ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

Scuola di ingegneria e architettura

-Sede di Forlì-

CORSO DI LAUREA

IN INGEGNERIA AEROSPAZIALE

Classe L-9

ELABORATO FINALE DI LAUREA

In Avionica e Strumentazione Spaziale

Interfaccia software di verifica e calibrazione a terra del sensore di terra di ESEO

Candidato
Giacomo Curzi

Relatore
Prof. Paolo Tortora

Correlatore
Ing. Claudio Bianchi

Anno accademico 2013/2014

Sessione I

Indice

Indice delle figure

Indice dell tabelle

Ringraziamenti

Desidero ricordare tutti coloro che mi hanno aiutato nella stesura di questo elaborato con critiche, suggerimenti ed osservazioni: a loro va la mia più sincera gratitudine.

Ringrazio prima di tutto il professore Paolo Tortora per avermi offerto la possibilità di prendere parte ad un progetto così importante ed internazionale come ESEO. Vorrei inoltre ringraziarlo per la completa disponibilità per domande, chiarimenti e suggerimenti.

Un ringraziamento speciale va al correlatore Ingegnere Claudio Bianchi per il supporto e la disponibilità che mi dato durante il lavoro di tesi, nonché durante la stesura del presente elaborato.

Vorrei anche ringraziare colleghi, amici e parenti per il supporto morale fornitomi sia durante il periodo di preparazione della tesi che durante i tre anni del corso.

1 Introduzione

Il presente elaborato è relativo allo sviluppo e validazione di un software di interfaccia per la calibrazione e debug a terra di un sensore di terra. L'elaborato è pertanto parte di un progetto più ampio che riguarda la progettazione del satellite European Student Earth Orbiter (ESEO).

1.1 Progetto ESEO

Il progetto ESEO è finanziato dalla European Space Agency (ESA), e nasce con lo scopo principale di formare studenti in uscita da scuole di indirizzo spaziale o aerospaziale. In particolare sono stati selezionati vari gruppi di studenti provenienti da varie università europee. Il progetto, attualmente in fase di progettazione di dettaglio (fase C) è coordinato dall' azienda ALMASpace partner dell'università di Bologna.

Lo scopo operativo del progetto ESEO è quello di mettere in orbita LEO un satellite in grado di scattare foto, misurare livelli di radiazione e testare nuove tecnologie da impiegare in missioni future simili.

Il satellite ESEO sarà lanciato presumibilmente nel 2016, si immetterà in una orbita bassa a circa 600km di quota e rimarrà in servizio per 6 mesi, durante i quali compirà le missioni per cui è stato progettato. Al termine della missione potrà essere previsto un prolungamento della vita operativa se possibile. Comunque sia al termine della vita operativa è previsto un rientro distruttivo come da norma ESA per il controllo dei detriti spaziali.

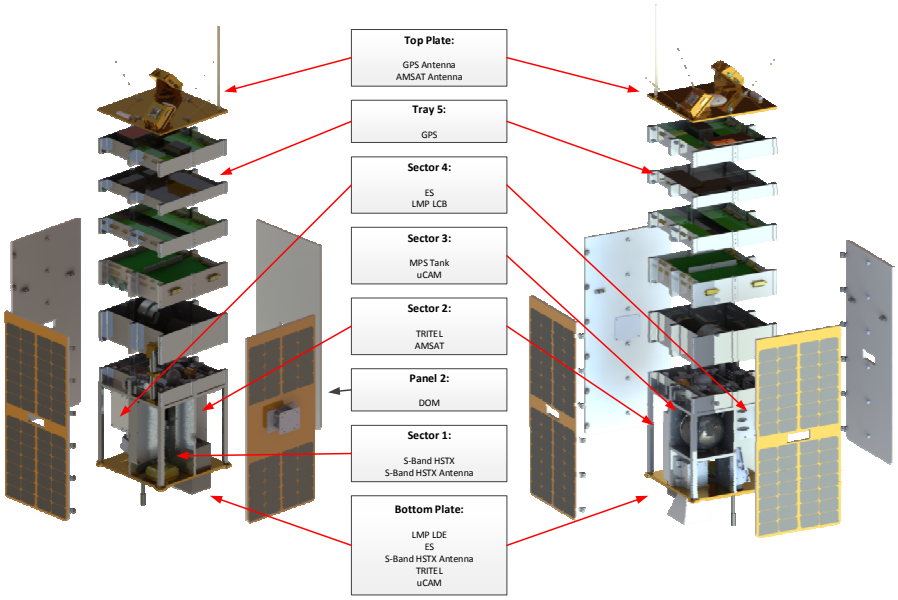

Figura 1: disposizione dei payload a bordo di ESEO

Dato lo scopo di ESEO di fare immagini della Terra si evince la necessità di un buon controllo d'assetto. Per realizzare un controllo d'assetto sono necessari punti di riferimento. In genere

per satelliti di questa fascia sono utilizzati come punti di riferimento il Sole e la Terra stessa. I sensori più comuni in orbite di tipo LEO infatti sono , sensori di sole, sensori di orizzonte terrestre e magnetometri. Altri sensori più complessi e ingombranti come startracker non vengono utilizzati in queste tipologie di missione . Nel caso specifico di ESEO si è scelto di dotare il satellite di tre tipologie di sensori, due sensori di sole, un sensore di Terra e due magnetometri.

1.2 Sensore di terra

Un sensore di terra è un dispositivo capace di stimare uno o più angoli di assetto di uno spacecraft attraverso la radiazione visibile o infrarossa emessa dalla Terra. Il sensore di terra a bordo di ESEO utilizza una camera che lavora nella banda elettromagnetica dell'infrarosso. La Terra, infatti, essendo più calda dallo spazio profondo si distingue da esso irradiando prevalentemente nella banda dell'infrarosso. Attraverso l'immagine della porzione di terra scattata con questo sensore è possibile stimare l'assetto del satellite. Elaborando la foto è possibile trovare la linea dell'orizzonte terrestre, sapendo la posizione e l'orientamento di quest' ultimo e l'orientamento della camera rispetto al satellite è possibile ricavare due angoli di assetto del satellite. Fornendo così un asse di riferimento diretto dal satellite al centro della terra.

