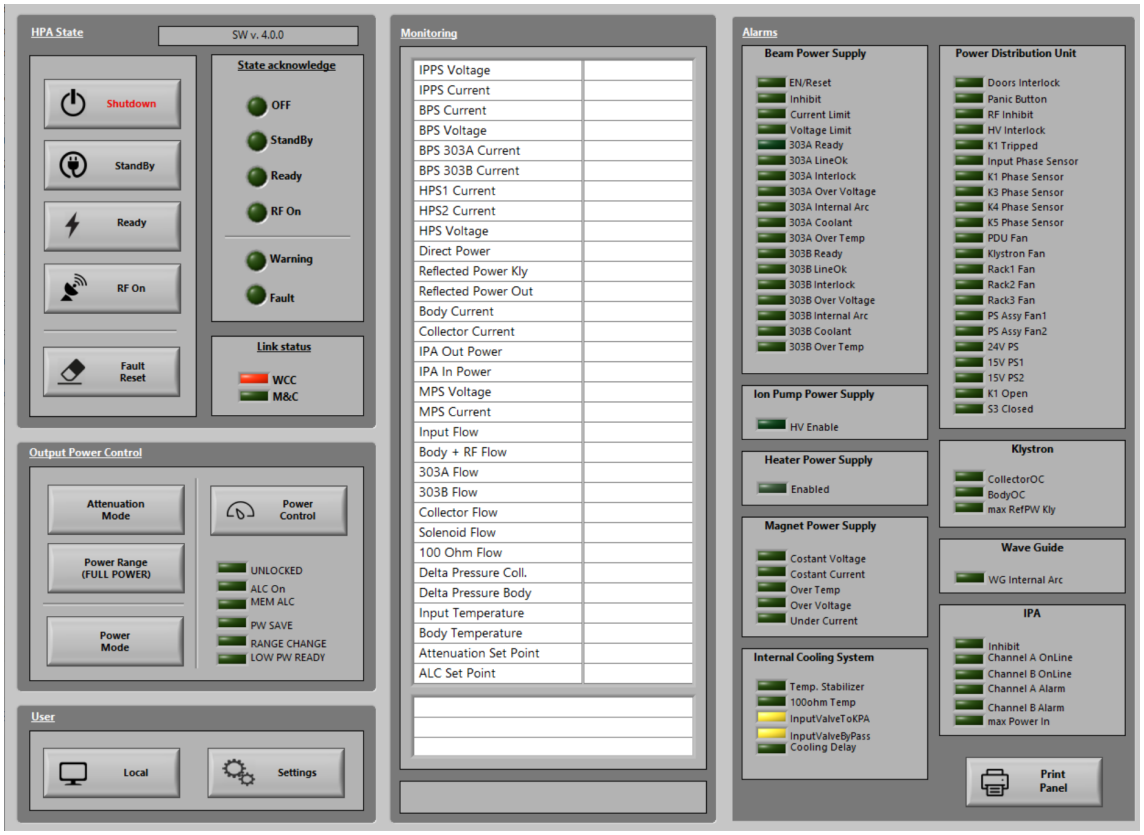


Figura 1.4: Screen Pannello M&C (Courtesy of Micrōsis s.r.l.®)

guida d'onda in uscita, in quanto è stato realizzato al fine di rispettare gli stringenti requisiti, relativi alla stabilità di fase.

Il CS fornisce un liquido di raffreddamento demineralizzato (acqua distillata o una miscela di acqua e glicolo) alla temperatura e pressione adeguate per raffreddare il klystron e dà particolare attenzione alla purità della miscela del liquido di raffreddamento per evitare fenomeni di corrosione all'interno del KPA e scambia calore con il sistema di raffreddamento esterno che opera, invece, con un liquido non demineralizzato.

Ogni sistema di raffreddamento è collegato ai due KPA e un sistema di ridondanza è stato creato con l'aggiunta di una terza pompa che è in grado di fornire pressione a uno dei due circuiti di raffreddamento in caso di guasto ad una delle principali.

Il CS è installato all'esterno del sistema di trasmissione ed ha attorno un sistema di protezione in grado di garantire lo standard IP55 contro la pioggia, getti d'acqua e polveri: come avviene per il KPA, il CS è controllato dallo stesso sistema di M&C interno e viene gestito attraverso la stessa interfaccia hardware e software che opera sotto il sistema operativo Linux.

Infatti il sistema M&C è anche in grado di registrare le operazioni di funzionamento e creare un file di log utile per le operazioni di diagnostica che vengono effettuate al centro di controllo ESOC in Germania.

L'utilizzo del sistema operativo Linux, insieme all'implmementazione di tutti i protocolli sviluppati da ESA®per la gestione e il controllo della stazione, è indubbiamente una delle più importanti caratteristiche di tutto il sistema: l'utilizzo di Linux, in opposizione ai software commerciali concorrenti di proprietà di Microsoft® garantisce una possibilità di manutenzione durevole nel tempo (10-20 anni) e l'uso di protocolli di comunicazioni standard rende questa stazione del tutto compatibile con le precedenti.

1.3 La nuova DSA di New Norcia

Il 29 Aprile 2021 ESA®e l'Agenzia Spaziale Australiana hanno annunciato la costruzione di una seconda DSA nel sito della DSA 1 a New Norcia, in Australia[4], e questo nuovo trasmettitore dovrà supportare la DSA già presente nel sito, con l'aggiunta di nuove funzionalità, quali il supporto per frequenze di trasmissione maggiori.

Un nuovo sistema di raffreddamento è stato completamente riprogettato al fine supportare temperature criogeniche inferiori a $-200\,^{\circ}\mathrm{C}$ in modo da garantire una potenza di ricezione molto maggiore.

Questa necessità di una nuova generazione di trasmettitori è nata dal sempre crescente numero di missioni nello spazio profondo, in quanto il numero di lanci per sonde e navicelle comporta una crescente necessità di inviare comandi e ricevere dati di notevole rilevanza per la ricerca scientifica in misura sempre più affidabile.

Le DSA comunicano con spacecraft anche a distanze di 1.44×10^9 km dalla Terra e ci sono progetti per spingersi a distanze ancora superiori in futuro: è importante riuscire a poter comunicare con più di una navicella per volta utilizzando una sola antenna se si trovano nella stessa direzione come ad esempio intorno al pianeta Marte.

Inoltre la costruzione di una seconda antenna permetterà le trasmissioni in contemporanea anche alle navicelle che si trovano in punti diversi dello spazio, quali ad esempio i pianeti Marte e Mercurio.

Per supportare questo aumento prestazionale, il sistema di M&C è stato anch'esso rivisitato e aggiornato: la Wired Control Card (WCC) che nella vecchia generazione di DSA era l'unica responsabile per il controllo del funzionamento del KPA è stata aggiornata per supportare una nuova scheda FastPro che di occupa sia di campionare ad alta frequenza i segnali, sia di salvarli in una memoria SDRAM e sia di inviare i file di log al computer in modo da poter ricostruire per punti le forme d'onda che hanno causato un malfunzionamento.

In questa è presente una FPGA che controlla l'intero sistema: la FastPro si occupa di campionare 6 segnali analogici ad una frequenza di 5 MHz; questa frequenza di campionamento è stata scelta in modo tale da garantire un'agevole ricostruzione per punti del segnale analogico in fase di diagnostica.

Questi segnali, che sono analogici, ricalcano le tensioni o correnti di funzionamento dei vari componenti del KPA e sono generati così da poterne analizzare il funzionamento in tempo reale.

I segnali vengono confrontati simultaneamente con delle soglie programmabili in modo da garantire un corretto funzionamento del sistema e gestire eventuali guasti: se uno dei 6 segnali ricade al di fuori delle soglie impostate, viene generato, in tempi brevi, un segnale di inhibit che va ad interrompere prontamente il funzionamento del componente che si trova in stato di comportamento anomalo.

Simultaneamente i valori digitali di questi segnali vengono registrati su di una SDRAM insieme ad un byte contenente le informazioni del superamento delle soglie da parte dei componenti soggetti a controllo. In caso di un evento anomalo, ovvero del superamento di una delle soglie prefissate, la scheda fornisce al computer centrale, attraverso un protocollo Universal Asynchronous Receiver-Transmitter (UART) che viene convertito nel protocollo Universal Serial Bus (USB), tutti i dati della memoria al fine di ricostruire, in misura completa e affidabile, gli istanti precedenti all'evento.

Il numero di dati inviati al computer può essere impostato attraverso un registro apposito che viene programmato attraverso la comunicazione con il sistema di controllo principale, ovvero la WCC: questa avviene attraverso l'utilizzo di un bus Inter Integrated Circuit (I2C).

All'interno del sistema digitale sono presenti due diverse frequenze di clock: 160 MHz per la parte di controllo degli ADC attraverso il protocollo Serial Peripheral Interface (SPI) e la comunicazione con la WCC attraverso I2C e una a 100 MHz che, in modo complementare, si occupa del controllo della SDRAM e della comunicazione UART-USB.

È stata operata questa scelta per avere la massima indipendenza tra i due sistemi al fine di poter agevolmente riutilizzare il codice in altre DSA che possono essere diverse da quella che è in costruzione a New Norcia: questo ha comportato la necessità di un sincronizzatore operante tra le due frequenze. Lo schema del sistema digitale è riassunto in figura 1.5.

Il sistema analogico è formato da una parte di signal conditioning che comprende un filtro attivo anti-aliasing a 4 poli: questo componente si occupa anche di operare la conversione da single ended a differenziale dei segnali di ingresso, di abbassare il valore di metà dinamica di uscita a 0 V in modo da poter sfruttare l'intera dinamica di ingresso dell'ADC e di riuscire ad attenuare i segnali al fine di poter essere correttamente campionati.

Il sistema di alimentazione invece è composto da un convertitore di tensione lineare a 5 V di uscita, due altri alimentatori lineari a 2,5 V per l'alimentazione

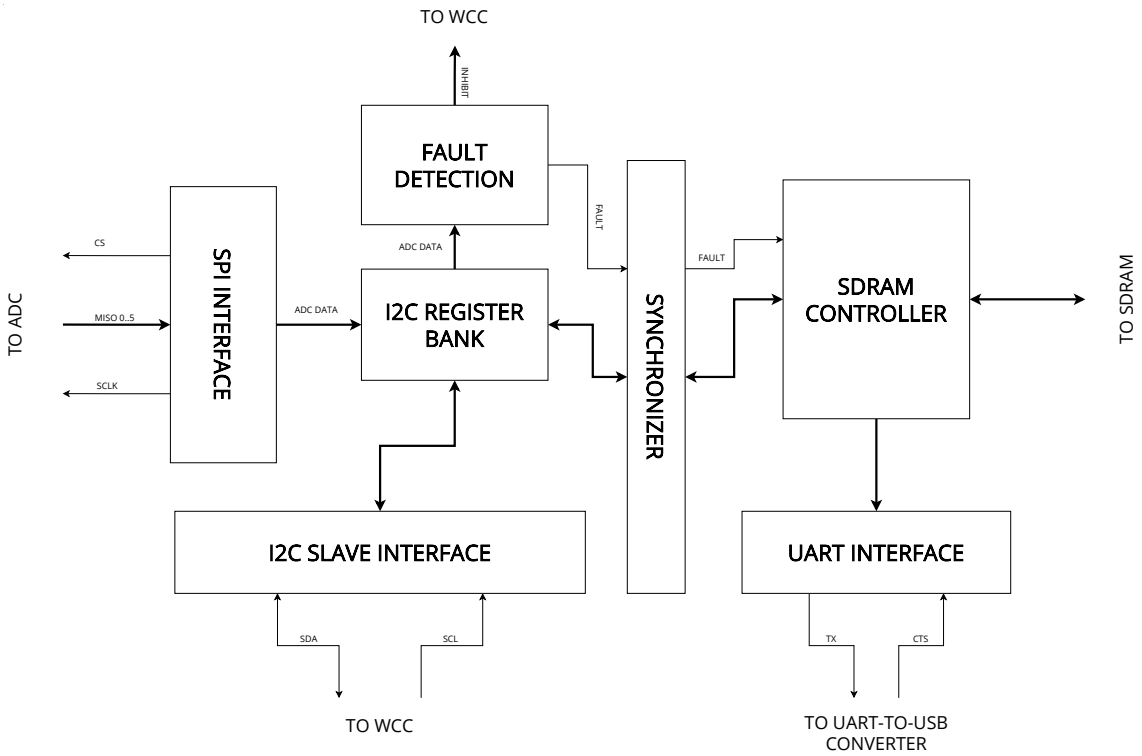


Figura 1.5: Diagramma a blocchi del Sistema Digitale

analogica della FPGA e ADC, un alimentatore switching a 3,3 V per alimentare i banchi di Input Output (I/O) della FPGA e alimentare la SDRAM e un ultimo alimentatore switching con uscita a 1,2 V per alimentare il core della FPGA. Lo schema della scheda è riassunto in figura 1.6.

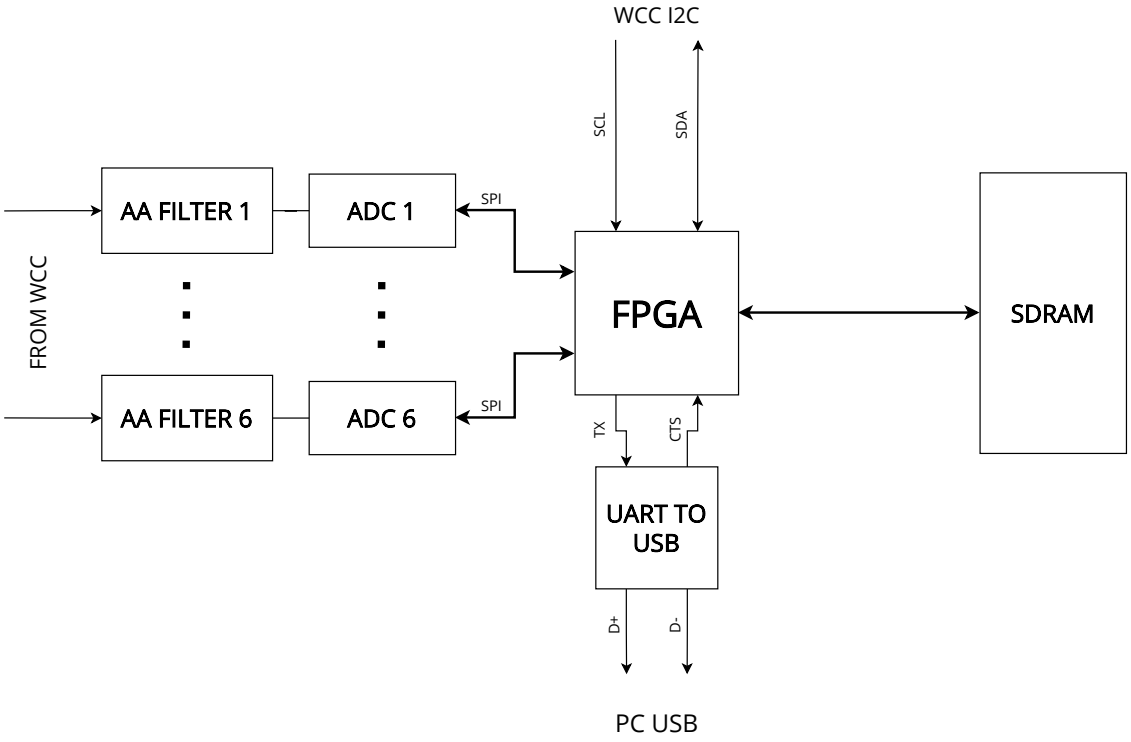


Figura 1.6: Diagramma a blocchi della scheda

Capitolo 2

Realizzazione della scheda

In questo capitolo viene affrontato il tema della realizzazione del circuito stampato, necessario per svolgere tutte le funzioni richieste dal sistema M&C della DSA e alla FastPro: per questo fine la scheda è stata realizzata su 6 layer e le dimensioni sono di 115 mm × 112 mm. In primo luogo verrà spigato il sistema di alimentazione della scheda: essendo presenti molti elementi diversi per il rispetto di tutte le specifiche richieste, questa parte è eterogenea in quanto comprende l'utilizzo di convertitori lineari e switching per generare le 5 alimentazioni necessarie.

Successivamente verranno esaminate le modalità di realizzazione del front-end analogico, ovvero si esporranno le linee di progettazione del sistema di acquisizione dei segnali analogici il quale comprende ADC e il relativo stadio di signal conditioning.

Dopo la parte analogica, si passerà alla trattazione della parte digitale della scheda: verrà spiegato il collegamento della FPGA alla SDRAM, quello seriale al bus I2C attraverso un level shifter e in ultimo la UART.

Infine si cercherà di spiegare nella maniera più esaustiva possibile le modalità che presiedono il collegamento con la comunicazione USB: in particolare la scelta del chip di conversione da UART a USB e il suo collegamento sulla scheda.

2.1 Alimentazione

La scheda riceve dalla WCC tre tensioni di alimentazione: una 8 V che presiede l'alimentazione analogica, una −8 V che in questo ambito non è utilizzata, in quanto ogni componente presente sulla FastPro è ingrado di funzionare correttamente utilizzando la sola alimentazione positiva, e infine è presente una 5 V per l'alimentazione della parte digitale della scheda: tutte queste alimentazioni di ingresso sono riferite alla massa digitale del sistema.

Dal momento che le tensioni di funzionamento dei componenti sono diverse da quelle di ingresso, si è deciso di usare dei convertitori di tensione DC-DC e per garantire una tensione di alimentazione più stabile.

Sulla scheda sono presenti due masse diverse: un ground digitale (DGND) che costituisce la massa indispensabile per le alimentazioni e i segnali della FPGA, SDRAM, UART, USB e I2C e un ground analogico(AGND) necessario ai segnali da campionare provenienti dai componenti della DSA 1.

Questa scelta è stata presa in accordo con le specifiche degli ADC AD7356: queste suggeriscono inoltre un collegamento a stella tra le due masse, ovvero in un solo punto, questa specifica è stata adottata anche perché garantisce il miglior isolamento dal rumore digitale per le componenti analogiche e di conseguenza permette di realizzare e ottenere la migliore precisione della misura da parte degli ADC.

Infatti avendo progettato il collegamento in un solo punto sulla scheda, utilizzando una resistenza da 0 Ω, si riesce a non creare un accoppiamento induttivo tra le due masse, in quanto le correnti di ground non possono chiudersi in un loop che intersechi attraversando le due masse.

Nel progetto della scheda sono stati utilizzati due alimentatori: il primo che è un convertitore di tensione lineare con uscita a 5 V e il secondo che è un alimentatore con uscite multiple, di cui due con tecnologia switching e due lineari.

Per quanto riguarda l'alimentatore lineare, è stato scelto, fra i tanti, il MC7805, che garantisce un'uscita fissa a 5 V, il che ha, come conseguenza, la possibilità di non realizzare un circuito di configurazione aggiuntivo, riuscendo a risparmiare sia componenti e sia la complessità della scheda, e una corrente di uscita di 1 A.

Inoltre, come già detto, la tensione di ingresso dell'alimentatore proviene direttamente dalla WCC ed è di 8 V, in aggiunta, dal momento che la potenza consumata da un convertitore lineare è data dalla seguente equazione:

$$P_D = (V_{IN} - V_{OUT}) \times I_{LOAD} \tag{2.1}$$

è possibile calcolare che la potenza massima dissipata dal componente risulta 3 W.

Il conseguente calore generato da questo componente viene dissipato, in misura adeguata, attraverso un package Double Decawatt Package (D2PAK) al cui pad di dissipazione termica è collegato il piano AGND che ha anche la funzione di dispersione del calore in quanto si estende per circa la metà della scheda misurando 6785 mm^2: in questo modo si riesce a garantire un corretto funzionamento in temperatura del componente.

Per quanto riguarda le altre 4 alimentazioni, nascono tutte dalla power rail a 5 V della WCC, riferita al sistema digitale, ed è collegata all'ingresso dell'alimentatore LM 26480 che comprende due buck converters con frequenza di commutazione a 2,1 MHz: questa garantisce un'oscillazione della tensione e della corrente di alimentazione di uscita molto limitata, anche senza usare induttanze e capacità estremamente consistenti.

Infatti l'oscillazione della corrente sull'induttore di uscita di un alimentatore switching con controller buck è data dalla seguente equazione:

$$\Delta I_L = \frac{(V_{IN} - V_{OUT}) \times D}{f_s \times L} \tag{2.2}$$

mentre l'oscillazione della tensione di ingresso è data da:

$$\Delta V_C = \frac{\Delta I_L}{8 \times f_s \times C_{OUT}} \tag{2.3}$$

Inoltre le tensioni di uscita dei convertitori buck sono quella del core della FPGA che è alimentato a 1,2 V e quella relativa ai restanti sistemi digitali come la SDRAM ed i piedini di uscita della FPGA alimentati a 3,3 V, mentre i due convertitori Low Drop Out (LDO) del LM 26480 sono riservati per l'alimentazione analogica sia del ADC e sia del Phase Locked Loop (PLL) della FPGA.

Affinché un alimentatore switching funzioni correttamente e con il minor rumore elettrico possibile, è di fondamentale importanza che il layout sulla scheda sia progettato con il massimo rigore.

Infatti, se le metallizzazioni di un circuito stampato non sono progettate accuratamente si verificano, di regola, dei fenomeni di electromagnetic interference (EMI), ground bounce e pericolose cadute di tensione, dovute alle resistenze delle piste.

Per ovviare a questi problemi per la realizzazione di questa parte di circuito sono state utilizzate le regole riassunte di seguito ed esposte nella scheda dati del LM26480 [5]: dal momento che gli induttori e condensatori di uscita vengono percorsi da correnti elevate, questi componenti sono stati posizionati vicino al convertitore buck e sono stati collegati con piste corte in modo tale da riuscire a minimizzare il rumore.

Inoltre i componenti sono stati posizionati in modo che le correnti di commutazione riescano a scorrere nello stesso verso: durante la prima metà del ciclo, la corrente fluisce dal condensatore di ingresso, attraversa il buck converter e l'induttore per arrivare infine al condensatore di uscita; durante la seconda metà del ciclo, questa viene presa dal ground, passa per il buck converter attraverso l'induttore per arrivare al condensatore di uscita.

Infatti per quanto riguarda il collegamento dei componenti, le piste sono state progettate in modo che la corrente fluisca nella stessa direzione per l'intero ciclo per evitare ogni cambiamento di campo magnetico.

In successione logico-organizzativa, i pin di ground del LM 26480 sono stati collegati, insieme a quello del condensatore di uscita, attraverso un piano di massa abbastanza esteso da essere in grado di garantire una connessione a bassa impedenza ed evitare il ricircolo di correnti di commutazione attraverso il piano di massa.

Infine i componenti del circuito che vengono attraversati da correnti elevate sono stati collegati con piste il più possibile larghe, mentre quelli relativi al feedback

per configurare la tensione di uscita, sono stati posizionati lontano dai componenti di potenza e connessi con piste strette in modo da minimizzare le interferenze e garantire, di conseguenza, una tensione di uscita il più possibile vicina a quella di progetto. Il risultato è riportato in figura 2.1.

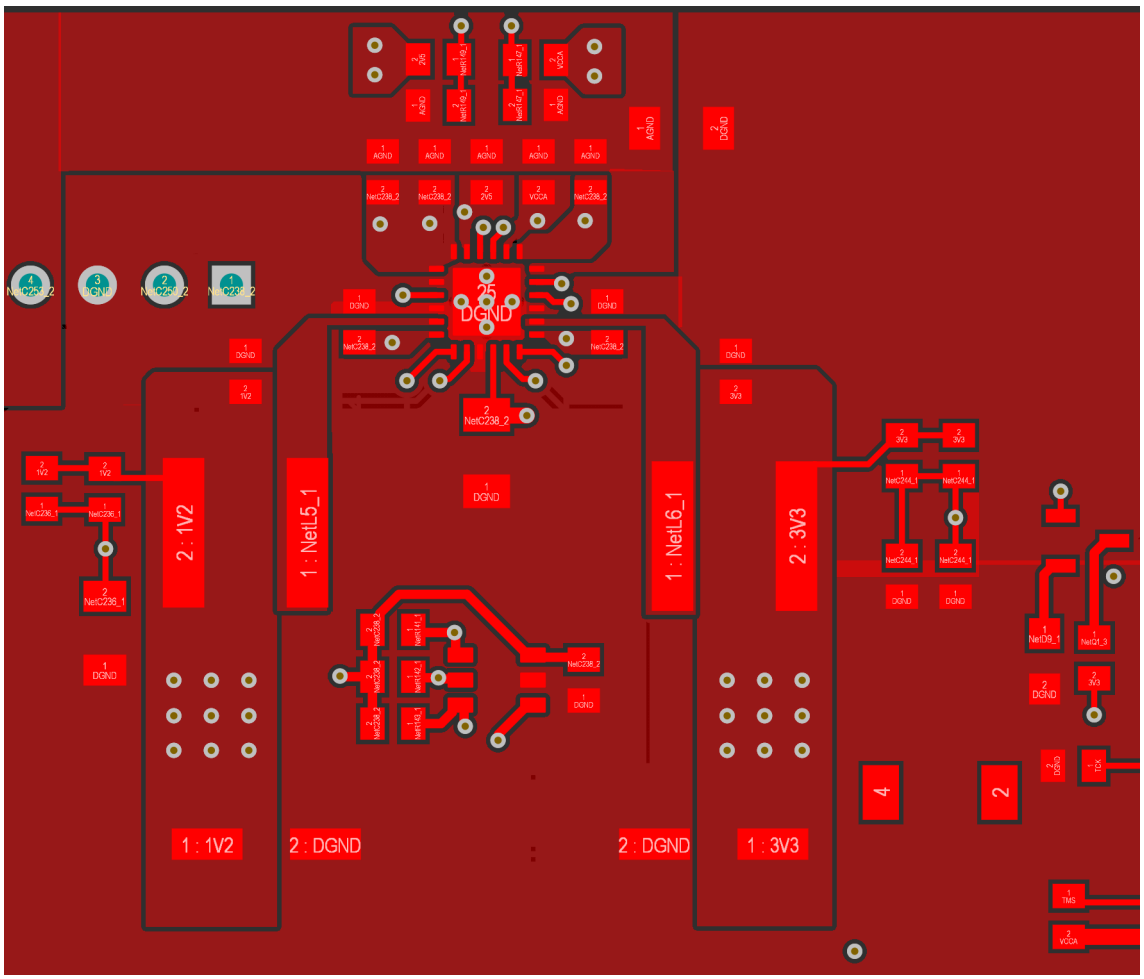


Figura 2.1: Top Layer del circuito di alimentazione switching

Per quanto riguarda la scelta di componenti quali le resistenze di configurazione, gli induttori e i condensatori di ingresso e di uscita, poiché gli alimentatori switching risultano sensibili alle instabilità che provocherebbero oscillazioni della tensione di uscita rendendo di conseguenza inutilizzabile tutta la scheda, sono stati scelti i componenti consigliati nel datasheet del LM 26480 [5].

2.2 Front-End Analogico

Al fine di garantire la migliore precisione della misura, al momento del campionamento attraverso gli ADC, e al fine di garantire un corretto funzionamento dei campionatori, risulta necessario che ci sia un circuito adeguato di signal conditioning, con il minimo rumore esterno presente alle frequenze di interesse, per questo è innanzitutto necessario avere un filtro anti aliasing.

Questo è l'unico strumento in grado di filtrare eventuali rumori che verrebbero riportati nella banda di campionamento e inoltre si evidezia l'ulteriore necessità di ridimensionare il range di tensione dei segnali in ingresso, in modo che riescano a ricoprire tutto il full scale range del ADC ed infine occorre eliminare l'offset causato dal fatto che i segnali di ingresso, nel caso oggetto di questa Tesi, sono sempre unipolari.

Il modello di ADC scelto per la conversione è il AD 7356, che è un convertitore analogico digitale con ingresso differenziale, frequenza di campionamento a 5 MHz e tensione di ingresso differenziale $-2.048 \div 2.048$ V e tensione di ingresso single ended $0 \div 2.048$ V e precisione di 12 bit.

Occorre puntualizzare che la scelta della frequenza di campionamento è stata condizionata fortemente dalla necessità che i segnali, una volta convertiti nel dominio digitale, vengano utilizzati per realizzare una ricostruzione per punti delle forme d'onda, riguardanti gli attimi precedenti ad un guasto, mentre la risoluzione è stata scelta per garantire un rumore di quantizzazione abbastanza esiguo da essere trascurabile, nel caso lo si confronti con la precisione richiesta dalla misura, queste specifiche sono state fornite in fase di definizione del progetto da Rheinmetall s.p.a.®

Di conseguenza occorre spiegare che il circuito di signal conditioning ha, come componente principale, un filtro attivo passa-basso del quarto ordine: per progettarlo ci si è serviti del tool FilterPro™di Texas Instruments® che è un programma di supporto al progetto, utilizzato in quanto rende possibile la realizzazione del design di filtri, anche in configurazione differenziale, funzione che non è disponibile con molti strumenti presenti sul mercato: una schermata del tool utilizzato è presente in figura 2.2.

Il filtro che è stato realizzato è di tipo passa basso la cui funzione di trasferimento è stata calcolata attraverso l'utilizzo di coefficienti di Butterworth in modo tale che la misura dei segnali in banda passante non sia influenzata da un ripple del guadagno del filtro: la configurazione scelta per questo filtro è stata quella multiple feedback in quanto, avendo utilizzato operazionali fully-differential, risulta l'unica possibile, considerando che altre, come Sallen-Key, richiedono una retroazione positiva che bisogna escludere per la realizzazione di circuiti basati sul tipo di amplificatori utilizzati per il progetto.

Inoltre la scelta degli operazionali è ricaduta, dopo attente analisi, sul ADA 4930, che è un amplificatore operazionale fully-differential molto usato per la realizzazione

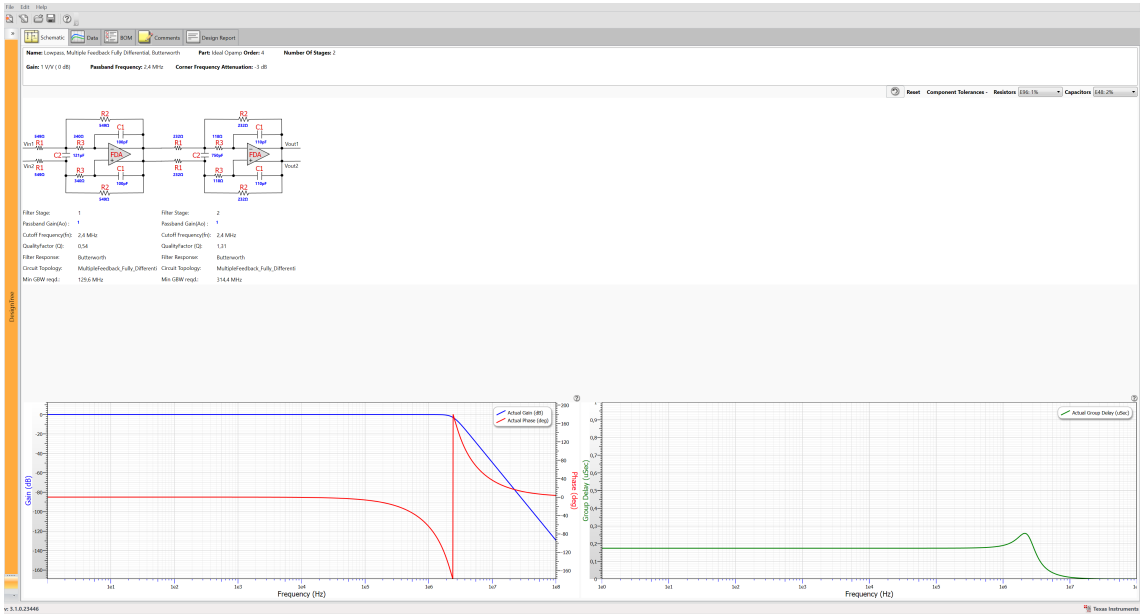


Figura 2.2: Schermata Software Progettazione Filtri

di circuiti di signal conditioning per ADC ad alte prestazioni, acquistato nel package contenente due op-amp su un solo circuito integrato.

Come riportato nella sua scheda tecnica[6], questo circuito analogico presenta e opera sotto le seguenti condizioni:

1. prodotto banda guadagno di 1,35 GHz;
2. offset in uscita massimo di 500 µV;
3. e densità di rumore dal valore tipico di 1,2 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ in output.

Le sue specifiche sono state scelte affinché il circuito di signal conditioning avesse un rumore totale necessariamente non superiore al rumore di quantizzazione e il prodotto banda guadagno è stato selezionato al fine di avere un comportamento di amplificatore ideale, anche a frequenze di almeno 10 volte superiori a quelle del funzionamento della scheda.

Utilizzando il componente sopracitato, è stato realizzato un filtro anti aliasing che ha una frequenza di taglio a -3 dB pari a 2,3 MHz e ha 4 poli, ovvero due stadi di amplificazione: non avendo ricevuto specifiche esplicite su questa parte del circuito, la definizione della frequenza di taglio è stata basata sulla priorità di conservare un rumore minimo, per cui è leggermente inferiore della frequenza di Nyquist teorica.

Simultaneamente la definizione del numero di poli è stata unicamente basata sulla complessità della scheda da realizzare, ovvero sullo spazio necessario per contenere comodamente tutti i componenti che il filtro richiede: a tale fine sono stati scelti due

stadi di filtraggio perché su questa scheda potevano essere posizionati solo questo numero di componenti, senza scegliere package per resistori eccessivamente piccoli, che avrebbero reso irrealizzabile un processo di saldatura manuale. Uno stadio del filtro è mostrato in figura 2.3.

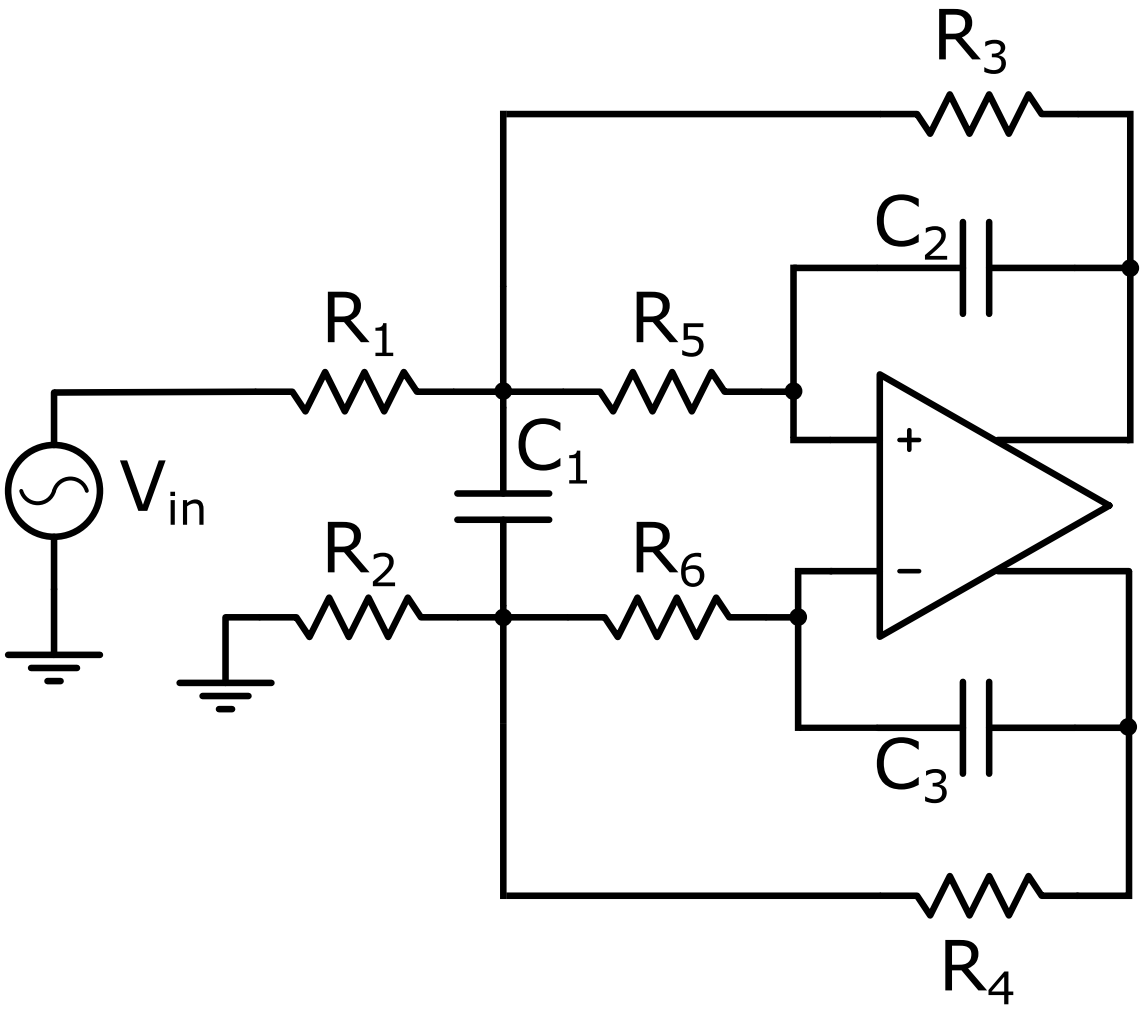


Figura 2.3: Stadio di un Filtro Passa-basso Multiple Feedback

Oltre al circuito descritto precedentemente, dal momento che i segnali analogici da controllare, provenienti dai vari componenti della DSA, sono single-ended unipolari e hanno uno swing dinamico di $0 \div 5\,\mathrm{V}$, è necessario regolare il guadagno del filtro attivo e allo stesso tempo eliminare l'offset di ingresso allo scopo di utilizzare tutta la dinamica di ingresso del ADC.

Dal momento che il tool di sviluppo FilterPro™non include la possibilità di progettare filtri con guadagno inferiore a 1, ci si è serviti necessariamente del loop di feedback resistivo: attraverso l'utilizzo della nomenclatura in figura 2.3, sapendo che:

$$V_{OUT\ DIFF} = V_{IN\ DIFF} \times \frac{R_3}{R_2} = V_{IN\ DIFF} \times \frac{R_4}{R_6} \tag{2.4}$$

è possibile, come è ampiamente noto in letteratura, definire il guadagno come $G = \frac{V_{OUT\ DIFF}}{V_{IN\ DIFF}}$ e impostarlo al valore di 0,82 per far rientrare in dinamica i segnali di ingresso al ADC.

Il passo successivo della fase progettuale è stato quello di eliminare l'offset per portare la dinamica dei segnali a $-2.5 \div 2.5\,\mathrm{V}$: ci sono diverse soluzioni che possono essere adottate in questi casi, come ad esempio, inserire un condensatore in serie al segnale: questa soluzione è però, in questo caso, non praticabile in quanto eliminerebbe il valore stazionario di funzionamento, che è codificato in DC, mentre questo costituisce un'informazione fondamentale per il sistema M&C.

È sta presa in esame un'altra possibilità cioè quella di inserire all'ingresso collegato a massa dell'amplificatore un riferimento di tensione pari al valore medio del segnale di ingresso, in questo caso 2,5 V: questa soluzione risulta più adatta, ma, allo stesso tempo poco pratica, in quanto richiede l'inserimento di un altro componente, andando ad aumentare la complessità della scheda il che avrebbe comportato l'impossibilità di utilizzare due stadi di filtraggio.

Di conseguenza si è deciso di utilizzare il riferimento presente nel ADC, utilizzato anche per fornire la polarizzazione del modo comune di uscita, come ingresso in una configurazione alle differenze: non essendo stato possibile trovare in letteratura questa configurazione, una dimostrazione formale del funzionamento del circuito è presentata di seguito, facendo riferimento alla figura 2.4.

Come è noto dalla letteratura:

$$V_{OUT\ P} - V_{OUT\ N} = a(f) \times (V_P - V_N) \tag{2.5}$$

dove $a(f)$ è l'amplificazione dell'operazionale che assumiamo $\gg 0$.