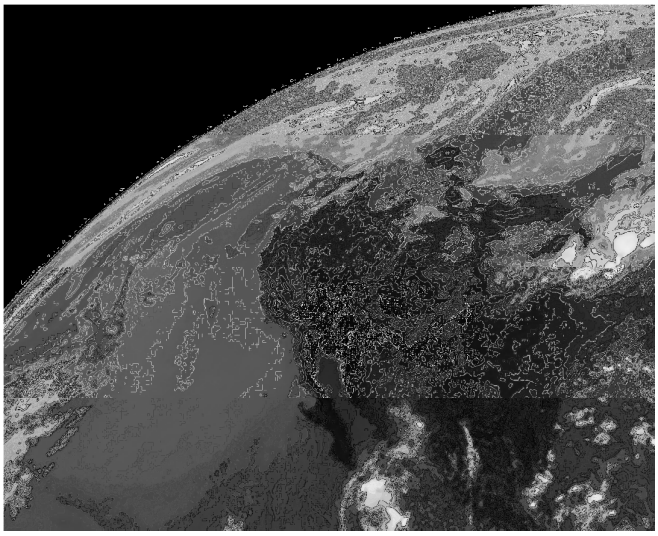

Figura 2: Vista all'infrarosso del sensore di terra

Il sensore di ESEO è fisso a bordo del satellite e posizionato in modo da inquadrare una porzione di orizzonte, una tipica vista è quella in Figura 2. Un grosso vantaggio dell'utilizzo della radiazione termica della terra per fare la stima di assetto è che non ci sono sostanziali differenze di irraggiamento tra quando la terra è illuminata dal sole o in ombra. Uno dei problemi intrinseci dell'utilizzo di sensori di terra è però quello di essere influenzati dagli altri corpi celesti con temperature proprie più alte di quelle dello spazio profondo, come ad esempio la Luna o il Sole. Infatti essendo questi corpi vicini alla terra, se cadessero nel campo di vista del sensore potrebbe essere impossibile tracciare il profilo della terra e

quindi determinare l'assetto.

1.3 Lavoro svolto

Il lavoro di tesi mira alla realizzazione di un' interfaccia di controllo tra il sensore di terra ed un PC attraverso il connettore di debug del sensore stesso. Questo è necessario per poter essere in grado di impostare i parametri di funzionamento del sensore. I comandi per la modifica delle varie impostazioni dovranno inoltre passare attraverso l' hardware di controllo del sensore di terra. Il passaggio attraverso questo hardware è necessario perché in fase di progettazione, si devono valutare le immagini scattate con parametri diversi. Le varie prove permetteranno quindi di ottenere un set di impostazioni ottimale per l'uso in orbita. La scheda di controllo del sensore di laboratorio che è stata utilizzata in questo lavoro, la Discovery F4, è analoga (in termini di componenti a bordo e di prestazioni) a quella utilizzata a bordo del satellite, permette infatti di essere programmata per fare analisi delle immagini e ricavarne i risultati utili.

Il lavoro di tesi è suddiviso principalmente in due parti, la prima parte è costituita dalla validazione dell'interfaccia GUI del lato PC precedentemente sviluppata durante il tirocinio. La seconda parte è composta dalla scrittura e validazione di un codice C a bordo dell' hardware Discovery con lo scopo di mettere in comunicazione il software Labview su PC con il sensore di terra.

Lo scopo ultimo di questo elaborato di tesi è quello di poter andare a variare le impostazioni della camera, attraverso la porta di debug, una volta che il sensore risulta assemblato nel proprio case e pronto per essere integrato a bordo di ESEO.

2 Hardware utilizzato

Il sensore di terra di ESEO è basato sull'utilizzo della camera commerciale Tau 320 dell'azienda Flir ed è l'hardware che il progetto di tesi intende comandare. Per raggiungere questo scopo anche i seguenti dispositivi sono stati utilizzati:

1. RS232 TAU Adapter
2. STM32F4DISCOVERY
3. Grabber Video

Questi saranno descritti qui di seguito, verranno inoltre illustrati le caratteristiche del protocollo RS232 e il funzionamento della camera ai fini di una miglior comprensione.

2.1 Tau camera

La camera Tau 320 utilizzata è la versione con ottica da 9mm e un angolo di vista orizzontale pari a 48°. Essa ha inoltre una potenza assorbita inferiore a 1 watt, pesa circa 70 grammi ed opera con temperature comprese tra -40 e +80 °C. Il sensore è in grado di captare la radiazione infrarossa da 8 a 14 µm ed è costituito da un array di microbolometri da 320x256.

Figura 3: Tau 320 FLIR

L'hardware in oggetto è dotato di :

1. Un canale di output digitale: che a sua volta può fornire immagini con diversi standard video (LVDS, CMOS e BT656)
2. Un canale di output analogico (NTSC o PAL)
3. Un canale di comando che utilizza il protocollo RS232

Il canale 3 è quello interessante dal punto di vista dell'argomento di tesi, è infatti attraverso questo che vengono impostati i parametri della fotocamera.

L'impostazione dei parametri della Tau avviene mediante l'invio di stringe di comando definite nel manuale attraverso il canale di comado. La camera, per ogni comando ricevuto, risponderà con un "echo" contenete o l'informazione richiesta (se il comando è di richiesta)

o il comando inviato (se il comando è di impostazione).

2.2 Protocollo RS232

Il protocollo RS232 è una convenzione di comunicazione tra dispositivi hardware nata negli anni 60 al fine di facilitare la connessione. La comunicazione mediante RS232 è ancora oggi utilizzata per la sua relativa semplicità ed è adottata dove la velocità di trasmissione non è un requisito stringente.
Di seguito verrà spiegato nel particolare il Protocollo "RS232 ridotto" poichè è quello interessante ai fini del progetto di tesi.

Il cavo che realizza la connessione ha nove linee e termina con il connettore mostrato in Figura 4.

Figura 4: Connettore del cavo seriale

Ogni pin è dedicato ad una funzione specifica riportata e spiegata in Tabella 1

Tabella 1: Funzioni dei pin dei connettori secondo il protocollo RS232

Numero pin	Denominazione	Funzione
1	DCD	Data Carrier Detect - E' imposto a livello logico 1 dall'evnetuale modem connesso e indica che la portante della linea telfonica è ricevuta. *
2	RXD	Receive Data - E' un pin di ingresso; va a costituire la linea dati ed è connesso al TXD dell'altro dispositivo.
3	TXD	Trasmit Data – E' un pin di uscita; va a costituire la linea dati ed è connesso al RXD dell'altro dispositivo.
4	DTR	Data Terminal Ready – E' un pin di uscita; è collegato al DSR dell'altro dispositivo e viene messo ad 1 quando è possibile la ricezione. Va a costituire la linea di sincronizzazione DTR-DSR.
5	SG	Ground – messa a terra del segnale
6	DSR	Data Set Ready – E' un pin di ingresso; è collegato al DTR dell'altro dispositivo, se a livello logico 1 indica che il dato può essere trasmesso. Va a costituire la linea di sincronizzazione DTR-DSR.

7	RTS	Request To Send – E' un pin di uscita ed è collegato al CTS dell'dispositivo; se portato ad 1 richiede la possibilità di inviare il dato e si aspetta una risposta sul proprio pin CTS.
8	CTS	Clear To Send – E' un pin di ingresso ed è collegato al RTS dell'altro dispositivo; se a livello logico 1 indica che l'altro dispositivo è poronto a ricevere.
9	RI	Ring Indicator – E' pilotato dall'eventuale modem ed indica la presenza di una chiamata sulla linea telefonica se impostato ad 1. *

* pin utilizzati solo in caso di connessione a modem telefonici.

Le informazioni vengono scambiate attraverso l'invio di pacchetti di bit successivi su una stessa linea dati, questo da il nome comune al protocollo che è anche conosciuto come seriale. I valori logici dei bit sono associati a due livelli di tensione distinti, in particolare vengono utilizzati solitamente livelli di ±5, ±10, ±12 o ±15 V, associando ad 1 il livellodi tensione basso e a 0 il livello di tensione alto. Il pacchetto è costituito da una sequenza precisa di invio dei bit:

1. Start bit
2. Data bits
3. Parity bit (evenetuale)
4. Stop bits

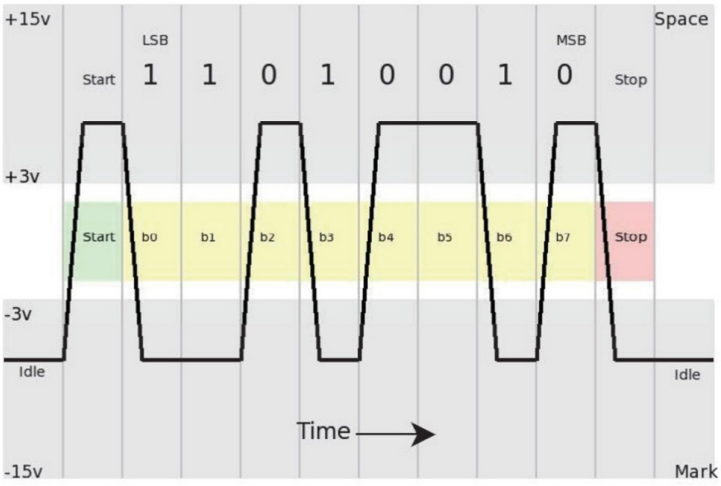

Figura 5: Tempi di comunicazione del protocollo RS232

Lo stato di riposo della linea è il livello logico alto (tensione negativa), per indicare l'inizio della trasmissione viene inviato lo start bit portando a livello logico basso la linea di trasmissione. Successivamente vengono inviati i veri dati partendo dal bit meno significativo (LSB). Al termine può essere inviato un parity bit che può a sua volta essere di quattro tipi: Eve, Odd, Mark o Space. Infine è presente uno (o più, a seconda delle esigenze) stop bit che riporta a livello logico alto la linea di trasmissione.

Il protocollo in oggetto prevede un funzionamento dei due dispositivi esclusivamente asincrono. Cioè non viene usato alcun segnale di clock (segnale periodico), comune per i due dispositivi, rispetto al quale fare riferimento per la trasmissione o ricezione. In modalità asincrona ogni dispositivo può trasmettere in qualsiasi istante, potenzialmente quindi il dispositivo che deve ricevere potrebbe non essere pronto a farlo, provocando un perdita di dati. Per evitare questo il protocollo RS232 ridotto implementa dei controlli di flusso hardware o software.

La modalità con controllo di flusso prevede l'utilizzo di linee apposite del cavo di connessione o controlli software appositi, per sincronizzare la trasmissione e la ricezione. Riguardo il controllo hardware vengono utilizzate a tale scopo le linee DTR-DSR ed eventualmente CTS-RTS a seconda della comodità, sfruttando il funzionamento descritto in Tabella 1. Il controllo software invece prevede di stabilire dei caratteri di inizio e fine trasmissione (Xon-Xoff), in questo modo i dispositivi sono "informati" se la corrente sessione di comunicazione è finita o meno. Questi controlli possono però non essere utilizzati e far lavorare la trasmissione in maniera puramente asincrona. Questa modalità è la più semplice ma anche quella più a rischio di perdita di informazione per il motivo sopra citato. Tale modo di comunicazione è quindi spesso adottato dove la comunicazione non è continua e la tempistica è nota, come succede infatti con la camera Tau 320 per l'impostazione dei parametri.

Tutti i dispositivi che vogliono interfacciarsi con questo protocollo devono avere una serie di specifici parametri impostati in modo uguale, altrimenti la comunicazione non avverrà in modo efficace. Infatti per la ricezione corretta dei bit del pacchetto i dispositivi fanno affidamento solamente: allo start bit, per identificare l'inizio del pacchetto, alla velocità di trasmissione (baudrate), per riconoscere i singoli bit, e alla conformazione e lunghezza del pacchetto (data bits + bit di parità + stop bits) per identificarne i dati e la fine. Lo standard prevede quindi una impostazione preliminare dei seguienti parametri di comunicazione, sia essa con o senza controllo di flusso:

- **Boudrate**: è la velocità di comunicazione ed è misurato in bit/sec. La scelta di questo parametro è influenzata da vari fattori tra cui il tempo in cui i dati devono essere trasmessi e dalla lunghezza del cavo di comunicazione.
- **Numero di data bit:** lunghezza del pacchetto di trasmissione in bit, può essere tra 5, 6, 7, 8 o più raramente 9 bit.
- **Bit di parità:** Se adottato ha quattro modalità di funzionamento: *Odd* - numero totale di 1 tra data bits e bit di parità deve essere pari. *Even* - numero totale di 1 tra data bits a bit di parità deve essere dispari. *Mark* – vale sempre 1, oppure *Space* vale sempre 0.
- **Bits di stop:** bit a valore 1 che vengono aggiunti per terminare la comunicazione, di solito è un singolo bit.