Scrivendo le equazioni ai nodi V_P e V_N otteniamo per V_P:

$$\frac{V_S - V_P}{R_G} + \frac{V_{OUT\ P} - V_P}{R_F} = \frac{V_P}{R_{OFFSET}} \tag{2.6}$$

e per V_N

$$\frac{V_{OUT\ P} - V_N}{R_F} + \frac{V_{OFFSET} - V_N}{R_{OFFSET}} = \frac{V_N}{R_G} \tag{2.7}$$

mettendo in evidenza V_P in 2.6 e V_N in 2.7 otteniamo rispettivamente le seguenti equazioni:

$$V_P = R_{EQ} \times \left(\frac{V_{OUT\ N}}{R_F} + \frac{V_S}{R_G} \right) \tag{2.8}$$

e

$$V_N = R_{EQ} \times \left(\frac{V_{OUT\ P}}{R_F} + \frac{V_{OFFSET}}{R_{OFFSET}} \right) \tag{2.9}$$

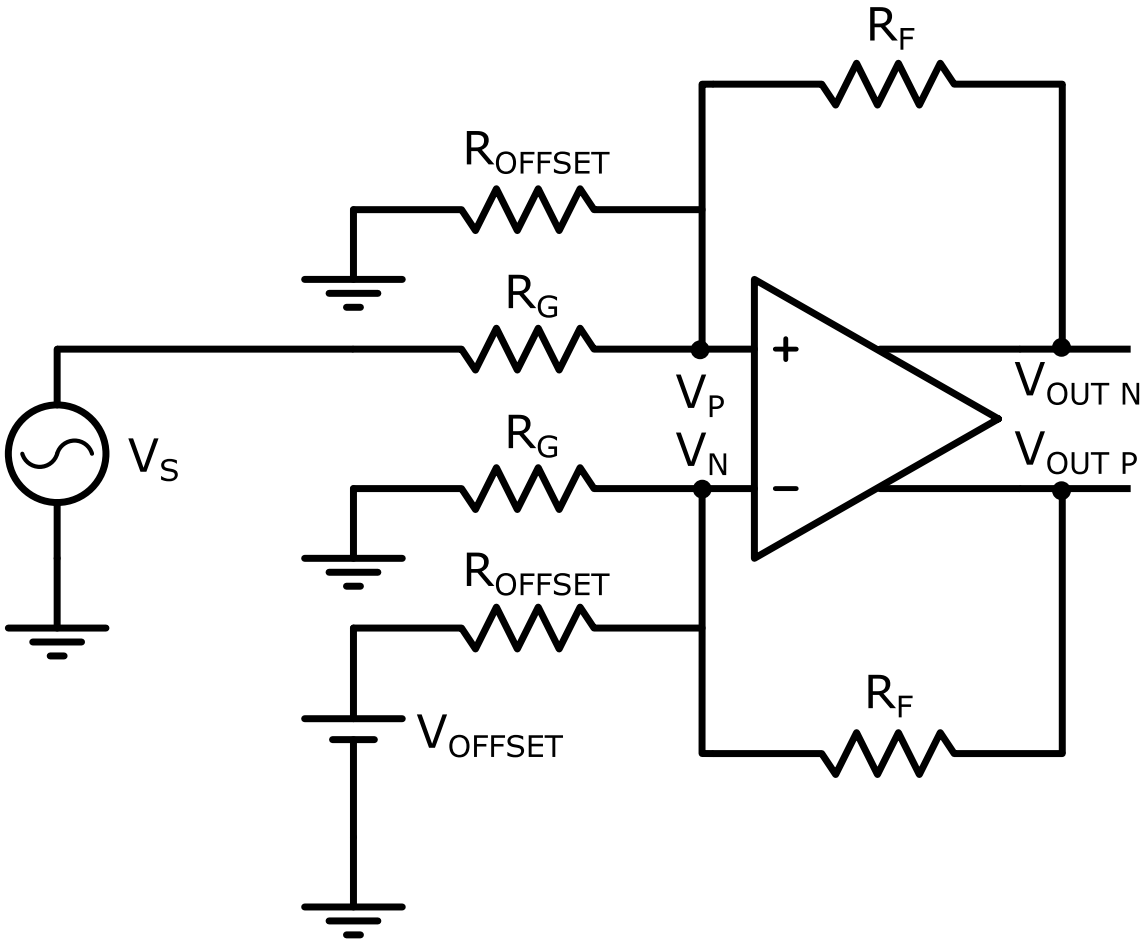


Figura 2.4: Circuito Configurazione Operazionale alle Differenze

dove $R_{EQ} = \left(\frac{1}{R_{OFFSET}} + \frac{1}{R_G} + \frac{1}{R_F}\right)$
Sostituendo le due equazioni 2.8 e 2.9 nella 2.5 otteniamo:

$$V_{OUT\ P} - V_{OUT\ N} = \frac{a(f) \times R_{EQ} \times \left(\frac{V_S}{R_G} - \frac{V_{OFFSET}}{R_{OFFSET}}\right)}{1 + \frac{R_{EQ}}{R_F} \times a(f)} \tag{2.10}$$

Assumendo $a(f) \gg 0$ alle frequenze di funzionamento del sistema otteniamo:

$$V_{OUT\ P} - V_{OUT\ N} = \frac{R_F}{R_G} \times V_S - \frac{R_F}{R_{OFFSET}} \times V_{OFFSET} \tag{2.11}$$

Nella dimostrazione, si è trascurato il mismatch delle resistenze e inoltre dall'equazione 2.11 è stato possibile calcolare il valore della resistenza R_{OFFSET}, ponendo

$V_{OFFSET} = 1{,}024\,\text{V}$; $V_S = 2{,}5\,\text{V}$; $V_{OUT\ P} - V_{OUT\ N} = 0\,\text{V}$. In figura 2.5, in figura 2.7, in figura 2.6 e in figura 2.8 sono rappresentati dei grafici che descrivono il funzionamento di un operazionale fully-differential con e senza l'aggiunta della tensione di offset, in caso di ingresso unipolare.

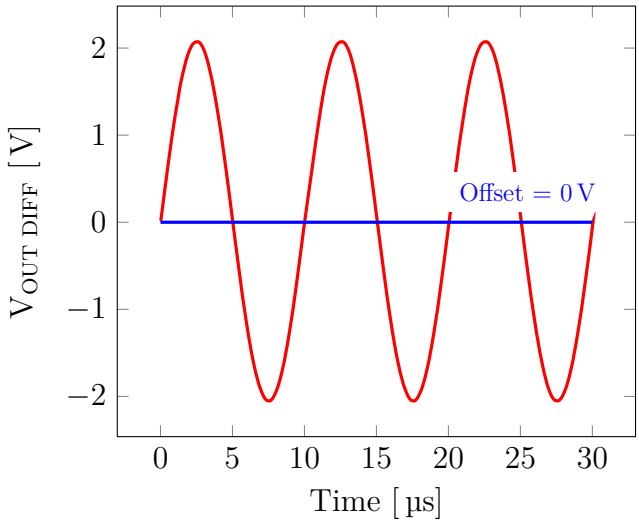


Figura 2.5: $V_{OUT\ DIFF}$ con Eliminazione Offset

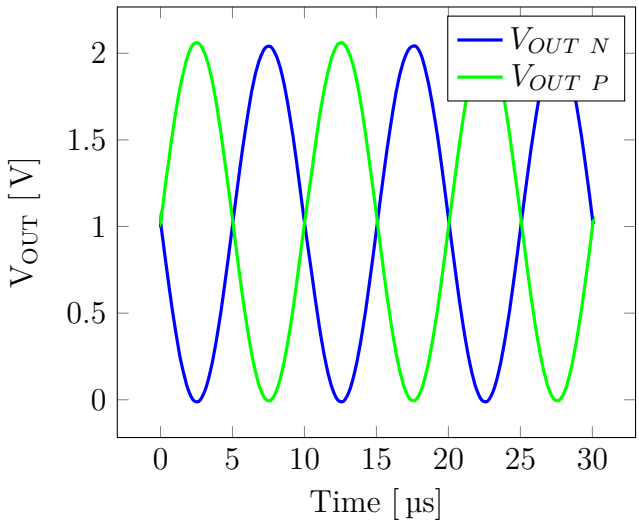


Figura 2.6: V_{OUT} Single Ended con Eliminazione Offset

Pertanto, per quanto riguarda la polarizzazione del modo comune di uscita e la tensione V_{OFFSET}, è stato utilizzato il riferimento interno al ADC che è stato prima isolato e poi diviso per 2, con un partitore resistivo collegato all'operazionale

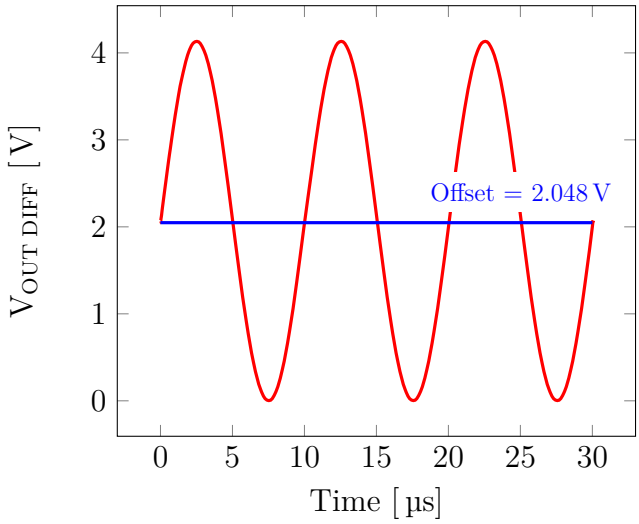


Figura 2.7: $V_{OUT\ DIFF}$ senza Eliminazione Offset

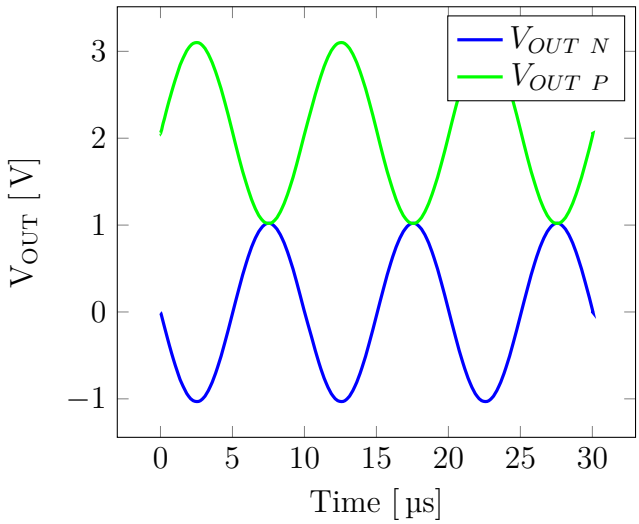


Figura 2.8: V_{OUT} Single Ended senza Eliminazione Offset

di precisione MCP6V97T che presenta una tensione di offset in uscita che misura appena $\pm 25\,\mu V$.

Inoltre, dal punto di vista della selezione dei componenti passivi, il filtro è stato realizzato con delle resistenze della serie E48, con tolleranza pari a $\pm 0{,}1\%$ del valore nominale: purtroppo, non è stato possibile usare la serie E96, in quanto non erano, al momento di acquisto dei componenti, ampiamente disponibili sul mercato.

L'ultimo elemento di considerevole rilevanza di cui è necessaio parlare è il disaccoppiamento delle alimentazioni: sono stati utilizzati condensatori al tantalio con

capacità di 10 µF che risultano attivi solo alle basse frequenze, mentre sono stati sfruttati dei condensatori ceramici da 100 nF o 1 µF, a seconda di quello che veniva specificato nella scheda dati di ciascun componente.

Infatti, per quanto riguarda gli amplificatori operazionali ADA 4930 sono stati utilizzati un condensatore di bulk al tantalio da 10 µF per ogni alimentazione, ovvero 1 componente per ogni operazionale presente nel chip, e uno in ceramica dal valore di 100 nF per ogni pin di alimentazione, ovvero 2 per ogni amplificatore presente nel chip.

Al contrario il disaccoppiamento dell'alimentazione del ADC AD 7356 è stato realizzato sempre con un condensatore al tantalio e uno ceramico, con capacità di 100 nF: il circuito è stato replicato sia per l'alimentazione analogica, sia per quella digitale, come indicato nella scheda dati del componente [7].

Per quanto invece riguarda l'op-amp di precisione MCP6V97T, come già enunciato nella sua scheda dati [8], è stato messo in atto un approccio differente, che consiste nell'utilizzo di due condensatori ceramici uno con capacità pari a 100 nF e uno da 1 µF, che riescono ad avere sia una capacità considerevole, sia riescono ad essere stabili ad alte frequenze.

2.3 FPGA

La scheda è pilotata, nella sua totalità, da una FPGA che si occupa di comandare e gestire tulle le sue funzionalità: di conseguenza il corretto funzionamento di questo componente è di vitale importanza per il successo della realizzazione della scheda e, essendo un circuito molto complesso, richiede numerose attenzioni sia nella progettazione dello schema elettrico, sia nella successiva definizione del circuito stampato.

Infatti, a differenza di un microcontrollore, una FPGA ha bisogno di un sistema di circuiti esterni complessi: il modello scelto, per le sue funzionalità adeguate, per il costo contenuto e per la sua ampia disponibilità sul mercato, è stato quello di Intel® Cyclone® 10 LP e sono state ampiamente seguite tutte le direttive di progettazione di un circuito stampato suggerite dalla casa produttrice al fine di garantirne il corretto funzionamento.

Di conseguenza, per quanto riguarda le alimentazioni, sono presenti 2 alimentazioni digitali obbligate: per il core dal valore di 1,2 V e per i pin di I/O dal valore di 3,3 V e un'ultima alimentazione analogica dal valore di 2,5 V.

Inoltre la stabilità di tutte queste tensioni di alimentazione è stata accuratamente progettata, utilizzando il tool fornito da Intel® per realizzare il disaccoppiamento delle alimentazioni per le sue FPGA. Come prima operazione è stato necessario selezionare lo stato del funzionamento a regime del circuito integrato: infatti maggiore

sarà l'occupazione di risorse e la frequenza di clock, maggiore sarà, di conseguenza, la corrente richiesta dalla FPGA e di conseguenza si dovrà usare una assoluta attenzione a questa fase del progetto.

Una volta selezionato il regime di funzionamento e la tensione di alimentazione per cui si vuole progettare il circuito in base a questi presupposti, il tool riesce a calcolare con precisione assoluta la Z_{target}, ovvero il valore massimo che può assumere l'impedenza vista ai piedini dell'alimentazione della FPGA, questo valore segue la formula:

$$Z_{target} = \frac{VoltageRail \times \frac{\%Ripple}{100}}{MaxTransientCurrent} \tag{2.12}$$

Inoltre, all'interno del tool, l'impedenza della linea di alimentazione viene di regola disegnata su un grafico che ne ritrae l'andamento in frequenza con precisione e fedeltà: quindi l'impedenza misurabile ai pin di alimentazione della FPGA deve essere al di sotto di Z_{target} per un range di frequenze più ampio possibile.

In questo programma inoltre sono presenti dei modelli di impedenze per vari package di condensatori che si possono utilizzare: ogni volta che si aggiunge uno di questi componenti alla lista dei condensatori presenti nel circuito di disaccoppiamento che si vuole realizzare, è possibile vedere con evidenza sul grafico come risulta l'andamento dell'impedenza in frequenza.

Infatti, come è noto in letteratura, ogni condensatore reale è formato da una capacità, una resistenza e un'induttanza serie che insieme forniscono un andamento dell'impedenza in frequenza, decrescente fino alla frequenza di risonanza e, per frequenze più alte, crescente dove diventa dominante la componente induttiva.

Di conseguenza inserendo più condensatori in parallelo, è sempre possibile diminuire l'impedenza totale e, operando una scelta accurata dei valori di capacità, si riesce ad ottenere un comportamento stabile anche a correnti di alimentazione molto elevate.

Dal momento che le parti parassite induttive e resistive di un condensatore sono dipendenti unicamente dal package, sono stati utilizzati componenti di dimensioni fisiche pari a (0603 $mils$) che, nonostante siano disponibili sul mercato package più piccoli, sono risultate un ottimo compromesso tra componenti parassiti e praticità durante l'operazione di saldatura manuale. Un esempio di selezione di condensatori di disaccoppiamento è in figura 2.9.

Sempre riguardo ai circuiti necessari al funzionamento della FPGA è indispensabile citare il circuito di programmazione: la Cyclone™10 LP è dodata di una memoria di configuraione basata su tecnologia volatile Static Random Access Memory (SRAM) che necessita di essere riscritta ad ogni accensione del dispositivo per poter funzionare correttamente.

Di conseguenza nel circuito è stata inserita una memoria Electrically Erasable Read Only Memory (EEPROM) non volatile in grado di consentire la tempestiva riprogrammazione della FPGA, ad ogni accensione.

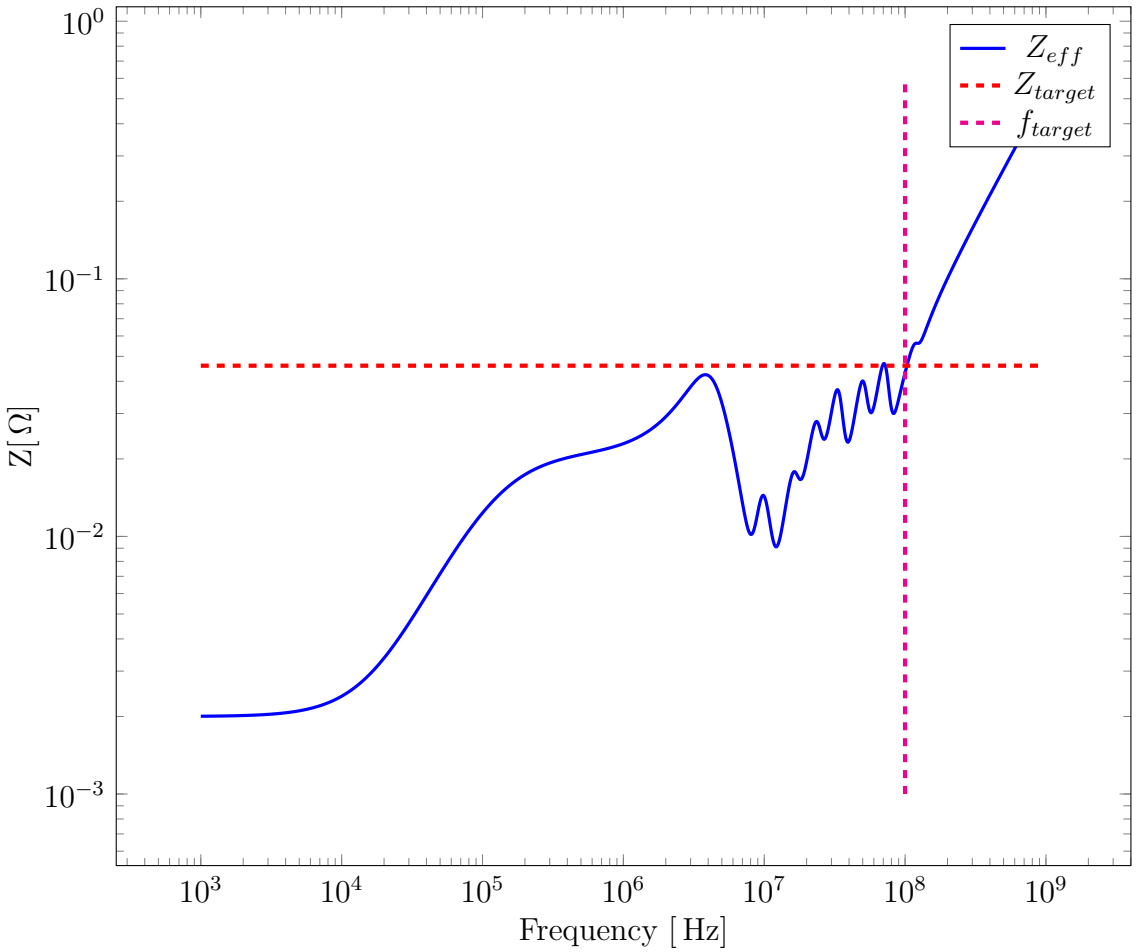


Figura 2.9: Z_{eff} in funzione della Frequenza

Per quanto riguarda la programmazione della memoria volatile e quella più lenta, ma indispensabile per il prodotto finale della EEPROM, è stato utilizzato il protocollo Join Test Action Group (JTAG) che è largamente supportato dai programmi di aiuto al progetto sviluppati da Intel™come Quartus.

Questa interfaccia seriale è costituita da 4 segnali: TCK, TDO, TMS, TDI: questi sono stati collegati secondo le indicazioni, riguardanti la programmazione dei dispositivi, fornite dal produttore del circuito integrato presenti nella propria scheda tecnica.

Infatti il segnale di TCK è stato collegato al corrispondente pin dedicato della FPGA e al DGND attraverso una resistenza da 1 kΩ,inoltre i segnali TMS e TDI sono stati collegati all'alimentazione attraverso delle resistenze da 10 kΩ ed, infine, per il segnale di TDO, è stata realizzata una semplice, ma funzionale connessione punto punto.

Successivamente ognuna delle tracce che compongono il bus JTAG sono state realizzate con un'impedenza caratteristica di 50 Ω con la finalità di non avere onde riflesse, fenomeno che verrà spiegato, seppure brevemente, nel proseguo della tesi.

Inoltre per configurare la FPGA all'accensione è stato implementato il protocollo sviluppato da Intel™per programmare tutte le FPGA con memoria volatile chiamato Active Serial: per utilizzare questa funzionalità, è necessario acquistare una memoria di configurazione proprietaria e, in questo caso, è stata scelta la serie EPCQA che risulta la più disponibile sul mercato ed è assolutamente compatibile con la FPGA, utilizzata nella scheda e il protocollo Active Serial.

Infatti i segnali che comandano questa memoria sono CSO, ASDI, ASDO, DCLK che sono stati collegati ai corrispondenti pin della FPGA attraverso connessioni punto punto, con l'eccezione di ADSI al quale è stata applicata una resistenza di terminazioni di 25 Ω per evitare onde riflesse.

Affinchè la EPCQA possa programmare la Cyclone™è stata utilizzata la funzione di programmazione JTAG Indirect Configuration che è supportata da Quartus per la parte software e, a livello hardware, da USB Blaster il quale è stato anche utilizzato per la programmazione della memoria volatile della FPGA.

Un'altra componente importante della scheda è il collegamento tra la FPGA e la SDRAM, in quanto la necessità di progettare un nuovo sistema di M&C è nata dal fatto che si voleva migliorare il sistema di diagnostica, andando ad immagazzinare in modo molto veloce e assolutamente affidabile i campioni prelevati dagli ADC.

Inoltre, il collegamento di un'interfaccia parallela ad alta frequenza è notoriamente un procedimento che richiede particolare precisione durante il layout di un circuito stampato: infatti, come è noto in letteratura, una linea di trasmissione presenta un ritardo che segue la seguente formula:

$$t_{PD} = \sqrt{L_0 \times C_0} \tag{2.13}$$

dove L_0 e C_0 sono rispettivamente l'induttanza e la capacità della pista di collegamento.

L'interfaccia tra la SDRAM e la FPGA sulla FastPro è composta da 39 collegamenti diversi e i segnali presenti su ciascuno di questi devono arrivare a destinazione tutti entro un intervallo di tempo molto breve che risponde a delle precise specifiche, in modo che la trasmissione funzioni correttamente.

Infatti esistono diversi standard che sono costituiti da altrettante specifiche sulle tolleranze dei ritardi: per la scheda oggetto di questa Tesi, si è operata la scelta di attenersi alle specifiche dello standard Double Data Rate (DDR) 2, nonostante la memoria fosse una Single Data Rate (SDR).

Questo protocollo prevede una differenza massima tra i ritardi delle piste di collegamento di soli 2 ps, e dal momento che il ritardo di propagazione di ogni pista è dipendente dalla sua lunghezza, risulta necessario che queste siano il più possibile uguali le une alle altre.

Al fine di realizzare con successo questo collegamento si è fatto uso di una funzione di Altium Designer® che è in grado di realizzare senza errori delle curve sulle piste, in modo da eguagliare la lunghezza di quelle più corte, una volta impostato correttamente.

Infatti, per utilizzare con successo questa funzionalità occorre, in prima istanza, racchiudere tutti i segnali della SDRAM sotto una classe di net e imporre successivamente una regola di progetto, per indicare al tool di sviluppo che questi collegamenti non possono avere differenze di ritardi superiori a 2 ps e usare in conclusione lo strumento per creare le serpentine.

Inoltre, essendo un collegamento ad alta velocità (100 MHz), è anche importante che l'impedenza caratteristica della pista rispetti in toto lo standard SDR SDRAM, ovvero sia uguale a 50 Ω: questa specifica risulta importante, in quanto i collegamenti sulla scheda sono abbastanza lunghi rispetto alle frequenze dei segnali.

Infatti dal momento che le piste si comportano come linee di trasmissione e di conseguenza, come è noto dalla teoria, presentano un coefficiente di riflessione che è definito dalla 2.14.

$$\Gamma = \frac{Z_0 - Z_L}{Z_0 + Z_L} \tag{2.14}$$

dove Z_0 è l'impedenza caratteristica della linea e Z_L è l'impedenza di carico che è propria del pin della SDRAM o della FPGA a seconda della direzione della comunicazione, è essenziale che le due impedenze sopracitate siano uguali.

Dal momento che nello standard SDR Z_L è definita a 50 Ω, allora anche Z_0 dovrà essere uguale allo stesso valore, in modo da rendere il coefficiente di riflessione nullo: questo comporta che non si verificano fenomeni di riflessione e, di conseguenza, i segnali sono il più possibile privi di difetti rendendo maggiore la probabilità di avere una comunicazione senza alcun errore.

A riguardo va prestata particolare attenzione al clock, in quanto ha una frequenza di commutazione doppia rispetto a tutti gli altri segnali, inoltre rappresenta il segnale di sincronia di tutto il sistema, compito che lo rende il più importante nel protocollo in questione.

Per verificare che le forme d'onda che si propagano nella scheda siano in specifica con i componenti che le generano o ricevono, è stato utilizzato un sotto-programma per la verifica della signal integrity presente in Altium Designer®: uno screen del risultato ottenuto per i segnali della SDRAM è in figura 2.10 e figura 2.11.

Occorre prendere in esame anche un'altra componente fondamentale della scheda, cioè la comunicazione I2C con la WCC, che è da sola responsabile di tutte le funzioni necessarie alla configurazione della FastPro e alla sua compatibilità con il precedente sistema di M&C.

Infatti il protocollo prevede solo due segnali: Serial Data Line (SDA) e Serial Clock Line (SCL), i quali rappresentano il bus dati e il segnale di clock rispettivamente e inoltre, dal momento che il precedente sistema di controllo risale al 2002,

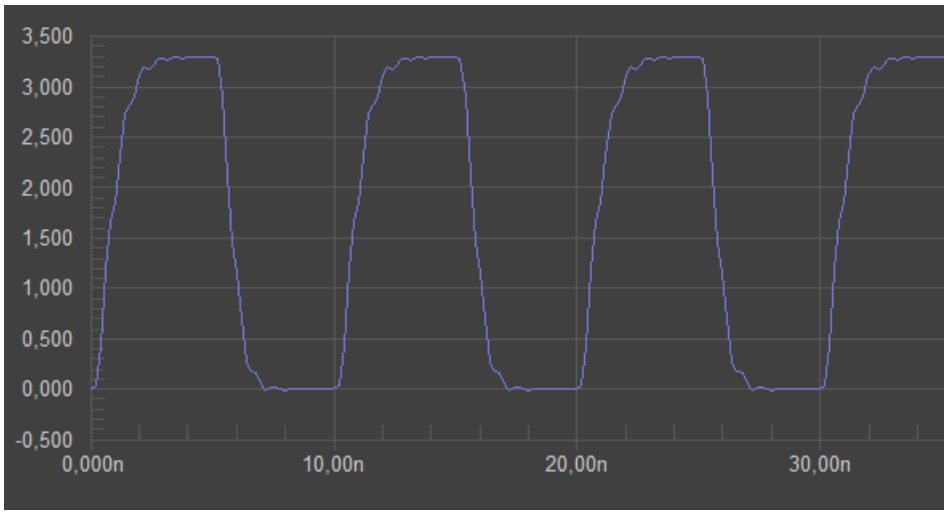


Figura 2.10: Signal Integity del Segnale di CLK della SDRAM

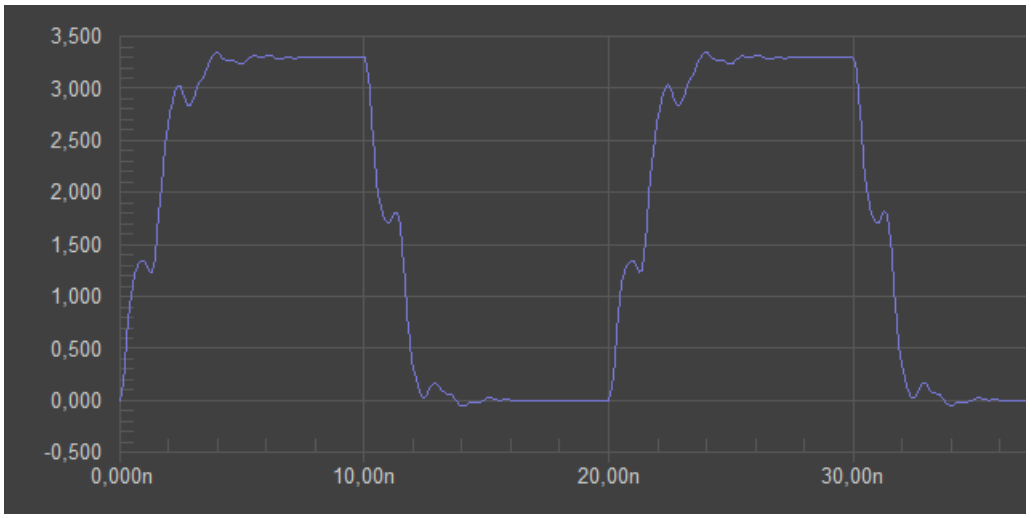


Figura 2.11: Signal Integity del Segnale di un bit della SDRAM

segue ancora uno standard elettrico a 5 V, mentre la più moderna Cyclone™10 LP è compatibile con uno standard massimo di 3,3 V.

Da questa differenza tra i due sistemi scaturisce la necessità di introdurre sulla scheda una traslazione di livello bidirezionale da 5 volt a 3,3 V e, per realizzarla, è stato utilizzato AN97055 di Philips® [9] che presenta un circuito uguale a quello in figura 2.12.

Come si può constatare un N-MOS è inserito sulla linea SDA e un altro sulla linea SCL, collegando il source al body ed entrambi alla parte del bus I2C ad alimentazione più bassa, mentre il gate è collegato alla tensione di alimentazione più bassa e il drain

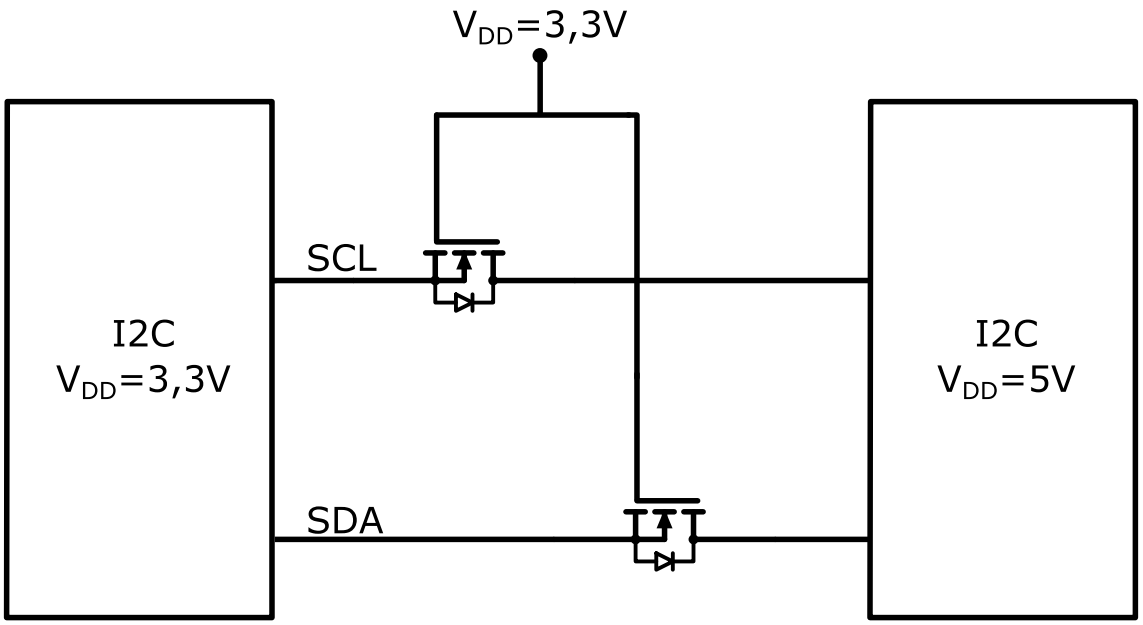


Figura 2.12: Circuito Level Shifter I2C

alla linea del bus alimentazione a tensione più alta.

Di conseguenza si può analizzare il funzionamento del circuito, identificando 3 condizioni di operazione: la prima consiste nella condizione in cui nessuna delle due parti del bus sta cercando di abbassare la tensione della linea e quindi il N-MOS ha il comportamento di un circuito aperto, in quanto il gate e il source sono entrambi a 3,3 V e di conseguenza la V_{GS} risulta nulla, in questo caso entrambe le parti del bus vedono la propria tensione di alimentazione senza interferenze.

La seconda condizione si verifica quando un dispositivo della parte ad alimentazione inferiore necessita di abbassare la tensione sul bus: in questo caso il gate è collegato a 3,3 V, mentre la tensione del source scende a 0 V, di conseguenza la V_{GS} è superiore alla tensione di soglia di accensione del N-MOS che diventa un circuito chiuso, abbassando la tensione, anche sulla parte del bus alimentato a 5 V.

La terza ed ultima condizione si verifica quando la parte ad alimentazione superiore ha bisogno di abbassare la tensione sul bus: in questo caso, la V_{GS} in un primo momento risulta nulla, mentre la giunzione p-n tra body e drain diventa polarizzata direttamente ed inizia a condurre, andando successivamente ad abbassare la tensione del source, cortocircuitato con il body, consentendo alla V_{GS} di superare la tensione di soglia e conseguentemente di accendere il transistor.

2.4 Comunicazione USB

I dati immagazzinati nella SDRAM, in caso di guasto della DSA, vengono prontamente inviati al computer centrale, per essere utilizzati nella fase di diagnostica: questa comunicazione avviene con il protocollo USB, invece la FPGA comunica attraverso il protocollo UART e, di conseguenza si presenta la necessità di convertire questo tipo di comunicazione nel primo.

Al fine di compiere correttamente questa operazione si è deciso di utilizzare un circuito integrato di FTDI® ovvero il FT232H: questo chip è compatibile con il protocollo UART, fino alle velocità di trasmissione di 12 Mbaud /s ed inoltre è in grado di convertirlo nel protocollo USB versione 2.0 Hi-Speed che ha una velocità di trasmissione di 480 Mb/s.

Per garantire il corretto funzionamento di questo componente, è necessario progettare un circuito complesso di supporto che presenti molti componenti.

In prima istanza si è optato per una configurazione bus-powered, ovvero,per l'alimentazione di questo sotto-sistema viene prelevata direttamente la tensione di alimentazione del protocollo USB che è definita al valore nominale di 5 V.

In questo modo è assolutamente possibile evitare l'utilizzo di un'alimentazione a parte per questa sezione di scheda: in questa modalità, il FT232H riesce a generare, attraverso dei convertitori DC-DC lineari tutte, le diverse alimentazioni utili al suo funzionamento.

Infatti l'alimentazione proveniente dal bus USB viene filtrata attraverso una ferrite, in modo da eliminare quasi totalmente i disturbi accoppiati sul cavo di collegamento: questa viene convertita a 3,3 V , in quanto destinata al PLL e al Phisical Layer della USB e sono ulteriormente filtrate da una ferrite, per ridurre al meglio i disturbi presenti sull'alimentazione.

Per quanto riguarda il disaccoppiamento sono stati utilizzati dei condensatori al tantalio da 10 µF come bulk in parallelo ed un condensatore in ceramica da 100 nF per l'alimentazione in ingresso e in uscita ai regolatori di tensione lineari, mentre sono stati utilizzati condensatori in ceramica da 100 nF collegati ad ogni pin di alimentazione per le tensioni di Phyisical Layer di USB, pin di I/O del FT232H e tensione di alimentazione del core del circuito integrato.

Il chip FT232H per svolgere le sue funzioni ha bisogno di essere configurato attraverso una EEPROM di configurazione: seguendo la scheda dati del componente [10] si è optato per la memoria 93LC56BT-I/OT di Microchip®, collegata alla stessa linea di alimentazione di 3,3 V e disaccoppiata da un condensatore ceramico di 100 nF. Il pin di Data-Out è connesso attraverso una resistenza di pull-up da 10 kΩ all'alimentazione e attraverso una resistenza serie da 2 kΩ al pin di Data-In, questi due pin sono stati collegati al bit di dato del bus seriale comandato da FT232H.

Il protocollo USB2.0 Hi-Speed ha bisogno di un clock a 480 MHz e viene realizzato attraverso un PLL interno al FT232H che riceve la frequenza base da un cristallo al

quarzo da 12 MHz il cui segnale viene amplificato internamente.

Rispettando le convenzioni USB si è optato per un connettore di tipo micro-b in quanto più piccolo del mini-b, ma sempre non della tipologia A in quanto la scheda non rappresenta l'HOST della comunicazione USB. In questo connettore il pin denominato ID è stato lasciato floating in quanto non utilizzato.

Essendo USB 2.0 High-Speed un protocollo ad alta velocità, è stato necessario prestare particolare attenzione alla realizzazione del layout della scheda relativa alle piste che portano il segnale di dato. Infatti sono state rispettate a pieno le specifiche del protocollo realizzando due tracce differenziali con impedenza differenziale di 90 Ω e impedenza single-ended di 45 Ω proprio per evitare segnali riflessi.

Le piste sono state lasciate il più corte possibile e sono state tracciate sopra un piano di riferimento senza interruzioni di metallizzazioni. Un'ultima ulteriore precauzione per quanto riguarda l'integrità dei segnali è stata quella di inserire nelle piste differenziali di USB, una coppia di diodi per la soppressione delle sovra tensioni modello RClamp0582B molto utilizzati per le comunicazioni USB 2.0 e altri protocolli differenziali.

Per verificare la correttezza del progetto della scheda sarebbe stato utilizzato il sotto-programma di Altium Designer® di signal integrity, il quale però richiede un modello ibis del componente per fornire risultati affidabili al progettista, in questo caso non è stato possibile utilizzarne uno in quanto il produttore e unico ente in grado di fornire uno di questi modelli non forniva questo ulteriore servizio.

Capitolo 3

Realizzazione del Sistema Digitale

In questo capitolo viene esposta l'architettura delle componenti della parte digitale del sistema FastPro.

In primo luogo verrà descritta l'interfaccia SPI, responsabile per la comunicazione tra FPGA e ADC: si elencheranno i componenti principali che compongono l'architettura come, ad esempio, la parte di de-serializzazione e la macchina a stati che gestisce la trasmissione e la ricezione dei comandi e dati.

Successivamente si esporrà il sistema responsabile della comunicazione tra FPGA e SDRAM: si indicherà come è stata utilizzata una Intellectual Property (IP) di Intel™contenente un controller per SDRAM a cui il sistema si interfaccia utilizzando un bus Avalon.

Inoltre verrà mostrato come è stato utilizzato il PLL presente nella FPGA per creare una differenza di fase tra il segnale di clock interno al sistema e quello che pilota la memoria in modo da annullare il ritardo che i segnali accumulano sulle piste della scheda.

Successivamente verrà esposto il sistema responsabile per la comunicazione con il pc di controllo che avviene attraverso una UART convertita esternamente nel protocollo USB.

Inoltre verrà spiegato come è stata utilizzata una IP di Intel™per implementare un'architettura UART e come è stato realizzato il circuito di supporto per controllare la IP e per metterla in comunicazione con la parte del sistema digitale responsabile della lettura della SDRAM in caso di guasto.

Infine verrà indicato come è stata utilizzata la IP di Intel™per realizzare la comunicazione tramite bus I2C tra la FastPro e la WCC: spiegando come sono stati organizzati i registri di stato, impostazione e dati in modo da facilitare la comunicazione e i comandi dati alla scheda dal sistema di controllo centrale rappresentato dalla WCC.

3.1 Interfaccia SPI

L'interfaccia SPI è responsabile della comunicazione tra FPGA e ADC ed è quindi finalizzata all'acquisizione delle misure in forma digitale effettuate dai 6 convertitori presenti sulla scheda e già descritti nella loro funzione analogica nel capitolo precedente.

Questa interfaccia è stata utilizzata anche per pilotare il convertitore analogico digitale AD 7356 in quanto è in grado di comandare il sistema di sample and hold, il convertitore scelto supporta ampiamente l'interfaccia utilizzata che è anche uno standard per il controllo degli ADC.

Questo modello di campionatore è stato scelto in quanto rispetta le specifiche di precisione e frequenza di campionamento esposte nel capitolo precedente, allo stesso tempo è compatibile con l'interfaccia SPI: infatti molti convertitori utilizzano il più complesso sistema di comunicazione Low Voltage Differential Signaling che, sebbene sia in grado di fornire prestazioni superiori in termini di velocità di comunicazione, richiede anche un hardware più performante e dal costo superiore.

Infatti la FPGA scelta, ovvero la Cyclone™10 LP non dispone del sistema di serializzazione e de-serializzazione dedicato e quindi risulta meno efficiente.

L'interfaccia consiste in un segnale di clock chiamato Serial Clock (SCLK) che è responsabile di scandire la velocità di comunicazione e della conversione da analogico a digitale.

Sono stati utilizzati 6 segnali di Master In Slave Out (MISO), uno per ogni ADC come è previsto dallo standard, che svolgono la funzionalità di inviare la misura del convertitore in forma seriale alla FPGA, mentre il segnale di chip select è utilizzato per selezionare gli ADC e per iniziare la fase di hold.

Per capire come è stata costruita l'architettura dell'interfaccia è importante spiegare come funziona il particolare convertitore da analogico a digitale: come si può vedere dal timing diagram presentato nella scheda tecnica del AD 7356 [7] in figura 3.1, il segnale di chip select (che utilizza una logica invertita) inizia il processo di acquisizione: il falling edge del chip select mette il track and hold in modalità hold, a questo punto il segnale analogico viene campionato e il bus SPI viene rimosso dallo stato ad alta impedenza.

Anche la conversione inizia a questo punto e richiede un minimo di 14 cicli di clock sul pin SCLK per essere completata. Come si può vedere dalla figura 3.1 dopo che sono intercorsi 13 colpi di clock il sistema torna nella modalità di track al rising edge del 14° ciclo di clock. Sul fronte di salita del segnale chip select la conversione viene conclusa definitivamente e il bus dati torna in alta impedenza.

Per quanto riguarda i dati: il falling edge di chip select da inizio alla trasmissione che presenta due 0 che non riguardano il dato effettivo da leggere e quindo possono essere scartati, i dati rimanenti sono presentati sul bus in modo seriale ogni fronte di discesa del segnale SCLK.

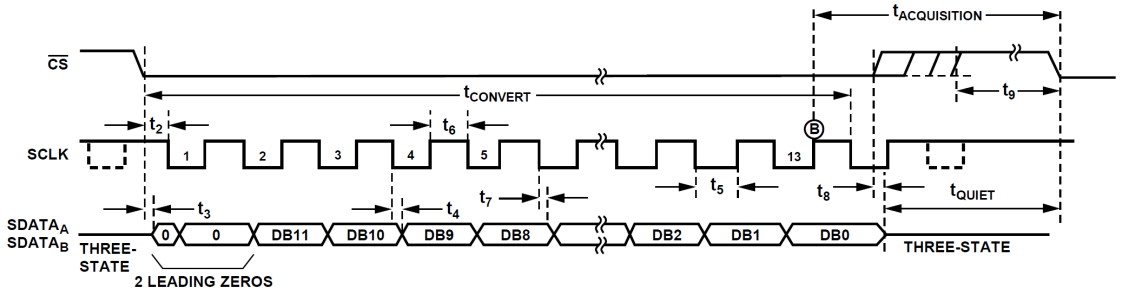


Figura 3.1: Timing Diagram dell'Interfaccia Seriale dell'ADC (Courtesy of Analog Devices)

Di conseguenza il segnale SCLK rappresenta il clock della trasmissione e il ricevitore, in questo caso la FPGA, dovrà campionare i dati sul fronte di salita del SCLK in modo da avere un corretto funzionamento.

Dal punto di vista della codifica dei dati l'ADC presenta una codifica straight-binary in cui il valore analogico differenziale di ingresso $-2{,}048\,\mathrm{V}$ rappresenta lo 0 in base decimale e il valore analogico differenziale $+2{,}048\,\mathrm{V}$ rappresenta il valore 4095 in base decimale.