2.3 RS232 Tau Adapter

Prima di utilizzare l'interfaccia del PC per programmare la Tau camera attraverso la Discovery F4, è stato necessario essere sicuri che la GUI spedisse i giusti comandi e codificasse inoltre in modo corretto le risposte della Tau. Da qui l'esigenza di collegare direttamente la camera con il PC. A tale scopo è stata realizzata una scheda di interfaccia con la funzione di trasformare i livelli di tensione RS232 compatibili con la porta seriale del PC nei livelli di tensione previsti dalla Tau 320.

Il lato PC infatti lavora con i parametri standard del protocollo RS232 mentre la Tau prevede l'eccezione di usare livelli di tensione compresi tra 0 e 3.3 volt, pur mantenedo il bit di valore 1 associato al livello di tensione basso.

Per realizzare questo circuito è stata quindi fatta una ricerca dei dispositivi presenti sul mercato in grado di traslare i livelli di tensione. La scelta è ricaduta sull'integrato MAX3232 che, attraverso dei traslatori di livello a pompa di carica (vedi Appendice E), riesce a trasformare livelli di tensione 0 e 3.3 volt in livelli a ±6 volt. Il MAX3232 inverte inoltre la corrispondenza dei livelli logici, esso infatti è costruito per adattare la seriale RS232 ai moduli USART (Universal Sycronous Asyncronous Reveiver Transmitter) dei microcontrollori commerciali. Questo impone di invertire il livello di tensione ottenuto per rendere il segnale adatto allo standard della Tau. Per fare questa inversione è stato scelto l'integrato commerciale 74HC04 che ha 6 porte invertenti al suo interno e può essere alimentato a 3.3 volt.

Il circuito deve inoltre essere alimentato con due diversi livelli di tensione, uno a 3.3 volt per gli integrati e uno a 5 volt per alimentare la camera. Lo schema del circuito è riportato in Figura 6.

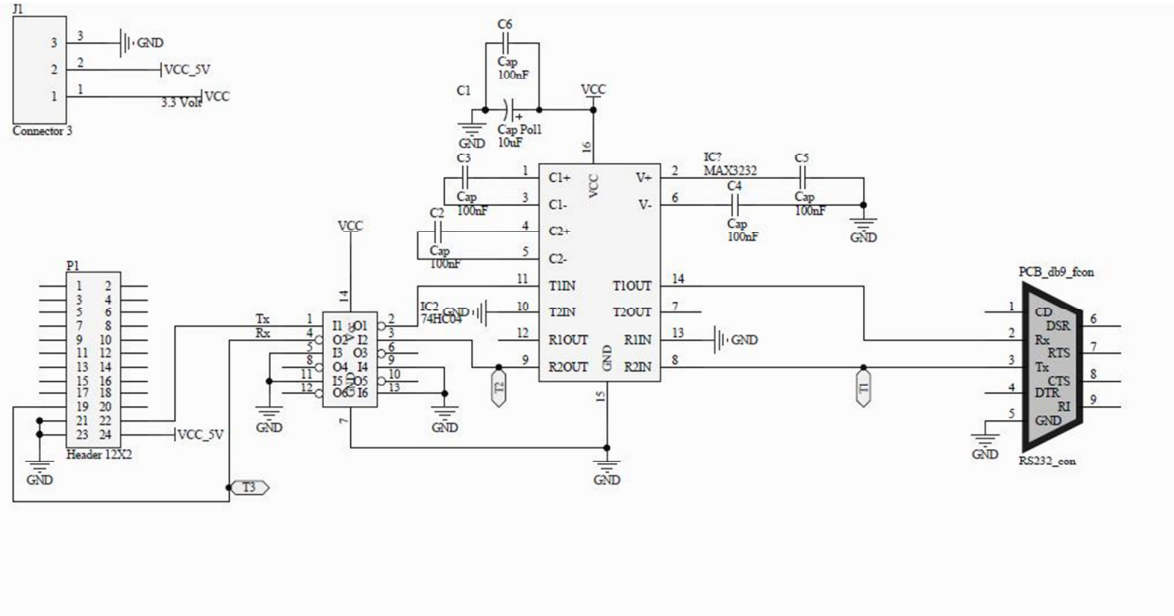

Figura 6: Schema RS232 Tau Adapter

La realizzazione del Tau Adapter è avvenuta su una basetta mille-fori come illustrato in Figura 7

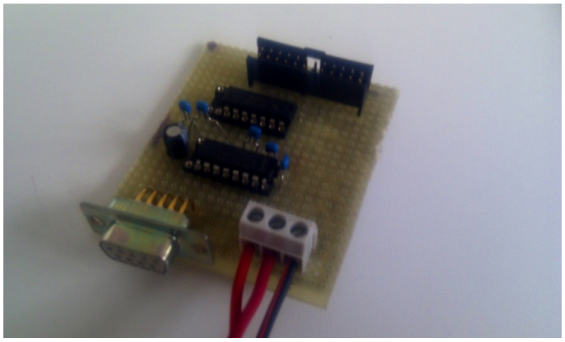

Figura 7: Scheda RS232 Tau Adapter

Una volta assemblato il circuito si è passati alla validazione per accertarsi dell'effettiva funzionalità. La prima verifica effettuata è stata quella di controllare con un multimetro digitale eventuali cortocircuiti e le giuste connessioni tra i pin rispetto allo schema (Figura 6). Successivamente sono state controllate le tensioni presenti in ogni punto del circuito, mentre questo era alimentato con doppia alimentazione come da progetto (3.3 e 5 volt). Inoltre, per garantire la funzionalità, è stato verificato con un oscilloscopio il passaggio corretto dei bit del pacchetto attraverso l'adattatore.