Questa codifica è in grado di rappresentare in modo molto accurato i segnali di ingresso in quanto nessuno di questi scende mai sotto lo zero e, di conseguenza non è necessario introdurre numeri negativi. Infatti i valori differenziali analogici negativi servono solo per avere una dinamica di ingresso del ADC doppia rispetto a quella teoricamente permessa dalla sua tensione di alimentazione.

Uno schema dell'architettura seguita sia per generare correttamente le forme d'onda necessarie al controllo degli ADC, sia per svolgere le funzioni logiche interne al sistema digitale, come il salvataggio dei dati in un registro di supporto per essere successivamente scritti nella memoria SDRAM, è presentato in figura 3.2.

Il clock del sistema è generato attraverso uno dei 4 PLL presenti all'interno della FPGA ed ha una frequenza di 160 MHz, questa frequenza è stata scelta in base al fatto che corrisponde ad un periodo di circa 6 ns che è compatibile con il tempo minimo in cui il segnale di chip select deve essere nello stato alto e quindi è possibile generare tutti i segnali in modo totalmente sincrono il che garantisce al sistema la massima stabilità, ovvero un comportamento indipendente da ritardi temporali assoluti.

Inoltre dividendo attraverso un clock divider la frequenza per due è possibile generare il segnale SCLK a 80 MHz necessario per ottenere la frequenza di campionamento richiesta. Il PLL presente all'interno della FPGA è fornito di un segnale che indica quando il clock di uscita è considerabile funzionante.

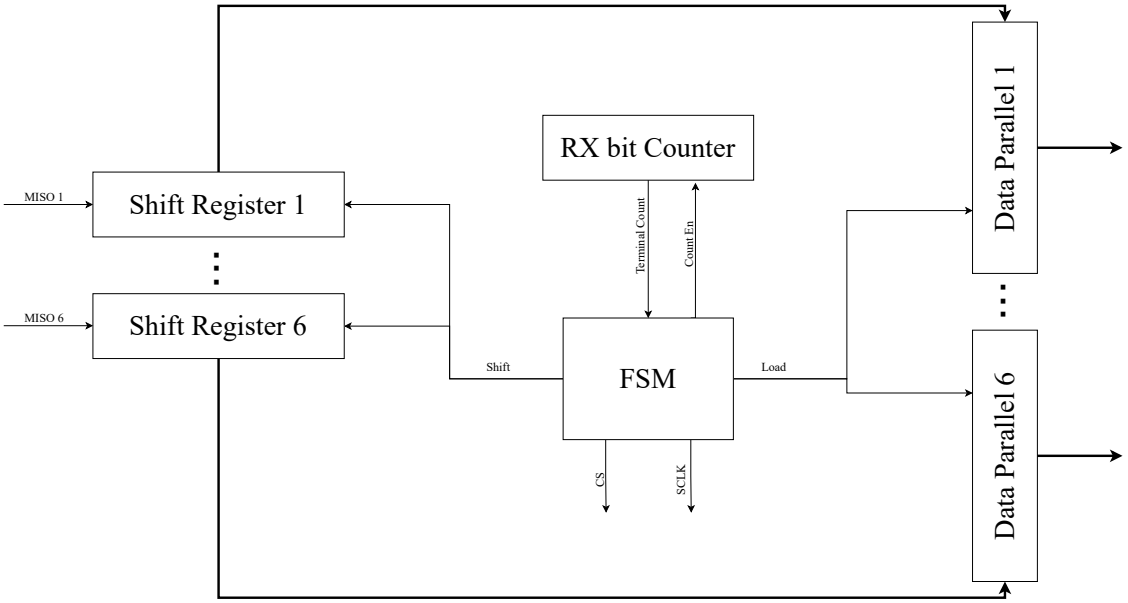


Figura 3.2: Diagramma a Blocchi interfaccia SPI

Questo segnale è stato utilizzato per mantenere in stato di reset, e quindi congelati, i componenti sequenziali in modo tale che il sistema non abbia comportamenti anomali dovuti ad un funzionamento fuori specifica durante il transitorio di agganciamento del clock da parte del PLL

Come si può vedere dalla figura 3.2 sono stati utilizzati 6 shift register, tanti quanti sono i convertitori da analogico a digitale, con parallelismo pari a 12 bit che vengono controllati da una macchina a stati finiti e un contatore.

Il dato una volta acquisito nella sua interezza viene inviato al banco di registri a cui ha accesso il modulo I2C, come si può vedere dalla figura 1.5.

La macchina a stati finiti schematizzata in figura 3.3, è stata utilizzata per il controllo di tutti i componenti del sistema di trasmissione SPI. Dopo uno stato di reset, in cui si inizializzano tutti i componenti sequenziali ad uno stato noto per evitare glitch anche solo transitori, inizia la conversione vera e propria.

Il primo stato operativo genera il fronte di discesa del segnale chip select e mette il sample and hold in modalità hold. Lo stato successivo è quello che controlla la trasmissione: questo stato ad ogni rising edge di SCLK cattura un bit della trasmissione su tutti e 6 i segnali di MISO, esclusi i primi due bit che non sono rappresentanti della misura, e lo salva in 6 shift register ognuno assegnato ad un ADC.

Per controllare la fine della trasmissione è stato utilizzato un counter abilitato dal fronte di salita del clock della stessa: una volta finita l'acquisizione dei dati lo stato successivo salva nei registri di I2C i valori delle misure in modo da essere letti dalla WCC e essere salvati dalla SDRAM.

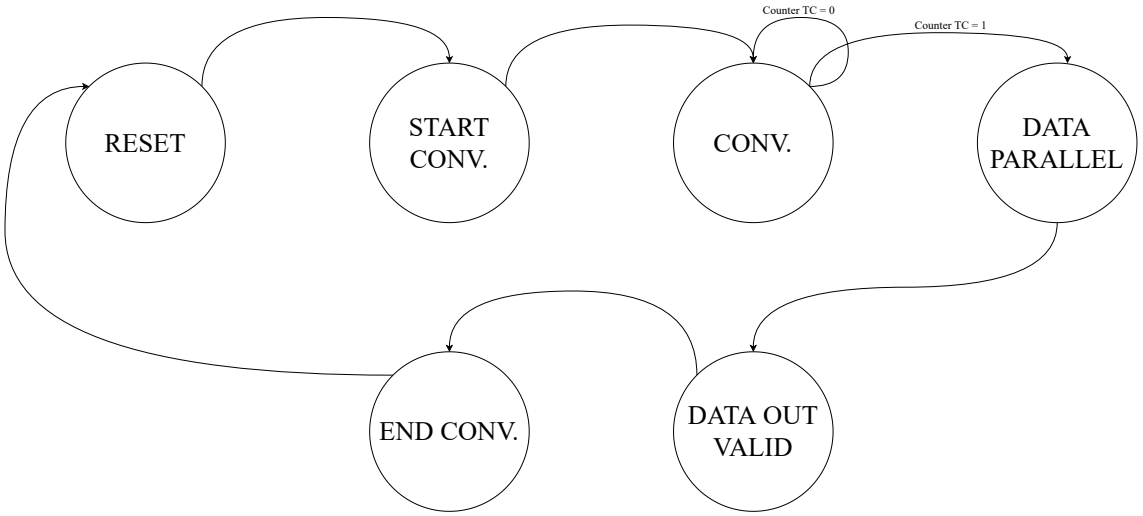


Figura 3.3: Macchina a Stati Finiti dell'Interfaccia SPI

Questo ulteriore set di registri è necessario in quanto il sistema inizia immediatamente una nuova conversione potenzialmente sovrascrivendo dati mai salvati nella SDRAM. Lo stato successivo pone ad 1 logico un bit responsabile per comunicare al resto del sistema che è disponibile un nuovo set di dati da salvare, segnale che è anche utilizzato nel protocollo di sincronizzazione che verrà spiegato nel capitolo successivo.

A seguire si passa ad uno stato responsabile per il termine della conversione e viene alzato il bit di chip select in modo da comandare il track and hold del ADC di tornare in modalità track. Si ritorna poi nello stato di reset in quanto è necessario aspettare un colpo di clock affinché il ADC abbia il tempo di configurarsi ed essere pronto per una nuova conversione.

Utilizzando il metodo appena descritto è stato possibile arrivare alla frequenza di conversione di 5 MHz.

3.2 Utilizzo di un controller SDRAM

Una volta acquisiti i dati dagli ADC questi vengono salvati in una SDRAM SDR, motivo per cui è nata la necessità di utilizzare un sistema M&C aggiornato per la nuova DSA di New Norcia. Per conseguire questo obiettivo è necessario, oltre ad una memoria SDRAM, anche il rispettivo sistema di controllo.

Un controller SDR per SDRAM, nonostante sia più semplice in confronto ad altri il cui funzionamento è definito da protocolli più avanzati come DDR 3 o il più recente DDR 4, riserva comunque molte difficoltà di progettazione, ragion per cui si è deciso, in questa Tesi, in accordo con Microsis s.r.l.®, azienda responsabile per il progetto,

di utilizzare una IP di Intel™in modo da ottimizzare i tempi per la progettazione dei sistemi digitali e analogico completi.

Come riportato nella sua scheda dati [11], il controller SDR rispetta le specifiche dello standard PC100 ed è in grado di interfacciarsi con sistemi che presentano diversi data width 8, 16, 32 o 64 bit, diverse capienze di memoria e un numero di chip select superiori ad 1.

Il controller comunica con il sistema interno, ovvero il restante hardware progettato all'interno FPGA, attraverso l'interfaccia Avalon Memory Mapped (Avalon MM) che è latency-aware e di conseguenza consente ai trasferimenti in lettura di usare il procedimento della pipeline.

L'interfaccia Avalon MM è utilizzata spesso dalle IP di Intel™ed è utile per collegare numerosi componenti di un sistema digitale quali: microprocessori, memorie, interfacce seriali come UART, sistemi DMA e timer. La peculiarità di questa interfaccia è il suo ampio raggio di complessità che permette di sviluppare sistemi semplici con hardware ridotto e di conseguenza utilizzare protocolli più complessi solo in caso di assoluta necessità.

Per quanto riguarda l'interfaccia con la memoria SDRAM sono state utilizzate le interfacce «memory mapped with wait request» per la scrittura e l'interfaccia «pipelined read transfer with variable latency» per quanto riguarda il processo di lettura. In figura 3.4 e in figura 3.5 sono rappresentati i timing diagram delle due interfacce utilizzate.

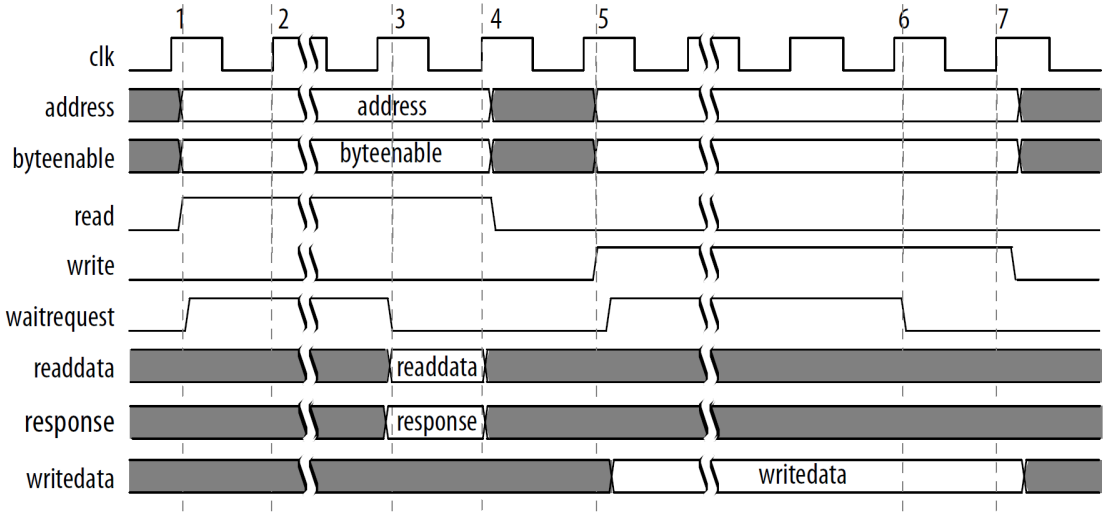


Figura 3.4: Interfaccia Avalon MM di Scrittura (Courtesy of Intel®)

Come si può vedere dalla figura 3.4 il processo di scrittura avviene utilizzando i segnali di write indicando l'indirizzo e il dato da scrivere, inoltre è presente un

Pipelined Read Transfers with Variable Latency

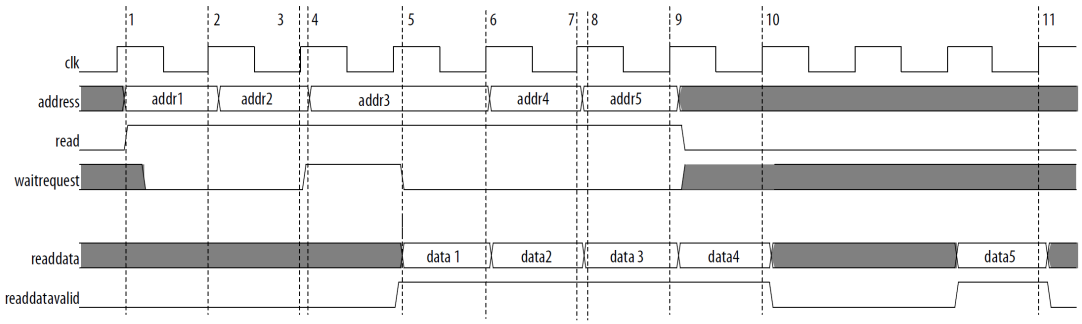


Figura 3.5: Interfaccia Avalon MM di Lettura (Courtesy of Intel®)

segnale di byte enable che è utilizzato come maschera per selezionare il parallelismo della word da scrivere, nel caso sotto esame 16 o 8 bit, invece il segnale di waitrequest svolge all'interno del protocollo Avalon MM la funzione di indicare che la memoria in quel momento è inaccessibile.

Per il ciclo di lettura sono stati utilizzati i segnali read, address, data e watrequest insieme ad un segnale di readdatavalid che viene asserito quando è presente un dato sul bus.

Come si può vedere dalla figura 3.5. è possibile richiedere all'interfaccia un dato successivo anche quando il precedente non è stato ancora ricevuto, utilizzando il processo di pipeline.

Come è noto dalla letteratura le SDRAM, nonostante siano memorie sincrone, necessitano di segnali che arrivino ad intervalli temporali precisi che esulano dal periodo di clock. Perché questo avvenga, la IP di Intel™consente di impostare queste tempistiche attraverso una tabella riportata in tabella 3.1 in cui sono schematizzati i valori inseriti nella IP per la SDRAM utilizzata durante il progetto del sistema trattato in questa Tesi.

Come si può vedere dalla tabella 3.1 è possibile impostare tutte le principali tempistiche: la latenza in termini di cicli di clock del segnale di Column Address Strobe (CAS), il secondo elemento della tabella 3.1 corrisponde al numero di comandi di refresh che bisogna inviare alla SDRAM prima di essere utilizzata, il parametro "Issue one refresh command every" indica ogni quanto inviare un segnale di refresh ed è calcolato tramite la seguente formula:

$$t_{ref\ command} = \frac{Refresh\ Period}{Number\ of\ Rows} \tag{3.1}$$

Il parametro "Delay after powerup, before initialization" indica quanto tempo deve

Parametro	Valore
CAS latency cylcles	3
Initialization refresh cycles	2
Issue one refresh command every	7,8125 µs
Delay after powerup, before initialization	200 µs
Duration of refresh command (t_{rfc})	63 ns
Duration of precharge command (t_{rp})	21 ns
ACTIVE to READ or WRITE delay (t_{rcd})	21 ns
Access time (t_{ac})	5,4 ns
Write recovery time (t_{wr} no auto precharge)	14 ns

Tabella 3.1: Tabella parametri di configurazione IP Intel® SDRAM Controller

passare da quando le alimentazioni e il clock sono completamente funzionanti per inviare il primo comando di refresh iniziale, i parametri t_{rfc}, t_{rp} indicano la durate dei rispettivi comandi.

Il parametro t_{rcd} indica quanto tempo deve passare, una volta attivata una riga, prima di inviare il comando di lettura o scrittura, t_{ac} indica il tempo necessario per avere il dato in uscita e t_{wr} indica quanto tempo deve passare dopo una scrittura prima di disattivare la riga selezionata.

Il controller consente di arrivare alla velocità di scrittura e lettura teorica di una word per ogni ciclo di clock, questo tempo però non può mai essere raggiunto in quanto sono necessarie alcune funzioni di controllo che rallentano la lettura e scrittura. In ogni caso la IP esegue automaticamente tutte le ottimizzazioni necessarie al raggiungimento della massima velocità di scrittura e lettura, ovvero utilizza la "Page Mode" che consente di salvare le word da scrivere in un set di registri interni alla SDRAM e infine scrive tutto in un solo ciclo, lo stesso concetto è applicabile alla lettura.

Inoltre, se si utilizza una scrittura in sequenza nell'interfaccia Avalon MM, ad ogni prima pagina scritta o letta, il controller cambia automaticamente banco in modo da rendere trasparenti le tempistiche dei controlli della SDRAM all'utente. In figura 3.6 è presente lo schema a blocchi della parte di interfaccia con la SDRAM.

Come si può vedere è presente una macchina a stati finiti che consente di generare tutti i segnali necessari al funzionamento del protocollo Avalon MM e acquisire i dati contenenti le misure degli ADC insieme ad un byte di stato, ovvero un byte contenente se e quali componenti si trovano in un funzionamento anomalo: questo byte è necessario in fase di diagnostica.

Dal momento che ogni ciclo di scrittura consiste nello scrivere più di una parola, viene utilizzato un multiplexer il cui selettore è controllato da un counter che fornisce al bus dati di Avalon MM una nuova word ogni volta che è stata scritta la precedente. Il counter è controllato dai segnali di write e waitrequest per avanzare di un ciclo.

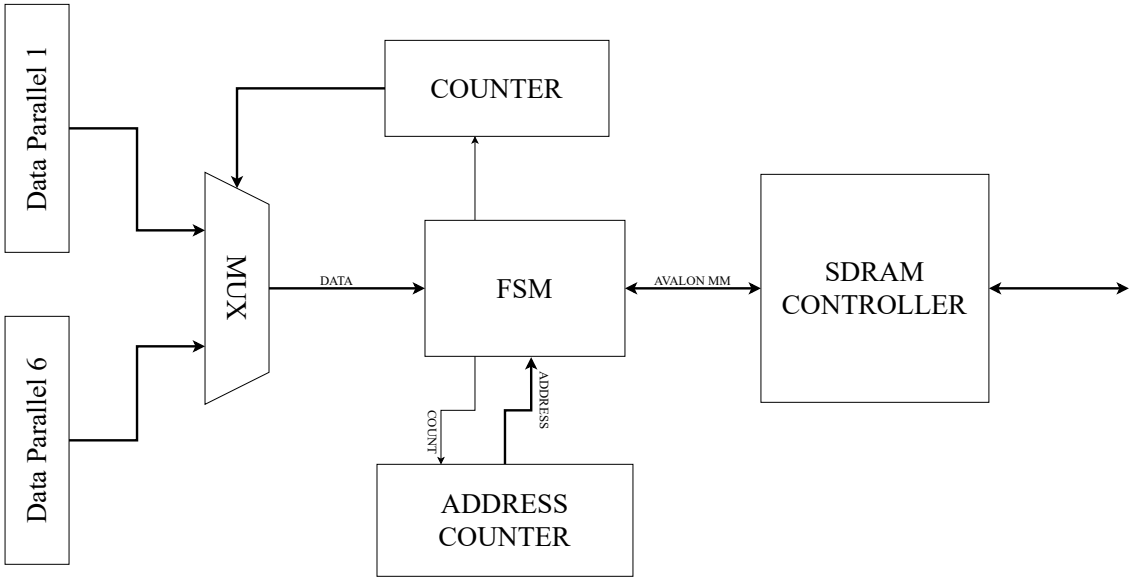


Figura 3.6: Diagramma a blocchi Interfaccia SDRAM

Un altro counter è utilizzato per la generazione dell'indirizzo per il bus Avalon MM che è controllato nello stesso modo del precedente. In caso di un fault il sistema passa in modalità lettura per leggere i dati da inviare all'interfaccia UART. In questo caso viene utilizzato un counter che è inizializzato tramite interfaccia I2C che decide il numero di dati da inviare al pc e decrementa questo numero ogni volta che viene letto un dato dalla SDRAM e, una volta arrivato a 0, viene interrotta la trasmissione e viene atteso un segnale di reset.

La macchina a stati finiti di questo sottosistema è rappresentata in figura 3.7, è da notare il fatto che questa è condivisa anche con il sottosistema UART in modo da aumentare le prestazioni.

Dopo un primo stato di reset in cui si portano tutti i componenti sequenziali in una configurazione nota, si passa ad uno stato in cui si attendono i dati inviati dall'interfaccia SPI degli ADC e, una volta ricevuto il segnale di data available, si manda un segnale di acknoledge e si procede con lo scrivere i dati sulla SDRAM. Finito il ciclo di scrittura si passa di nuovo nello stato in cui si attendono i dati per ricominciare il processo.

Questo ciclo continuo di scrittura può essere interrotto al verificarsi di un fault: in questo caso, dopo un colpo di clock "a vuoto", si passa alla lettura della SDRAM per permettere al counter degli indirizzi Avalon MM di riconfigurarsi per la lettura e tutto questo dà inizio al ciclo di lettura fino a quando tutti i dati richiesti non sono stati inviati, successivamente si passa ad uno stato di attesa di un comando di reset proveniente dall'interfaccia I2C.

Bisogna prestare particolare attenzione al timing di questo sottosistema: il segnale

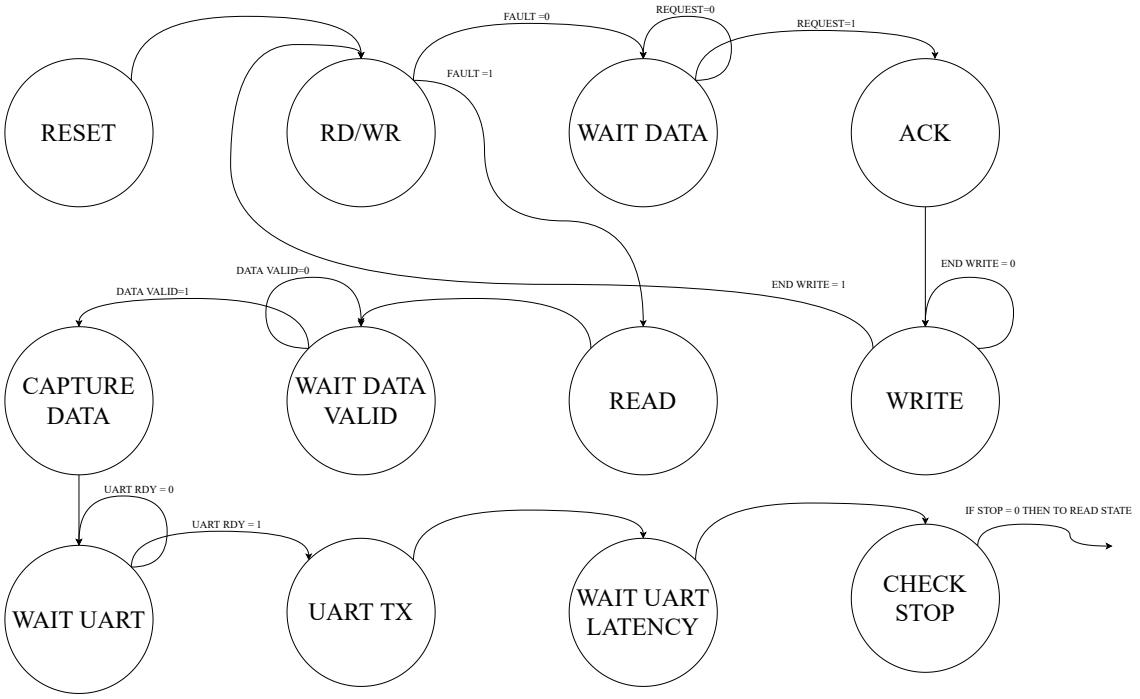


Figura 3.7: Macchina a Stati Finiti dell'Interfaccia SDRAM e UART

di clock è stato generato con un PLL presente nella Cyclon® 10 LP ed è impostato a 100 MHz, che è diverso da quello inviato al chip SDRAM. Infatti durante una transazione con la SDRAM l'indirizzo, il dato e i segnali di controllo devono essere validi per una finestra temporale durante la quale deve avvenire il fronte di salita relativo al clock della SDRAM in modo che questa possa correttamente campionarli.

Questo problema è molto evidente a frequenze alte e di conseguenza è stata calcolata per prima la differenza di fase in termini temporali necessaria per far funzionare il sistema in modo corretto e successivamente è stata utilizzata la funzione di sfasamento tra due clock presente nel PLL.

Per calcolare lo sfasamento è stato utilizzato il seguente metodo, presente nella scheda dati di Intel™per la loro IP, il metodo si divide in tre fasi:

1. calcolare di quanto è possibile anticipare il clock della SDRAM
2. calcolare di quanto il clock può essere ritardato
3. calcolare la media per posizionarsi esattamente a metà della finestra temporale individuata nei punti precedenti

Il valore di anticipo del clock è dato dal massimo tra le 3.2 e 3.3

$$Read\ Lag = t_{OH}(SDRAM) - t_{H\_MAX}(FPGA) \tag{3.2}$$

$$Write\ Lag = t_{CLK}(FPGA) - t_{CO\_MAX}(FPGA) - t_{DS}(SDRAM) \tag{3.3}$$

mentre per quanto riguarda il tempo che il clock della SDRAM può essere ritardato rispetto a quello del sistema è dato dal massimo tra le 3.4 e 3.5

$$Read\ Lead = t_{CO\_MIN}(FPGA) - t_{DH}(SDRAM) \tag{3.4}$$

$$Write\ Lead = t_{CLK}(FPGA) - t_{HZ}(SDRAM) - t_{SU\_MAX}(FPGA) \tag{3.5}$$

3.3 Interfaccia UART

Una volta che i dati vengono salvati nella SDRAM, in caso di un fault di uno dei componenti della DSA, questi vengono inviati al pc di controllo attraverso il protocollo UART, convertito in USB da un chip e letto dal computer attraverso la funzione "Virtual COM" di Windows®.

Per svolgere questa funzione è stata utilizzata, in accordo con l'azienda responsabile del progetto, una IP di Intel® per la generazione di un'architettura UART compatibile con il timing del protocollo RS-232. Questa scelta è stata presa per risparmiare tempo e concentrarsi sul restante sistema digitale e analogico.

Il clock del sottosistema è lo stesso di quello dell'interfaccia SDRAM (descritto nel paragrafo 3.2) in quanto questi sono fortemente legati dal punto di vista implementativo e funzionale. Di conseguenza è stato utilizzato lo stesso PLL presente nella Cyclone® 10 LP.

La IP utilizza l'interfaccia Avalon MM con i seguenti segnali: address, writedata, readdata, write, read e begintransfer, la funzione dei primi 5 segnali è intuibile dal loro nome inglese, mentre l'ultimo è un segnale, che al momento della scrittura di questa tesi, è considerato obsoleto da Intel®, ma comunque presente nella versione di questa IP.Il suo funzionamento consiste nella generazione di un impulso della durata di un ciclo di clock ogni volta che avviene un trasferimento in uscita o entrata dalla IP.

Il controllo di questa interfaccia è sviluppato chiaramente per essere utilizzato tramite software da un processore, nel sistema, oggetto di questa Tesi, è stato invece costruito un controller specifico: come riportato nella scheda tecnica [12], la logica responsabile per la trasmissione consiste in un registro di buffer da 7, 8 o 9 bit a seconda delle impostazioni utilizzate alla generazione dell'hardware e un registro a scorrimento con lo stesso parallelismo del precedente utilizzato per la trasmissione vera e propria dei dati.

Utilizzando l'interfaccia Avalon MM è possibile scrivere esclusivamente nel registro di buffer, mentre lo spostamento dei dati nel registro a scorrimento avviene in

automatico. Questa configurazione garantisce un registro di buffer e, di conseguenza, consente di inviare un nuovo dato all'interfaccia UART mentre questa è impegnata nella trasmissione di un altro dato, massimizzando in questo modo la quantità di dati inviabile per unità di tempo.

La logica responsabile, per la ricezione di un dato, è del tutto simile a quella per la trasmissione in quanto consiste in un registro di buffer e un altro registro a scorrimento per la lettura del dato sul bus UART: con questa modalità è possibile ricevere un dato anche quando quello precedente non è ancora stato letto.

Il controllo dello stato dell'architettura, così come la scrittura dei dati da trasmettere, viene effettuato tramite l'interazione con appositi registri, ovvero come avverrebbe in una periferica di un microprocessore o microcontrollore.

La mappa dei registri è presente in tabella 3.2. Infine all'interno della IP è presente uno stadio di sincronizzazione a due registri che introduce una latenza di due colpi di clock.

Offset	Reg Name	R/W	Description/Register Bits		
			15:8	7	6:8
0	rxdata	RO	Reserved	Receive Data	
1	txdata	WO	Reserved	Transmit Data	
2	status	RW	N.U. Status Bits	rrdy	N.U. Status Bits
3	control	RW	Not Used		
4	divisor	RW	Baud Rate Divisor (N.U.)		
5	endofpacket	RW	Reserved	End-of-Packet-Value(N.U.)	

Tabella 3.2: Tabella Registri Interfaccia UART (Courtesy of Intel®)

Il controller, per questa interfaccia, è presentato nel suo schema a blocchi in figura 3.8. Come è possibile vedere sono presenti: una macchina a stati finiti per la generazione dei segnali dell'interfaccia Avalon MM, e una coda First In First Out (FIFO) che ha la funzione di formare i pacchetti da inviare alla UART andando anche a comprimere i dati salvati nella SDRAM, eliminando il bit stuffing utilizzato durante la scrittura dei dati.

Infatti le misure provenienti dai convertitori ADC vengono rappresentate su 12 bit, mentre la larghezza di una word della SDRAM si attesta sui 16 bit, per questo motivo vengono aggiunti 4 bit a 0 per completare il dato da inviare alla SDRAM.

La necessità di eliminare il bit stuffing nasce dal fatto che la trasmissione avviene a 3 Mbaud /s e per inviare una quantità di dati utile al computer centrale occorrerebbe un tempo dell'ordine dei secondi, allora si è deciso di riportare il parallelismo dei dati da 16 a 12 bit e la FIFO si occupa di convertire il questo parallelismo da 12 a 8 bit, dimensione del pacchetto UART selezionata, e successivamente inserendo i 4 bit mancanti in un nuovo pacchetto.

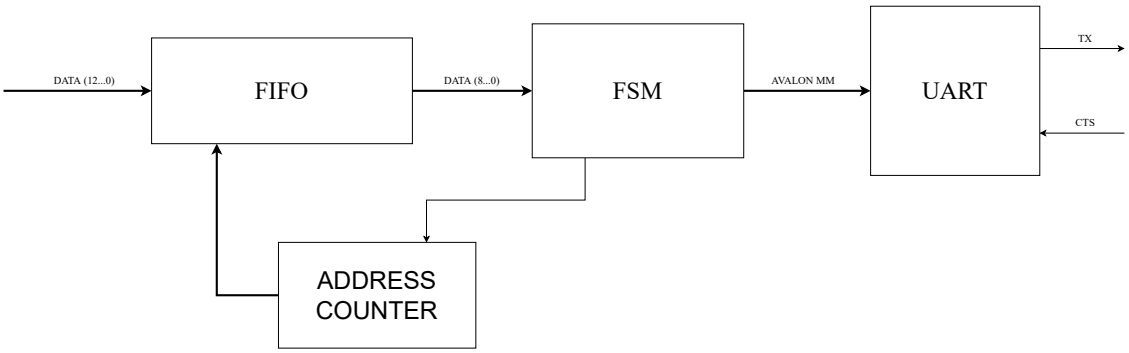


Figura 3.8: Schema a Blocchi dell'Iterfaccia UART

La macchina a stati finiti, rappresentata in figura 3.7, è in comune a quella del sottosistema SDRAM descritto nel paragrafo 3.2.

Il primo stato controlla se la UART è disponibile per inviare un dato, successivamente si passa nello stato in cui si invia, attraverso l'interfaccia Avalon MM, un dato alla UART e ulteriori due stati in cui si aspetta la latenza introdotta dal sincronizzatore presente nella IP.

Dopo aver atteso due colpi di clock è possibile controllare nuovamente se la UART è disponibile per l'invio di un nuovo dato in quanto la lettura del registro di stato rappresenta un dato aggiornato all'ultimo comando inviato e di conseguenza si cerca di inviare nuovamente un dato fino a svuotare la FIFO. Una volta che la FIFO è vuota è necessario leggere dei nuovi dati dalla SDRAM.

I dati vengono inviati al computer centrale in una struttura dati chiamata "record" che è rappresentata nella tabella 3.3, come è possibile vedere, oltre alle misure degli ADC ogni record presenta lo stato dei fault del sistema in quell'istante mentre il bit di ARC_DET proviene da un sensore che indica in formato digitale se è presente un arco voltaico nella guida d'onda a fascio descritta nel Capitolo 1.

Record Content		
ADC 1		
ADC 2		
ADC 3		
ADC 4		
ADC 5		
ADC 6		
Fault Max	Fault Min	ARC_DET

Tabella 3.3: Struttura Dati del Record

3.4 Interfaccia I2C

Il sistema FastPro, oggetto di questa Tesi, è controllato nella definizione del suo funzionamento attraverso un'interfaccia I2C con la WCC che serve anche per mantenere un livello di retrocompatibilità con il precedente sistema di M&C.

Per risparmiare tempo e concentrarsi sul resto del sistema digitale e analogico è stato deciso, in accordo con Microsis s.r.l.®, azienda responsabile del progetto, di utilizzare una IP di Intel® che traduce un'interfaccia I2C Slave in Avalon MM.

Come descritto nella scheda tecnica della IP I2C di Intel® [13], questa supporta il metodo classico per l'interfaccia di un sistema con un set di registri accessibili tramite I2C.

Per quanto riguarda la lettura di un dato viene inviato sul bus un primo byte di controllo in modalità scrittura che contiene l'indirizzo della periferica, successivamente si invia l'indirizzo del registro che deve essere letto e infine, dopo un byte di controllo in modalità lettura, la IP invia sul bus il byte per cui è stata richiesta la lettura.

Un'altra modalità di lettura è quella sequenziale: questa modalità si usa quando si intendono leggere dati ad indirizzi contigui. In questo caso viene inviato un primo byte di controllo in modalità scrittura che contiene l'indirizzo della periferica, successivamente viene inviato l'indirizzo del primo registro che si vuole leggere, inoltre viene inviato un altro byte di controllo in modalità lettura, ed infine l'interfaccia I2C Slave invia il primo byte richiesto. A questo punto il master I2C trasmette un segnale di ACK che indica allo Slave di inviare il byte all'indirizzo contiguo, la transazione termina quando il master invia il NACK e la condizione di stop.

Per quanto riguarda la scrittura viene utilizzato un sistema simile a quello della lettura sequenziale in quanto questo consiste nell'invio da parte del master del primo byte di controllo, in modalità scrittura, contenente l'indirizzo dello slave, successivamente viene inviato l'indirizzo del primo registro da scrivere e a seguire vengono inviati uno dopo l'altro i dati da scrivere in indirizzi contigui fino a quando il master non invia la condizione di stop ponendo fine alla trasmissione. Il protocollo utilizzato è riportato in figura 3.9 e in figura 3.10

Come si può vedere in figura 3.11, il sottosistema è composto dal register file contenente tutti i registri che possono essere letti o scritti dalla WCC il cui contenuto è organizzato come nella tabella 3.4, per quanto riguarda l'interfaccia, sono stati usati semplicemente dei registri per ritardare i comandi del corretto numero cicli di clock: tutto questo rende il sottosistema meno complesso e potenzialmente più performante.

Random Address Read

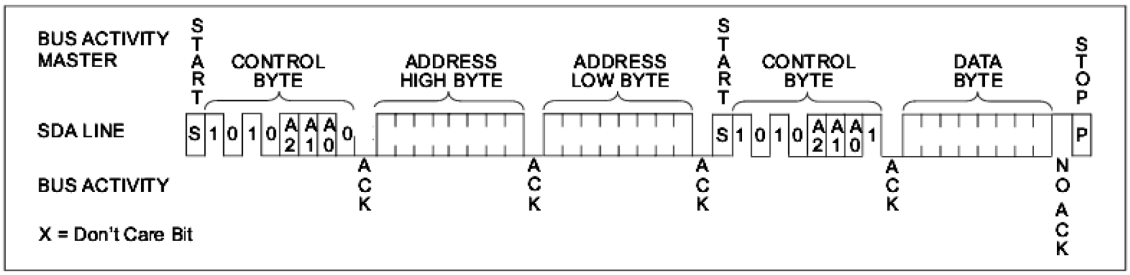


Figura 3.9: I2C Timing Diagram Lettura (Courtesy of Intel®)

Write Operation

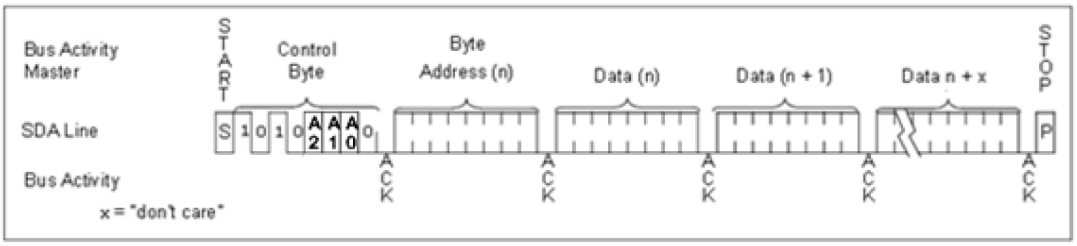


Figura 3.10: I2C Timing Diagram Scrittura (Courtesy of Intel®)

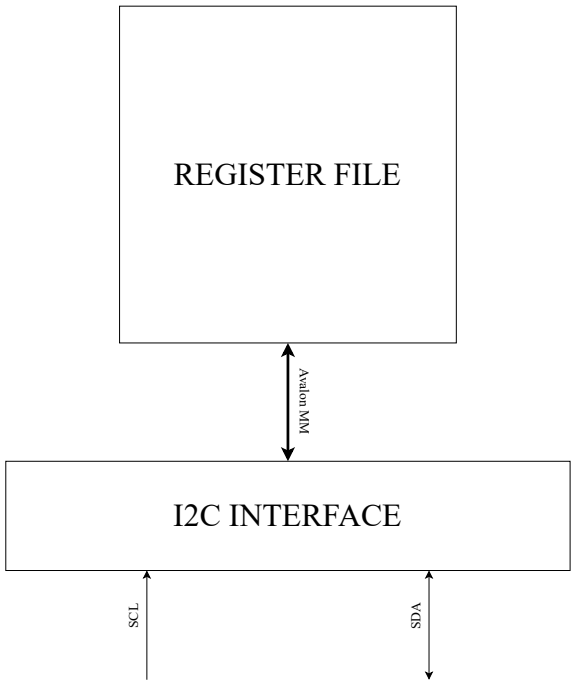


Figura 3.11: Diagramma a Blocchi Interfaccia I2C

Address	Content
0	ADC 1
...	...
10	ADC 6
12	TH MAX 1
...	...
22	TH MAX 6
24	TH MIN 1
...	...
34	TH MIN 6
36	FAULT MAX
37	FAULT MIN
38	STATUS
39	COMMANDS

Tabella 3.4: Tabella Mappa Registri I2C

3.5 Fault Detection

L'altra funzione portante del sistema FastPro è quello della generazione dei segnali di fault e di inibizione dei componenti che sono in uno stato di funzionamento anomalo. La generazione di questi segnali deve essere assolutamente precisa e stabile, ma allo stesso tempo veloce. Per questo è stata implementata l'architettura in figura 3.12.

I dati dei registri degli ADC vengono costantemente confrontati con delle soglie attraverso dei comparatori di maggioranza o minoranza a seconda se le soglie sono massime o minime. Una volta che una soglia non è rispettata viene messo ad 1 logico il corrispondente bit del registro di stato della FastPro e si provede ad inviare un segnale di SET ad un flip flop set-reset in modo che questo porti l'uscita ad 1 che è responsabile della generazione del segnale di fault interno al sistema e simultaneamente ad un altro flip flop set reset che asserisce il segnale di inhibit del componente interessato secondo lo schema in tabella 3.5 in cui ciascun flip flop viene attivato dalla funzione OR dei segnali presenti nella colonna corrisponde.

Inhibit Type	ADC NumberTriggering Faults
IPA Inibit	3(max), 5(max), ARC_DET
BPS Inhibit	1(max), 2(max), 4(max), 5(max), 6(min)

Tabella 3.5: Tabella Numero degli ADC corrispondente ai segnali di Inhibit

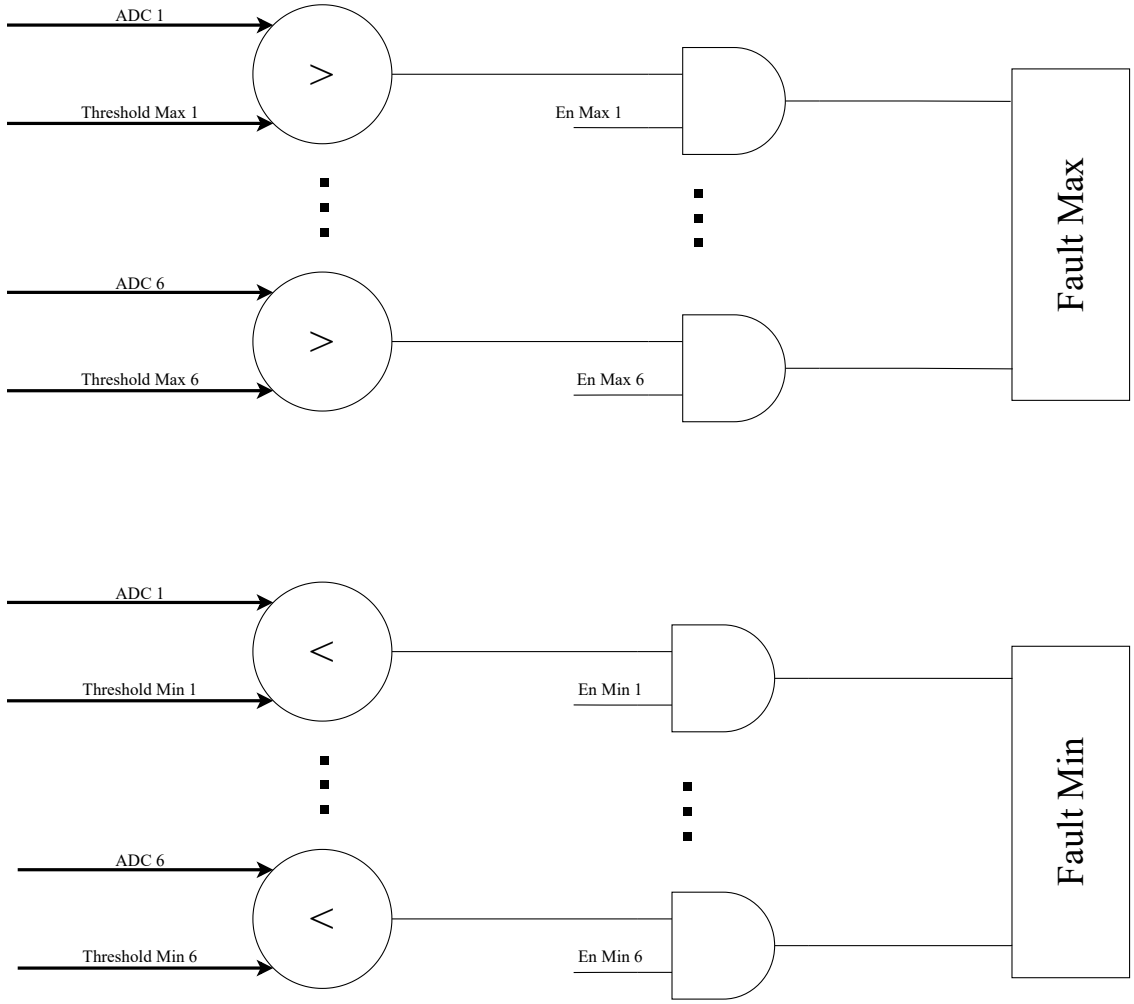


Figura 3.12: Diagramma a Blocchi del Sistema di Fault Detection

Dal confronto con lo schema in figura 3.12 e quello in tabella 3.5 è possibile vedere come non tutte le soglie sono utilizzate: si è voluto, però, mantenere un'architettura più ampia di quella necessaria in modo tale da avere la possibilità di aumentare le prestazioni su richiesta, senza dover modificare il sistema digitale in modo sostanziale.

Capitolo 4

Collegamento e Verifica del Sistema Digitale

In questo capitolo vengono trattati: in primo luogo i temi della sincronizzazione della trasmissione tra i componenti che vedono il clock a 160 MHz e quelli che sono sotto il regime a 100 MHz, in secondo luogo viene esposto il tema della verifica funzionale dei vari sottosistemi digitali descritti nel capitolo 3.

In una prima fase verranno esposti i principi teorici su cui è basato il componente di sincronizzazione, mentre successivamente verrà descritta e motivata la scelta dell'architettura utilizzata e della sua implementazione.

Nel secondo paragrafo verrà descritta la fase di verifica dei vari sottosistemi necessaria per assicurare che il comportamento dell'hardware sviluppato rispecchi pienamente le specifiche di progetto: verrà descritto il sistema di testing utilizzato e verranno illustrati i procedimenti seguiti per garantire una verifica assolutamente affidabile dal punto di vista comportamentale e dal punto di vista di un'analisi di timing statica.

4.1 Sincronizzatore

Molti sistemi digitali hanno la necessità di comunicare con diverse periferiche esterne, queste possono avere regimi di clock completamente diversi da quelli del sistema principale.

Questa differenza tra i segnali di sincronizzazione può portare ad errori dovuti al fatto che gli input, provenienti da un regime di clock, e il clock del ricevitore commutano in modo totalmente scollegato a livello temporale e di conseguenza si verificano stati di metastabilità nei componenti sequenziali.

Di conseguenza è importante che questo stato incerto evolva in un tempo breve e fissato verso uno stato stabile, indipendentemente dal fatto che quest'ultimo sia logicamente corretto o meno.

Appare, quindi, evidente la necessità di avere un componente con funzione di arbitro con il compito di decidere quale evento è accaduto per prima: ovvero la commutazione del clock o del segnale in ingresso.

Inoltre è importante sottolineare che risulta impossibile rendere un sistema di questo tipo completamente immune da errori in quanto i principi teorici e pratici su cui questo è basato sono per loro costruzione esenti da un qualsiasi controllo sulla coerenza del dato.

La funzione di arbitro può essere svolta a pieno da un flip flop, il cui circuito interno di base è descritto in figura 4.1: come si può vedere se i due inverter fossero uguali e se il dato venisse campionato esattamente a metà della tensione il tempo per uscita dalla metastabilità risulterebbe infinito, ma visto che un componente reale ha sempre delle diversità questo si traduce in tempi finiti, ma molto lunghi.

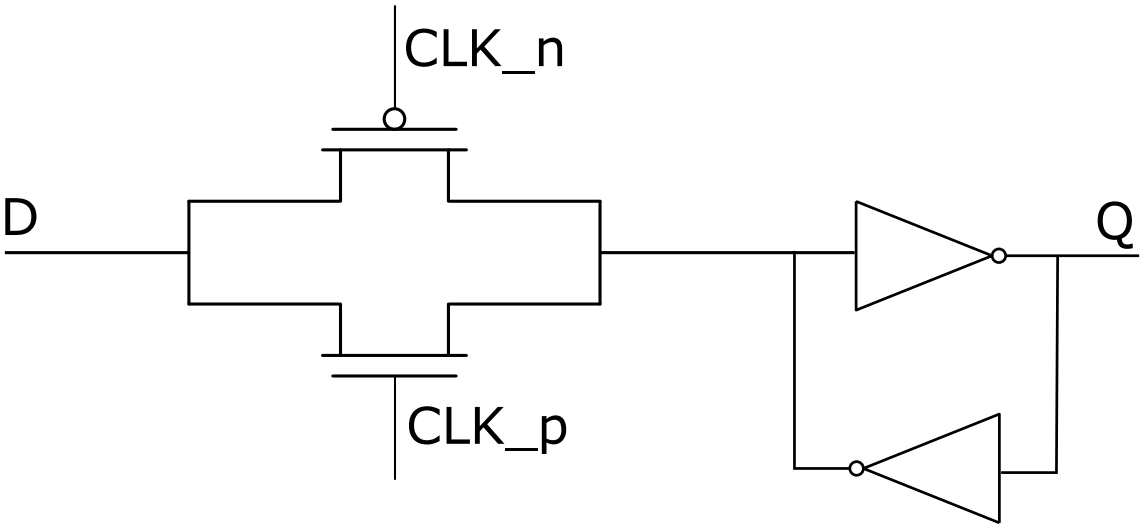


Figura 4.1: Circuito Arbitro

Inoltre è noto in letteratura come all'aumentare del tempo che intercorre tra il momento del campionamento al momento in cui il segnale deve essere valido, diminuisce la probabilità che questo sia ancora in uno stato instabile.

Dal momento che è impossibile avere un sistema completamente esente da errori, è importante calcolare la frequenza con cui questi si verificano e individuare la modalità per diminuirla fino a renderla trascurabile per l'applicazione in cui è utilizzata. Per fare questo si utilizzerà il diagramma temporale in figura 4.2, in cui è presentato un fronte di salita di un segnale, ma le modalità sono analoghe per il fronte di discesa.

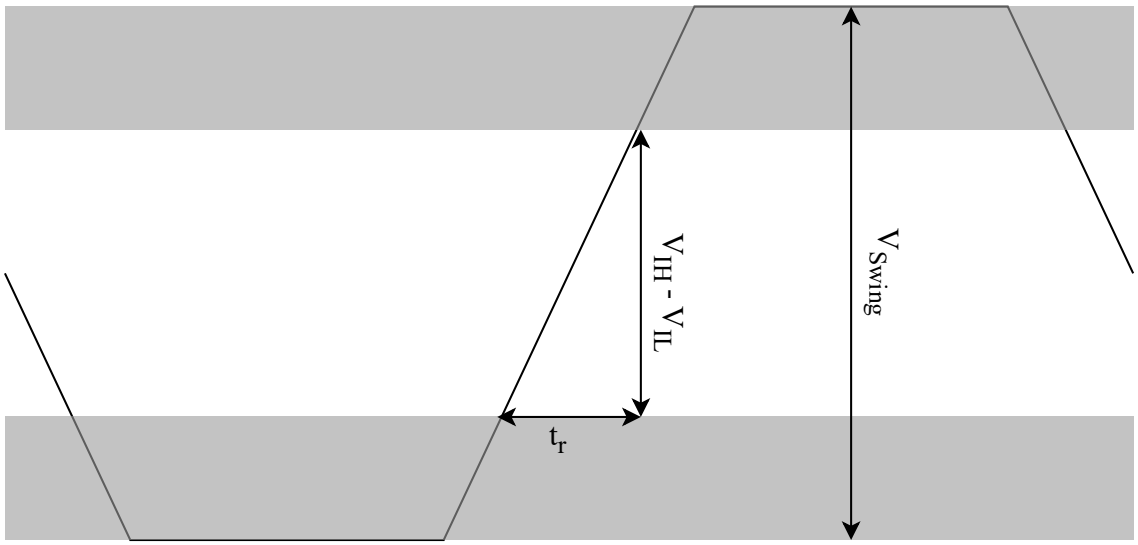


Figura 4.2: Timing Diagram per Calcolare il MTBF

Se il segnale viene campionato in un punto in cui è fortemente stabile, la probabilità di errore è la stessa di un sistema puramente sincrono, il periodo in cui il segnale comporta uno stato metastabile, se campionato, è quello in cui, durante la transizione basso alto, il segnale attraversa la zona proibita. Da cui segue la seguente equazione:

$$N_{sync}(t) = \frac{(V_{IH} - V_{IL}) \times e^{-t/\tau}}{V_{swing}} \times \frac{t_r}{T_{signal} \times T_{sample}} \tag{4.1}$$

dove $N_{sync}(t)$ è il numero medio di errori quando si campiona all'istante di tempo t, τ è un fattore che dipende dalla tecnologia utilizzata per realizzare il circuito, t_r è il tempo di salita, T_{signal} e T_{sample} sono rispettivamente il periodo di commutazione del segnale di ingresso e la frequenza di campionamento.

Il parametro che viene usato per caratterizzare la probabilità di errore è chiamato Mean Time Before Failure (MTBF) ed è l'inverso di $N_{sync}(t)$ come si può vedere dalla seguente equazione, secondo un Application Note di Intel® [14] :

$$MTBF = \frac{e^{t_{MET}/C_2}}{C_1 \times f_{CLK} \times f_{DATA}} \tag{4.2}$$

dove t_{MET} è il tempo disponibile per il risolversi dello stato metastabile ed è corrispondente ad un tempo di propagazione, C_1 e C_2 sono parametri costanti che dipendono dalla tecnologia scelta per la realizzazione di un circuito integrato come le FPGA e dalle sue particolari condizioni di operazione, f_{CLK} è la frequenza di clock del sistema che deve campionare il segnale e infine f_{DATA} è la frequenza con cui commuta il segnale asincrono in ingresso.

Dall'equazione 4.2 è possibile notare come aggiungendo più sincronizzatori in cascata, ovvero più registri, è possibile ridurre in modo esponenziale il MTBF in quanto si aumenta il parametro t_{MET} perché il tempo di propagazione da un registro ad un altro è del tutto utilizzabile per risolvere lo stato di metastabilità. Quando si utilizza un approccio di questo tipo, uno dei più rilevanti aspetti negativi consiste nell'introduzione di una latenza al sistema dovuta ai registri di sincronizzazione inseriti.

Il circuito realizzato per la FastPro fa uso di bus asincroni per scrivere i dati nella SDRAM: è noto in letteratura come semplicemente inserire una serie di sincronizzatori in parallelo, uno su ogni bit che compone il bus, non porta ad avere un risultato altrettanto affidabile.

Risulta necessario introdurre sistemi più complessi come un protocollo di handshake: i due esponenti principali sono il protocollo a 4-fasi e il più complesso protocollo a 2-fasi.

I protocolli sono molto simili tra di loro, ma differiscono nell'interpretazione dei segnali di scambio, per cui una prima parte di trattazione viene esposta una sola volta in modo da avere una forma del testo più concisa e priva di ripetizioni.

I protocolli utilizzano due segnali di controllo chiamati request e acknoledge che vengono sincronizzati come descritto precedentemente, e di conseguenza sono gli unici segnali di tutta la comunicazione che vengono campionati più volte e per cui viene risolta la metastabilità.

Questi segnali sono anche quelli che introducono la latenza aggiuntiva all'intero sistema. I segnali di dato, invece non vengono sincronizzati attraverso dei registri, ma, dal momento che sono presenti dei segnali di controllo, è possibile sapere che il bus dati è completamente stabile al momento del campionamento in ricezione.

Il protocollo a 4-fasi, riassunto in figura 4.3, ha il seguente comportamento per quanto riguarda i segnali di handshake: il trasmettitore, ogni qual volta che ha un dato da inviare al ricevitore, asserisce il segnale di request, il ricevitore, una volta ricevuto questo comando, può in qualsiasi momento successivo campionare i dati sapendo che questi sono fissi e di conseguenza non ci sono problemi di stabilità.

Una volta campionati i dati dal ricevitore, questo invia il segnale di acknoledge al trasmettitore, in modo da comunicare che è possibile inviare un altro dato, a questo punto il ricevitore abbassa il segnale di request e si prepara ad un eventuale successivo ciclo di trasmissione, infine il ricevitore abbassa il segnale di acknoledge.

Questo metodo ha un ottimo grado di precisione ed è meno complesso da implementare e di conseguenza possono essere utilizzati sistemi FPGA più semplici ed economici, a sfavore ha il difetto di essere abbastanza lento in quanto è necessario, tutte le volte, portare al livello logico basso i segnali di request ed acknoledge, da cui il termine 4-fasi.

Per ovviare a questa limitazione viene introdotto il protocollo a 2-fasi: in questo caso si utilizza una codifica dei segnali di controllo diversa, ovvero il fronte di discesa

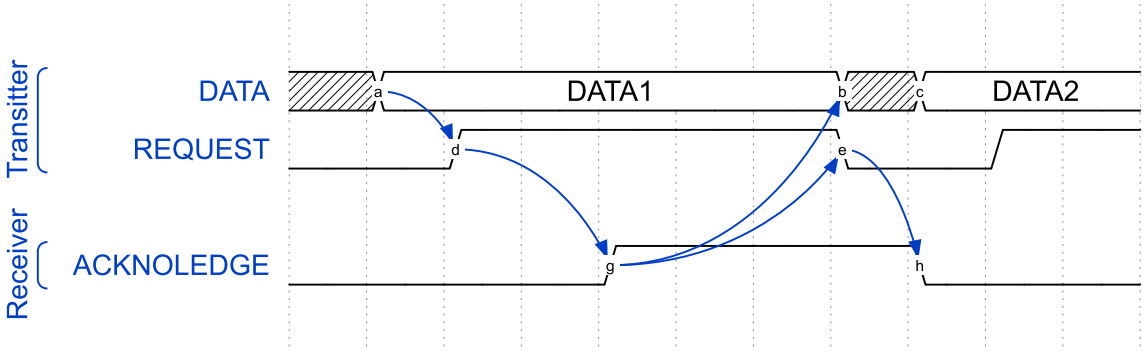


Figura 4.3: Timing Diagram del Prtocollo di Handshake a 4 fasi

o salita è l'evento scatenante per la fase successiva, ad esempio se il request ha un fronte e ci si trova nella condizione in cui è necessario ricevere un nuovo dato, allora il ricevitore campionerà il dato.

Il problema di queste due modalità di protocollo consiste nel fatto che è rilevante lo skew tra i segnali di controllo e i segnali di dato, infatti, ad esempio, il segnale di request deve ritardare i dati di almeno un tempo di set-up dei flip-flop in modo che questi possano campionare correttamente il dato in ingresso.

Per la particolare applicazione, oggetto di questa Tesi, sia la velocità a cui viaggiano i dati, sia il fatto di essere all'interno di un circuito integrato, che presenta differenze di ritardi trascurabili, sia perché i dati sono in uscita a dei flip-flop e quindi non hanno tempi di propagazione dovuti a percorsi combinatori che possono introdurre skew sui vari bit dei dati, si è potuta trascurare la differenza dei ritardi tra i vari bit che compongono il dato.

Il protocollo implementato nel sistema hardware digitale, oggetto di questa Tesi, è stato quello a handshake a 4 fasi, in quanto richiede il minor numero di risorse per essere implementato e le frequenze di trasmissione permettono di assorbire la latenza aggiuntiva caratteristica di questo protocollo rispetto a quello a 2 fasi.

In figura 4.4 è riassunta l'architettura di implementazione di questo sotto-sistema all'interno del progetto sviluppato in questa Tesi: come è possibile vedere due flip-flop set-reset sono utilizzati per generare i segnali di request e ack, mentre i dati escono da dei registri nel dominio di clock a 160 MHz ed entrano in altri registri che campionano secondo il regime di clock di 100 MHz; i segnali di set per i registri che generano i comandi di controllo della trasmissione sono presi dalle macchine a stati finiti del sistema SPI, descritto nel paragrafo 3.1 per quanto riguarda il request e dalla macchina a stati finiti del sistema SDRAM descritto nel paragrafo 3.2.

Per quanto riguarda la sincronizzazione dei segnali di handshake è stato implementato un sincronizzatore che utilizza due stadi flip-flop in cascata, per misurare il MTBF è possibile impostare delle direttive di compilazione apposite.

Inoltre questa operazione consente anche di eliminare i messaggi di violazione del

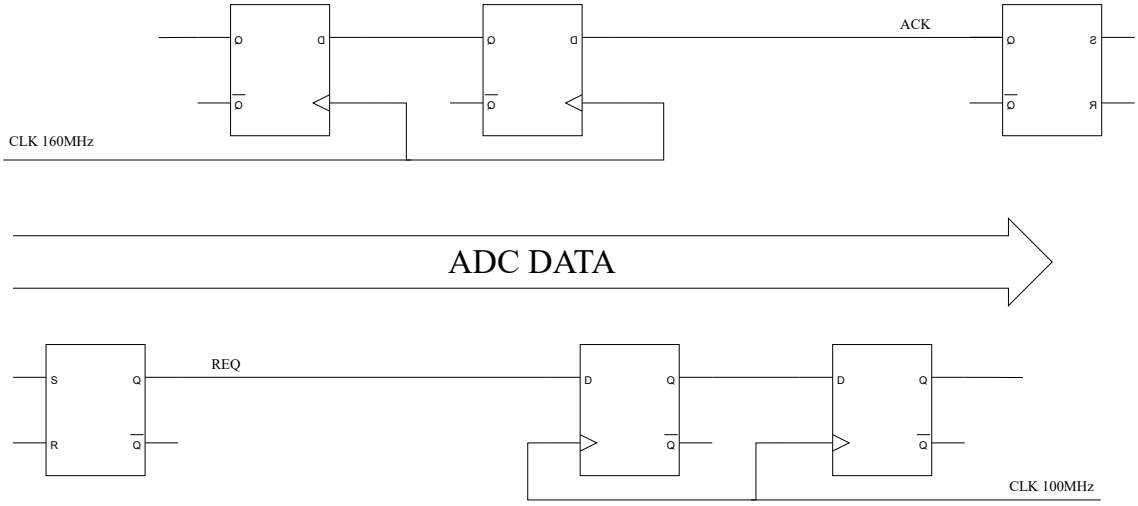


Figura 4.4: Diagramma a Blocchi Sincronizzatore

tempo di hold nel report della timing analisys e il risultato riportato dal software di compilazione stima un valore di MTBF maggiore di 1 Miliardo di anni, ovvero il limite massimo calcolabile dal programma utilizzato.

4.2 Functional Verification

Al fine di garantire un corretto funzionamento del sistema digitale, è stata effettuata una fase di verifica del completo rispetto delle specifiche per ogni componente descritto nel capitolo 3, utilizzando dei modelli funzionali ideali in formato puramente software, la cui implementazione è stata eseguita tramite l'impiego del linguaggio di descrizione hardware VHDL.

Infatti ogni componente è stato sottoposto ad una fase di Test Bench in cui sono stati forniti degli input all'unità sotto esame, ovvero uno dei sottosistemi, e successivamente i risultati in uscita sono stati confrontati con quelli di un modello ideale realizzato in software, creando un file di log in cui venivano riportati i risultati in uscita dall'unità sotto test e l'inserimento di assert in simulazione che permettevano di individuare direttamente nell'interfaccia del simulatore, in quale istante temporale si verificava l'errore.

Per prima cosa viene analizzato il processo di verifica dell'interfaccia SPI descritta nel paragrafo 3.1: è stato codificato in linguaggio VHDL un modello di interfaccia SPI slave in grado di rispondere ai comandi della unità sotto test, e fornire un dato in uscita; questo test comprendeva il corretto pilotaggio del ADC da parte del componente digitale sviluppato e il corretto campionamento e de serializzazione del dato ricevuto.

Infatti è stato realizzato in VHDL un'interfaccia seriale che, una volta ricevuto il fronte di discesa del segnale di Chip Select iniziava ad inviare i dati in forma seriale cambiando bit ad ogni falling edge del segnale SCLK. I dati inviati sul bus, non essendo possibile replicare in VHDL il comportamento analogico del convertitore ADC, sono stati generati in modo casuale attraverso le apposite funzioni del linguaggio di descrizione hardware utilizzato.

Inoltre il valore del dato inviato sul bus è stato considerato trascurabile, purché fosse ricevuto correttamente; un fattore che non è stato trascurato, invece, è stato quello dei tempi di ritardo dei dati e del bus in alta impedenza: sono stati inseriti nel modello di simulazione creato tutti i ritardi e le caratteristiche di timing da rispettare, presenti nella scheda dati del convertitore AD 7356 [7], questo è stato accompagnato da appositi messaggi di errore in caso di violazione di queste specifiche.

La verifica formale è stata condotta utilizzando, oltre ai messaggi di errore presenti nell'interfaccia grafica del simulatore generati dagli assert, anche attraverso un confronto tra due file: uno di input e uno di output, controllando che i valori dei dati che questi due file contenevano fossero identici: una schermata contenente delle forme d'onda della simulazione è presente in figura 4.5

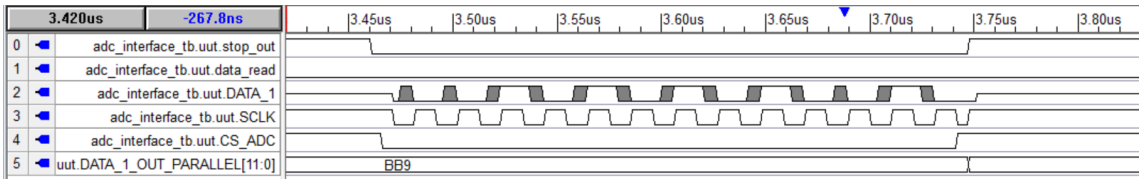


Figura 4.5: Forme d'Onda della Simulazione dell'interfaccia SPI

Successivamente è stata analizzata l'interfaccia tra la FPGA e la SDRAM descritta nel paragrafo 3.2: è stato generato un test funzionale che verificasse che l'interfaccia Avalon MM venisse rispettata in tutte le sue specifiche. Non è stato invece possibile generare un modello, né di timing, né RTL per la SDRAM in quanto il modello insito nella IP di Intel richiedeva una quantità di memoria superiore a quella gestibile dal software di simulazione in relazione alla licenza utilizzata.

Inoltre l'unico modello di timing presente su internet era scritto nel linguaggio di descrizione hardware Verilog, mentre il progetto del sistema digitale di questa Tesi è stato realizzato in VHDL e pertanto la licenza del programma di simulazione impediva di utilizzare due linguaggi contemporaneamente.

Conseguentemente è stato creato un test bech in VHDL che invia all'unità sotto analisi 6 word da 12 bit ciascuna il cui dato è stato generato in modo casuale utilizzando le funzioni specifiche del linguaggio di descrizione hardware e sono stati controllati i valori in uscita all'interfaccia Avalon MM, generando due file di log: uno

con i dati generati dal test bench e l'altro con i dati scritti direttamente dall'unità sotto test.

Inoltre sono stati creati dei messaggi di errore in caso di un comportamento anomalo in modo da poter individuare in modo efficace l'istante in cui questo avviene attraverso l'interfaccia grafica del programma di simulazione.

Per quanto riguarda la verifica del sotto-sistema completo comprendente sia la FPGA che la memoria SDRAM è stata utilizzata una scheda di sviluppo TerasIC® DE1-SoC contenente una Intel® Cyclone® FPGA che, a sua volta, conteneva i componenti necessari alla realizzazione di un test del codice VHDL scritto per il sistema del paragrafo 3.2, in figura 4.6 si mostra una schermata del software di simulazione contenente le forme d'onda prese in esame.

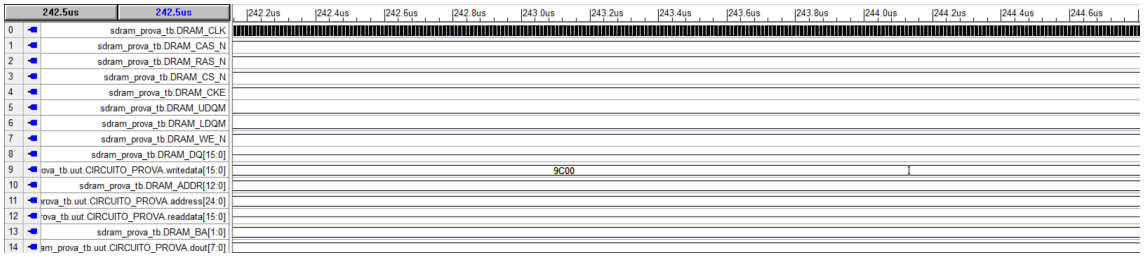


Figura 4.6: Forme d'Onda della Simulazione dell'interfaccia SDRAM

Il test è stato svolto utilizzando un contatore che permetteva simultaneamente la generazione dell'indirizzo di scrittura e il dato da scrivere, che corrispondeva ad un sottoinsieme di bit dell'indirizzo. Dopo una scrittura completa della memoria, si passava ad una fase di lettura in cui il contatore veniva inizializzato, in modo da avviare la lettura dal primo indirizzo scritto.

Infatti è stata letta ogni word della memoria SDRAM e successivamente confrontata con il valore precedentemente scritto: in caso di una mancata corrispondenza tra questi due valori, veniva acceso un LED presente sulla scheda di sviluppo e veniva fatto suonare un cicalino, anch'esso connesso alla DE1-SoC.

Dal momento che, in un collegamento con una SDRAM, è possibile avere un comportamento corretto, in alcuni casi, anche se il circuito presenta problemi di timing relativi all'individuazione della finestra di data valid, come spiegato nel paragrafo 3.2, è stato ripetuto il test ininterrottamente per circa 30 minuti.

Successivamente è stato verificato il modulo di comunicazione UART descritto nel paragrafo 3.3: la verifica è stata svolta utilizzando un test bench in cui venivano forniti in ingresso al circuito dei dati generati in modo casuale da trasmettere sul bus. Per prima cosa è stato osservato il comportamento dell'interfaccia Avalon MM utilizzata per pilotare la IP di comunicazione, successivamente è stato esaminato il dato in uscita dall'interfaccia UART, configurando la simulazione, in modo tale, che il baud rate dell'interfaccia fosse accelerato di diverse unità in modo da poter

occupare un tempo minore, pur mantenendo un livello di verifica assolutamente valido, e di conseguenza utilizzare meno risorse computazionali.

Dal momento che non era possibile creare un sistema di verifica, in modo rapido ma allo stesso tempo rigoroso ed efficace, motivato dal fatto che si sarebbe resa necessaria la creazione di un modello software per la ricezione e interpretazione del bus UART, si è eseguito un test di funzionamento utilizzando la scheda di sviluppo DE1-SoC collegata ad un pc attraverso una scheda contenente un convertitore di protocollo UART a USB, ovvero la UM232H-B.

Attraverso una semplice interfaccia software utilizzando una API del produttore del convertitore è stato possibile creare una rapida interfaccia seriale UART che permetteva di salvare in modo sicuro i dati ricevuti dal sistema digitale: in questo modo, utilizzando un file di log che, attraverso un confronto con un file contenente i dati inviati dal sistema sotto esame eseguito da uno scrip in Python in grado di evidenziare eventuali errori di trasmissione, è stato possibile verificare che i dati venissero inviati correttamente.

Un ultimo test sui sotto-sistemi descritti nei paragrafi 3.2 e 3.3 è stato quello dell'unione di questi due componenti: la verifica si è svolta utilizzando la DE1-Soc e la scheda UM232H-B sopracitate, andando a salvare sulla SDRAM un'insieme di dati prefissati.

Successivamente, simulando un segnale di fault attraverso un timer implementato all'interno del sistema, si è scaricato il contenuto della memoria attraverso l'interfaccia UART, infine si sono confrontati i dati inviati dalla UART con quelli teorici prefissati.

Infine è stato verificato il funzionamento del banco di registri accessibile dall'esterno tramite I2C, descritto nel paragrafo 3.4: per la fase di testing di questa interfaccia è stato utilizzato un approccio differente rispetto a quello usato per gli altri sotto-sistemi, infatti la fine dello sviluppo del codice relativo a questo componente è coincisa con l'inizio della fase di aggiornamento del firmware della WCC.

Di conseguenza la fase di testing è stata fortemente collegata alla parte software: infatti è stato ultimato il collegamento alla scheda madre WCC ed è stato utilizzato un oscilloscopio in grado di decodificare il protocollo seriale I2C e rappresentarne agevolmente il contenuto su uno schermo. Attraverso questa funzionalità dello strumento, è stato possibile verificare la scrittura, la lettura e, in generale, il corretto funzionamento del sistema andando ad inviare comandi dal master, ovvero la WCC, e osservando le risposte della FastPro.

Capitolo 5

Collaudo del Sistema

Al termine del capitolo 4, avendo stabilito le norme di sicurezza per il sistema FastPro, ora è consequenziale trattare le norme e le procedure per assicurare il collaudo del sistema il più possibile rispondente a canoni di assoluta sicurezza.

Di conseguenza nella prima parte di questo capitolo viene trattato un test effettuato sul circuito di conversione da analogico a digitale che verifica le prestazioni in termini di precisione della misura e di distorsione in frequenza.

Infatti prima di convalidare l'ADC AD 7356 sono state eseguite delle verifiche su un sistema prototipo in cui si valutavano le sue prestazioni in modo da essere sicuri di avere un sistema rispondente alle specifiche richieste.

Per prima cosa si è proceduto con un test con un ingresso costante ed è stato valutato quanto la misura cambiasse valore, successivamente è stato utilizzato un ingresso sinusoidale a 1 MHz per la quale è stato condotto lo studio in frequenza attraverso una Fast Fourier Trasform (FFT) evidenziando distorsione e rumore di fase.

Nella seconda parte del capitolo viene esposto il processo di collegamento del sistema FastPro con la WCC e vengono descritte le fasi di collaudo che sono state necessarie per la totale validazione del progetto presentato in questa Tesi, essendo delle verifiche reali e non semplici simulazioni teoriche al computer.

5.1 Test Performance ADC

Prima di convalidare l'utilizzo del convertitore AD 7356, è stato svolto un test per verificare se il componente, insieme al suo necessario circuito di signal conditioning, rispondesse a pieno alle caratteristiche minime richieste al fine di garantire un corretto monitoraggio e controllo del funzionamento della DSA.

Conseguentemente c'è stata la verifica di due comportamenti diversi con due test: il primo test è stato quello di verifica della precisione e della stabilità della misura, ovvero ad un ingresso a tensione costante, si è studiata l'adeguatezza della risposta

del sistema di misura; il secondo processo di verifica verteva sul comportamento del circuito stimolato da una sorgente a tensione variabile che, per semplicità di calcolo e verifica delle performance, è stata scelta come tensione sinusoidale.

La configurazione utilizzata per il primo test è stata quella rappresentata in figura 5.1, che prevedeva l'utilizzo di un alimentatore BK PRECISION 9129B per la generazione di un segnale costante, caratterizzato da un'elevata precisione nel valore e per l'alimentazione della parte analogica del circuito.

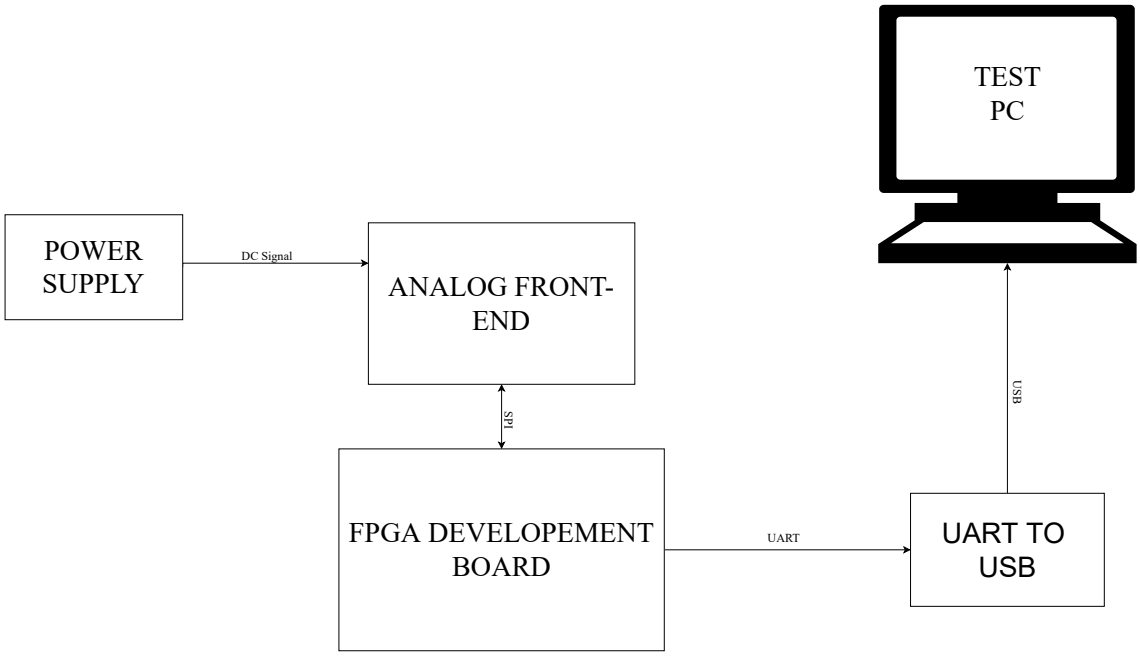


Figura 5.1: Set up della modalità di Test ADC

Inoltre è stato realizzato nella sua interezza il circuito da validare, montando i componenti richiesti su una millefori ed effettuando tutti i collegamenti richiesti.

Successivamente è stata utilizzata la scheda di sviluppo DE1-SoC, enunciata nei capitoli precedenti, per l'implementazione della parte del sistema digitale necessaria al test, ovvero l'interfaccia SPI, la SDRAM per salvare i dati e l'interfaccia UART per comunicarli ad un PC: il metodo seguito consiste nel salvare all'interno della memoria i dati prelevati dall'ADC. Questo metodo è stato scelto in quanto consentiva di ridurre la velocità di trasmissione dei dati al PC rispetto ad un loro invio in tempo reale, soluzione che avrebbe richiesto un'interfaccia parallela in quanto la velocità da raggiungere sarebbe stata di 60 Mb /s.

Come si può vedere dalla figura 5.1 la configurazione utilizzata per questo tipo di test è molto simile al sistema descritto nel capitolo 3, infatti è stato implementato utilizzando lo stesso codice VHDL leggermente modificato in modo tale da essere adattato alla situazione di utilizzo.

Mentre nel sistema FastPro il segnale di fault è asserito nel caso in cui viene superata una soglia, in questo caso si è utilizzato lo stesso metodo con l'unica differenza che il fault viene messo ad 1 logico dopo che vengono salvati all'interno della memoria SDRAM un numero prefissato di campioni che in questo caso ammontano a 65000.

Per la comunicazione con il computer è stata utilizzata la scheda UM232H-B che converte il protocollo UART in quello più diffuso ed usato per collegare periferiche hardware ai PC, ovvero USB.

Dal momento che, per garantire un tempo di trasmissione totale esiguo, sono stati applicati bit rate elevati, i software commerciali faticavano ad acquisire in modo completo tutti i campioni inviati attraverso l'interfaccia USB, per questo motivo si è ricorsi ad una soluzione custom grazie all'impiego dell'insieme di funzioni presenti nella API di FTDI D2XX, la quale ha permesso una trasmissione assolutamente priva di errori o perdita di dati.

Questo programma, sviluppato attraverso il linguaggio di programmazione C, consiste nella seguende sequenza di operazioni:

1. Eseguire una scansione dei dispositivi USB collegati
2. Aprire una comunicazione con il dispositivo di interesse
3. Resettare il dispositivo in modo da svuotare il buffer USB associato
4. Impostare i parametri della comunicazione USB tali da garantirne il corretto funzionamento
5. Selezionare il baud rate della trasmissione UART
6. Leggere il numero di dati desidarato dal bus

Successivamente i dati ricevuti dal PC sono stati elaborati attraverso degli script nel linguaggio di programmazione Python, che utilizza delle funzioni della libreria "matplotlib" e "numpy".

In primis i dati vengono letti da un file generato al momento della ricezione, successivamente vengono organizzati in un array per generare in modo corretto un istogramma rappresentativo e infine vengono organizzati in modo tale da essere salvati su un file facilmente interpretabile da altri software commerciali per la realizzazione di grafici. Il risultato del test è riportato in figura 5.2.

Il secondo test eseguito prevedeva l'utilizzo di una sorgente sinusoidale e verifica il comportamento del sistema nella condizione di uno stimolo variabile per andare ad evidenziare comportamenti di distorsione o rumore di fase che rendono la misura di segnali, il cui valore cambia in modo molto rapido, poco fedele e di conseguenza porta ad un possibile errore per quanto riguarda le funzioni di controllo della FastPro.

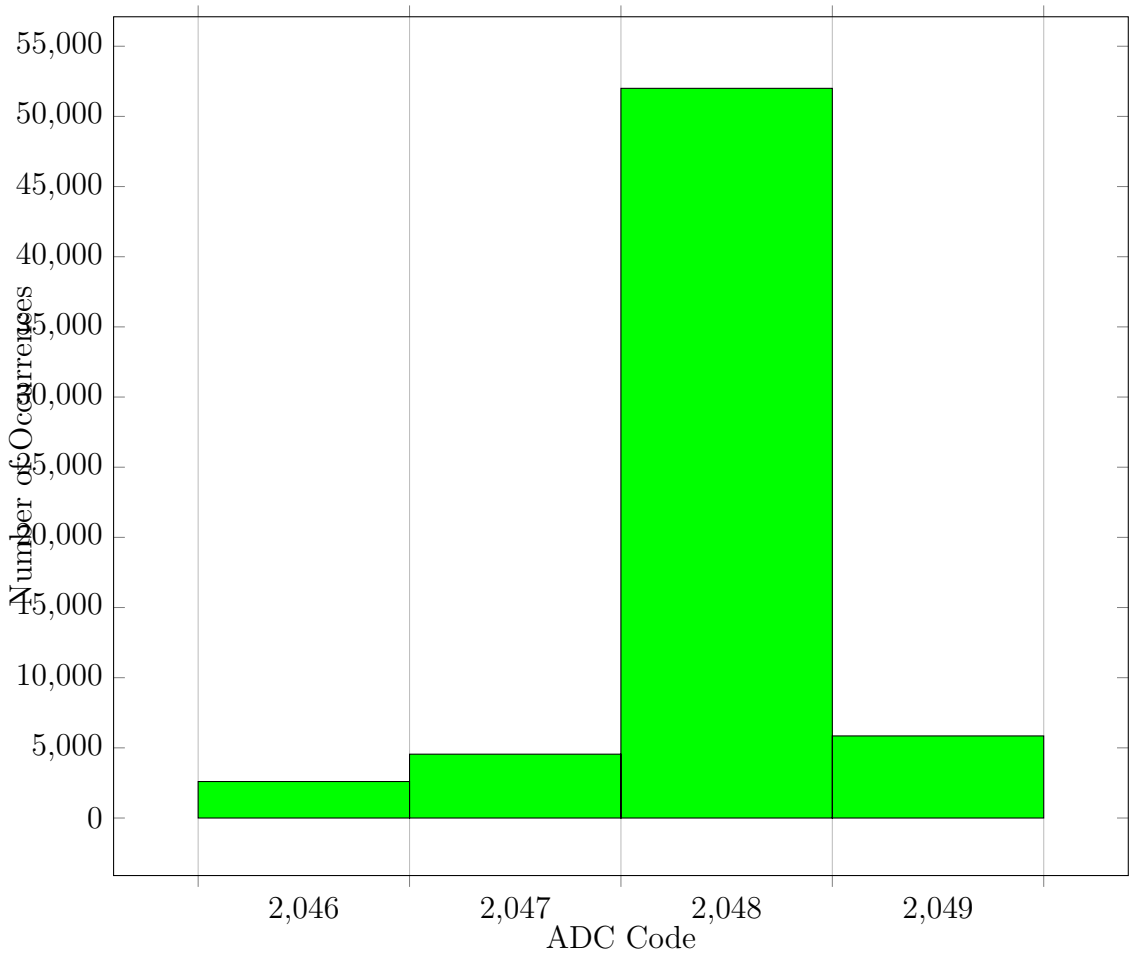


Figura 5.2: Numero di occorrenze di un codice ADC con ingresso costante (65000 Campioni)

Questa verifica è stata effettuata con modalità simili a quelle utilizzate per il test descritto precedentemente: su una millefori è stato mantenuto lo stesso circuito, mentre per il segnale di ingresso è stato utilizzato un generatore di funzioni arbitrarie modello BK PRECISION 4054B.

Il punto in cui i due test descritti in questo paragrafo divergono sostanzialmente è nell'elaborazione dei dati, una volta raccolti: se per la prima fase di verifica era bastato un istogramma in questa sono stati utilizzati necessariamente meccanismi più complessi, infatti è stato necessario impiegare una funzione che implementasse la FFT.