In questa fase il circuito era alimentato e connesso alla seriale del PC, mentre la camera non era connessa per evitare danni fisici o compromissioni del firmware. In questa configurazione è stato quindi spedito dal PC un carattere ASCII di prova ("a"), mentre l'oscilloscopio era connesso ai punti di controllo T1, T2 e T3 come specificato in Figura 6. I

risultati ottenuti nei diversi punti sono riportati rispettivamente in Figura 8, Figura 9 e Figura 10

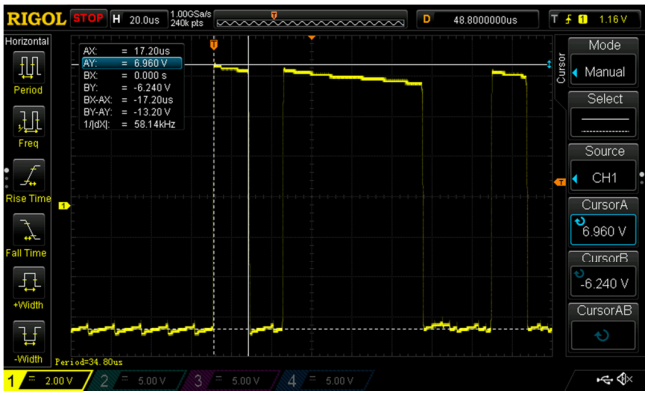

Figura 8: Segnale di tensione al punto di controllo T1

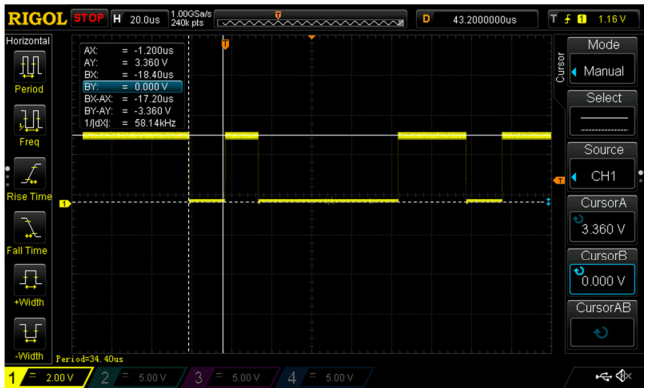

Figura 9: Segnale di tensione al punto di controllo T2

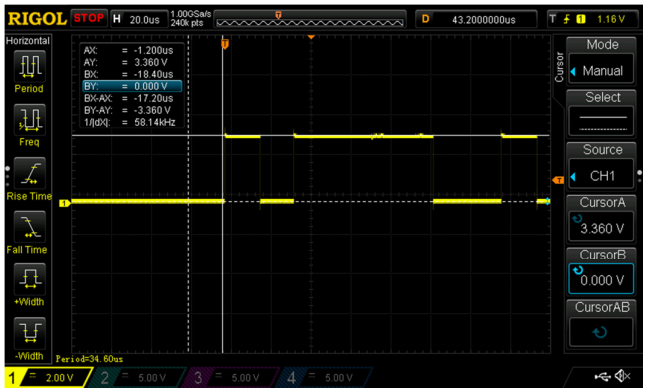

Figura 10: Segnale di tensione al punto di controllo T3

Come si può notare il segnale nel punto T1 rappresentato in Figura 8 (proveniente dal pc) varia tra ± 6 volt, e parte da un livello di tensione basso come da standard, si possono quindi distinguere i vari bit spediti notando che il baudrate impostato (57600 bps) corrisponde circa alla frequenza misurata dall'oscilloscopio sulla sequenza di bit che è 58.14kHz. Nell'ordine da sinistra verso destra, dato che il carattere "a" corrisponde al numero binario

“01100001” e che il protocollo prevede l’inversione dell’ordine di invio, si possono notare i bit del pacchetto spediti al punto di controllo T1: start-bit (+6 volt), 8 bit del pacchetto nell’ordine “10000110” e stop-bit (-6 volts). Passando al punto di controllo successivo T2 in Figura 9, si può notare come il livello di tensione passi tra 0 e 3.3 volts ma con forma d’onda invertita. Come già spiegato, questo non è il risultato finale voluto. Il segnale in uscita al MAX3232 deve essere fatto passare quindi attraverso un invertitore che inverte i livelli logici. In uscita dall’ invertitore, al punto di controllo T3 riportato in Figura 10, si ha il segnale corretto compatibile con la camera Tau.

Per fare una verifica globale dell’adattatore e del corretto funzionamento dei driver della GUI, è stata spedita successivamente una intera stringa di comando per la Tau (richiesta della ROI: vedi (FLIR, 2010)), ottenendo al punto di controllo T3 il risultato illustrato in Appendice A. Controllando bit a bit analogamente a quanto fatto per il carattere “a”, si vede la correttezza della stringa.

Alla fine di questo processo l’adattatore è risultato validato. Va notato però che è stata controllata solo la linea in trasmissione dal PC verso la Tau, la camera non era infatti connessa e non c’era quindi possibilità di fare funzionare la linea di ritorno al pc. La validazione da quindi per scontato che il circuito funzioni assolutamente in modo analogo per la ricezione.
Questo ha dato origine ad un’inconveniente, nonostante tutti i controlli dei collegamenti fatti una volta realizzato l’ hardware. Infatti durante la prova di connessione alla Tau 320 la risposta di quest’ultima non veniva recepita dal PC. Dopo opportuni test, si è scoperto che il mancato funzionamento era dovuto alla presenza di un errore di disegno. La connessione del segnale in ricezione del PC era collegata in modo errato. Dopo aver fatto le opportune correzioni è stato possibile utilizzare il circuito per validare l’interfaccia GUI.

2.4 Discovery F4

La scheda STM32F4DISCOVERY è una piattaforma hardware multiuso programmabile. Nell’ambito del progetto ESEO è utilizzata come piattaforma di sviluppo per la gestione della camera Tau.

In particolare nella fase di test a terra o di debug la Discovey F4 elabora l’immagine con lo stesso codice che sarà caricato sul reale dispositivo destinato al volo. Allora è importante che l’impostazione dei parametri della camera avvengano dal PC attraverso il dispositivo in oggetto, essenzialmente per una questione di rapidità di impostazione. Se così non fosse infatti, sarebbe necessario scollegare la camera dalla Discovery e collegarla al PC ogni volta si desideri cambiare qualche parametro.