Di conseguenza è stata utilizzata la libreria "numpy" propria del linguaggio di programmazione Python e sono stati effettuati i seguenti passaggi:

1. Calcolo della FFT attraverso la funzione della libreria

2. Calcolo del numero di campioni su cui è stata calcolata la FFT
3. Calcolo della durata temporale del segnale di ingresso
4. Calcolo dei valori di frequenza per l'asse delle ascisse del grafico

Il risultato è presente in figura 5.3

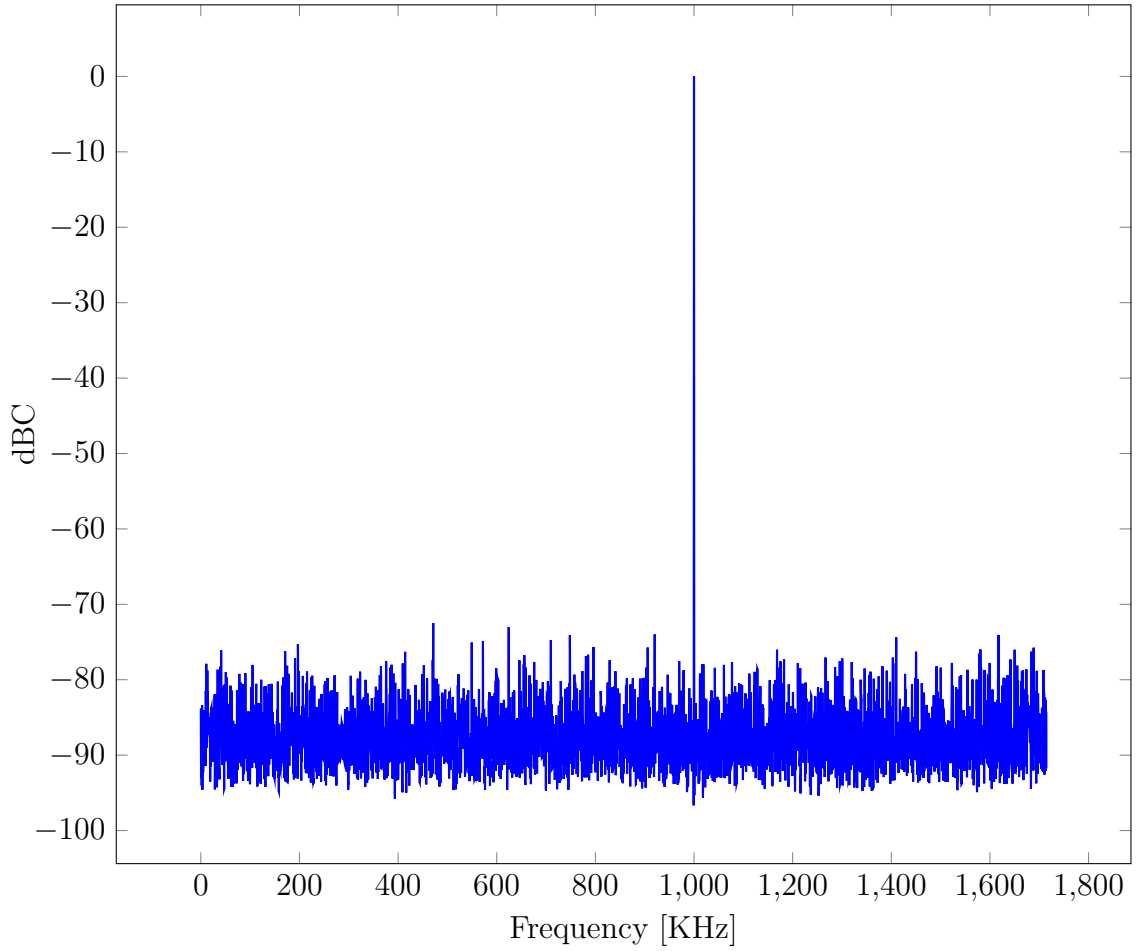


Figura 5.3: FFT ADC e signal conditioning, input sinusoide ad 1 MHz

5.2 Collaudo del Sistema per Rheinmetall

Dopo la fase di verifica funzionale del sistema in simulazione, utilizzando i programmi appropriati per la parte digitale e analogica, si è passati alla produzione della scheda, alla sua saldatura e infine alla fase di collaudo oggetto di questo capitolo.

Sono state effettuate 3 fasi distinte, dopo un primo test da banco sulla FastPro, si è passati alla verifica delle due schede di controllo (FastPro e WCC collegate insieme) e infine al collaudo del sistema KPA totale.

Per quanto riguarda la prima fase è stato utilizzato un generatore di funzioni che forniva un ingresso sinusoidale ad un canale di acquisizione della scheda ed è stato implementato un timer nell'architettura digitale, per generare il segnale di fault in modo da poter trasmettere i dati ad un PC di prova. Questo test è stato ripetuto per ogni canale di acquisizione in modo da validare l'intera parte di front-end analogico.

I segnali analogici sono stati trasmessi attraverso la stessa interfaccia fisica che è stata usata per il sistema: ovvero un cavo piatto da 26 fili per il quale, al posto di uno dei due connettori normalmente presenti alle estremità, è stato lasciato il cavo libero affinché potesse essere tolto il rivestimento isolante ed effettuare i collegamenti elettrici necessari a questa prima fase di collaudo.

Questo metodo di test è stato ripetuto per tutti e 6 i canali di acquisizione del sistema in modo da validare il funzionamento dell'intero sotto-sistema analogico della scheda progettata. Inoltre questo procedimento è in grado di testare il corretto funzionamento delle parti digitali del progetto poiché per un corretto funzionamento dei campionatori è necessario che la FPGA di controllo sia perfettamente funzionante.

Inoltre, dal momento che la trasmissione UART è troppo lenta per garantire una trasmissione dei dati in tempo reale, questi sono stati salvati nella memoria SDRAM che di conseguenza è stata potuta validare, infine la trasmissione UART e il convertitore UART-USB è stato testato dal momento che l'acquisizione del dato da parte del PC doveva essere corretta.

Una volta che i dati sono stati comunicati nella loro interezza al PC, questi sono stati salvati in un file binario, successivamente, dal momento che è stata effettuata la rimozione dei bit di stuffing nella fase di trasmissione UART, è necessario ricomporre il dato da 12 bit a 16 bit in modo da poter essere analizzato utilizzando i linguaggi di programmazione più comuni, infine attraverso uno script in Python è stato realizzato un grafico e un'analisi in frequenza del segnale.

Successivamente si è passati al test della parte del sistema digitale relativa alla rilevazione delle condizioni di comportamento anomalo del sistema. La configurazione di questo collaudo da banco è stata attuata impiegando un generatore di tensione per fornire l'alimentazione e un segnale costante conrollabile in ingresso agli ADC e infine è stato utilizzato un voltmetro per avere una corretta misura della tensione di ingresso.

Per eseguire questo test, per prima cosa ci si è avvalsi delle soglie di tensione preimpostate, non utilizzando il sistema WCC, e successivamente è stata fatta variare la tensione di ingresso dell'alimentatore, misurando la sua variazione attraverso il multimetro inserito in parallelo al circuito ed infine una volta arrivati alla tensione di soglia è stato acceso un LED presente nella scheda FastPro.

Il sistema di test per questa parte del collaudo è in figura 5.4.

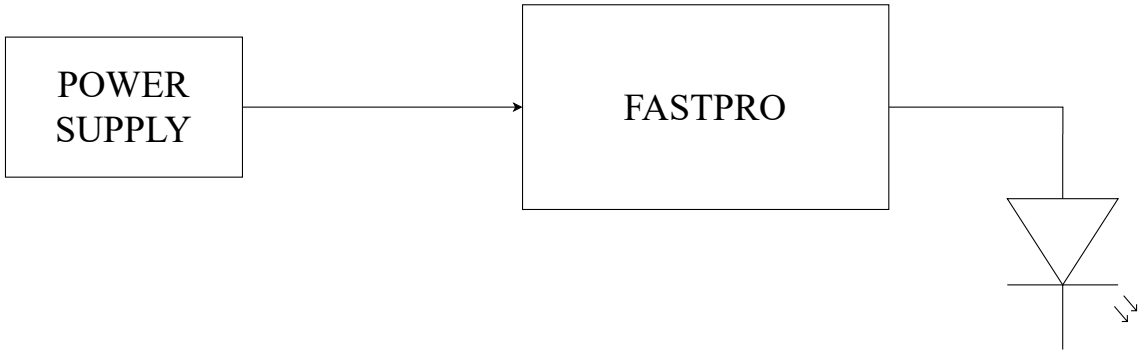


Figura 5.4: Setup Test delle Soglie

Una volta superata con successo questa prima fase di collaudo, la scheda è stata inserita all'interno del sistema di Monitoring and Control completo, che contiene la WCC: così è stato possibile verificare l'effettivo funzionamento della parte di comunicazione tra i due sistemi che compongono il sistema M&C in modo tale da garantire un prodotto funzionante e, in caso di problemi durante questa fase, applicare le modifiche necessarie.

La configurazione per questo test è mostrata in figura 5.5: la scheda madre WCC è inserita all'interno del suo "case" che presenta tutti i collegamenti necessari al suo funzionamento. La FastPro, naturalmente, è anch'essa nella stessa configurazione, e il collegamento tra le due schede avviene attraverso un cavo piatto da 26 fili che è responsabile del trasporto di tutti i segnali analogici e digitali che servono per il corretto funzionamento del sistema e un connettore "header" a 4 pin responsabile per il collegamento dell'alimentazione.

Infine tutto il sistema è collegato attraverso un cavo USB ad un computer nel quale è presente il software del sistema M&C della DSA e la cui schermata è presentata in figura 1.4.

Questa fase di collaudo ha previsto la verifica della corretta comunicazione I2C e il corretto trasferimento dei segnali analogici e digitali tra le due scheda WCC e FastPro: il primo obiettivo è stato collaudare il funzionamento del protocollo I2C, andando a leggere sul pannello del software di M&C il valore corrispondente riportato per un confronto con quello teorico.

Successivamente, utilizzando un alimentatore da banco, come generatore di segnale costante, è stata verificata la corretta comunicazione dei fault da parte della FastPro alla scheda WCC: la tensione in ingresso veniva fatta salire gradualmente fino a quando non veniva superata la soglia impostata dalla WCC tramite il protocollo I2C, a questo punto si controllava il software di M&C presente nel computer per rilevare se evidenziava correttamente lo stato di fault e visualizzava quale dispositivo lo avesse provocato.

Un'ulteriore verifica eseguita durante questa fase è stata quella dei segnali di

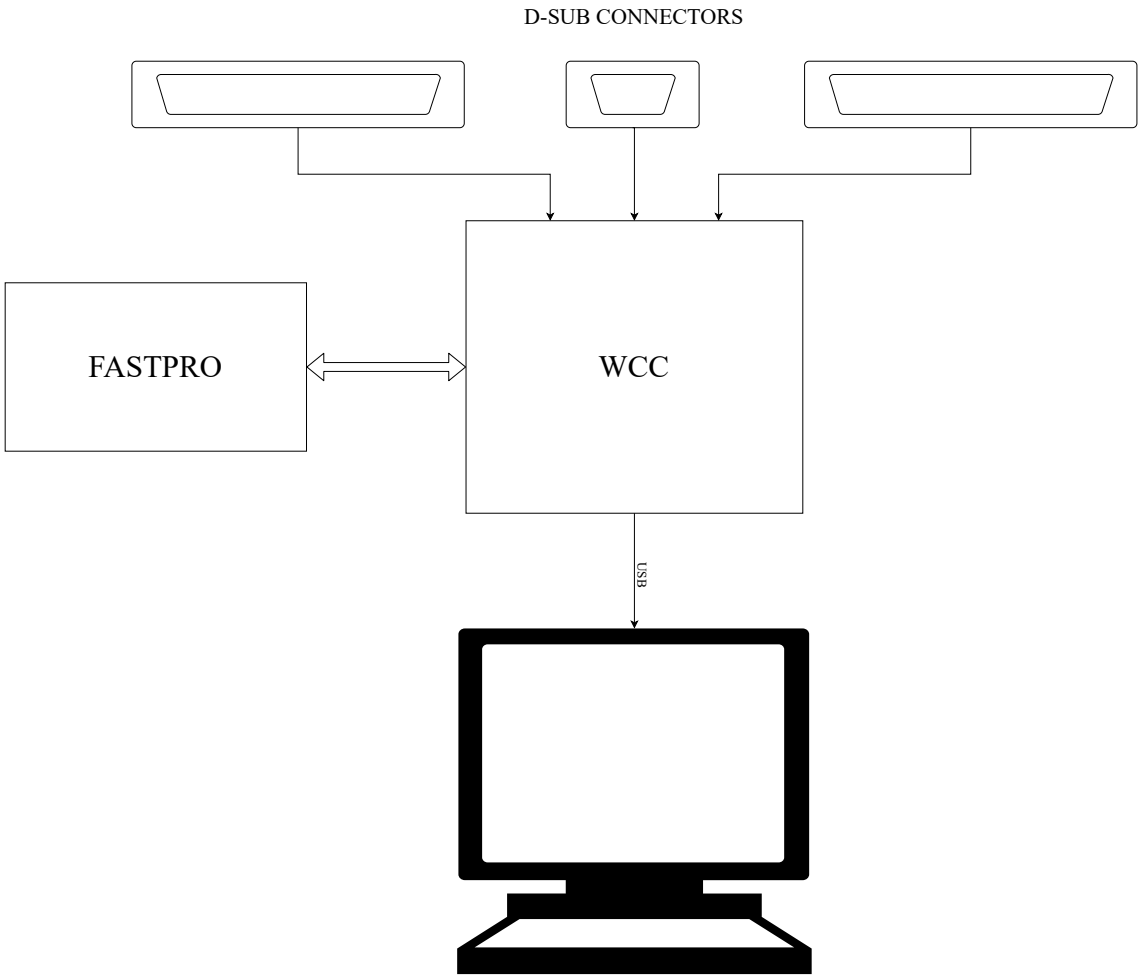


Figura 5.5: Setup del Collaudo FastPro e WCC

inhibit per i vari componenti: per prima cosa, una volta generato un segnale superiore alla soglia, attraverso lo stesso metodo esposto in precedenza, è stato misurato il valore di tensione in uscita alla FastPro, è stato misurato il valore durante il suo percorso all'interno della WCC e infine in un'uscita a questa scheda. Un ulteriore prova è stata effettuata monitorando il corrispondente indicatore sul pannello del software M&C.

Una volta superata la seconda fase di collaudo si è passati a quella riguardante l'intero sistema KPA: procedendo in questo modo si è utilizzato un sito esterno presso la società responsabile del progetto e dei contatti con ESA®, ovvero Rheinmetall s.p.a.®, dove è presente una replica del sistema, così è stato possibile testare il corretto funzionamento del KPA e, in particolare, la scheda oggetto di questa Tesi.

Infatti, attraverso dei fault simulati, è stato possibile verificare il comportamento

del sistema di Monitoring and Control: si è partiti dalla necessità di accendere il sistema e portarlo al regime di funzionamento, operazione che richiede più di 30 minuti, successivamente è stato possibile generare uno stato di funzionamento anomalo, senza però rischiare di danneggiare il sistema in caso di malfunzionamento del sistema di protezione attraverso i seguenti metodi:

1. Eliminare il collegamento con un sensore
2. Impostare attraverso il protocollo I2C una soglia inferiore (o superiore) al normale regime di funzionamento dei componenti del KPA

Entrambi questi metodi generano un fault all'interno del sistema M&C e quindi è possibile verificarne il corretto funzionamento direttamente sul sistema. Una volta superata quest'ultima fase di collaudo questo nuovo sistema è pronto per essere spedito in Germania e infine in Australia, nella sua sede di destinazione finale, dove sarà operativo per la prima volta supportando la missione "Mars Express".

Appendice A

Listati dei Programmi

A.1 Codice Sorgente VHDL

A.1.1 Sorgente Strutturale del Sistema

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4
5  entity fastPRO is
6      port(
7          ref_clk : in std_logic;
8          NPOR : in std_logic;
9
10         --ADC SIGNALS--
11         DATA_1 : in std_logic;
12         DATA_2 : in std_logic;
13         DATA_3 : in std_logic;
14         DATA_4 : in std_logic;
15         DATA_5 : in std_logic;
16         DATA_6 : in std_logic;
17         CS_ADC : out std_logic;
18         SCLK   : out std_logic;
19
20         --I2C SIGNALS--
21         I2C_SCL : inout std_logic;
22         I2C_SDA : inout std_logic;
23
24
25         -- SEGNALI DA DEFINIRE --
26         ARC_DET   : in  std_logic;
27         INHIBIT_1 : out std_logic;
28         INHIBIT_2 : out std_logic;
29
```

```
30          -- SDRAM ADDRESS --
31          A: out std_logic_vector(12 downto 0);
32
33          -- SDRAM BANK ADDRESS --
34          BA: out std_logic_vector(1 downto 0);
35
36          -- SDRAM DATA BUS --
37          DQ: inout std_logic_vector(15 downto 0);
38          LDQM: out std_logic;
39          UDQM: out std_logic;
40
41          -- SDRAM COMMANDS AND CLOCK --
42          CLK: out std_logic;
43          CS_N: out std_logic;
44          CKE: out std_logic;
45          WE_N: out std_logic;
46          CAS_N: out std_logic;
47          RAS_N: out std_logic;
48
49          -- UART SIGNALS --
50          TXD : out std_logic;
51          RXD : in std_logic;
52
53          -- LED --
54          LED :   out std_logic
55
56      );
57  end fastPRO;
58
59  architecture main_structural of fastPRO is
60
61  component SDRAM_Interface is
62
63  port (
64
65      ref_clk: in std_logic;
66      rst: in std_logic;
67
68      -- DATA TO BE WRITTEN TO THE RAM --
69      WR_DATA1: in std_logic_vector (11 downto 0);
70      WR_DATA2: in std_logic_vector (11 downto 0);
71      WR_DATA3: in std_logic_vector (11 downto 0);
72      WR_DATA4: in std_logic_vector (11 downto 0);
73      WR_DATA5: in std_logic_vector (11 downto 0);
74      WR_DATA6: in std_logic_vector (11 downto 0);
75      fault_min : in std_logic_vector (5 downto 0);
76      fault_max : in std_logic_vector (5 downto 0);
77      ARC_DET_sync : in std_logic;
78
```

```
79      -- UART SIGNALS --
80      TXD : out std_logic;
81      RXD : in std_logic;
82
83      -- DATA AVAILABLE FROM EXTERNAL INTERFACE --
84      data_available_sync: in std_logic;
85
86      -- DATA READ TO EXTERNAL INTERFACE --
87      data_read: out std_logic;
88
89      -- CONTROL SIGNALS --
90      fault_signal_sync: in std_logic; -- 1 = FAULT | 0 = NOT FAULT
91      fault_reset_sync : in std_logic;
92
93
94      -- SDRAM ADDRESS --
95      A: out std_logic_vector(12 downto 0);
96
97      -- SDRAM BANK ADDRESS --
98      BA: out std_logic_vector(1 downto 0);
99
100     -- SDRAM DATA BUS --
101     DQ: inout std_logic_vector(15 downto 0);
102     LDQM: out std_logic;
103     UDQM: out std_logic;
104
105     -- SDRAM COMMANDS AND CLOCK --
106     CLK: out std_logic;
107     CS_N: out std_logic;
108     CKE: out std_logic;
109     WE_N: out std_logic;
110     CAS_N: out std_logic;
111     RAS_N: out std_logic;
112
113     number_of_records_sync : std_logic_vector(15 downto 0);
114
115     LED_out : out std_logic
116
117 );
118
119 end component SDRAM_Interface;
120
121 component ADC_Interface is
122
123 port(
124
125 -- CLOCK FROM THE OSCILLATOR --
126     ref_clk: in std_logic;
127     rst_n: in std_logic;
```

```
128
129 -- DATA READ SIGNAL FROM THE SRAM SYSTEM --
130     data_read_sync: in std_logic;
131
132 -- SIGNAL USED TO HALT THE INTERFACE WHEN NEEDED --
133     fault_signal_sync: in std_logic;
134     fault_reset_sync : in std_logic;
135
136 -- ADC 1 INPUTS --
137     DATA_1: in std_logic;
138     DATA_2: in std_logic;
139
140 -- ADC 2 INPUTS --
141     DATA_3: in std_logic;
142     DATA_4: in std_logic;
143
144 -- ADC 3 INPUTS --
145     DATA_5: in std_logic;
146     DATA_6: in std_logic;
147
148 -- ADC SERIAL CLOCK --
149     SCLK: out std_logic;
150
151 -- ADC CHIP SELECT (ACTIVE LOW) --
152     CS_ADC: out std_logic;
153
154 -- PARALLEL OUTPUT DATA --
155     DATA_1_OUT_PARALLEL :out std_logic_vector(11 downto 0);
156     DATA_2_OUT_PARALLEL :out std_logic_vector(11 downto 0);
157     DATA_3_OUT_PARALLEL :out std_logic_vector(11 downto 0);
158     DATA_4_OUT_PARALLEL :out std_logic_vector(11 downto 0);
159     DATA_5_OUT_PARALLEL :out std_logic_vector(11 downto 0);
160     DATA_6_OUT_PARALLEL :out std_logic_vector(11 downto 0);
161
162 -- OUTPUT DATA VALID --
163     data_valid: out std_logic;
164
165     I2C_Interface_clock : out std_logic;
166     I2C_Interface_reset_n : out std_logic
167
168 );
169 end component ADC_Interface;
170
171 component I2C_Interface is
172     port(
173     clk : in std_logic;
174     rst_n : in std_logic;
175     SCL : inout std_logic;
176     SDA : inout std_logic;
```

```
177
178     ADC_DATA1 : in std_logic_vector(11 downto 0);
179     ADC_DATA2 : in std_logic_vector(11 downto 0);
180     ADC_DATA3 : in std_logic_vector(11 downto 0);
181     ADC_DATA4 : in std_logic_vector(11 downto 0);
182     ADC_DATA5 : in std_logic_vector(11 downto 0);
183     ADC_DATA6 : in std_logic_vector(11 downto 0);
184
185     ADC_TH_MAX1 : out std_logic_vector(11 downto 0);
186     ADC_TH_MAX2 : out std_logic_vector(11 downto 0);
187     ADC_TH_MAX3 : out std_logic_vector(11 downto 0);
188     ADC_TH_MAX4 : out std_logic_vector(11 downto 0);
189     ADC_TH_MAX5 : out std_logic_vector(11 downto 0);
190     ADC_TH_MAX6 : out std_logic_vector(11 downto 0);
191     ADC_TH_MIN1 : out std_logic_vector(11 downto 0);
192
193     ADC_TH_MIN2 : out std_logic_vector(11 downto 0);
194     ADC_TH_MIN3 : out std_logic_vector(11 downto 0);
195     ADC_TH_MIN4 : out std_logic_vector(11 downto 0);
196     ADC_TH_MIN5 : out std_logic_vector(11 downto 0);
197     ADC_TH_MIN6 : out std_logic_vector(11 downto 0);
198
199     TH_MIN_EN : out std_logic_vector(7 downto 0);
200     TH_MAX_EN : out std_logic_vector(7 downto 0);
201
202     commands : out std_logic_vector(7 downto 0);
203     faults_min : in std_logic_vector(7 downto 0);
204     faults_max : in std_logic_vector(7 downto 0);
205
206     debug_out : out std_logic_vector(15 downto 0)
207
208 );
209 end component I2C_Interface;
210
211 component Fault_Detection is
212
213 port(
214
215     clk : in std_logic;
216     rst_n : in std_logic;
217     fault_reset : in std_logic;
218     BPS_Vmon : in std_logic_vector(11 downto 0);
219     BPS_Imon : in std_logic_vector(11 downto 0);
220     Kly_REF_Pw : in std_logic_vector(11 downto 0);
221     Kly_Icoll : in std_logic_vector(11 downto 0);
222     Kly_Ibody : in std_logic_vector(11 downto 0);
223     MPS_Imon : in std_logic_vector(11 downto 0);
224
225     BPS_Vmon_TH_MAX : in std_logic_vector(11 downto 0);
```

```
226     BPS_Imon_TH_MAX : in std_logic_vector(11 downto 0);
227     Kly_REF_Pw_TH_MAX : in std_logic_vector(11 downto 0);
228     Kly_Icoll_TH_MAX : in std_logic_vector(11 downto 0);
229     Kly_Ibody_TH_MAX : in std_logic_vector(11 downto 0);
230     MPS_Imon_TH_MAX : in std_logic_vector(11 downto 0);
231
232     BPS_Vmon_TH_MIN : in std_logic_vector(11 downto 0);
233     BPS_Imon_TH_MIN : in std_logic_vector(11 downto 0);
234     Kly_REF_Pw_TH_MIN : in std_logic_vector(11 downto 0);
235     Kly_Icoll_TH_MIN : in std_logic_vector(11 downto 0);
236     Kly_Ibody_TH_MIN : in std_logic_vector(11 downto 0);
237     MPS_Imon_TH_MIN : in std_logic_vector(11 downto 0);
238
239     fault_min : out std_logic_vector(7 downto 0);
240     fault_max : out std_logic_vector(7 downto 0);
241
242     fault_max_en : in std_logic_vector(7 downto 0);
243     fault_min_en : in std_logic_vector(7 downto 0);
244     fault_signal : out std_logic;
245
246
247
248     ARC_DET : in std_logic;
249     INHIBIT_1 : out std_logic;
250     INHIBIT_2 : out std_logic
251
252 );
253 end component Fault_Detection;
254
255 -- SIGNALS FOR Interconnection --
256     signal WR_DATA1 : std_logic_vector (11 downto 0);
257     signal WR_DATA2 : std_logic_vector (11 downto 0);
258     signal WR_DATA3 : std_logic_vector (11 downto 0);
259     signal WR_DATA4 : std_logic_vector (11 downto 0);
260     signal WR_DATA5 : std_logic_vector (11 downto 0);
261     signal WR_DATA6 : std_logic_vector (11 downto 0);
262     signal data_available: std_logic;
263     signal data_read: std_logic;
264     signal fault_signal: std_logic;
265     signal fault_reset : std_logic;
266     signal debug : std_logic_vector (11 downto 0);
267     signal i2c_debug : std_logic_vector(15 downto 0);
268     signal clk_i2c : std_logic;
269     signal rst_n_i2c: std_logic;
270     signal ADC_TH_MAX1 : std_logic_vector(11 downto 0);
271     signal ADC_TH_MAX2 : std_logic_vector(11 downto 0);
272     signal ADC_TH_MAX3 : std_logic_vector(11 downto 0);
273     signal ADC_TH_MAX4 : std_logic_vector(11 downto 0);
274     signal ADC_TH_MAX5 : std_logic_vector(11 downto 0);
```

```
275     signal ADC_TH_MAX6 : std_logic_vector(11 downto 0);
276     signal ADC_TH_MIN1 : std_logic_vector(11 downto 0);
277     signal ADC_TH_MIN2 : std_logic_vector(11 downto 0);
278     signal ADC_TH_MIN3 : std_logic_vector(11 downto 0);
279     signal ADC_TH_MIN4 : std_logic_vector(11 downto 0);
280     signal ADC_TH_MIN5 : std_logic_vector(11 downto 0);
281     signal ADC_TH_MIN6 : std_logic_vector(11 downto 0);
282     signal TH_MIN_EN : std_logic_vector(7 downto 0);
283     signal TH_MAX_EN : std_logic_vector(7 downto 0);
284     signal faults_max : std_logic_vector(7 downto 0);
285     signal faults_min : std_logic_vector(7 downto 0);
286     signal commands : std_logic_vector(7 downto 0);
287
288
289
290
291 begin
292
293 SDRAM_and_UART : SDRAM_Interface port map(
294
295     ref_clk => ref_clk,
296     rst => NPOR,
297     WR_DATA1 => WR_DATA1,
298     WR_DATA2 => WR_DATA2,
299     WR_DATA3 => WR_DATA3,
300     WR_DATA4 => WR_DATA4,
301     WR_DATA5 => WR_DATA5,
302     WR_DATA6 => WR_DATA6,
303     fault_min_sync => faults_min(5 downto 0),
304     fault_max_sync => faults_max(5 downto 0),
305     ARC_DET_sync => ARC_DET,
306     TXD => TXD,
307     RXD => RXD,
308     data_available_sync => data_available,
309     data_read => data_read,
310     fault_signal_sync => fault_signal,
311     fault_reset_sync  => commands(0),
312     A => A,
313     BA => BA,
314     DQ => DQ,
315     LDQM => LDQM,
316     UDQM => UDQM,
317     CLK => CLK,
318     CS_N => CS_N,
319     CKE => CKE,
320     WE_N => WE_N,
321     CAS_N => CAS_N,
322     RAS_N => RAS_N,
323     number_of_records_sync => "0000111111111111",
```

```
324
325     LED_out => LED
326
327 );
328
329 ADC: ADC_Interface port map (
330
331     ref_clk=>ref_clk,
332     rst_n=>NPOR,
333     data_read_sync=>data_read,
334     fault_signal_sync=>fault_signal,
335     fault_reset_sync=> commands(0),
336     DATA_1=>DATA_1,
337     DATA_2=>DATA_2,
338     DATA_3=>DATA_3,
339     DATA_4=>DATA_4,
340     DATA_5=>DATA_5,
341     DATA_6=>DATA_6,
342     SCLK=>SCLK,
343     CS_ADC=>CS_ADC,
344     DATA_1_OUT_PARALLEL=>WR_DATA1,
345     DATA_2_OUT_PARALLEL=>WR_DATA2,
346     DATA_3_OUT_PARALLEL=>WR_DATA3,
347     DATA_4_OUT_PARALLEL=>WR_DATA4,
348     DATA_5_OUT_PARALLEL=>WR_DATA5,
349     DATA_6_OUT_PARALLEL=>WR_DATA6,
350     data_valid=>data_available,
351     I2C_Interface_clock => clk_i2c,
352     I2C_Interface_reset_n => rst_n_i2c
353
354
355 );
356
357 I2C_Interface0 : I2C_Interface port map (
358
359     clk => clk_i2c,
360    rst_n => rst_n_i2c,
361    SCL   => I2C_SCL,
362    SDA   => I2C_SDA,
363     ADC_DATA1 => WR_DATA1,
364     ADC_DATA2 => WR_DATA2,
365     ADC_DATA3 => WR_DATA3,
366     ADC_DATA4 => WR_DATA4,
367     ADC_DATA5 => WR_DATA5,
368     ADC_DATA6 => WR_DATA6,
369     ADC_TH_MAX1 => ADC_TH_MAX1,
370     ADC_TH_MAX2 => ADC_TH_MAX2,
371     ADC_TH_MAX3 => ADC_TH_MAX3,
372     ADC_TH_MAX4 => ADC_TH_MAX4,
```

```
373     ADC_TH_MAX5 => ADC_TH_MAX5,
374     ADC_TH_MAX6 => ADC_TH_MAX6,
375     ADC_TH_MIN1 => ADC_TH_MIN1,
376     ADC_TH_MIN2 => ADC_TH_MIN2,
377     ADC_TH_MIN3 => ADC_TH_MIN3,
378     ADC_TH_MIN4 => ADC_TH_MIN4,
379     ADC_TH_MIN5 => ADC_TH_MIN5,
380     ADC_TH_MIN6 => ADC_TH_MIN6,
381     TH_MIN_EN => TH_MIN_EN,
382     TH_MAX_EN => TH_MAX_EN,
383     commands => commands,
384     faults_min => faults_min,
385     faults_max => faults_max,
386     debug_out=> i2c_debug
387 );
388
389
390 Fault_Detection0 : Fault_Detection port map (
391
392
393     clk => clk_i2c,
394     rst_n => rst_n_i2c,
395     fault_reset => commands(0),
396     BPS_Vmon     => WR_DATA1,
397     BPS_Imon => WR_DATA2,
398     Kly_REF_Pw => WR_DATA3,
399     Kly_Icoll => WR_DATA4,
400     Kly_Ibody => WR_DATA5,
401     MPS_Imon => WR_DATA6,
402
403     BPS_Vmon_TH_MAX => ADC_TH_MAX1,
404     BPS_Imon_TH_MAX => ADC_TH_MAX2,
405     Kly_REF_Pw_TH_MAX => ADC_TH_MAX3,
406     Kly_Icoll_TH_MAX => ADC_TH_MAX4,
407     Kly_Ibody_TH_MAX => ADC_TH_MAX5,
408     MPS_Imon_TH_MAX => ADC_TH_MAX6,
409
410     BPS_Vmon_TH_MIN => ADC_TH_MIN1,
411     BPS_Imon_TH_MIN => ADC_TH_MIN2,
412     Kly_REF_Pw_TH_MIN => ADC_TH_MIN3,
413     Kly_Icoll_TH_MIN => ADC_TH_MIN4,
414     Kly_Ibody_TH_MIN => ADC_TH_MIN5,
415     MPS_Imon_TH_MIN => ADC_TH_MIN6,
416
417     fault_min => faults_min,
418     fault_max => faults_max,
419
420     fault_max_en => TH_MAX_EN,
421     fault_min_en => TH_MIN_EN,
```

```
422     fault_signal => fault_signal,
423
424     ARC_DET => ARC_DET,
425     INHIBIT_1 => INHIBIT_1,
426     INHIBIT_2 => INHIBIT_2
427
428
429
430 );
431
432
433 end main_structural;
```

A.1.2 Sorgente Interfaccia ADC

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_UNSIGNED.all;
4
5  --INTERFACCIA A 3,43 MSPS CON IL CLOCK DIVIDER--
6
7  -- Interfaccia tra 3 ADC "AD7352" e la FPGA, questa interfaccia usa un
       protocollo
8  -- SPI a 48MHz dove i segnali di controllo (CS_ADC e SCLK) sono comuni
       per tutti gli
9  -- ADC in modo da garantire una conversione parallela. Il clock è
       generato da un PLL,
10 -- un counter gestisce il tempo per la conversione che è di 14 periodi
       di SCLK.
11
12
13 entity ADC_Interface is
14
15 port (
16
17 -- CLOCK FROM THE OSCILLATOR --
18     ref_clk: in std_logic;
19     rst_n: in std_logic;
20
21 -- DATA READ SIGNAL FROM THE SRAM SYSTEM --
22     data_read_sync: in std_logic;
23
24 -- SIGNAL USED TO HALT AND RESET THE INTERFACE WHEN NEEDED --
25     fault_signal_sync: in std_logic;
26     fault_reset_sync: in std_logic;
27
28 -- ADC 1 INPUTS --
29     DATA_1: in std_logic;
```

```
30      DATA_2: in std_logic;
31
32  -- ADC 2 INPUTS --
33      DATA_3: in std_logic;
34      DATA_4: in std_logic;
35
36  -- ADC 3 INPUTS --
37      DATA_5: in std_logic;
38      DATA_6: in std_logic;
39
40  -- ADC SERIAL CLOCK --
41      SCLK: out std_logic;
42
43  -- ADC CHIP SELECT (ACTIVE LOW) --
44      CS_ADC: out std_logic;
45
46  -- PARALLEL OUTPUT DATA --
47      DATA_1_OUT_PARALLEL :out std_logic_vector(11 downto 0);
48      DATA_2_OUT_PARALLEL :out std_logic_vector(11 downto 0);
49      DATA_3_OUT_PARALLEL :out std_logic_vector(11 downto 0);
50      DATA_4_OUT_PARALLEL :out std_logic_vector(11 downto 0);
51      DATA_5_OUT_PARALLEL :out std_logic_vector(11 downto 0);
52      DATA_6_OUT_PARALLEL :out std_logic_vector(11 downto 0);
53
54  -- OUTPUT DATA VALID --
55      data_valid: out std_logic;
56
57  -- I2C CLOCK AND RESET --
58
59      I2C_Interface_clock : out std_logic;
60      I2C_Interface_reset_n : out std_logic
61
62  );
63
64  end ADC_Interface;
65
66
67
68  architecture SPI of ADC_Interface is
69
70  -- PLL COMPONENT DECLARATION --
71  component ADC_PLL is
72    port (
73          clk_clk        : in  std_logic := 'X'; -- clk
74          reset_reset_n  : in  std_logic := 'X'; -- reset_n
75          clk_1_clk      : out std_logic;        -- clk
76          locked_export  : out std_logic         -- export
77    );
78  end component ADC_PLL;
```

```
79
80
81
82 -- FSM POSSIBLE STATES --
83 type state_t is (RESET,START_CONVERSION,CONVERSION,DATA_ACQUISITION,
       DATA_OUT_VALID,END_CONVERSION,HALT);
84
85 -- FSM STATE SIGNALS --
86 signal current_state: state_t;
87 signal next_state: state_t;
88
89 -- FSM CONTROL SIGNALS --
90 signal FSM_reset: std_logic;
91 signal sr_en: std_logic;
92 signal out_reg_en: std_logic;
93 signal SCLK_en_n: std_logic;
94 signal convst_n: std_logic;
95 signal data_valid_en: std_logic;
96 signal count_en: std_logic;
97
98 -- COUNTER SIGNALS --
99 signal counter: std_logic_vector(3 downto 0);
100 signal conv_tc:std_logic;
101
102 --PLL SIGNALS --
103 signal clk: std_logic;
104 signal pll_locked: std_logic;
105
106
107 --SHIFT REGISTER SIGNALS --
108 signal DATA_1_SR: std_logic_vector(11 downto 0);
109 signal DATA_2_SR: std_logic_vector(11 downto 0);
110 signal DATA_3_SR: std_logic_vector(11 downto 0);
111 signal DATA_4_SR: std_logic_vector(11 downto 0);
112 signal DATA_5_SR: std_logic_vector(11 downto 0);
113 signal DATA_6_SR: std_logic_vector(11 downto 0);
114
115
116 --DATA_VALID SET RESET CONTROL SIGNALS--
117 signal data_valid_reset: std_logic;
118 signal data_valid_set: std_logic;
119
120 --CLOCK DIVIDER SIGNALS--
121 signal clock_div: std_logic_vector(1 downto 0);
122 signal clock_en: std_logic;
123 signal clock_div_resync:std_logic;
124
125 --SYNCHRONIZER SIGNALS--
126 signal data_read:std_logic;
```

```
127 signal data_read_sync1:std_logic;
128 signal fault_signal:std_logic;
129 signal fault_signal_sync1:std_logic;
130 signal fault_reset:std_logic;
131 signal fault_reset_sync1:std_logic;
132
133 attribute altera_attribute : string;
134
135 attribute altera_attribute of fault_reset_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {ADC_Interface:ADC|
    fault_reset_sync1}]""";
136 attribute altera_attribute of fault_reset : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
137
138 attribute altera_attribute of fault_signal_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {ADC_Interface:ADC|
    fault_signal_sync1}]""";
139 attribute altera_attribute of fault_signal : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
140
141
142 attribute altera_attribute of data_read_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {ADC_Interface:ADC|
    data_read_sync1}]""";
143 attribute altera_attribute of data_read : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
144
145
146
147
148 begin
149
150 -- PLL INSTANCE --
151 PLL0 : ADC_PLL port map (
152
153     clk_clk       => ref_clk,
154     clk_1_clk     => clk,
155     locked_export => pll_locked,
156     reset_reset_n => rst_n
157
```

```
158 );
159
160 -- FSM --
161 FSM_SEQUENTIAL: process (clk, pll_locked)
162 begin
163
164     if pll_locked = '0' then
165         current_state<=RESET;
166
167     elsif rising_edge(clk) then
168
169         current_state<=next_state;
170
171     end if;
172
173 end process;
174
175 FSM_SIGNALS_AND_SEQUENCER: process(fault_signal,current_state,conv_tc,
        fault_reset)
176
177 begin
178
179     case current_state is
180
181         when RESET =>
182
183             fsm_reset <='1';
184             sr_en <='0';
185             out_reg_en <='0';
186             SCLK_en_n<='1';
187             convst_n <='1';
188             data_valid_en <='0';
189             count_en <='0';
190             clock_div_resync <='0';
191
192
193             if fault_signal = '1' then
194
195                 next_state<=RESET;
196
197             else
198
199                 next_state<=START_CONVERSION;
200
201             end if;
202
203         when START_CONVERSION =>
204
205             fsm_reset <='0';
```

```
206             sr_en <= '1';
207             out_reg_en <= '0';
208             SCLK_en_n<= '1';
209             convst_n <= '0';
210             data_valid_en <= '0';
211             count_en <= '0';
212             clock_div_resync <= '0';
213
214
215             next_state<=CONVERSION;
216
217
218         when CONVERSION =>
219
220             fsm_reset <= '0';
221             sr_en <= '1';
222             out_reg_en <= '0';
223             SCLK_en_n<= '0';
224             convst_n <= '0';
225             data_valid_en <= '0';
226             count_en <= '1';
227             clock_div_resync <= '0';
228
229
230             if conv_tc = '1' then
231
232                 next_state<= DATA_ACQUISITION;
233
234             else
235
236                 next_state<=CONVERSION;
237
238             end if;
239
240         when DATA_ACQUISITION =>
241
242             fsm_reset <= '0';
243             sr_en <= '0';
244             out_reg_en <= '1';
245             SCLK_en_n<= '0';
246             convst_n <= '1';
247             data_valid_en <= '0';
248             count_en <= '0';
249             clock_div_resync <= '0';
250
251
252             if fault_signal = '1' then
253
254                 next_state<=HALT;
```

```
255
256             else
257
258                 next_state<=DATA_OUT_VALID;
259
260             end if;
261
262         when DATA_OUT_VALID=>
263
264             fsm_reset <='0';
265             sr_en <='0';
266             out_reg_en <='0';
267             SCLK_en_n<='0';
268             convst_n <='1';
269             data_valid_en <='1';
270             count_en <='0';
271             clock_div_resync <='0';
272
273             next_state<=END_CONVERSION;
274
275
276         when END_CONVERSION=>
277
278             fsm_reset <='0';
279             sr_en <='0';
280             out_reg_en <='0';
281             SCLK_en_n<='0';
282             convst_n <='1';
283             data_valid_en <='0';
284             count_en <='0';
285             clock_div_resync <='0';
286
287             if fault_signal = '1' then
288
289                 next_state<=HALT;
290
291             else
292
293                 next_state<=RESET;
294
295             end if;
296
297         when HALT =>
298
299             fsm_reset <='0';
300             sr_en <='0';
301             out_reg_en <='0';
302             SCLK_en_n<='1';
303             convst_n <='1';
```

```
304                 data_valid_en <= '0';
305                 count_en <= '0';
306                 clock_div_resync <= '1';
307 
308                 if fault_reset = '1' then
309 
310                     next_state<=RESET;
311 
312                 else
313 
314                     next_state<=HALT;
315 
316                 end if;
317 
318             when others =>
319 
320                 fsm_reset <= '0';
321                 sr_en <= '0';
322                 out_reg_en <= '0';
323                 SCLK_en_n<= '1';
324                 convst_n <= '1';
325                 data_valid_en <= '0';
326                 count_en <= '0';
327                 clock_div_resync <= '1';
328 
329                 next_state<=RESET;
330 
331     end case;
332 
333 end process;
334 
335 
336 -- COUNTER --
337 SPI_bit_counter: process(clk,fsm_reset)
338 
339 begin
340 
341     if  fsm_reset = '1' then
342 
343         counter<="0000";
344 
345     elsif rising_edge(clk) then
346 
347         if conv_tc = '1' then
348 
349             counter<="0000";
350 
351         else
352 
```

```
353             if (count_en = '1' and clock_en = '1') then
354
355                 counter<=counter+1;
356
357             end if;
358
359         end if;
360
361     end if;
362
363 end process;
364
365 -- CONVERSION TERMINAL COUNT --
366 process(counter)
367
368 begin
369
370     if counter = "1101" then
371
372         conv_tc<='1';
373
374     else
375
376         conv_tc<='0';
377
378     end if;
379
380 end process;
381
382
383 -- CLOCK DIVIDER --
384 process(pll_locked,clock_div_resync,clk)
385 begin
386
387     if (pll_locked='0' or clock_div_resync='1') then
388
389         clock_div<="10";
390
391     elsif rising_edge(clk) then
392
393         clock_div<=clock_div+1;
394
395     end if;
396
397 end process;
398
399 clock_en<=clock_div(1) and not(clock_div(0));
400 SCLK<=clock_div(1) or sclk_en_n;
401
```

```
402
403 -- SHIFT REGISTER --
404 data_in_shift_register: process(clk,sr_en, fsm_reset)
405
406 begin
407
408     if  fsm_reset = '1' then
409
410         DATA_1_SR<=(others => '0');
411         DATA_2_SR<=(others => '0');
412         DATA_3_SR<=(others => '0');
413         DATA_4_SR<=(others => '0');
414         DATA_5_SR<=(others => '0');
415         DATA_6_SR<=(others => '0');
416
417     elsif rising_edge(clk) then
418
419         if (sr_en = '1' and clock_en = '1') then
420
421             for I IN 11 downto 1 loop
422
423                 DATA_1_SR(I)<=DATA_1_SR(I-1);
424                 DATA_2_SR(I)<=DATA_2_SR(I-1);
425                 DATA_3_SR(I)<=DATA_3_SR(I-1);
426                 DATA_4_SR(I)<=DATA_4_SR(I-1);
427                 DATA_5_SR(I)<=DATA_5_SR(I-1);
428                 DATA_6_SR(I)<=DATA_6_SR(I-1);
429
430
431             end loop;
432
433             DATA_1_SR(0)<=DATA_1;
434             DATA_2_SR(0)<=DATA_2;
435             DATA_3_SR(0)<=DATA_3;
436             DATA_4_SR(0)<=DATA_4;
437             DATA_5_SR(0)<=DATA_5;
438             DATA_6_SR(0)<=DATA_6;
439
440         end if;
441
442     end if;
443
444
445 end process;
446
447 -- OUTPUT REGISTERS --
448 process(clk,pll_locked)
449
450 begin
```

```
451
452     if pll_locked = '0' then
453
454         DATA_1_OUT_PARALLEL<=(others => '0');
455         DATA_2_OUT_PARALLEL<=(others => '0');
456         DATA_3_OUT_PARALLEL<=(others => '0');
457         DATA_4_OUT_PARALLEL<=(others => '0');
458         DATA_5_OUT_PARALLEL<=(others => '0');
459         DATA_6_OUT_PARALLEL<=(others => '0');
460
461     elsif rising_edge(clk) then
462
463         if out_reg_en = '1' then
464
465             DATA_1_OUT_PARALLEL<=DATA_1_SR;
466             DATA_2_OUT_PARALLEL<=DATA_2_SR;
467             DATA_3_OUT_PARALLEL<=DATA_3_SR;
468             DATA_4_OUT_PARALLEL<=DATA_4_SR;
469             DATA_5_OUT_PARALLEL<=DATA_5_SR;
470             DATA_6_OUT_PARALLEL<=DATA_6_SR;
471
472         end if;
473
474     end if;
475
476 end process;
477
478 --DATA VALID SET RESET --
479 --process( data_valid_reset, data_valid_set)
480 process (pll_locked, clk)
481 begin
482 if pll_locked = '0' then
483     data_valid<='0';
484
485 elsif rising_edge(clk) then
486     if (data_valid_reset = '1') then
487
488         data_valid<='0';
489
490     end if;
491
492     if (data_valid_set = '1') then
493
494         data_valid<='1';
495
496     end if;
497 end if;
498
499
```

```
500 end process;
501
502
503 -- SYNCHRONIZER --
504 process(clk,pll_locked)
505 begin
506
507     if pll_locked = '0' then
508
509         data_read<='0';
510         data_read_sync1<='0';
511
512         fault_signal<='0';
513         fault_signal_sync1<='0';
514
515         fault_reset<='0';
516         fault_reset_sync1<='0';
517
518     elsif rising_edge(clk) then
519
520         data_read<=data_read_sync1;
521         data_read_sync1<=data_read_sync;
522
523         fault_signal<=fault_signal_sync1;
524         fault_signal_sync1<=fault_signal_sync;
525
526         fault_reset<=fault_reset_sync1;
527         fault_reset_sync1<=fault_reset_sync;
528
529     end if;
530
531 end process;
532
533 --DATA_VALID SET and RESET SIGNAL GENERATION--
534 data_valid_set<=data_valid_en;
535 --data_valid_reset<=(not(rst_n) or data_read) and not(data_valid_en);
536 data_valid_reset<=(data_read) and not(data_valid_en);
537
538 CS_ADC<=convst_n;
539
540
541 I2C_Interface_clock <= clk;
542 I2C_Interface_reset_n <= pll_locked;
543
544
545
546 end SPI;
```

A.1.3 Sorgente Interfaccia SDRAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.NUMERIC_STD.all;

entity SDRAM_interface is

port(

    ref_clk: in std_logic;
    rst: in std_logic;

    -- DATA TO BE WRITTEN TO THE RAM --
    WR_DATA1 : in std_logic_vector (11 downto 0);
    WR_DATA2 : in std_logic_vector (11 downto 0);
    WR_DATA3 : in std_logic_vector (11 downto 0);
    WR_DATA4 : in std_logic_vector (11 downto 0);
    WR_DATA5 : in std_logic_vector (11 downto 0);
    WR_DATA6 : in std_logic_vector (11 downto 0);
    fault_min_sync : in std_logic_vector (5 downto 0);
    fault_max_sync : in std_logic_vector (5 downto 0);
    ARC_DET_sync : in std_logic;

    -- UART SIGNALS --
    TXD : out std_logic;
    RXD : in std_logic;

    -- DATA AVAILABLE FROM EXTERNAL INTERFACE --
    data_available_sync: in std_logic;

    -- DATA READ TO EXTERNAL INTERFACE --
    data_read: out std_logic;

    -- CONTROL SIGNALS --
    fault_signal_sync: in std_logic; -- 1 = FAULT | 0 = NOT FAULT
    fault_reset_sync : in std_logic;


    -- SDRAM ADDRESS --
    A: out std_logic_vector(12 downto 0);

    -- SDRAM BANK ADDRESS --
    BA: out std_logic_vector(1 downto 0);

    -- SDRAM DATA BUS --
    DQ: inout std_logic_vector(15 downto 0);
    LDQM: out std_logic;
```

```
48      UDQM: out std_logic;
49
50      -- SDRAM COMMANDS AND CLOCK --
51      CLK: out std_logic;
52      CS_N: out std_logic;
53      CKE: out std_logic;
54      WE_N: out std_logic;
55      CAS_N: out std_logic;
56      RAS_N: out std_logic;
57
58      -- NUMBER OF RECORDS TO BE SENT TO SYSTEM PC --
59      number_of_records_sync : std_logic_vector(15 downto 0);
60
61
62      LED_out : out std_logic
63
64  );
65
66
67  end SDRAM_interface;
68
69
70  architecture behav of SDRAM_interface is
71
72
73
74
75  -- SDRAM SUBSYSTEM COMPONENT DECLARATION --
76  component SDRAM_Subsys is
77    port (
78          clk_clk                       : in    std_logic
        := 'X';             -- clk
79          reset_reset_n                 : in    std_logic
        := 'X';             -- reset_n
80          pll_export                    : out   std_logic;
                            -- export
81          sdram_avalon_address          : in    std_logic_vector(24 downto
     0) := (others => 'X'); -- address
82          sdram_avalon_byteenable_n     : in    std_logic_vector(1 downto 0)
        := (others => 'X'); -- byteenable_n
83          sdram_avalon_chipselect       : in    std_logic
        := 'X';             -- chipselect
84          sdram_avalon_writedata        : in    std_logic_vector(15 downto
     0) := (others => 'X'); -- writedata
85          sdram_avalon_read_n           : in    std_logic
        := 'X';             -- readc_n
86          sdram_avalon_write_n          : in    std_logic
        := 'X';             -- write_n
```

```
87          sdram_avalon_readdata       : out   std_logic_vector(15 downto
    0);                     -- readdata
88          sdram_avalon_readdatavalid : out   std_logic;
                            -- readdatavalid
89          sdram_avalon_waitrequest    : out   std_logic;
                            -- waitrequest
90          sdram_addr                  : out   std_logic_vector(12 downto
    0);                     -- addr
91          sdram_ba                    : out   std_logic_vector(1 downto 0)
    ;                       -- ba
92          sdram_cas_n                 : out   std_logic;
                            -- cas_n
93          sdram_cke                   : out   std_logic;
                            -- cke
94          sdram_cs_n                  : out   std_logic;
                            -- cs_n
95          sdram_dq                    : inout std_logic_vector(15 downto
    0) := (others => 'X'); -- dq
96          sdram_dqm                   : out   std_logic_vector(1 downto 0)
    ;                       -- dqm
97          sdram_ras_n                 : out   std_logic;
                            -- ras_n
98          sdram_we_n                  : out   std_logic;
                            -- we_n
99          sys_clk                     : out   std_logic;
                            -- clk
100         ram_clk                     : out   std_logic
                            -- ram_clk
101
102 );
103 end component SDRAM_Subsys;
104
105
106 -- UART INTERFACE COMPONENT DECLARATION --
107 component uart is
108   port (
109         clk_clk                     : in  std_logic
    := 'X';             -- clk
110         reset_reset_n               : in  std_logic
    := 'X';             -- reset_n
111         uart_rxd                    : in  std_logic
    := 'X';             -- rxd
112         uart_txd                    : out std_logic;
                        -- txd
113         uart_avalon_address         : in  std_logic_vector(2 downto 0)
    := (others => 'X'); -- address
114         uart_avalon_begintransfer : in  std_logic
    := 'X';             -- begintransfer
```

```
115         uart_avalon_chipselect     : in  std_logic
    := 'X';                 -- chipselect
116         uart_avalon_read_n         : in  std_logic
    := 'X';                 -- read_n
117         uart_avalon_write_n        : in  std_logic
    := 'X';                 -- write_n
118         uart_avalon_writedata      : in  std_logic_vector(15 downto 0)
    := (others => 'X'); -- writedata
119         uart_avalon_readdata       : out std_logic_vector(15 downto 0)
                        -- readdata
120 );
121 end component uart;
122
123 -- PLL SIGNALS --
124 signal sys_clk:std_logic; -- clock for the whole system except for the
    sdram --
125 signal pll_locked:std_logic;
126
127 -- SDRAM CONTROLLER AVALON INTERFACE SIGNALS --
128 signal sdram_avalon_address : std_logic_vector (24 downto 0);
129 signal sdram_avalon_byteenable_n : std_logic_vector (1 downto 0);
130 signal sdram_avalon_writedata : std_logic_vector (15 downto 0);
131 signal sdram_avalon_read_n : std_logic;
132 signal sdram_avalon_write_n : std_logic;
133 signal sdram_avalon_readdata : std_logic_vector (15 downto 0);
134 signal sdram_avalon_readdatavalid : std_logic;
135 signal sdram_avalon_waitrequest : std_logic;
136
137 -- UART AVALON INTERFACE SIGNALS --
138 signal uart_avalon_address : std_logic_vector(2 downto 0);
139 signal uart_avalon_begintransfer : std_logic;
140 signal uart_avalon_chipselect : std_logic;
141 signal uart_avalon_read_n : std_logic;
142 signal uart_avalon_write_n : std_logic;
143 signal uart_avalon_writedata : std_logic_vector(15 downto 0);
144 signal uart_avalon_readdata : std_logic_vector(15 downto 0);
145
146 -- DATA READ S/R SIGNALS --
147 signal data_read_set: std_logic;
148 signal data_read_reset: std_logic;
149 signal data_available_edge_1: std_logic;
150 signal data_available_edge_2: std_logic;
151 signal data_read_sig: std_logic;
152
153 -- FSM SIGNALS --
154 type state_t is (RESET,RD_WR,WAIT_DATA,WR,RD,WAIT_DATAVALID,
    CAPTURE_RAM_DATA,WAIT_UART,UART_TX,WAIT_DELAY1,WAIT_DELAY2,
    DATA_READ_ACK,
155                             CHECK_STOP, HALT,FLUSH_FIFO, ADDRESS_RECOVERY);
```

```
156
157 signal current_state : state_t;
158 signal next_state : state_t;
159 signal FSM_reset : std_logic;
160 signal in_mux_cnt_en:std_logic;
161 signal address_counter_up_en : std_logic;
162 signal address_counter_down_en : std_logic;
163 signal uart_write_en : std_logic;
164 signal uart_counter : std_logic;
165 signal sdram_read_counter : std_logic;
166 signal RD_DATA_en : std_logic;
167 signal record_counter_en : std_logic;
168 signal UART_fifo_flushed_en : std_logic;
169
170 -- INPUT MUX SIGNALS --
171 signal in_mux_cnt : std_logic_vector (2 downto 0);
172 signal in_mux_cnt_tc : std_logic;
173
174 -- ADDRESS COUNTER SIGNALS --
175 signal address_counter : std_logic_vector (24 downto 0);
176 signal address_incr : std_logic_vector (24 downto 0);
177
178 -- RECORD COUNTER --
179 signal record_counter : std_logic_vector(2 downto 0);
180
181 -- STOP READ ADDRESS REGISTER --
182 signal stop_read_address_register : std_logic_vector (15 downto 0);
183 signal stop_read : std_logic;
184
185 --INPUT REGISTERS SIGNALS --
186 signal WR_DATA1_REG : std_logic_vector (11 downto 0);
187 signal WR_DATA2_REG : std_logic_vector (11 downto 0);
188 signal WR_DATA3_REG : std_logic_vector (11 downto 0);
189 signal WR_DATA4_REG : std_logic_vector (11 downto 0);
190 signal WR_DATA5_REG : std_logic_vector (11 downto 0);
191 signal WR_DATA6_REG : std_logic_vector (11 downto 0);
192 signal fault_min_reg : std_logic_vector (5 downto 0);
193 signal fault_max_reg : std_logic_vector (5 downto 0);
194 signal ARC_DET_reg : std_logic;
195
196
197
198 signal input_reg_en : std_logic;
199
200
201 -- DATA READ FROM RAM REGISTER SIGNAL --
202 signal RD_DATA : std_logic_vector (15 downto 0);
203
204 -- SYNCHRONIZER SIGNALS --
```

```
205 signal WR_DATA1 : std_logic_vector (11 downto 0);
206 signal WR_DATA2 : std_logic_vector (11 downto 0);
207 signal WR_DATA3 : std_logic_vector (11 downto 0);
208 signal WR_DATA4 : std_logic_vector (11 downto 0);
209 signal WR_DATA5 : std_logic_vector (11 downto 0);
210 signal WR_DATA6 : std_logic_vector (11 downto 0);
211 signal fault_signal : std_logic;
212 signal fault_signal_sync1 : std_logic;
213 signal fault_reset : std_logic;
214 signal fault_reset_sync1 : std_logic;
215 signal data_available : std_logic;
216 signal data_available_sync1 : std_logic;
217 signal fault_min_sync1 : std_logic_vector (5 downto 0);
218 signal fault_min : std_logic_vector (5 downto 0);
219 signal fault_max_sync1 : std_logic_vector (5 downto 0);
220 signal fault_max : std_logic_vector (5 downto 0);
221 signal ARC_DET_sync1 : std_logic;
222 signal ARC_DET : std_logic;
223 signal number_of_records_sync1 : std_logic_vector(15 downto 0);
224 signal number_of_records : std_logic_vector(15 downto 0);
225
226
227 signal addr_cnt_rst : std_logic;
228 signal prova : std_logic;
229 signal debug : std_logic;
230
231 attribute altera_attribute : string;
232
233 attribute altera_attribute of data_available_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {SDRAM_Interface:
    SDRAM_and_UART|data_available_sync1}]""";
234 attribute altera_attribute of data_available : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
235
236 attribute altera_attribute of fault_reset_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {SDRAM_Interface:
    SDRAM_and_UART|fault_reset_sync1}]""";
237 attribute altera_attribute of fault_reset : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
238
```

```
239 attribute altera_attribute of fault_signal_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {SDRAM_Interface:
    SDRAM_and_UART|fault_signal_sync1}]""";
240 attribute altera_attribute of fault_signal : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
241
242 attribute altera_attribute of fault_min_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
243 attribute altera_attribute of fault_min : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
244
245 attribute altera_attribute of fault_max_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
246 attribute altera_attribute of fault_max : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
247
248 attribute altera_attribute of ARC_DET_sync1 : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON; -name
    SDC_STATEMENT ""set_false_path -to [get_keepers {SDRAM_Interface:
    SDRAM_and_UART|ARC_DET_sync1}]""";
249 attribute altera_attribute of ARC_DET : signal is "-name
    synchronizer_identification FORCED_IF_ASYNCHRONOUS; -name
    DONT_MERGE_REGISTER ON; -name PRESERVE_REGISTER ON";
250
251
252 begin
253
254 -- SDRAM SUBSYSTEM INSTANCE --
255 SDRAM_contr_and_PLL : SDRAM_Subsys
256       port map (
257             clk_clk                       =>ref_clk,
258             reset_reset_n                 =>rst,
259             pll_export                    =>pll_locked,
260             sdram_avalon_address          =>sdram_avalon_address,
261             sdram_avalon_byteenable_n     =>sdram_avalon_byteenable_n,
262             sdram_avalon_chipselect       =>'1',
263             sdram_avalon_writedata        =>sdram_avalon_writedata,
264             sdram_avalon_read_n           =>sdram_avalon_read_n,
265             sdram_avalon_write_n          =>sdram_avalon_write_n,
266             sdram_avalon_readdata         =>sdram_avalon_readdata,
267             sdram_avalon_readdatavalid    =>sdram_avalon_readdatavalid,
```

```
268                 sdram_avalon_waitrequest    =>sdram_avalon_waitrequest,
269                 sdram_addr                  =>A,
270                 sdram_ba                    =>BA,
271                 sdram_cas_n                 =>CAS_N,
272                 sdram_cke                   =>CKE,
273                 sdram_cs_n                  =>CS_N,
274                 sdram_dq                    =>DQ,
275                 sdram_dqm(1)                =>UDQM,
276                 sdram_dqm(0)                =>LDQM,
277                 sdram_ras_n                 =>RAS_N,
278                 sdram_we_n                  =>WE_N,
279                 sys_clk                     =>sys_clk,
280                 ram_clk                     =>CLK
281         );
282
283 --UART INSTANCE --
284 uart0 : uart
285           port map (
286                 clk_clk                     => sys_clk,
287                 reset_reset_n               => pll_locked,
288                 uart_rxd                    => RXD,
289                 uart_txd                    => TXD,
290                 uart_avalon_address         => uart_avalon_address,
291                 uart_avalon_begintransfer   => uart_avalon_begintransfer,
292                 uart_avalon_chipselect      => uart_avalon_chipselect,
293                 uart_avalon_read_n          => uart_avalon_read_n,
294                 uart_avalon_write_n         => uart_avalon_write_n,
295                 uart_avalon_writedata       => uart_avalon_writedata,
296                 uart_avalon_readdata        => uart_avalon_readdata
297           );
298
299
300
301 -- AVALON ADDRESS --
302 sdram_avalon_address<=address_counter;
303
304
305 -- FSM --
306 FSM_SEQUENTIAL: process(sys_clk,pll_locked)
307
308 begin
309
310     if pll_locked='0' then
311
312         current_state<=RESET;
313
314     elsif rising_edge(sys_clk) then
315
316         current_state<=next_state;
```

```
317
318     end if;
319
320 end process;
321
322 FSM_SIGNAL_AND_SEQUENCER: process (current_state,
    sdram_avalon_waitrequest,sdram_avalon_readdatavalid, fault_signal,
    fault_reset, data_available,
323                                                    in_mux_cnt_tc,
    uart_counter,uart_avalon_readdata,data_read_sig,stop_read,
    sdram_read_counter)
324 begin
325
326     case current_state is
327
328         when RESET =>
329
330             FSM_reset<='1';
331             in_mux_cnt_en<='0';
332             address_counter_up_en<='0';
333             address_counter_down_en<='0';
334             sdram_avalon_byteenable_n<="00";
335             sdram_avalon_read_n <= '1';
336             sdram_avalon_write_n <= '1';
337             uart_avalon_write_n<='1';
338             uart_avalon_address<="000";
339             uart_avalon_begintransfer<='0';
340             data_read_set<='0';
341             input_reg_en<='0';
342             uart_avalon_chipselect<='1';
343             uart_avalon_read_n<='1';
344             RD_DATA_en<='0';
345             record_counter_en<='0';
346             UART_fifo_flushed_en<='0';
347
348
349
350
351             addr_cnt_rst<='0';
352
353             next_state<=RD_WR;
354
355         when RD_WR =>
356
357             FSM_reset<='0';
358            in_mux_cnt_en<='0';
359            address_counter_up_en<='0';
360             address_counter_down_en<='0';
361            sdram_avalon_byteenable_n<="00";
```

```
362             sdram_avalon_read_n <= '1';
363             sdram_avalon_write_n <= '1';
364              uart_avalon_write_n <='1';
365              uart_avalon_address<="000";
366              uart_avalon_begintransfer <='0';
367              data_read_set <='0';
368              input_reg_en <='0';
369              uart_avalon_chipselect <='0';
370              uart_avalon_read_n <='1';
371             RD_DATA_en<='0';
372              record_counter_en <='0';
373              UART_fifo_flushed_en <='0';
374
375
376
377              addr_cnt_rst <='0';
378
379              if fault_signal = '0' then
380
381                  next_state<=WAIT_DATA;
382
383              else
384
385                  --next_state<=RD;
386                  next_state<=ADDRESS_RECOVERY;
387
388              end if;
389
390          when WAIT_DATA =>
391
392              FSM_reset <='0';
393             in_mux_cnt_en<='0';
394             address_counter_up_en <='0';
395              address_counter_down_en <='0';
396             sdram_avalon_byteenable_n<="00";
397             sdram_avalon_read_n <= '1';
398             sdram_avalon_write_n <= '1';
399              uart_avalon_write_n <='1';
400              uart_avalon_address<="000";
401              uart_avalon_begintransfer <='0';
402              data_read_set <='0';
403              input_reg_en <='0';
404              uart_avalon_chipselect <='0';
405              uart_avalon_read_n <='1';
406             RD_DATA_en<='0';
407              record_counter_en <='0';
408              UART_fifo_flushed_en <='0';
409
410
```

```
411             addr_cnt_rst <='0';
412
413             if fault_signal = '1' then
414
415                 next_state<=ADDRESS_RECOVERY;
416
417             elsif data_available = '1' and data_read_sig = '0' then
418
419                     next_state <= DATA_READ_ACK;
420
421             else
422
423                 next_state <= WAIT_DATA;
424
425             end if;
426
427         when DATA_READ_ACK =>
428
429             FSM_reset<='0';
430             in_mux_cnt_en<='0';
431             address_counter_up_en<='0';
432             address_counter_down_en<='0';
433             sdram_avalon_byteenable_n<="00";
434             sdram_avalon_read_n <= '1';
435             sdram_avalon_write_n <= '1';
436             uart_avalon_write_n<='1';
437             uart_avalon_address<="000";
438             uart_avalon_begintransfer<='0';
439             data_read_set<='1';
440             input_reg_en<='1';
441             uart_avalon_chipselect<='0';
442             uart_avalon_read_n<='1';
443             RD_DATA_en<='0';
444             record_counter_en<='0';
445             UART_fifo_flushed_en<='0';
446
447
448             addr_cnt_rst <='0';
449
450             if fault_signal = '0' then
451
452                 next_state<= WR;
453
454             else
455
456                 next_state<=RD;
457
458             end if;
459
```

```
460         when WR =>
461
462              FSM_reset<='0';
463             in_mux_cnt_en<='1';
464             address_counter_up_en<='1';
465              address_counter_down_en<='0';
466             sdram_avalon_byteenable_n<="00";
467             sdram_avalon_read_n <= '1';
468             sdram_avalon_write_n <= '0';
469              uart_avalon_write_n<='1';
470              uart_avalon_address<="000";
471              uart_avalon_begintransfer<='0';
472              data_read_set<='0';
473              input_reg_en<='0';
474              uart_avalon_chipselect<='0';
475              uart_avalon_read_n<='1';
476             RD_DATA_en<='0';
477              record_counter_en<='0';
478              UART_fifo_flushed_en<='0';
479
480
481              addr_cnt_rst<='0';
482
483              if (sdram_avalon_waitrequest = '0' and in_mux_cnt_tc = '1')
    then
484
485                  next_state<=RD_WR;
486
487              else
488
489                  next_state<=WR;
490
491              end if;
492
493         when RD =>
494
495              FSM_reset<='0';
496             in_mux_cnt_en<='0';
497             address_counter_up_en<='0';
498              address_counter_down_en<='1';
499             sdram_avalon_byteenable_n<="00";
500             sdram_avalon_read_n <= '0';
501             sdram_avalon_write_n <= '1';
502              uart_avalon_write_n<='1';
503              uart_avalon_address<="000";
504              uart_avalon_begintransfer<='0';
505              data_read_set<='0';
506              input_reg_en<='0';
507              uart_avalon_chipselect<='0';
```

```
508             uart_avalon_read_n <='1';
509            RD_DATA_en<='0';
510             record_counter_en <='0';
511             UART_fifo_flushed_en <='0';
512
513
514
515             addr_cnt_rst <='0';
516
517
518             if   sdram_avalon_waitrequest = '0' then
519
520                  next_state <= WAIT_DATAVALID;
521
522             else
523
524                  next_state <= RD;
525
526             end if;
527
528         when WAIT_DATAVALID =>
529
530              FSM_reset<='0';
531            in_mux_cnt_en<='0';
532            address_counter_up_en<='0';
533             address_counter_down_en<='0';
534            sdram_avalon_byteenable_n<="00";
535            sdram_avalon_read_n <= '1';
536            sdram_avalon_write_n <= '1';
537             uart_avalon_write_n<='1';
538             uart_avalon_address<="000";
539             uart_avalon_begintransfer<='0';
540             data_read_set<='0';
541             input_reg_en<='0';
542             uart_avalon_chipselect<='1';
543             uart_avalon_read_n<='1';
544            RD_DATA_en<='0';
545             record_counter_en<='0';
546             UART_fifo_flushed_en<='0';
547
548
549             addr_cnt_rst<='0';
550
551             if sdram_avalon_readdatavalid = '1' then
552
553                  next_state <= CAPTURE_RAM_DATA;
554
555             else
556
```

```
557                 next_state <= WAIT_DATAVALID;
558
559               end if;
560
561
562           when CAPTURE_RAM_DATA =>
563
564               FSM_reset<='0';
565              in_mux_cnt_en<='0';
566              address_counter_up_en<='0';
567               address_counter_down_en<='0';
568              sdram_avalon_byteenable_n<="00";
569              sdram_avalon_read_n <= '1';
570              sdram_avalon_write_n <= '1';
571               uart_avalon_write_n<='1';
572               uart_avalon_address<="000";
573               uart_avalon_begintransfer<='0';
574               data_read_set<='0';
575               input_reg_en<='0';
576               uart_avalon_chipselect<='1';
577               uart_avalon_read_n<='1';
578              RD_DATA_en<='1';
579               record_counter_en<='1';
580               UART_fifo_flushed_en<='0';
581
582
583               addr_cnt_rst<='0';
584
585               next_state<=WAIT_UART;
586
587           when WAIT_UART =>
588
589               FSM_reset<='0';
590              in_mux_cnt_en<='0';
591              address_counter_up_en<='0';
592               address_counter_down_en<='0';
593              sdram_avalon_byteenable_n<="00";
594              sdram_avalon_read_n <= '1';
595              sdram_avalon_write_n <= '1';
596               uart_avalon_write_n<='1';
597               uart_avalon_address<="010";
598               uart_avalon_begintransfer<='0';
599               data_read_set<='0';
600               input_reg_en<='0';
601               uart_avalon_chipselect<='1';
602               uart_avalon_read_n<='0';
603               addr_cnt_rst<='0';
604              RD_DATA_en<='0';
605               record_counter_en<='0';
```

```
606             UART_fifo_flushed_en<='0';
607
608
609             addr_cnt_rst<='0';
610
611
612             if uart_avalon_readdata(6) = '1' then
613
614                 next_state<=UART_TX;
615
616             else
617
618                 next_state<=WAIT_UART;
619
620             end if;
621
622
623         when UART_TX =>
624
625              FSM_reset<='0';
626            in_mux_cnt_en<='0';
627            address_counter_up_en<='0';
628             address_counter_down_en<='0';
629            sdram_avalon_byteenable_n<="00";
630            sdram_avalon_read_n <= '1';
631            sdram_avalon_write_n <= '1';
632             uart_avalon_write_n<='0';
633             uart_avalon_address<="001";
634             uart_avalon_begintransfer<='1';
635             data_read_set<='0';
636             input_reg_en<='0';
637             uart_avalon_chipselect<='1';
638             uart_avalon_read_n<='1';
639            RD_DATA_en<='0';
640             record_counter_en<='0';
641             UART_fifo_flushed_en<='0';
642
643
644             addr_cnt_rst<='0';
645
646             next_state<=WAIT_DELAY1;
647
648         when WAIT_DELAY1 =>
649
650              FSM_reset<='0';
651            in_mux_cnt_en<='0';
652            address_counter_up_en<='0';
653             address_counter_down_en<='0';
654            sdram_avalon_byteenable_n<="00";
```

```
655             sdram_avalon_read_n <= '1';
656             sdram_avalon_write_n <= '1';
657              uart_avalon_write_n <='1';
658              uart_avalon_address<="000";
659              uart_avalon_begintransfer <='0';
660              data_read_set <='0';
661              input_reg_en <='0';
662              uart_avalon_chipselect <='1';
663              uart_avalon_read_n <='1';
664             RD_DATA_en<='0';
665              record_counter_en <='0';
666              UART_fifo_flushed_en <='0';
667
668
669              addr_cnt_rst <='0';
670
671              next_state<=WAIT_DELAY2;
672
673
674         when WAIT_DELAY2 =>
675
676              FSM_reset <='0';
677             in_mux_cnt_en<='0';
678             address_counter_up_en <='0';
679              address_counter_down_en <='0';
680             sdram_avalon_byteenable_n<="00";
681             sdram_avalon_read_n <= '1';
682             sdram_avalon_write_n <= '1';
683              uart_avalon_address<="000";
684              uart_avalon_begintransfer <='0';
685              uart_avalon_write_n <='1';
686              data_read_set <='0';
687              input_reg_en <='0';
688              uart_avalon_chipselect <='1';
689              uart_avalon_read_n <='1';
690             RD_DATA_en<='0';
691              record_counter_en <='0';
692              UART_fifo_flushed_en <='0';
693
694
695              addr_cnt_rst <='0';
696
697              if uart_counter = '1' then
698
699                   next_state<=CHECK_STOP;
700
701              else
702
703                   next_state<=WAIT_UART;
```

```
704
705             end if;
706
707
708         when CHECK_STOP =>
709
710             FSM_reset<='0';
711             in_mux_cnt_en<='0';
712             address_counter_up_en<='0';
713             address_counter_down_en<='0';
714             sdram_avalon_byteenable_n<="00";
715             sdram_avalon_read_n <= '1';
716             sdram_avalon_write_n <= '1';
717             uart_avalon_address<="000";
718             uart_avalon_write_n<='1';
719             uart_avalon_begintransfer<='0';
720             data_read_set<='0';
721             input_reg_en<='0';
722             uart_avalon_chipselect<='0';
723             uart_avalon_read_n<='1';
724             RD_DATA_en<='0';
725             record_counter_en<='0';
726             UART_fifo_flushed_en<='0';
727
728             addr_cnt_rst<='0';
729
730             if stop_read = '1' then
731
732                 next_state<= HALT;
733
734             else
735
736                 next_state<= RD;
737
738             end if;
739
740         when HALT =>
741
742             FSM_reset<='0';
743             in_mux_cnt_en<='0';
744             address_counter_up_en<='0';
745             address_counter_down_en<='0';
746             sdram_avalon_byteenable_n<="00";
747             sdram_avalon_read_n <= '1';
748             sdram_avalon_write_n <= '1';
749             uart_avalon_address<="000";
750             uart_avalon_write_n<='1';
751             uart_avalon_begintransfer<='0';
752             data_read_set<='0';
```

```
753                 input_reg_en <= '0';
754                 uart_avalon_chipselect <= '0';
755                 uart_avalon_read_n <= '1';
756                 RD_DATA_en<= '0';
757                 record_counter_en <= '0';
758                 UART_fifo_flushed_en <= '0';
759
760
761                 addr_cnt_rst <= '0';
762
763                 if fault_reset = '1' then
764
765                     next_state<=RESET;
766
767                 else
768
769                     next_state<=HALT;
770
771                 end if;
772
773             when ADDRESS_RECOVERY =>
774
775                 FSM_reset <= '0';
776                 in_mux_cnt_en <= '0';
777                 address_counter_up_en <= '0';
778                 address_counter_down_en <= '1';
779                 sdram_avalon_byteenable_n<="00";
780                 sdram_avalon_read_n <= '1';
781                 sdram_avalon_write_n <= '1';
782                 uart_avalon_address<="000";
783                 uart_avalon_write_n <= '1';
784                 uart_avalon_begintransfer <= '0';
785                 data_read_set <= '0';
786                 input_reg_en <= '0';
787                 uart_avalon_chipselect <= '0';
788                 uart_avalon_read_n <= '1';
789                 RD_DATA_en<= '0';
790                 record_counter_en <= '0';
791                 UART_fifo_flushed_en <= '0';
792
793
794
795                 addr_cnt_rst <= '0';
796
797                 next_state<=RD;
798
799
800
801
```

```
802         when others =>
803
804             FSM_reset<='0';
805             in_mux_cnt_en<='0';
806             address_counter_up_en<='0';
807             address_counter_down_en<='0';
808             sdram_avalon_byteenable_n<="00";
809             sdram_avalon_read_n <= '1';
810             sdram_avalon_write_n <= '1';
811             uart_avalon_address<="000";
812             uart_avalon_write_n<='1';
813             data_read_set<='0';
814             input_reg_en<='0';
815             uart_avalon_chipselect<='0';
816             uart_avalon_read_n<='1';
817             RD_DATA_en<='0';
818             record_counter_en<='0';
819             UART_fifo_flushed_en<='0';
820
821
822             addr_cnt_rst<='0';
823
824
825             next_state<=RESET;
826
827     end case;
828
829 end process;
830
831 -- INPUT REGISTERS --
832 process(sys_clk,FSM_reset)
833 begin
834     if FSM_reset='1' then
835
836         WR_DATA1_REG <= (others => '0');
837         WR_DATA2_REG <= (others => '0');
838         WR_DATA3_REG <= (others => '0');
839         WR_DATA4_REG <= (others => '0');
840         WR_DATA5_REG <= (others => '0');
841         WR_DATA6_REG <= (others => '0');
842         fault_min_reg <= (others => '0');
843         fault_max_reg <= (others => '0');
844         ARC_DET_reg <= '0';
845
846
847
848     elsif rising_edge(sys_clk) then
849
850         if input_reg_en = '1' then
```

```
851
852                 WR_DATA1_REG <= WR_DATA1;
853                 WR_DATA2_REG <= WR_DATA2;
854                 WR_DATA3_REG <= WR_DATA3;
855                 WR_DATA4_REG <= WR_DATA4;
856                 WR_DATA5_REG <= WR_DATA5;
857                 WR_DATA6_REG <= WR_DATA6;
858                 fault_min_reg <= fault_min;
859                 fault_max_reg <= fault_max;
860                 ARC_DET_reg <= ARC_DET;
861
862             end if;
863
864         end if;
865
866 end process;
867
868 -- INPUT MULTIPLEXER --
869 process (sdram_avalon_address,WR_DATA1_REG, WR_DATA2_REG, WR_DATA3_REG,
        WR_DATA4_REG, WR_DATA5_REG, WR_DATA6_REG, in_mux_cnt, ARC_DET_REG,
        fault_max_reg, fault_min_reg)
870 begin
871
872     case in_mux_cnt is
873
874         when "000" => sdram_avalon_writedata<=sdram_avalon_address(15
        downto 0);
875         when "001" => sdram_avalon_writedata<=sdram_avalon_address(15
        downto 0);
876         when "010" => sdram_avalon_writedata<=sdram_avalon_address(15
        downto 0);
877         when "011" => sdram_avalon_writedata<=sdram_avalon_address(15
        downto 0);
878         when "100" => sdram_avalon_writedata<=sdram_avalon_address(15
        downto 0);
879         when others => sdram_avalon_writedata<= (others=>'0');
880
881     end case;
882
883 end process;
884
885 -- INPUT MULTIPLEXER SELECTOR GENERATION WITH A COUNTER --
886 process(sys_clk, FSM_reset)
887
888 begin
889
890     if FSM_reset = '1' then
891
892         in_mux_cnt<= (others =>'0');
```

```
893
894     elsif rising_edge(sys_clk) then
895
896         if sdram_avalon_waitrequest='0' and sdram_avalon_write_n='0'
        then
897
898             if  in_mux_cnt_tc = '1' then
899
900                 in_mux_cnt <= (others => '0');
901
902             elsif (in_mux_cnt_en = '1') then
903
904                 in_mux_cnt <= in_mux_cnt + 1;
905
906             end if;
907
908         end if;
909
910     end if;
911
912 end process;
913
914 in_mux_cnt_tc<= in_mux_cnt(2) and not(in_mux_cnt(1)) and not(in_mux_cnt
        (0));
915
916 process(sys_clk,FSM_reset)
917 begin
918
919     if FSM_reset='1' then
920
921         stop_read_address_register <= (others=>'1');
922
923     elsif rising_edge(sys_clk) then
924
925         if input_reg_en ='1' then
926
927             stop_read_address_register<=number_of_records;
928
929         elsif record_counter = "100" and record_counter_en = '1' then
930
931             stop_read_address_register<=stop_read_address_register-1;
932
933         end if;
934
935     end if;
936
937 end process;
938
939
```

```
940 -- RECORD COUNTER --
941 process(sys_clk, FSM_reset)
942 begin
943
944 if FSM_reset = '1' then
945
946     record_counter <= (others => '0');
947
948 elsif rising_edge(sys_clk) then
949
950     if record_counter = "100" and record_counter_en='1' then
951
952         record_counter<=(others=>'0');
953
954     elsif record_counter_en = '1' then
955
956         record_counter<= record_counter + 1;
957
958     end if;
959
960 end if;
961
962 end process;
963
964
965 process(FSM_reset,sys_clk)
966 begin
967 if(FSM_reset = '1') then
968
969     stop_read<='0';
970
971 elsif rising_edge(sys_clk) then
972
973     if  stop_read_address_register = "0000000000000000" then
974
975         stop_read<='1';
976
977     end if;
978
979 end if;
980
981 end process;
982
983 -- DATA READ FROM RAM REGISTER --
984 process(sys_clk,pll_locked)
985
986 begin
987
988     if pll_locked = '0' then
```

```
989
990         RD_DATA<=(others => '0');
991
992         debug<='0';
993
994     elsif rising_edge(sys_clk) then
995
996         if  RD_DATA_en = '1' then
997
998             debug <= not(debug);
999             RD_DATA<=sdram_avalon_readdata;
1000
1001        end if;
1002
1003    end if;
1004
1005end process;
1006
1007-- UART AVALON WRITE DATA --
1008process(uart_avalon_address, RD_DATA, uart_avalon_writedata)
1009begin
1010
1011    if uart_avalon_address="001" then
1012
1013        if uart_counter='0' then
1014            uart_avalon_writedata(7 downto 0)<=RD_DATA(15 downto 8);
1015        else
1016            uart_avalon_writedata(7 downto 0)<=RD_DATA(7 downto 0);
1017        end if;
1018
1019    else
1020
1021        uart_avalon_writedata(7 downto 0)<="00000000";
1022
1023    end if;
1024
1025end process;
1026
1027-- ADDRESS COUNTER --
1028process(sys_clk,FSM_reset)
1029begin
1030
1031    if FSM_reset = '1' then
1032
1033        address_counter<=(others => '0');
1034
1035    elsif rising_edge(sys_clk) then
1036
```

```
1037        if (address_counter_up_en = '1' and sdram_avalon_waitrequest =
        '0') then
1038
1039            address_counter <= address_incr;
1040
1041        elsif (address_counter_down_en = '1' and
        sdram_avalon_waitrequest = '0') then
1042
1043            address_counter<=address_counter - 1;
1044
1045        end if;
1046
1047    end if;
1048
1049 end process;
1050
1051 -- ADDRESS COUNTER ADDER --
1052 address_incr<=address_counter+1;
1053
1054 -- DATA READ SIGNAL SET RESET --
1055 process(sys_clk,pll_locked)
1056 begin
1057 if pll_locked='0' then
1058
1059     data_read_sig<='0';
1060
1061 elsif rising_edge(sys_clk) then
1062
1063     if data_read_set = '1' then
1064
1065         data_read_sig<='1';
1066
1067     elsif (data_read_reset = '1') then
1068
1069         data_read_sig<='0';
1070
1071     end if;
1072 end if;
1073
1074 end process;
1075
1076 -- DATA AVAILABLE EDGE DETECTOR --
1077 process(sys_clk,FSM_reset)
1078 begin
1079
1080     if FSM_reset ='1' then
1081
1082         data_available_edge_1 <= '0';
1083        data_available_edge_2 <= '0';
```