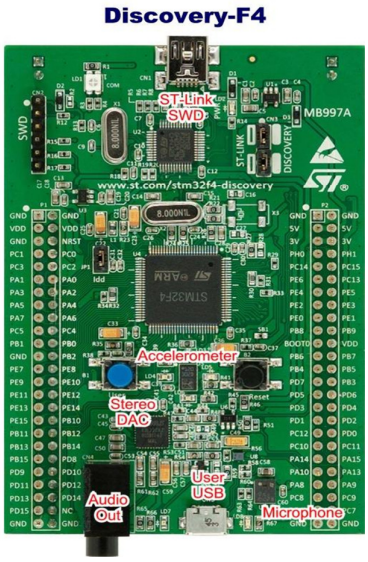

Figura 11: Discovery F4

La Discovery F4 ha la possibilità di interfacciarsi con molti dispositivi esterni grazie ai diversi pin multiuso disponibili. Alcuni dei pin disponibili all'utilizzatore possono inoltre essere programmati in diversi modi per sfruttare vari protocolli di comunicazione. La scheda in oggetto mette anche a disposizione una serie di sensori e output di default utilizzabili dall'utente, che in particolare sono:

- 4 Led programmabili di diversi colori
- 2 Pulsanti
- Una porta di programmazione/debug o USB generica (selezionabile meccanicamente)
- Un microfono
- Un set di tre accelerometri disposti in modo triassiale

Il "cuore" della scheda è il microcontrollore STM32F407 che gestisce tutte le periferiche a lui afferenti. Il controllore possiede le seguenti caratteristiche fondamentali:

- Architettura a 32 bit
- Memoria Flash da 1Mbyte
- Debug mode
- Clock interno programmabile da 4 a 26 MHz
- 3x12 A/D e 2x12 D/A converters
- 17 timers
- 140 pins input/output (I/O) programmabili
- 15 interfacce di comunicazione, tra cui 4 moduli USART
- Interfaccia digitale per camera da 8 bit e 14 bit

2.5 Grabber video

Per verificare effettivamente l'applicazione del comando inviato è necessario visualizzare l'uscita video della camera. Il modo più comodo per farlo è quello di rendere visibile l'uscita analogica della camera. Questo può essere fatto sul PC utilizzando proprio il Grabber video DVC80, rappresentato in Figura 12.

Figura 12: Grabber video

L'immagine può così essere visualizzata con un qualsiasi software che offra la possibilità di acquisizione dell'immagine real time e supporti il dispositivo DVC80.

3 Graphical User Interface

La Graphical User Interface o GUI costituisce la parte più importante del progetto per quanto riguarda il lato PC. Essa permette all'operatore di avere un controllo diretto dei parametri della camera. Lo scopo effettivo di questo strumento è quindi quello di tradurre l'input dell'utente, in una stringa di byte ed inviarla alla camera che la interpreta ed esegue. La schermata principale della GUI è riportata in Figura 13.

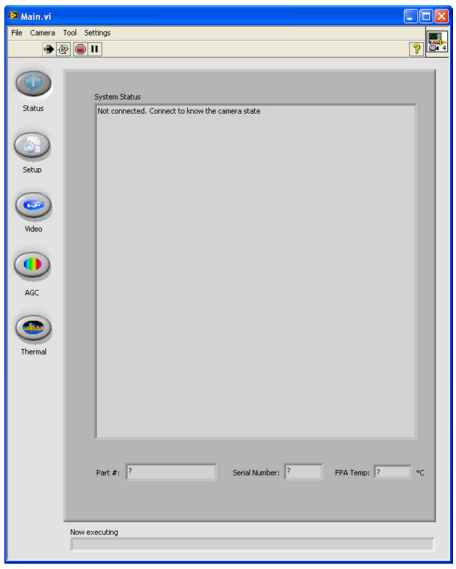

Figura 13: Schermata principale della GUI

Il modo più efficace di realizzare la GUI è stato di usare il software di programmazione LabView. Questo software di sviluppo inoltre, permette anche di fare complesse elaborazioni e di usare agevolmente le periferiche del PC. La realizzazione è stata fatta durante il periodo di tirocinio, cercando di verificarne il funzionamento attraverso un simulatore della Tau, anche esso fatto appositamente. Seppur con la massima accortezza, è stato però impossibile replicare esattamente il comportamento della Tau; quindi il primo passo del lavoro di tesi è stato quello di validare sulla camera la GUI, evidenziando ed eliminando eventuali errori.

Di seguito verrà spiegato brevemente come funziona l'interfaccia ai fini di una migliore comprensione della verifica. Si faccia riferimento alla relazione di tirocinio per eventuali approfondimenti specifici (Curzi, 2014).

All'apertura della GUI l'utente ha a disposizione una barra del menù, scegliendo tra le opzioni si può ad esempio selezionare la porta seriale da utilizzare e poi connettersi. All'atto della connessione l'interfaccia invia una serie di comandi di richiesta, la Tau quindi risponde e le varie schermate di comando possono essere inizializzate con i valori presenti nella Tau in quel momento. Avvenuta questa fase iniziale, l'eseguibile entra in un "loop" che richiede

il parametro FPA_temp (temperatura del piano focale) in modo da testare continuamente la connessione. Contemporaneamente l' utente ha a disposizione cinque diverse schermate selezionabili che raggruppano i comandi per tipologia (come da manuale). Cliccando o spostando un comando, la GUI compone la stringa da inviare secondo il protocollo descritto dal manuale e la invia attraverso la porta seriale. La camera a seconda del comando, ad esempio di impostazione o richiesta, risponderà con una stringa di "feedback" che verrà esaminata. In particolare alla risposta vengono fatti due controlli nell'ordine:

1. Di correttezza: consiste nel verificare se ci sono stati errori di comunicazione, controllando i CRC (cyclic redoundancy check: vedi Appendice B) ed altri parametri significativi.
2. Di coerenza: consiste nel verificare che la camera abbia recepito il giusto comando. In particolare se il comando è di impostazione la camera deve rispondere con la stessa stringa ricevuta (vedi (FLIR, 2010)). Se invece il comando è di richiesta il controllo non viene fatto, in quanto le due stringe sono sicuramente diverse poichè quella ricevuta dal PC contiene il dato richiesto.

Esaminata la risposta l'interfaccia aggiorna il comando virtuale solo in base alla risposta stessa della Tau, in questo modo l'utente ha sempre sulla GUI la situazione della camera. A questo punto riparte il ciclo di attesa selezione di un comando, finchè l'operatore non decide di disconnettersi attraverso l'opportuno tasto della barra del menu.

3.1 Validazione della GUI

La validazione mira alla verifica del corretto invio del comando e della corretta elaborazione della risposta. Il processo è stato suddiviso in tre parti principali per essere realizzato:

1. Controllo della correttezza di ogni stringa potenzialmente inviabile
2. Controllo associazione stringa/comando.
3. Verifica dell'esecuzione dei comandi

Il punto 1 è stato realizzato intercettando i parametri principali delle stringe inviate dalla GUI al simulatore della Tau. Sebbene infatti non si abbia la certezza dell'esatta replica della camera da parte del simulatore, questo non ne influenza il risultato e non si rischia la comprimossione della Tau. Variando tutti i possibili comandi si è creato un file di log che riportava tutti i parametri essenziali della stringa per ogni pulsante. La tracciabilità, dato l'elevato numero di combinazioni dei comandi, è fondamentale, per questo il file è stato creato secondo una check-list riportata in Appendice C. Una sezionedel file di log è riportato in Appendice D, mentre l'esempio di un solo comando è riportato nella Figura 14.

6E-00-00-04-00-00-03-7B-00-00 4 00 00 crcs OK

Figura 14: Esempio di logging della stringa inviata dalla GUI

Come si vede in Figura 14 il primo parametro è la stringa completa, i successivi replicano informazioni essenziali già contenute nella stringa per comodità di controllo. Nell'ordine

sono: stringa inviata, numero della funzione, numero di byte dell'argomento della funzione, il valore del CRC fatto su tutta la stringa e un "feedback" sul controllo fatto dei due CRC (*crcs OK/KO*). *Per informazioni dettagliate sulla stringa si rimanda al manuale della Tau320* (FLIR, 2010). Si può inoltre notare che non è stato necessario controllare i limiti degli argomenti delle funzioni; questo perchè si sono verificati precedentemente i vari range di argomento direttamente nel codice LabView in quanto più semplici da controllare, si era così sufficientemente sicuri di non andare fuori range.

Ogni riga del file di log è stata confrontata con il format previsto dal manuale, per scongiurare che il software inviasse comandi sconosciuti, incompleti o non corretti. Se cosi non fosse si correrebbe infatti il rischio di compromettere il firmware interno della camera.

Il punto 2 è stato invece realizzato confrontando il comando inviato dall'interfaccia costruita con quella già fornita dalla casa costruttrice. Essendo il file di log precedente stato creato secondo una check-list in cui è stato riportato l'ordine dei comandi spostati dell'interfaccia, dal punto 1 si è già in possesso dell'associazione stringa/comando della GUI. Per confrontarla con quella dell'interfaccia originale è stata eseguita nuovamente la check-list del punto 1, questa volta con la Tau collegata al software originale. Ad ogni passo è stata intercettata la stringa inviata e confrontata con quella ottenuta al punto 1 nello stesso passo. Per tenere traccia dei comandi inviati, è comunque stato creato un secondo file di log (Appendice D) che riporta l'attività dell'interfaccia originale.

Il punto 3 è stato realizzato inviando i comandi dalla GUI alla Camera reale. Mediante il dispositivo DVC80 (le cui funzioni sono state spiegate al paragrafo 2.5) è stato possibile verificare se effettivamente il comando inviato fosse attuato o meno. In questa fase finale si è verificata inoltre la corretta elaborazione della risposta della camera. Per una esecuzione ordinata ci si è avvalsi ancora una volta della check-list sopra menzionata.

3.2 Risultati della validazione

Qui di seguito verranno riportati quali sono stati i principali problemi affrontati nella validazione e come sono stati risolti, seguite da considerazioni finali sul lavoro svolto. Anche qui si farà uso di un linguaggio specifico in riferimento alla GUI, pertanto si rimanda alla relazione di tirocinio per maggiori informazioni (Curzi, 2014).

Le difficoltà maggiori si sono verificate al punto 2 della validazione, infatti la correlazione stringa/comando del software di fabbrica era in alcuni punti diversa da come specificato sul manuale, quindi anche da come è implementata nella riproduzione della GUI. Questo forse per motivi di protezione della camera o del know-how dell'azienda.

In particolare i comandi in cui si è riscontrato questo tipo di problema sono stati i seguenti:

- Zoom: si è scoperta una correlazione tra lo stato dello zoom (unzoom, 2x, 4x) con altri comandi che sono: "freeze/realtime" e "video on/off". E' stato risolto

semplicemente andando a replicare la stringa inviata dal software originale nelle diverse combinazioni.

- BT656-CMOS (modalità di uscita delle immagini digitali): le stringhe inviate per variare le modalità erano diverse da quelle indicate sul manuale, inoltre alcune combinazioni non erano consentite. Anche qui il problema è stato risolto andando a verificare le combinazioni possibili dal software originale e replicandole.

In questa fase si sono inoltre scoperti alcuni limiti di comandi che per errore non erano stati specificati sul manuale. Si è inoltre risolto il problema del comando DDE rimasto aperto dal tirocinio, questo infatti non aveva una corrispondenza nominativa sul manuale ma si è scoperto impostare un parametro chiamato SPATIAL_THRASHOLD. Si è trovato inoltre che, diversamente da come riportato sul manuale, l'ordine di invio dei byte dell'argomento di tale funzione era invertito.

Sono stati scoperti anche comandi proprietari della FLIR non presenti sull'interfaccia inviati in automatico alla camera dal software originale. Dopo un esame di quali veramente erano i comandi necessari e non essendo esattamente noto che cosa andassero a modificare (in quanto non presenti sul manuale) si è deciso di non implementarli.

Un altro errore riscontrato in questa fase, che non poteva essere evitato, è l'attivazione contestuale di alcuni tasti della scheda AGC. Era infatti impossibile capire dal solo manuale quali dei tasti erano contestualmente attivi. La schermata in oggetto è tra l'altro quella più importante perchè fornisce i comandi su luminosità, contrasto, algoritmo di ottimizzazione e altro ancora. Il problema è anche qui stato risolto replicando il funzionamento del software originale.

I problemi riscontrati al punto 3 della validazione sono stati invece più facili da risolvere, questo grazie alle proprietà di debug di LabView.

In questa fase si è scoperto che, diversamente da come scritto sul manuale, per il comando DO_FFC non andava fatto il controllo di coerenza. La camera infatti rispondeva con una stringa diversa da quella aspettata. Si è evidenziato un problema analogo con il comando READ_SENSOR (per prendere la temperatura del piano focale) ma in questo caso si trattava di un errore della GUI, in quanto è un comando di richiesta e non di settaggio, era quindi ovvio che non andasse fatto il controllo.

Si è poi riscontrato un altro errore dell'interfaccia, in particolare il settaggio della ROI (Region of Interest) non avveniva in modo corretto finché non si impostava una "ROI zone" per ogni modalità di zoom. Era già nota da manuale infatti la correlazione tra ROI e Zoom, ma appunto per questo ci si aspettava un corretto funzionamento. L'errore era nell'inizializzazione dell'argomento della funzione ROI, questo infatti era inizializzato a 0 nelle quattro coordinate e per tutti e tre i campi di zoom, ma il settaggio delle quattro

coordinate a 0 contemporaneamente non era permesso. Questo spiega il comportamento iniziale, settando infatti tutti i campi di zoom si eliminava l'inizializzazione a 0 permettendo così il funzionamento le successive volte. Il problema è stato semplicemente risolto cambiando l'inizializzazione.