```
1084
1085        elsif rising_edge(sys_clk) then
1086
1087            data_available_edge_1 <= data_available;
1088            data_available_edge_2 <= data_available_edge_1;
1089
1090        end if;
1091
1092    end process;
1093
1094    -- UART COUNTER --
1095    process(sys_clk,FSM_reset)
1096    begin
1097
1098        if FSM_reset='1' then
1099
1100            uart_counter<='0';
1101
1102        elsif rising_edge(sys_clk) then
1103
1104            if uart_avalon_begintransfer='1' then
1105
1106                uart_counter<=not(uart_counter);
1107
1108            end if;
1109
1110        end if;
1111
1112    end process;
1113
1114
1115    -- SYNCHRONIZER --
1116    process(sys_clk,FSM_reset)
1117    begin
1118
1119        if FSM_reset = '1' then
1120
1121            fault_signal<='0';
1122            fault_signal_sync1<='0';
1123            fault_reset<='0';
1124            fault_reset_sync1<='0';
1125            data_available<='0';
1126            data_available_sync1<='0';
1127            ARC_DET_sync1<='0';
1128            ARC_DET<='0';
1129
1130        elsif rising_edge(sys_clk) then
1131
1132            fault_signal<=fault_signal_sync1;
```

```
1133            fault_signal_sync1<=fault_signal_sync;
1134            fault_reset<=fault_reset_sync1;
1135            fault_reset_sync1<=fault_reset_sync;
1136            data_available<=data_available_sync1;
1137            data_available_sync1<=data_available_sync;
1138            fault_min_sync1 <= fault_min_sync;
1139            fault_min <= fault_min_sync1;
1140            fault_max_sync1 <= fault_max_sync;
1141            fault_max <= fault_max_sync1;
1142            ARC_DET_sync1 <= ARC_DET_sync;
1143            ARC_DET <= ARC_DET_sync1;
1144            number_of_records<=number_of_records_sync1;
1145            number_of_records_sync1<=number_of_records_sync;
1146
1147
1148        end if;
1149
1150    end process;
1151
1152
1153    -- DATA READ SET AND RESET SIGNAL GENERATION --
1154    data_read_reset<=not(data_read_set) and (not(data_available_edge_1) and
         data_available_edge_2);
1155
1156    data_read<=data_read_sig;
1157
1158
1159
1160    process(sys_clk,FSM_reset)
1161    begin
1162
1163        if fsm_reset = '1' then
1164
1165            prova<='0';
1166
1167        elsif rising_edge(sys_clk) then
1168
1169            prova<=sdram_avalon_readdatavalid;
1170
1171        end if;
1172
1173    end process;
1174
1175    LED_out<=stop_read;
1176
1177    end behav;
```

A.1.4 Sorgente I2C

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.numeric_std.all;

entity I2C_Interface is
port(
    clk : in std_logic;
    rst_n : in std_logic;
    SCL : inout std_logic;
    SDA : inout std_logic;

    ADC_DATA1 : in std_logic_vector(11 downto 0);
    ADC_DATA2 : in std_logic_vector(11 downto 0);
    ADC_DATA3 : in std_logic_vector(11 downto 0);
    ADC_DATA4 : in std_logic_vector(11 downto 0);
    ADC_DATA5 : in std_logic_vector(11 downto 0);
    ADC_DATA6 : in std_logic_vector(11 downto 0);
    status : in std_logic_vector(7 downto 0);

    ADC_TH_MAX1 : out std_logic_vector(11 downto 0);
    ADC_TH_MAX2 : out std_logic_vector(11 downto 0);
    ADC_TH_MAX3 : out std_logic_vector(11 downto 0);
    ADC_TH_MAX4 : out std_logic_vector(11 downto 0);
    ADC_TH_MAX5 : out std_logic_vector(11 downto 0);
    ADC_TH_MAX6 : out std_logic_vector(11 downto 0);
    ADC_TH_MIN1 : out std_logic_vector(11 downto 0);

    ADC_TH_MIN2 : out std_logic_vector(11 downto 0);
    ADC_TH_MIN3 : out std_logic_vector(11 downto 0);
    ADC_TH_MIN4 : out std_logic_vector(11 downto 0);
    ADC_TH_MIN5 : out std_logic_vector(11 downto 0);
    ADC_TH_MIN6 : out std_logic_vector(11 downto 0);

    TH_MIN_EN : out std_logic_vector(7 downto 0);
    TH_MAX_EN : out std_logic_vector(7 downto 0);

    commands : out std_logic_vector(7 downto 0);
    faults_min : in std_logic_vector(7 downto 0);
    faults_max : in std_logic_vector(7 downto 0);

    debug_out : out std_logic_vector(15 downto 0)

);

end I2C_Interface;


```

```
48 architecture I2C of I2C_Interface is
49
50 component I2C_Slave is
51     port (
52         clk_clk                    : in  std_logic                     :=
       'X';              -- clk
53       i2c_conduit_data_in       : in  std_logic                     := '
       X';               -- conduit_data_in
54       i2c_conduit_clk_in        : in  std_logic                     := '
       X';               -- conduit_clk_in
55       i2c_conduit_data_oe       : out std_logic;
                         -- conduit_data_oe
56       i2c_conduit_clk_oe        : out std_logic;
                         -- conduit_clk_oe
57       i2c_avalon_address        : out std_logic_vector(31 downto 0);
                         -- address
58       i2c_avalon_read           : out std_logic;
                         -- read
59       i2c_avalon_readdata       : in  std_logic_vector(31 downto 0) := (
       others => 'X'); -- readdata
60       i2c_avalon_readdatavalid  : in  std_logic                     := '
       X';               -- readdatavalid
61       i2c_avalon_waitrequest    : in  std_logic                     := '
       X';               -- waitrequest
62       i2c_avalon_write          : out std_logic;
                         -- write
63       i2c_avalon_byteenable     : out std_logic_vector(3 downto 0);
                         -- byteenable
64       i2c_avalon_writedata      : out std_logic_vector(31 downto 0);
                         -- writedata
65       reset_reset_n             : in  std_logic                     := '
       X'                -- reset_n
66         );
67 end component I2C_Slave;
68
69 -- I2C CONTROLLER SIGNALS --
70     signal I2C_avalon_address        : std_logic_vector(31 downto 0);
71     signal I2C_avalon_read           : std_logic;
72     signal I2C_avalon_readdata       : std_logic_vector(31 downto 0);
73     signal I2C_avalon_readdatavalid  : std_logic;
74     signal I2C_avalon_waitrequest    : std_logic;
75     signal I2C_avalon_write          : std_logic;
76     signal I2C_avalon_byteenable     : std_logic_vector(3 downto 0);
77     signal I2C_avalon_writedata      : std_logic_vector(31 downto 0);
78     signal i2c_data_in               : std_logic;
79     signal i2c_clk_in                : std_logic;
80     signal i2c_data_oe               : std_logic;
81     signal i2c_clk_oe                : std_logic;
82
```

```
83 -- I/O REGISTERS --
84     signal ADC_DATA1_REG : std_logic_vector(15 downto 0);
85     signal ADC_DATA2_REG : std_logic_vector(15 downto 0);
86     signal ADC_DATA3_REG : std_logic_vector(15 downto 0);
87     signal ADC_DATA4_REG : std_logic_vector(15 downto 0);
88     signal ADC_DATA5_REG : std_logic_vector(15 downto 0);
89     signal ADC_DATA6_REG : std_logic_vector(15 downto 0);
90     signal faults_max_reg : std_logic_vector(7 downto 0);
91     signal faults_min_reg : std_logic_vector(7 downto 0);
92     signal ADC_TH_MAX1_REG : std_logic_vector(15 downto 0);
93     signal ADC_TH_MAX2_REG : std_logic_vector(15 downto 0);
94     signal ADC_TH_MAX3_REG : std_logic_vector(15 downto 0);
95     signal ADC_TH_MAX4_REG : std_logic_vector(15 downto 0);
96     signal ADC_TH_MAX5_REG : std_logic_vector(15 downto 0);
97     signal ADC_TH_MAX6_REG : std_logic_vector(15 downto 0);
98     signal ADC_TH_MIN1_REG : std_logic_vector(15 downto 0);
99     signal ADC_TH_MIN2_REG : std_logic_vector(15 downto 0);
100    signal ADC_TH_MIN3_REG : std_logic_vector(15 downto 0);
101    signal ADC_TH_MIN4_REG : std_logic_vector(15 downto 0);
102    signal ADC_TH_MIN5_REG : std_logic_vector(15 downto 0);
103    signal ADC_TH_MIN6_REG : std_logic_vector(15 downto 0);
104    signal commands_REG :  std_logic_vector(7 downto 0);
105    signal TH_MAX_EN_REG : std_logic_vector(7 downto 0);
106    signal TH_MIN_EN_REG : std_logic_vector(7 downto 0);
107    signal status_reg : std_logic_vector(7 downto 0);
108
109 -- CONTROL REGISTER --
110    signal i2c_avalon_read_reg : std_logic;
111
112
113
114    type avalon_data_t is array (9 downto 0) of std_logic_vector(31
       downto 0);
115    type avalon_data_prova_t is array (41 downto 0) of std_logic_vector
       (7 downto 0);
116
117    signal avalon_array_read : avalon_data_prova_t;
118    signal avalon_array_write : avalon_data_prova_t;
119
120
121 -- FSM SIGNALS --
122    type state_t is (RESET, WAIT_MASTER, DATA_TX, DATA_RX);
123    signal next_state, current_state: state_t;
124    signal input_en : std_logic;
125    signal output_en : std_logic;
126    signal dataread_en : std_logic;
127    signal datawrite_en : std_logic;
128
129 begin
```

```
130
131
132 I2C_Slave0 : I2C_Slave port map (
133
134     clk_clk                       => clk,
135     i2c_avalon_address            => I2C_avalon_address,
136     i2c_avalon_read               => I2C_avalon_read,
137     i2c_avalon_readdata           => I2C_avalon_readdata,
138     i2c_avalon_readdatavalid      => I2C_avalon_readdatavalid,
139     i2c_avalon_waitrequest        => I2C_avalon_waitrequest,
140     i2c_avalon_write              => I2C_avalon_write,
141     i2c_avalon_byteenable         => I2C_avalon_byteenable,
142     i2c_avalon_writedata          => I2C_avalon_writedata,
143     reset_reset_n                 => rst_n,
144     i2c_conduit_data_in           => SDA,
145     i2c_conduit_clk_in            => SCL,
146     i2c_conduit_data_oe           => i2c_data_oe,
147     i2c_conduit_clk_oe            => i2c_clk_oe
148
149 );
150
151 avalon_array_read(0) <= ADC_DATA1_REG(15 downto 8);
152 avalon_array_read(1) <= ADC_DATA1_REG(7 downto 0);
153 avalon_array_read(2) <= ADC_DATA2_REG(15 downto 8);
154 avalon_array_read(3) <= ADC_DATA2_REG(7 downto 0);
155 avalon_array_read(4) <= ADC_DATA5_REG(15 downto 8);
156 avalon_array_read(5) <= ADC_DATA5_REG(7 downto 0);
157 avalon_array_read(6) <= ADC_DATA3_REG(15 downto 8);
158 avalon_array_read(7) <= ADC_DATA3_REG(7 downto 0);
159 avalon_array_read(8) <= ADC_DATA6_REG(15 downto 8);
160 avalon_array_read(9) <= ADC_DATA6_REG(7 downto 0);
161 avalon_array_read(10)<= ADC_DATA4_REG(15 downto 8);
162 avalon_array_read(11)<= ADC_DATA4_REG(7 downto 0);
163 avalon_array_read(12)<= avalon_array_write(12);
164 avalon_array_read(13)<= avalon_array_write(13);
165 avalon_array_read(14)<= avalon_array_write(14);
166 avalon_array_read(15)<= avalon_array_write(15);
167 avalon_array_read(16)<= avalon_array_write(16);
168 avalon_array_read(17)<= avalon_array_write(17);
169 avalon_array_read(18)<= avalon_array_write(18);
170 avalon_array_read(19)<= avalon_array_write(19);
171 avalon_array_read(20)<= avalon_array_write(20);
172 avalon_array_read(21)<= avalon_array_write(21);
173 avalon_array_read(22)<= avalon_array_write(22);
174 avalon_array_read(23)<= avalon_array_write(23);
175 avalon_array_read(24)<= avalon_array_write(24);
176 avalon_array_read(25)<= avalon_array_write(25);
177 avalon_array_read(26)<= avalon_array_write(26);
178 avalon_array_read(27)<= avalon_array_write(27);
```

```
179 avalon_array_read(28)<= avalon_array_write(28);
180 avalon_array_read(29)<= avalon_array_write(29);
181 avalon_array_read(30)<= avalon_array_write(30);
182 avalon_array_read(31)<= avalon_array_write(31);
183 avalon_array_read(32)<= avalon_array_write(32);
184 avalon_array_read(33)<= avalon_array_write(33);
185 avalon_array_read(34)<= avalon_array_write(34);
186 avalon_array_read(35)<= avalon_array_write(35);
187 avalon_array_read(36)<= avalon_array_write(36);
188 avalon_array_read(37)<= avalon_array_write(37);
189 avalon_array_read(38)<= faults_max_reg;
190 avalon_array_read(39)<= faults_min_reg;
191 avalon_array_read(40)<= status_reg;
192 avalon_array_read(41)<= avalon_array_write(40);
193
194
195 --ADC_DATA1_REG <= avalon_array_write(0)(31 downto 16);
196 --ADC_DATA2_REG <= avalon_array_write(0)(15 downto 0);
197 --ADC_DATA3_REG <= avalon_array_write(1)(31 downto 16);
198 --ADC_DATA4_REG <= avalon_array_write(1)(15 downto 0);
199 --ADC_DATA5_REG <= avalon_array_write(2)(31 downto 16);
200 --ADC_DATA6_REG <= avalon_array_write(2)(15 downto 0);
201 ADC_TH_MAX2<= avalon_array_write(12)(3 downto 0)& avalon_array_write
      (13);
202 ADC_TH_MAX1<= avalon_array_write(14)(3 downto 0)& avalon_array_write
      (15);
203 ADC_TH_MAX3<= avalon_array_write(16)(3 downto 0)& avalon_array_write
      (17);
204 ADC_TH_MAX5<= avalon_array_write(18)(3 downto 0)& avalon_array_write
      (19);
205 ADC_TH_MAX4<= avalon_array_write(20)(3 downto 0)& avalon_array_write
      (21);
206 ADC_TH_MAX6<= avalon_array_write(22)(3 downto 0)& avalon_array_write
      (23);
207 ADC_TH_MIN2<= avalon_array_write(24)(3 downto 0)& avalon_array_write
      (25);
208 ADC_TH_MIN1<= avalon_array_write(26)(3 downto 0)& avalon_array_write
      (27);
209 ADC_TH_MIN3<= avalon_array_write(28)(3 downto 0)& avalon_array_write
      (29);
210 ADC_TH_MIN5<= avalon_array_write(30)(3 downto 0)& avalon_array_write
      (31);
211 ADC_TH_MIN4<= avalon_array_write(32)(3 downto 0)& avalon_array_write
      (33);
212 ADC_TH_MIN6<= avalon_array_write(34)(3 downto 0) & avalon_array_write
      (35);
213 TH_MAX_EN <= avalon_array_write(36);
214 TH_MIN_EN <= avalon_array_write(37);
215 commands <= avalon_array_write(41);
```

```
216
217 -- TRISTATE OUTPUT BUFFER --
218 SDA<='0' when i2c_data_oe = '1' else 'Z';
219 SCL<='0' when i2c_clk_oe = '1' else 'Z';
220
221 -- FSM SEQUENTIAL --
222 process(clk,rst_n)
223
224 begin
225
226 if rst_n='0' then
227
228     current_state<=RESET;
229
230 elsif rising_edge(clk) then
231
232     current_state<=next_state;
233
234 end if;
235
236 end process;
237
238
239 -- CONTROL REGISTERS --
240
241 process (rst_n, clk)
242 begin
243
244     if rst_n = '0' then
245
246         i2c_avalon_read_reg<= '0';
247
248     elsif rising_edge(clk) then
249
250         i2c_avalon_read_reg<=i2c_avalon_read;
251
252     end if;
253
254
255 end process;
256
257
258
259 -- I/O REGISTERS --
260
261 process(clk,rst_n)
262
263 begin
264
```

```
265     if rst_n = '0' then
266 
267         ADC_DATA1_REG<=(others =>'0');
268         ADC_DATA2_REG<=(others =>'0');
269         ADC_DATA3_REG<=(others =>'0');
270         ADC_DATA4_REG<=(others =>'0');
271         ADC_DATA5_REG<=(others =>'0');
272         ADC_DATA6_REG<=(others =>'0');
273          faults_max_reg<=(others => '0');
274          faults_min_reg<=(others =>'0');
275          status_reg<=(others => '0');
276 
277     elsif rising_edge(clk) then
278 
279 
280             ADC_DATA1_REG<=std_logic_vector(to_unsigned(2,8)) &
    std_logic_vector(to_unsigned(1,8));
281             ADC_DATA2_REG<=std_logic_vector(to_unsigned(4,8)) &
    std_logic_vector(to_unsigned(3,8));
282             ADC_DATA3_REG<=std_logic_vector(to_unsigned(6,8)) &
    std_logic_vector(to_unsigned(5,8));--"0000" & ADC_DATA3;
283             ADC_DATA4_REG<=std_logic_vector(to_unsigned(8,8)) &
    std_logic_vector(to_unsigned(7,8));--"0000" & ADC_DATA4;
284             ADC_DATA5_REG<=std_logic_vector(to_unsigned(10,8)) &
    std_logic_vector(to_unsigned(9,8));--"0000" & ADC_DATA5;
285             ADC_DATA6_REG<=std_logic_vector(to_unsigned(12,8)) &
    std_logic_vector(to_unsigned(11,8));--"0000" & ADC_DATA6;
286             faults_max_reg<=faults_max;--"01010101";
287             faults_min_reg<=faults_min;--"11101111";
288             status_reg<=status;
289 
290     end if;
291 
292 end process;
293 
294 
295 process(i2c_avalon_address,i2c_avalon_read_reg,avalon_array_read)
296 begin
297 
298 if i2c_avalon_read_reg = '1' then
299 
300     i2c_avalon_readdata <= avalon_array_read(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))) & avalon_array_read(to_integer(
    unsigned(i2c_avalon_address(7 downto 0)))+1) & avalon_array_read(
    to_integer(unsigned(i2c_avalon_address(7 downto 0)))+2) &
    avalon_array_read(to_integer(unsigned(i2c_avalon_address(7 downto
    0)))+3);
301 
302 
```

```
303 else
304
305     i2c_avalon_readdata<=(others => '1');
306
307 end if;
308
309 end process;
310
311
312
313 process (rst_n,clk)
314
315 begin
316
317     if rst_n = '0' then
318
319         avalon_array_write<=(others =>(others=>'1'));
320
321     elsif rising_edge(clk) then
322
323         if i2c_avalon_write = '1' then
324
325             case i2c_avalon_byteenable is
326
327                 when "0001"=>
328
329                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0))))<=i2c_avalon_writedata(7 downto 0)
    ;
330
331                 when "0010" =>
332
333                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+1)<=i2c_avalon_writedata(15 downto
     8);
334
335
336                 when "0011" =>
337
338                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0))))<=i2c_avalon_writedata(7 downto 0)
    ;
339                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+1)<=i2c_avalon_writedata(15 downto
     8);
340
341
342                 when "0100" =>
343
```

```
344                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+2)<=i2c_avalon_writedata(23 downto
     16);
345
346                 when "1000" =>
347
348                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+3)<=i2c_avalon_writedata(31 downto
     24);
349
350                 when "1100" =>
351
352                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+3)<=i2c_avalon_writedata(31 downto
     24);
353                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+2)<=i2c_avalon_writedata(23 downto
     16);
354
355                 when "1111" =>
356
357                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0))))<=i2c_avalon_writedata(7 downto 0)
    ;
358                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+1)<=i2c_avalon_writedata(15 downto
     8);
359                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+2)<=i2c_avalon_writedata(23 downto
     16);
360                     avalon_array_write(to_integer(unsigned(
    i2c_avalon_address(7 downto 0)))+3)<=i2c_avalon_writedata(31 downto
     24);
361
362                 when others =>
363
364             end case;
365
366         end if;
367
368     end if;
369
370 end process;
371
372 i2c_avalon_waitrequest<=not(rst_n);
373 i2c_avalon_readdatavalid<=i2c_avalon_read_reg;
374
375 end architecture;
```

A.1.5 Sorgente Fault Detection

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;


entity Fault_Detection is

port(

    clk : in std_logic;
    rst_n : in std_logic;
    fault_reset : in std_logic;
    BPS_Vmon : in std_logic_vector(11 downto 0);
    BPS_Imon : in std_logic_vector(11 downto 0);
    Kly_REF_Pw : in std_logic_vector(11 downto 0);
    Kly_Icoll : in std_logic_vector(11 downto 0);
    Kly_Ibody : in std_logic_vector(11 downto 0);
    MPS_Imon : in std_logic_vector(11 downto 0);

    BPS_Vmon_TH_MAX : in std_logic_vector(11 downto 0);
    BPS_Imon_TH_MAX : in std_logic_vector(11 downto 0);
    Kly_REF_Pw_TH_MAX : in std_logic_vector(11 downto 0);
    Kly_Icoll_TH_MAX : in std_logic_vector(11 downto 0);
    Kly_Ibody_TH_MAX : in std_logic_vector(11 downto 0);
    MPS_Imon_TH_MAX : in std_logic_vector(11 downto 0);

    BPS_Vmon_TH_MIN : in std_logic_vector(11 downto 0);
    BPS_Imon_TH_MIN : in std_logic_vector(11 downto 0);
    Kly_REF_Pw_TH_MIN : in std_logic_vector(11 downto 0);
    Kly_Icoll_TH_MIN : in std_logic_vector(11 downto 0);
    Kly_Ibody_TH_MIN : in std_logic_vector(11 downto 0);
    MPS_Imon_TH_MIN : in std_logic_vector(11 downto 0);

    fault_min : out std_logic_vector(7 downto 0);
    fault_max : out std_logic_vector(7 downto 0);
    status : out std_logic_vector(7 downto 0);

    fault_max_en : in std_logic_vector(7 downto 0);
    fault_min_en : in std_logic_vector(7 downto 0);
    fault_signal : out std_logic;


    ARC_DET : in std_logic;
    INHIBIT_1 : out std_logic; -- IPA INHIBIT
    INHIBIT_2 : out std_logic  -- BPS INHIBIT

);
```

```
48
49 end Fault_Detection;
50
51 architecture threshold_comparator of Fault_Detection is
52
53 signal fault_min_temp : std_logic_vector(7 downto 0);
54 signal fault_max_temp : std_logic_vector(7 downto 0);
55 signal INHIBIT_1_set : std_logic;
56 signal INHIBIT_2_set : std_logic;
57 signal INHIBIT_1_reset : std_logic;
58 signal INHIBIT_2_reset : std_logic;
59
60 begin
61
62 fault_min_temp(1)<= '1' when fault_min_en(1)='1' and (BPS_Vmon <
       BPS_Vmon_TH_MIN) else '0';
63 fault_min_temp(0)<= '1' when fault_min_en(0)='1' and (BPS_Imon <
       BPS_Imon_TH_MIN) else '0';
64 fault_min_temp(2)<= '1' when fault_min_en(2)='1' and (Kly_REF_Pw <
       Kly_REF_Pw_TH_MIN) else '0';
65 fault_min_temp(4)<= '1' when fault_min_en(4)='1' and (Kly_Icoll <
       Kly_Icoll_TH_MIN) else '0';
66 fault_min_temp(3)<= '1' when fault_min_en(3)='1' and (Kly_Ibody <
       Kly_Ibody_TH_MIN) else '0';
67 fault_min_temp(5)<= '1' when fault_min_en(5)='1' and (MPS_Imon <
       MPS_Imon_TH_MIN) else '0';
68
69
70 fault_max_temp(1)<= '1' when fault_max_en(1)='1' and (BPS_Vmon >
       BPS_Vmon_TH_MAX) else '0';
71 fault_max_temp(0)<= '1' when fault_max_en(0)='1' and (BPS_Imon >
       BPS_Imon_TH_MAX) else '0';
72 fault_max_temp(2)<= '1' when fault_max_en(2)='1' and (Kly_REF_Pw >
       Kly_REF_Pw_TH_MAX) else '0';
73 fault_max_temp(4)<= '1' when fault_max_en(4)='1' and (Kly_Icoll >
       Kly_Icoll_TH_MAX) else '0';
74 fault_max_temp(3)<= '1' when fault_max_en(3)='1' and (Kly_Ibody >
       Kly_Ibody_TH_MAX) else '0';
75 fault_max_temp(5)<= '1' when fault_max_en(5)='1' and (MPS_Imon >
       MPS_Imon_TH_MAX) else '0';
76
77
78 fault_signal<= fault_min_temp(0) or fault_min_temp(1) or fault_min_temp
       (2) or fault_min_temp(3) or fault_min_temp(4) or fault_min_temp(5)
       or
79                     fault_max_temp(0) or fault_max_temp(1) or
       fault_max_temp(2) or fault_max_temp(3) or fault_max_temp(4) or
       fault_max_temp(5);
80
```

```
81  fault_max<=fault_max_temp;
82  fault_min<=fault_min_temp;
83
84
85  process(clk, rst_n)
86  begin
87
88      if rst_n = '0' then
89
90          INHIBIT_1<='0';
91          INHIBIT_2<='0';
92
93      elsif rising_edge(clk) then
94
95          if INHIBIT_1_reset = '1' then
96
97              INHIBIT_1<='0';
98
99          elsif INHIBIT_1_set = '1' then
100
101             INHIBIT_1<='1';
102
103         end if;
104
105
106         if INHIBIT_2_reset = '1' then
107
108             INHIBIT_2<='0';
109
110         elsif INHIBIT_2_set = '1' then
111
112             INHIBIT_2<='1';
113
114         end if;
115
116     end if;
117
118
119 end process;
120
121
122
123
124 INHIBIT_1_reset<=fault_reset and not(INHIBIT_1_set);
125 INHIBIT_2_reset<=fault_reset and not(INHIBIT_2_set);
126
127 INHIBIT_1_set<=fault_max_temp(2) or fault_max_temp(3) or ARC_DET;
128 INHIBIT_2_set<=fault_max_temp(1) or fault_max_temp(0) or fault_max_temp
    (4) or fault_max_temp(3) or fault_min_temp(5);
```

```
129 status<="000000" & ARC_DET & (fault_min_temp(0) or fault_min_temp(1) or
        fault_min_temp(2) or fault_min_temp(3) or fault_min_temp(4) or
    fault_min_temp(5) or
130                     fault_max_temp(0) or fault_max_temp(1) or
    fault_max_temp(2) or fault_max_temp(3) or fault_max_temp(4) or
    fault_max_temp(5));
131
132
133
134 end threshold_comparator;
```

A.2 Codice di Test per ADC

A.2.1 Codice comunicazione USB

```
1  #include<stdio.h>
2  #include"ftd2xx.h"
3  #include<signal.h>
4  #include <windows.h>
5
6
7  bool terminate = false;
8
9  void sig_func(int signum);
10 byte* buffer;
11 FT_HANDLE ftHandle;
12 FILE* dumpFile;
13 FILE* decodeFile;
14 UINT16 sample;
15 byte temp;
16
17
18 int main() {
19
20     FT_STATUS ftStatus;
21     DWORD numDevs;
22     DWORD bytesToRead;
23     DWORD bytesRead;
24     DWORD bytesWritten;
25     DWORD totalBytesWritten = 0;
26     DWORD threadID;
27     DWORD totalBytesRead = 0;
28     char deviceName[18];
29     FT_DEVICE_LIST_INFO_NODE list[10];
30     errno_t errorOpeningFile;
31     int i, stopWriteFile;
32     bool oddBytes = false;
```

```
33
34
35      signal(SIGBREAK, sig_func);
36
37
38      ftStatus = FT_CreateDeviceInfoList(&numDevs);
39
40      if (ftStatus == FT_OK) {
41
42          printf("Num devs = %d\n", numDevs);
43
44      }
45      else {
46
47          printf("Errore nella funzione FT_CreateDeviceInfoList errore
    numero %d\n", ftStatus);
48          return -1;
49      }
50
51      ftStatus = FT_GetDeviceInfoList(list, &numDevs);
52      if (ftStatus == FT_OK) {
53
54      }
55      else {
56
57          printf("Errore nella funzione FT_GetDeviceInfoList errore
    numero %d\n", ftStatus);
58          return -1;
59      }
60      sprintf_s(deviceName, 18, "FT230X Basic UART");
61      ftStatus = FT_OpenEx((LPVOID)deviceName, FT_OPEN_BY_DESCRIPTION, &
    ftHandle);
62
63      FT_ResetDevice(ftHandle);
64
65
66      if (ftStatus == FT_OK) {
67
68      }
69      else {
70
71          printf("Impossibile aprire il dispositivo \"%s\" errore numero
    %d\n", deviceName,ftStatus);
72          return -1;
73      }
74
75      ftStatus = FT_SetLatencyTimer(ftHandle, 1);
76
77      if (ftStatus == FT_OK) {
```

```
78
79      }
80      else {
81
82          printf("Errore nella funzione FT_SetLatencyTimer errore numero
        %d\n", ftStatus);
83          return -1;
84      }
85
86      ftStatus = FT_SetUSBParameters(ftHandle, 256, 256);
87
88      if (ftStatus == FT_OK) {
89
90      }
91      else {
92
93          printf("Errore nella funzione FT_SetUSBParameters errore numero
        %d\n", ftStatus);
94          return -1;
95      }
96
97      FT_SetChars(ftHandle, false, 0, false, 0);
98
99      ftStatus = FT_SetBaudRate(ftHandle, 2995200);
100
101     if (ftStatus == FT_OK) {
102
103     }
104     else {
105
106         printf("Errore nella funzione FT_SetBaudRate errore numero %d\n
        ", ftStatus);
107         return -1;
108     }
109
110     errorOpeningFile = fopen_s(&dumpFile, "Uart.bin", "wb");
111
112     if (errorOpeningFile == 0) {
113
114         printf("Aperto il file di log\n");
115
116     }
117     else {
118
119         printf("Impossibile aprire il file di log errore numero %d\n",
        errorOpeningFile);
120         return -1;
121
122     }
```

```
123
124     buffer = (byte*)malloc(65536);
125
126
127
128     while (totalBytesWritten<8191) {
129         FT_GetQueueStatus(ftHandle, &bytesToRead);
130         FT_Read(ftHandle,buffer , bytesToRead, &bytesRead);
131         totalBytesRead += bytesRead;
132         bytesWritten = fwrite(buffer,sizeof(unsigned char) , bytesRead,
    dumpFile);
133         totalBytesWritten += bytesWritten;
134         if (bytesRead != 0) {
135             printf("%d Bytes read\n", totalBytesWritten);
136         }
137
138     }
139     printf("closing file \n");
140     Sleep(100);
141     fclose(dumpFile);
142     FT_Close(ftHandle);
143     free(buffer);
144
145     fopen_s(&dumpFile, "Uart.bin", "rb");
146     fopen_s(&decodeFile, "Decode.txt", "w");
147
148     buffer = (byte *) malloc(3);
149
150     while (!feof(dumpFile)) {
151
152         fread(buffer,3,1,dumpFile);
153
154         temp = buffer[1];
155
156         sample = (UINT16)((UINT16)buffer[0]<<4 | ((UINT16)((buffer[1] &
    0b11110000)))>>4);
157
158         fprintf_s(decodeFile,"%d\n",sample);
159
160         sample = (UINT16)(((UINT16)(temp & 0b00001111))<<8 | ((UINT16)(
    buffer[2])));
161
162         fprintf_s(decodeFile,"%d\n", sample);
163
164
165     }
166
167     free(buffer);
168
```

```
169     return 0;
170 }
171
172
173 void sig_func(int signum) {
174
175     printf("Termino\n");
176     terminate = false;
177     fclose(dumpFile);
178     FT_Close(ftHandle);
179     free(buffer);
180
181     fopen_s(&dumpFile, "Uart.bin", "rb");
182     fopen_s(&decodeFile, "Decode.txt", "w");
183
184     buffer = (byte*)malloc(3);
185
186     while (!feof(dumpFile)) {
187
188         fread(buffer, 3, 1, dumpFile);
189
190         temp = buffer[1];
191
192         sample = (UINT16)((UINT16)buffer[0] << 4 | ((UINT16)((buffer[1]
    & 0b11110000))) >> 4);
193
194         fprintf_s(decodeFile, "%d\n", sample);
195
196         sample = (UINT16)(((UINT16)(temp & 0b00001111)) << 8 | ((UINT16
    )(buffer[2])));
197
198         fprintf_s(decodeFile, "%d\n", sample);
199     }
200
201     free(buffer);
202
203     return;
204
205 }
```

A.2.2 Codice Analisi Dati

```
1 import numpy as np
2 import os
3 import matplotlib.pyplot as plt
4 import sys
5 import bitarray
6
```

```
ADCData = []
print("Write File")
logFile = open("C:/Users/crist/Documents/Visual Studio 2019/fastPRO/
    Project1/Decode.txt","r")
ADCFile = open("ADC.log","w")

sample = logFile.readline()
data_to_write = [2]
sample_integer = [4]

while sample:


    if int(sample) != 222:
        ADCFile.write(sample)

    sample = logFile.readline()

logFile.close()


ADCFile.close()

print("Reading file")
ADCFile = open("ADC.log", "r")
for x in ADCFile:
     ADCData.append(int(x[:len(x)-1]))

ADCData.pop(len(ADCData)-1)
print(max(ADCData))
print(min(ADCData))

(graph,b) = np.histogram(ADCData, bins=max(ADCData)-min(ADCData))
print(graph)
print(b)
plt.hist(ADCData, bins=b, rwidth=1)
plt.show()
plt.figure()
plt.plot(ADCData)
plt.ylim([0,max(ADCData)+100])
plt.show()
plt.figure()
F = np.fft.fft(ADCData)/len(ADCData)
F = F[range(int(len(ADCData)/2))]
tpCount = len(ADCData)
values = np.arange(int(tpCount/2))
timePeriod = tpCount/3430000
frequencies = values/timePeriod
plt.plot(frequencies,abs(F))
```

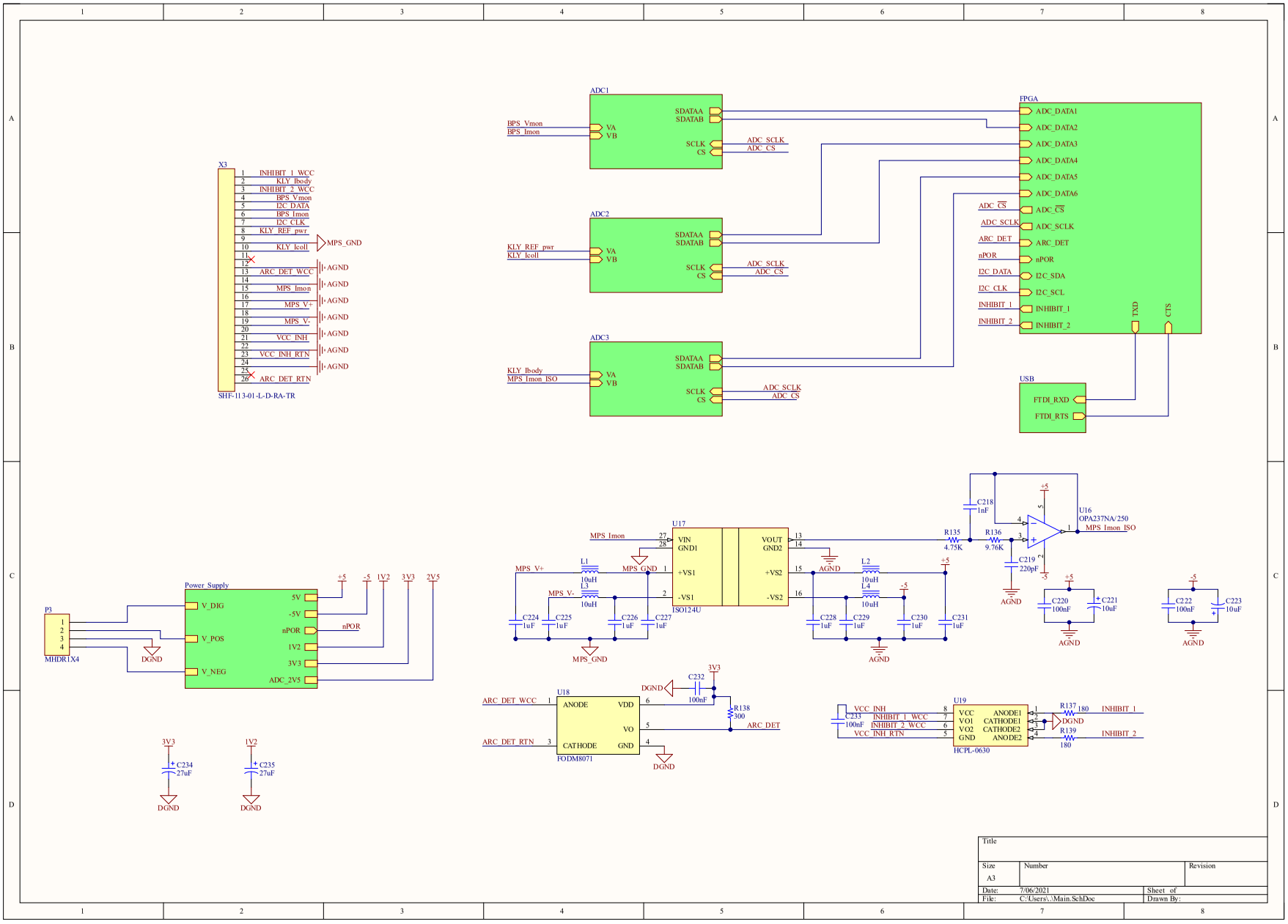
```
55 plt.show()
56 ADCFile.close()
```

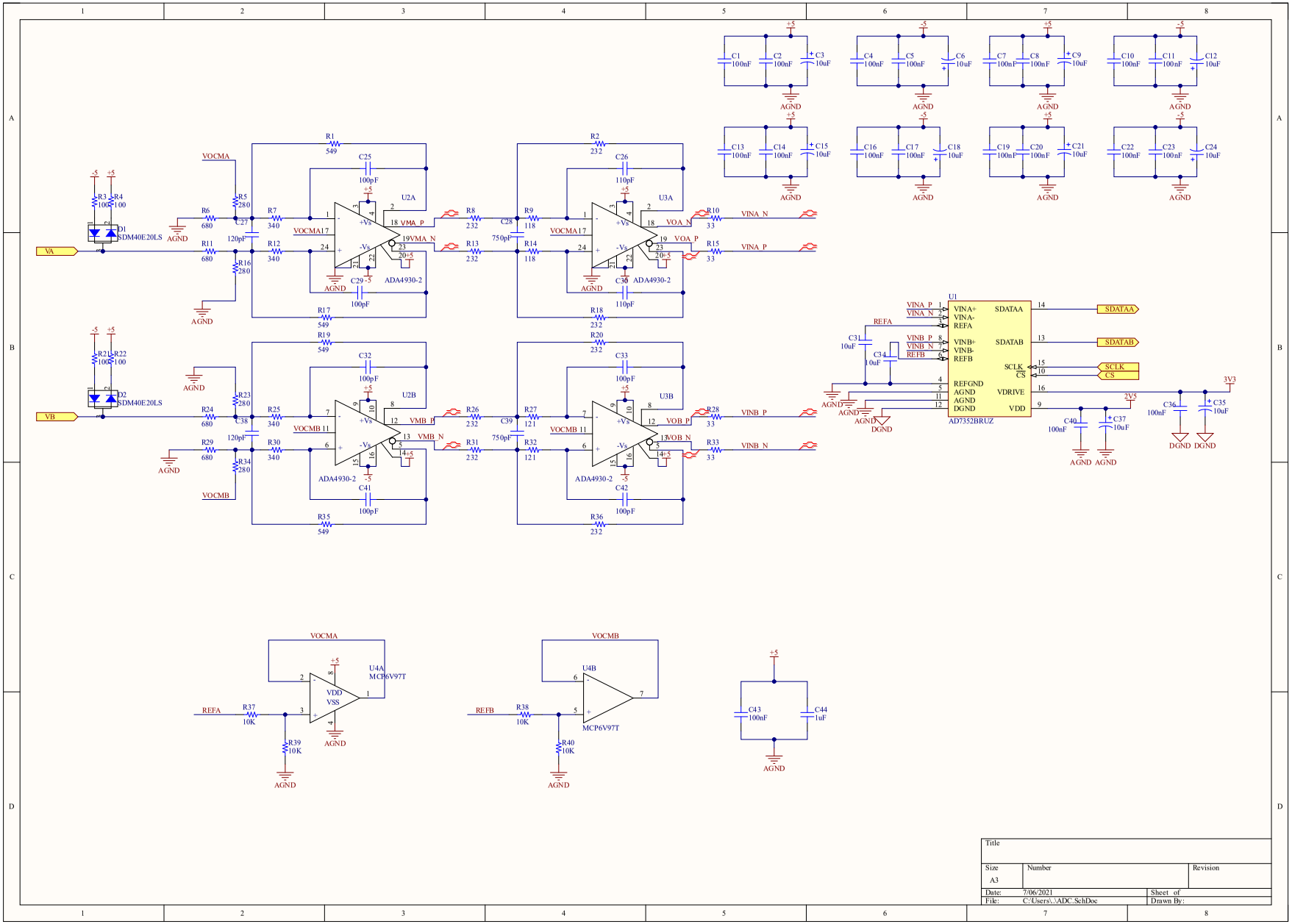
Appendice B

Schemi Elettrici

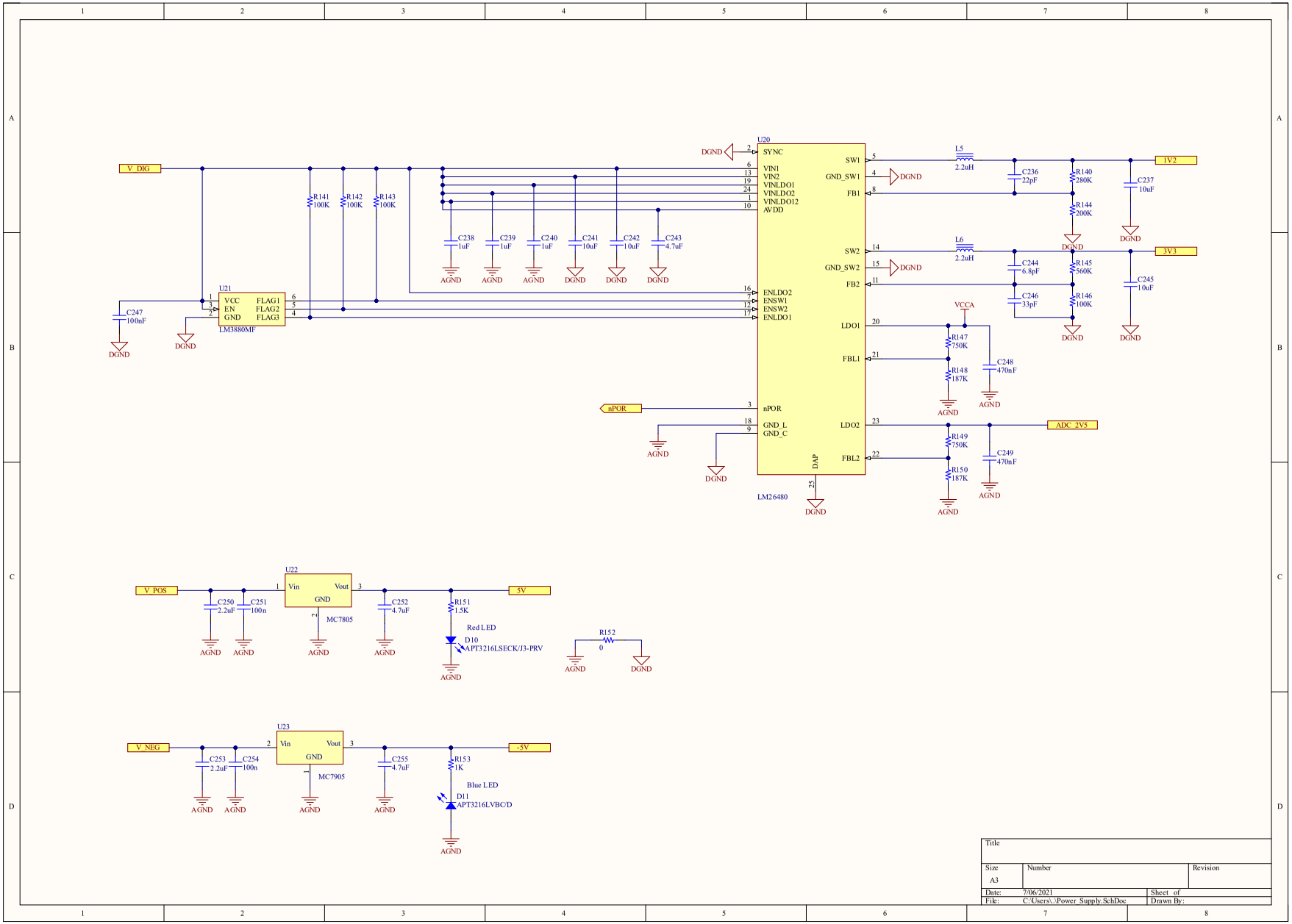
Nelle pagine seguenti di questa appendice sono presentati gli schemi elettrici che sono stati utilizzati per il successivo progetto del layout della scheda, sono in ordine:

1. Main (Sistema intero diviso in sotto-sistemi)
2. Front-end analogico
3. Power Supply
4. Circuito USB
5. Circuito FPGA
6. Disaccoppiamento FPGA

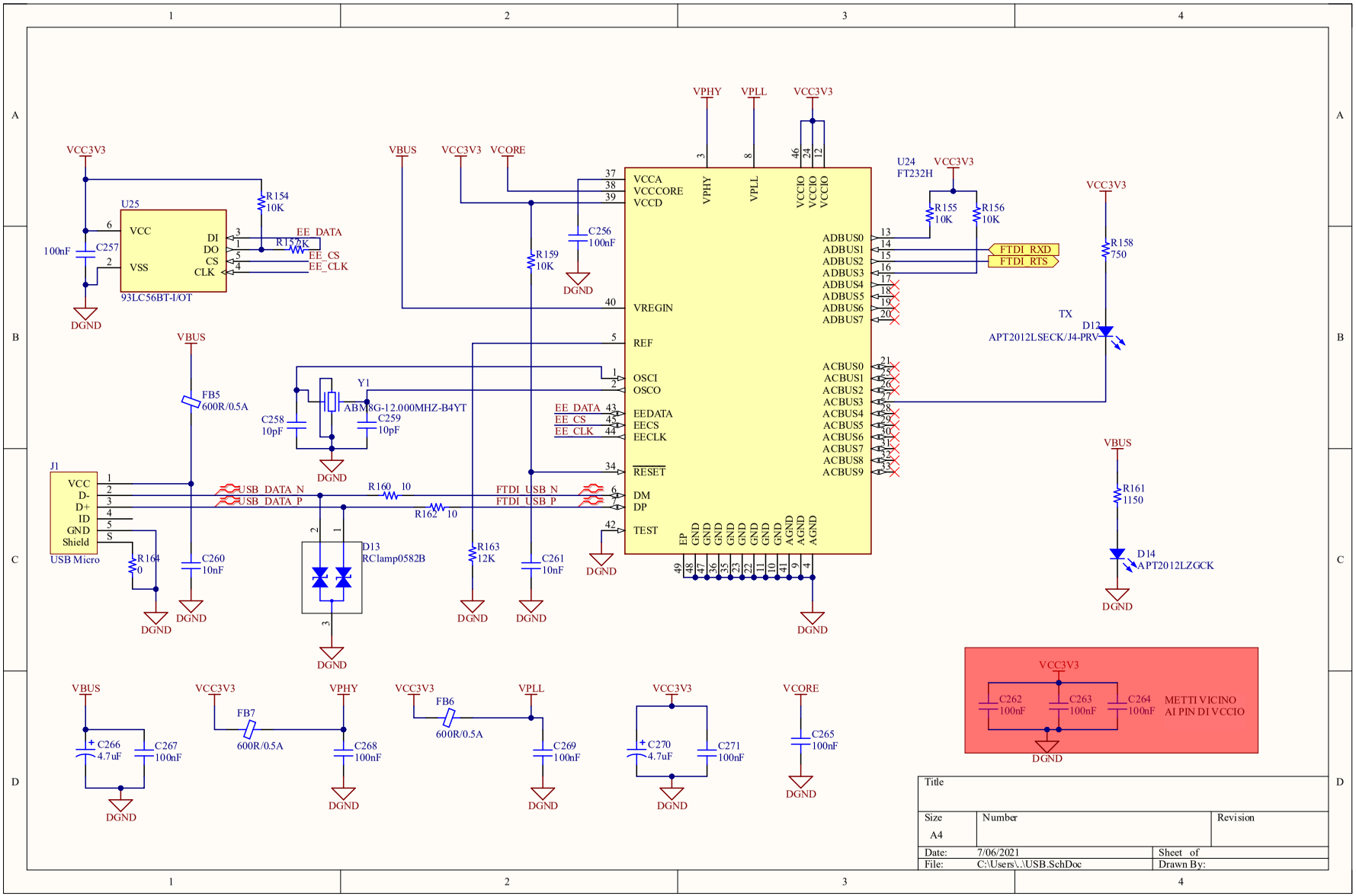

X3
1 INHIBIT_1_WCC
2 KLY_Ibody
3 INHIBIT_2_WCC
4 BPS_Vmon
5 I2C_DATA
6 BPS_Imon
7 I2C_CLK
8 KLY_REF_pwr
9
10 KLY_Icoil
11
12
13 ARC_DET_WCC
14
15 MPS_Imon
16
17 MPS_V+
18
19 MPS_V-
20
21 VCC_INH
22
23 VCC_INH_RTN
24
25
26 ARC_DET_RTN
MPS_GND
AGND
SHF-113-01-L-D-RA-TR
ADC1
ADC2
ADC3
SDATAA
SDATAB
SCLK
CS
VA
VB
BPS_Vmon
BPS_Imon
KLY_REF_pwr
KLY_Icoil
KLY_Ibody
MPS_Imon_ISO
ADC_SCLK
ADC_CS
FPGA
ADC_DATA1
ADC_DATA2
ADC_DATA3
ADC_DATA4
ADC_DATA5
ADC_DATA6
ADC_CS
ADC_SCLK
ARC_DET
nPOR
I2C_SDA
I2C_SCL
INHIBIT_1
INHIBIT_2
TXD
CTS
USB
FTDI_RXD
FTDI_RTS
P3
MHDR1X4
DGND
Power_Supply
V_DIG
V_POS
V_NEG
5V
-5V
nPOR
1V2
3V3
ADC_2V5
+5
-5
1V2
3V3
2V5
U17
ISO124U
VIN
GND1
+VS1
-VS1
VOUT
GND2
+VS2
-VS2
MPS_Imon
MPS_V+
MPS_V-
MPS_GND
AGND
L1
L2
L3
L4
10uH
C224 1uF
C225 1uF
C226 1uF
C227 1uF
C228 1uF
C229 1uF
C230 1uF
C231 1uF
R135 4.75K
R136 9.76K
C218 1nF
C219 220pF
U16
OPA237NA/250
MPS_Imon_ISO
C220 100nF
C221 10uF
C222 100nF
C223 10uF
U18
FODM8071
ANODE
CATHODE
VDD
VO
GND
ARC_DET_WCC
ARC_DET_RTN
C232 100nF
R138 300
ARC_DET
U19
HCPL-0630
VCC
VO1
VO2
GND
ANODE1
CATHODE1
CATHODE2
ANODE2
VCC_INH
INHIBIT_1_WCC
INHIBIT_2_WCC
VCC_INH_RTN
C233 100nF
R137 180
R139 180
INHIBIT_1
INHIBIT_2
C234 27uF
C235 27uF
Title
Size A3
Number
Revision
Date: 7/06/2021
File: C:\Users\..\Main.SchDoc
Sheet of
Drawn By:

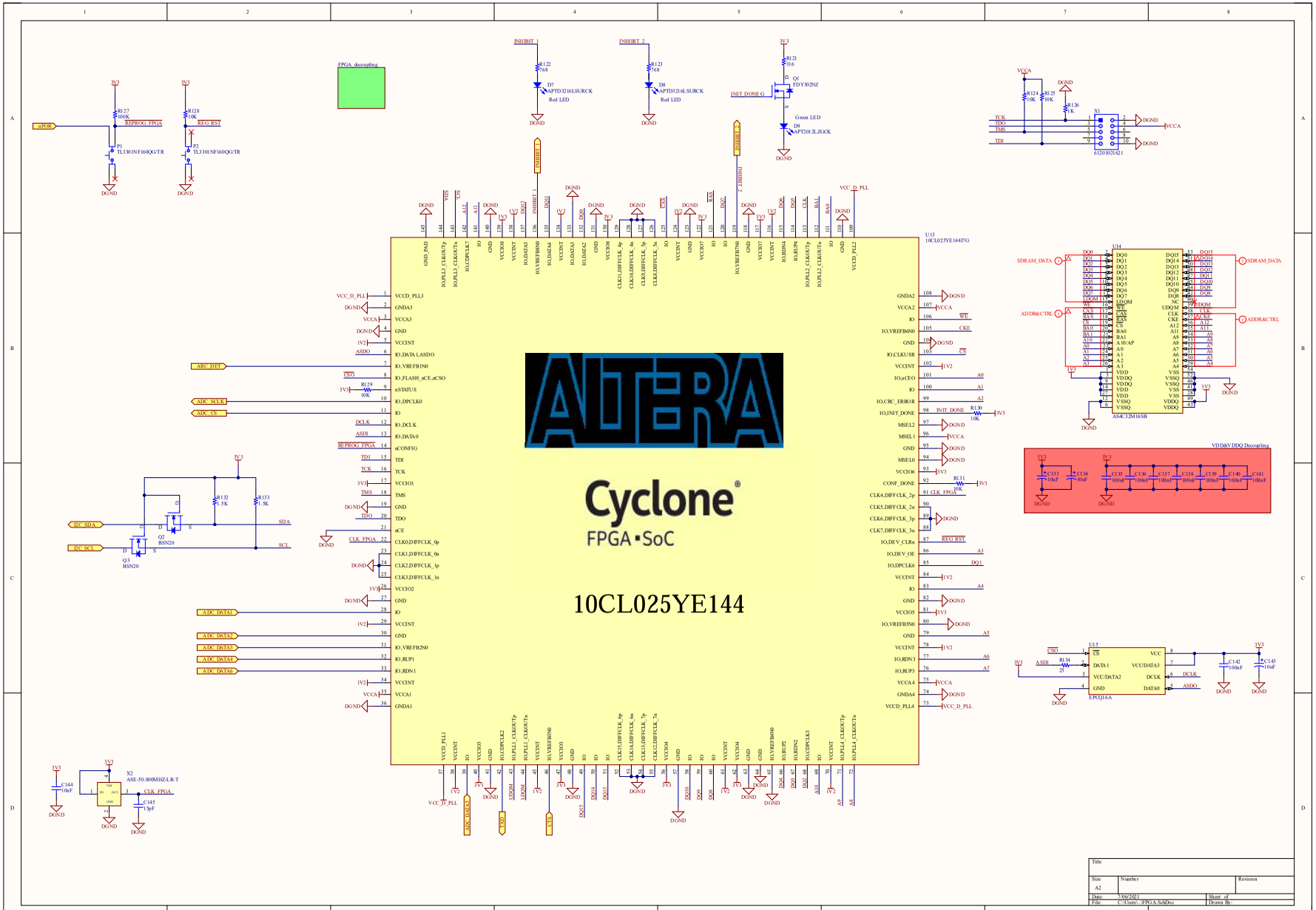


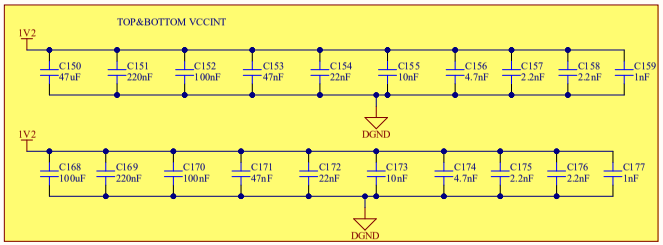
Title			
Size A3	Number		Revision
Date: 7/06/2021		Sheet of	
File: C:\Users\..\ADC.SchDoc		Drawn By:	

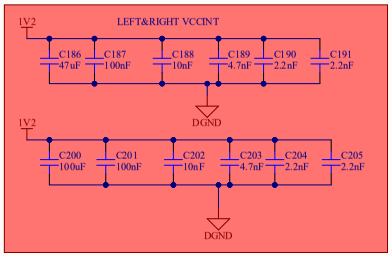


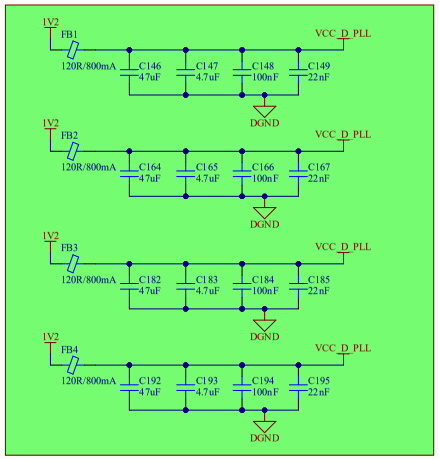
Title		
Size: A3	Number	Revision
Date: 7/06/2021		Sheet of
File: C:\Users\..\Power_Supply.SchDoc		Drawn By:

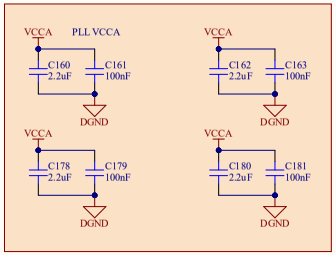

VCC3V3
U25
93LC56BT-I/OT
VCC
VSS
DI
DO
CS
CLK
R154
10K
R157K
EE_DATA
EE_CS
EE_CLK
C257
100nF
DGND
VBUS
VCORE
VPHY
VPLL
VCCA
VCCORE
VCCD
VREGIN
REF
OSCI
OSCO
VCCIO
EEDATA
EECS
EECLK
RESET
DM
DP
TEST
EP
GND
AGND
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
ACBUS0
ACBUS1
ACBUS2
ACBUS3
ACBUS4
ACBUS5
ACBUS6
ACBUS7
ACBUS8
ACBUS9
U24
FT232H
R155
10K
R156
10K
FTDI_RXD
FTDI_RTS
R158
750
TX
D12
APT2012LSECK/J4-PRV
R159
10K
C256
100nF
FB5
600R/0.5A
Y1
ABM8G-12.000MHZ-B4YT
C258
10pF
C259
10pF
J1
USB Micro
USB_DATA_N
USB_DATA_P
FTDI_USB_N
FTDI_USB_P
R160
10
R162
10
R164
0
C260
10nF
D13
RClamp0582B
R163
12K
C261
10nF
R161
1150
D14
APT2012LZGCK
FB7
600R/0.5A
FB6
600R/0.5A
C262
100nF
C263
100nF
C264
100nF
METTI VICINO AI PIN DI VCCIO
C265
100nF
C266
4.7uF
C267
100nF
C268
100nF
C269
100nF
C270
4.7uF
C271
100nF
Title
Size
A4
Number
Revision
Date:
7/06/2021
File:
C:\Users\..\USB.SchDoc
Sheet of
Drawn By:

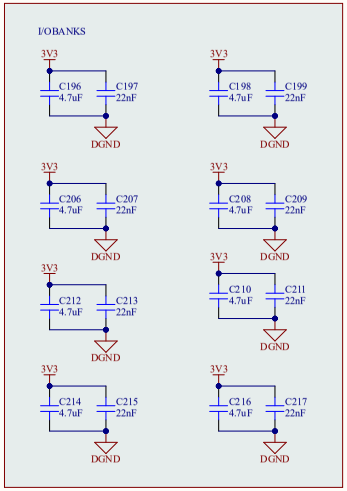

10CL025YE144
Cyclone
FPGA • SoC
ALTERA
nPOR
RETPROG_FPGA
REG_RST
P1
P2
TL3301NF160QG/TR
FPGA_decoupling
INHIBIT_1
INHIBIT_2
INHIBIT_3
D7
D8
D9
APTD3216LSURCK
Red LED
Green LED
APT2012LZGCK
Q1
FDY302NZ
INIT_DONE_G
U13
10CL025YE144I7G
U14
SDRAM_DATA
ADDR&CTRL
AS4C32M16SB
VDD&VDDQ Decoupling
U15
EPCQ16A
X1
X2
ASE-50.000MHZ-LR-T
Q2
Q3
BSN20
I2C_SDA
I2C_SCL
SDA
SCL
ADC_DATA1
ADC_DATA2
ADC_DATA3
ADC_DATA4
ADC_DATA5
ADC_DCLK
ADC_CS
ASDI
CSO
DCLK
ASDO
TCK
TDI
TMS
TDO
CLK_FPGA
Title
Size
A2
Number
Revision
Date:
7/06/2021
Sheet of
File:
C:\Users\...\FPGA.SchDoc
Drawn By:











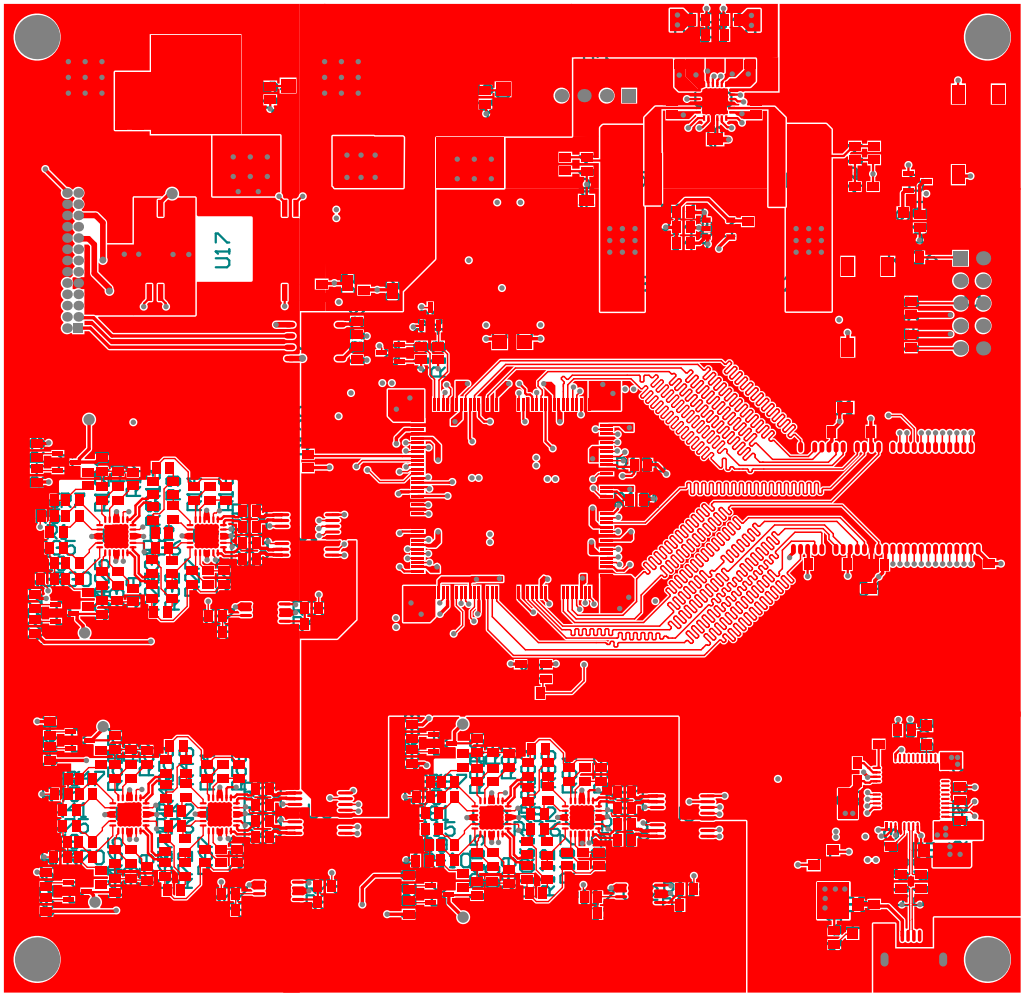
Title		
Size A3	Number	Revision
Date: 7/06/2021		Sheet of
File: C:\Users\..\FPGA decoupling.SchDoc		Drawn By:

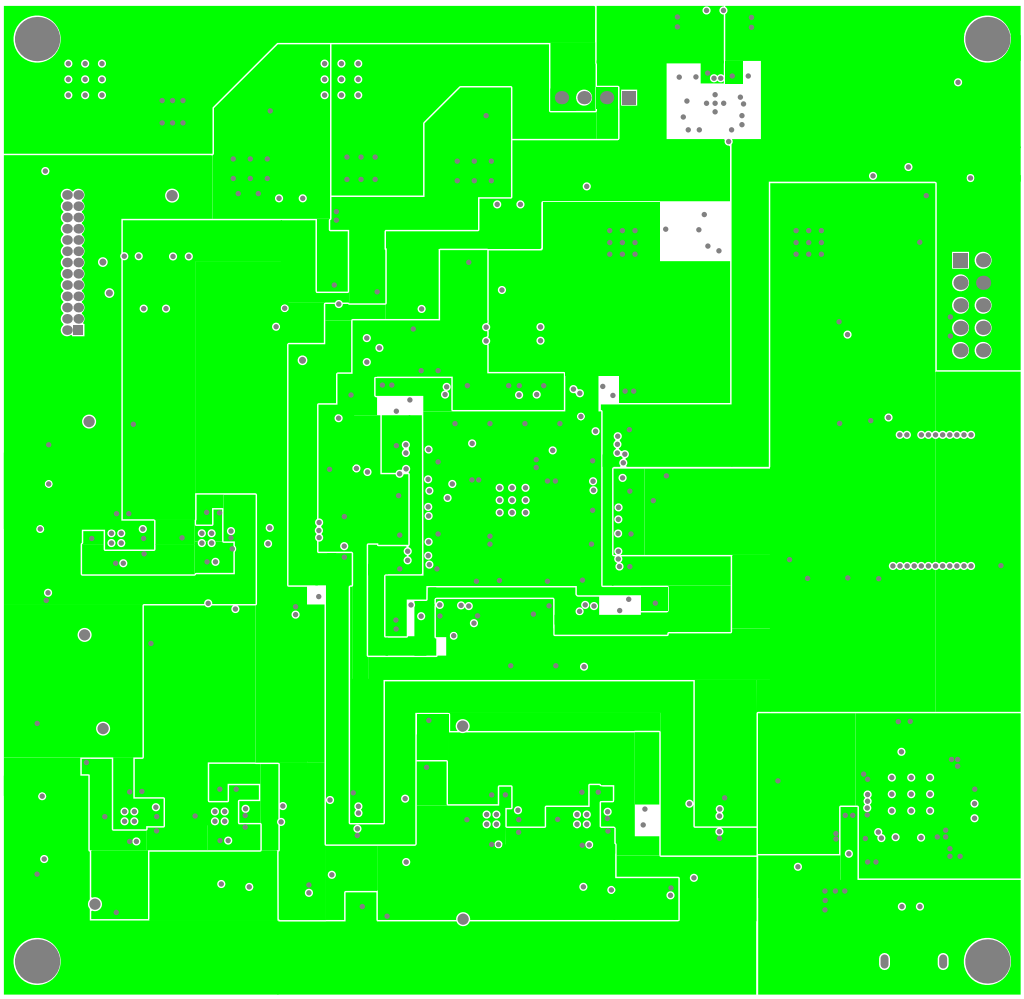
Appendice C

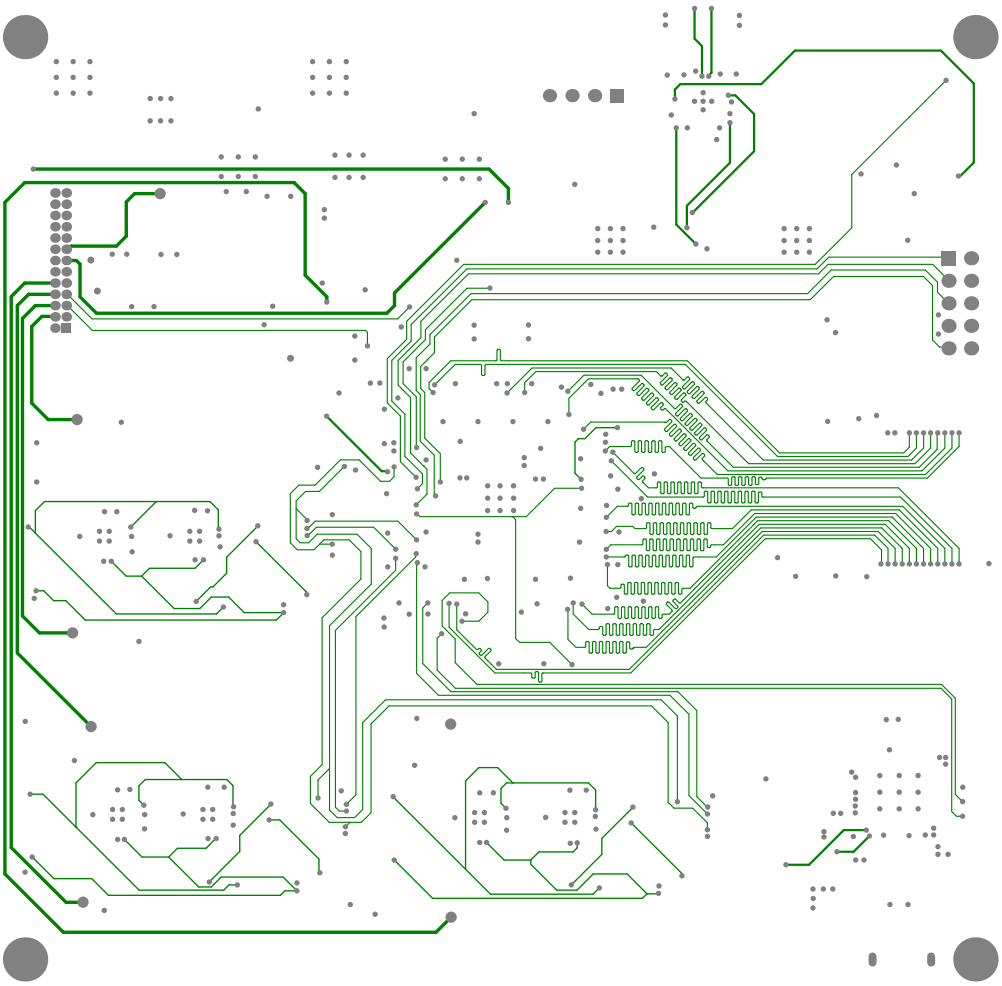
Layout della Scheda e Modello 3D

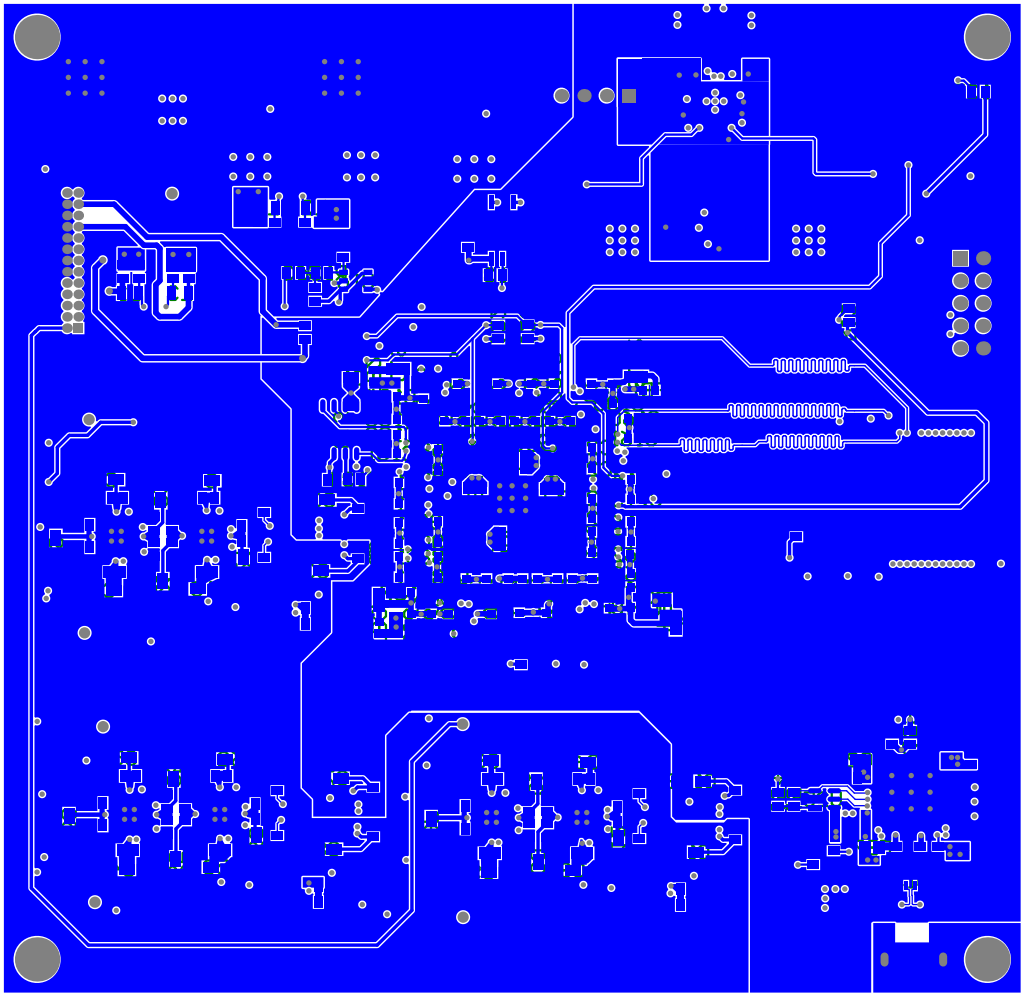
In questa appendice vengono presentati i layout finali dei vari livelli della scheda relativi ai segnali e al piano di alimentazione in quest'ordine:

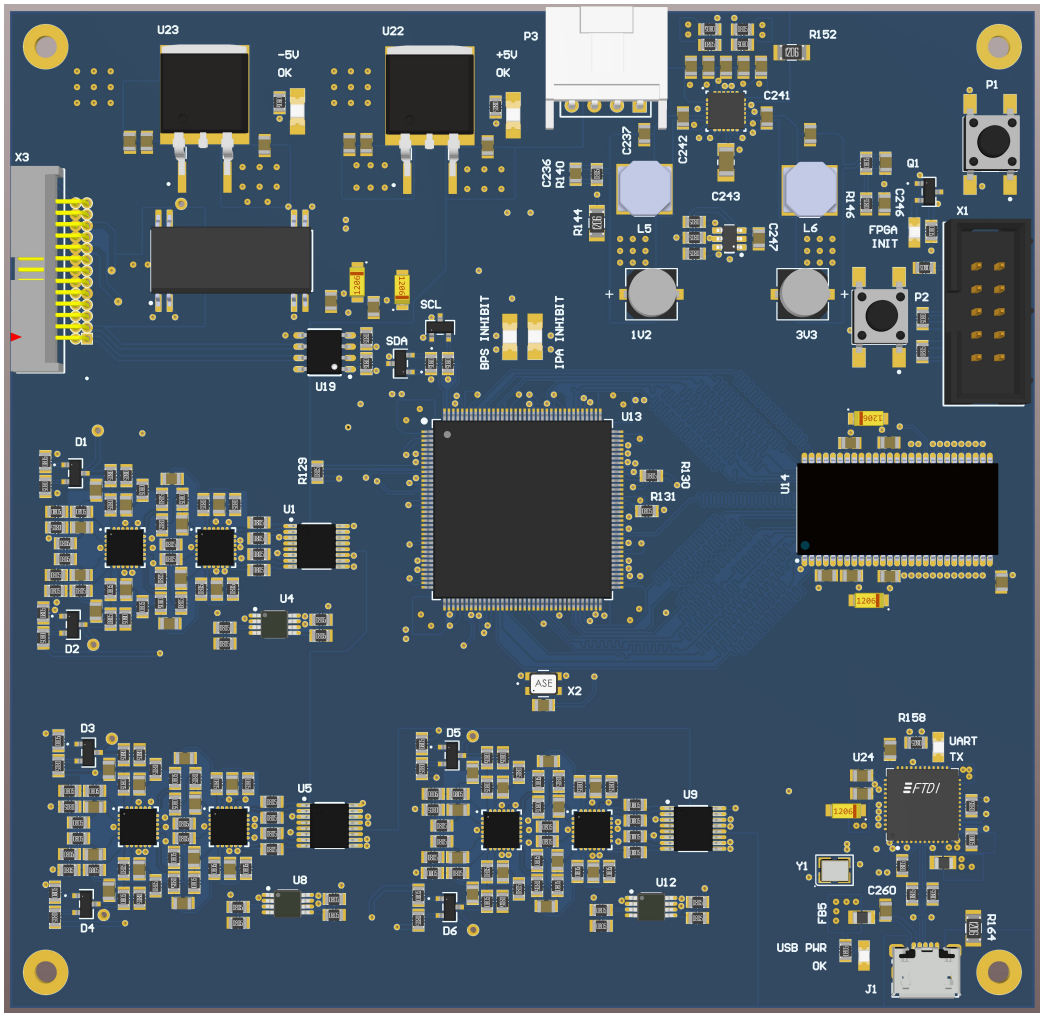
1. Top Layer
2. Power Layer
3. Internal Layer
4. Bottom Layer
5. Modello 3D (Top View)
6. Modello 3D (Bottom View)

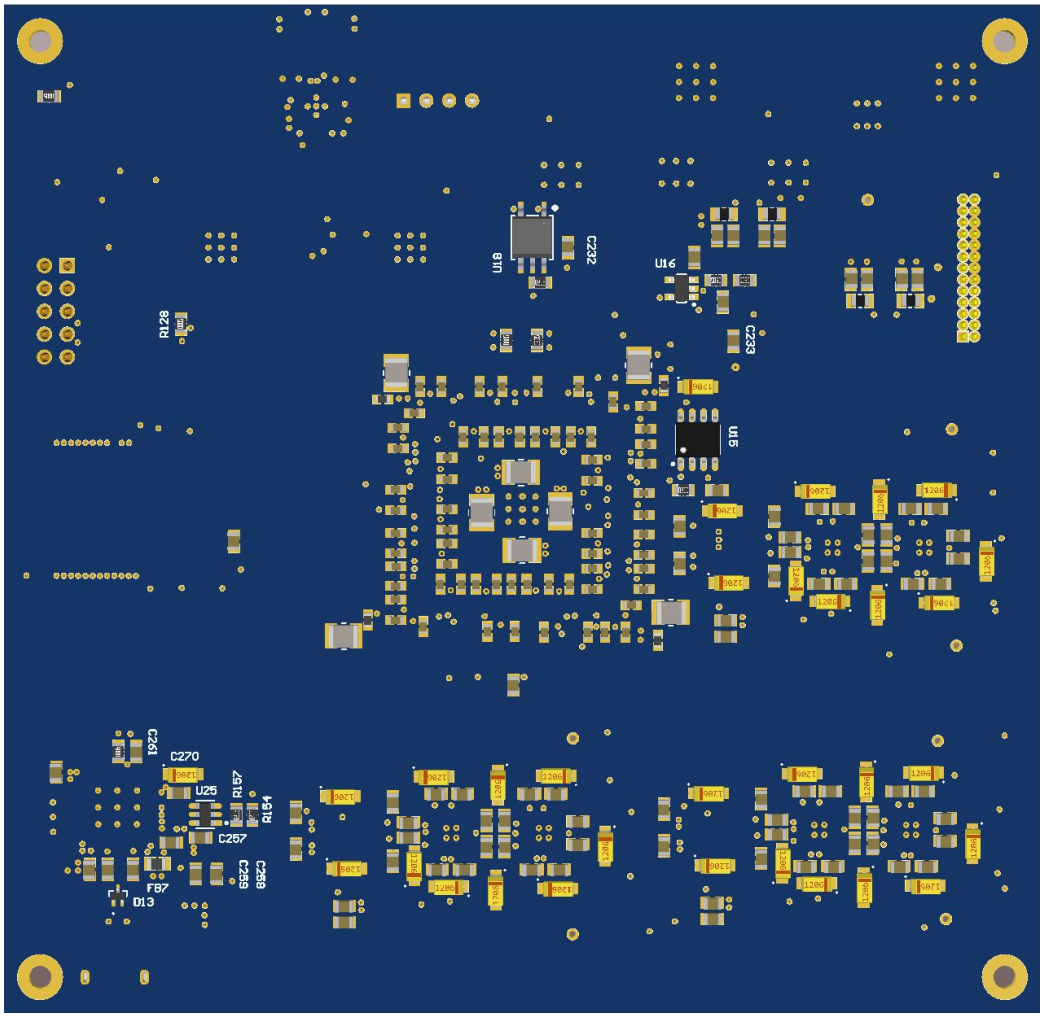

U17








U23
-5V
OK
U22
+5V
OK
P3
R152
C241
C242
C237
C236
R140
C243
P1
Q1
X3
R144
L5
L6
R146
C246
FPGA
INIT
X1
C247
SCL
SDA
BPS INHIBIT
IPA INHIBIT
1V2
3V3
P2
U19
U13
D1
R129
R130
R131
U1
U14
U4
D2
X2
D3
D5
U5
U9
R158
UART
TX
U24
FTDI
U8
U12
Y1
C260
D4
D6
FB1
R164
USB PWR
OK
J1


U18
C232
U16
C233
U15
R82
C251
C270
U25
R157
R164
C257
C256
C258
FB7
D13

Bibliografia

[1] ESA. *Europe's Access to Deep Space Space, The Deep Space Ground Station in Australia*. URL: https://esamultimedia.esa.int/multimedia/esoc/esoc_newnorcia_brochure.pdf (cit. a p. 1).

[2] ESA. *New Norcia - DSA 1*. URL: https://www.esa.int/Enabling_Support/Operations/ESA_Ground_Stations/New_Norcia_-_DSA_1 (cit. a p. 1).

[3] V. Dainelli, F. Serrano, L. Tomasi, C. Tuninetti, S. Marti, G. Alessi e R. Madde. «A 20 kW X band High Power Amplifier for ESA Deep Space Ground Stations». In: *2009 IEEE International Vacuum Electronics Conference*. 2009, pp. 313–314. DOI: 10.1109/IVELEC.2009.5193405 (cit. a p. 1).

[4] ESA. *ESA to build second deep space dish in Australia*. Apr. 2021. URL: https://www.esa.int/Enabling_Support/Operations/ESA_to_build_second_deep_space_dish_in_Australia (cit. alle pp. 1, 8).

[5] Texas Instruments. *LM26480 Dual 2-MHz, 1.5-A Buck Regulators and Dual 300-mA LDOs With Individual Enable and Power Good*. Rapp. tecn. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/lm26480.pdf?ts=1626333867379&ref_url=https%5C%253A%5C%252F%5C%252Fwww.ti.com%5C%252Fproduct%5C%252FLM26480%5C%253FHQS%5C%253DTI-null-null-octopart-df-pf-null-wwe (cit. alle pp. 14, 15).

[6] Analog Devices. *Data Sheet ADA4930-1/ADA4930-2*. Rapp. tecn. Analog Devices. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADA4930-1_4930-2.pdf (cit. a p. 17).

[7] Analog Devices. *Differential Input, Dual, Simultaneous Sampling, 5 MSPS, 12-Bit, SAR ADC*. Rapp. tecn. Analog Devices. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7356.pdf (cit. alle pp. 23, 33, 55).

[8] Microchip. *MCP6V96/6U/7/9, 10 MHz, Zero-Drift Op Amps*. Rapp. tecn. Microchip. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP6V96-Family-Data-Sheet-DS20006467A.pdf (cit. a p. 23).

[9] Herman Schutte. *Bi-directional level shifter for I^2C-bus and other systems. AN97055*. Rapp. tecn. Philips Semiconductors Systems Laboratory Eindhoven, The Netherlads (cit. a p. 28).

[10] FTDI. *FT232H SINGLE CHANNEL HI-SPEED USB TO MULTIPURPOSE UART/FIFO IC Datasheet Version 2.0*. Rapp. tecn. FTDI. URL: `http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232H.pdf` (cit. a p. 30).

[11] Intel. *SDRAM Controller Core*. Rapp. tecn. Intel. URL: `https://www.intel.cn/content/dam/altera-www/global/zh_CN/pdfs/literature/hb/nios2/n2cpu_nii51005.pdf` (cit. a p. 37).

[12] Intel. *UART Core*. Rapp. tecn. Intel. URL: `https://www.intel.cn/content/dam/altera-www/global/zh_CN/pdfs/literature/hb/nios2/n2cpu_nii51010.pdf` (cit. a p. 42).

[13] Intel. *Embedded Peripherals IP User Guide*. Rapp. tecn. Intel, pp. 196–205. URL: `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_embedded_ip.pdf` (cit. a p. 45).

[14] Intel. *Managing Metastability with the Quartus II Software*. Rapp. tecn. Intel. URL: `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii51018.pdf` (cit. a p. 51).