Alla fine di un lungo processo l'interfaccia è risultata valida, seppur con minime differenze rispetto al software originale. E' stato sicuramente un lavoro dove l'organizzazione è stata un fattore essenziale, si dovevano infatti gestire e controllare uno ad uno 42 diversi comandi. Alcuni comandi avevano poi correlazioni con altri non specificate nel manuale, si è dovuto quindi prestare attenzione alle diverse combinazioni e intuire la logica di programmazione della camera.

4 Programmazione della Discovery F4

Per quanto riguarda il titolo di tesi il programma che va sviluppato deve soddisfare due requisiti:

1. Trasferire, senza alcun tipo di manipolazione, la stringa ricevuta dal PC alla camera e viceversa.
2. Convivere con un programma già esistente che si occupa dell'elaborazione dell'immagine proveniente dalla camera.

Durante lo sviluppo del codice si è cercato inoltre di sfruttare codici già fatti (dal costruttore o provenienti da codici già funzionanti) che implementano funzioni di basso livello, questo infatti assicura una migliore comprensione poiché riduce il volume di script. Inoltre così facendo si ha la garanzia che quella specifica funzione adottata operi correttamente in quanto già testata.

Data poi l'internazionalità del progetto si è cercato di spiegare in loco le funzioni in inglese e di scrivere inoltre gli script con metodi convenzionali, cioè introducendo direttive, file header (saranno spiegati di seguito) e brevi descrizioni per ogni funzione.

Di seguito verranno riportati dapprima i vari collegamenti per comprendere meglio le varie impostazioni del programma, verranno poi date alcune informazioni di base sul linguaggio C per poi essere specializzato nel caso dei microcontrollori. Successivamente viene riportato il progetto del programma, seguito da conlusioni sui risultati raggiunti e problemi incontrati.

4.1 Collegamenti

Per quanto riguarda il collegamento PC-Discovery F4 esso è realizzato tramite un adattatore già fornito da ALMASpace, infatti il problema di connessione è analogo a quello risolto con la scheda RS232 Tau adapter. La Discovery lavora con livelli di tensione 0 – 3.3 volt e implementa il modulo USART quindi il problema è risolvibile con un semplice traslatore di livello invertente (MAX3232).

Dato che non si è scelto di utilizzare alcun tipo di controllo di flusso basta collegare solamente la linea dati, cioè i pin RXD e TXD del cavo seriale uscente dal PC. Questi due linee passando attraverso il MAX3232 sono collegati rispettivamente ai GPIO (General Purpuse Input Output pin) PA8 e PA9 della Discovery. Su questi pin infatti è possibile impostare da programma il modulo USART3. In realtà, come si vede in Figura 15, il collegamento permette l'utilizzo della linea di controllo RTS-CTS, i pin PD11 e PD12 saranno fisicamente occupati ma non utilizzati.

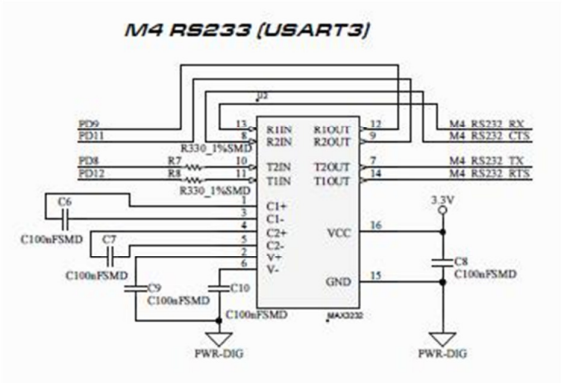

Figura 15: Schema di connessione USART3

Per quanto riguarda invece il collegamento Discovery - Tau si necessità solamente di una inversione (negazione logica del segnale) a 3.3 volt. Questa connessione utilizza il modulo USART2 che è implementato sui GPIO PA2 e PA3, corrispondenti rispettivamente al TXD e RXD della Discovery. La linea dati in oggetto passa quindi per l'invertitore, collegandosi poi rispettivamente ai pin RX e TX della Tau. Anche qui non vi è alcun controllo di flusso poiché la camera non ne prevede. Lo schema di collegamento è riportato in Figura 16.

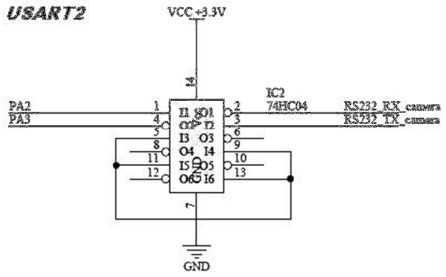

Figura 16: Schema di connessione USART2

La realizzazione completa del collegamento e riportata in Figura 17.

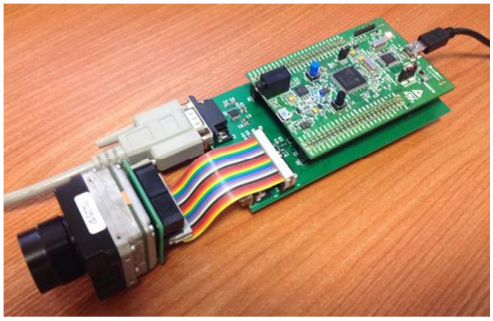

Figura 17: Discovery F4 connessa alla Tau e al PC attraverso gli adattatori

4.2 Linguaggio C per microcontrollori

Un linguaggio di programmazione è un set di parole chiave che consentono di realizzare una routine su un dispositivo elettronico programmabile. Il linguaggio C è un linguaggio di programmazione il cui set di parole o simboli chiave vanno impiegete secondo una opportuna forma, si crea così un codice strutturato comunemente chiamato script o codice sorgente. Questo potrà poi essere interpretato da un compilatore che lo tradurrà in linguaggio macchina e verrà successivamente trasferito sulla memoria del dispositivo di destinazione.

Uno script in C è costituito da:

- Strutture di controllo
- Operatori
- Variabili
- Funzioni
- Direttive

Le strutture di controllo costituiscono la base del C che è per questo detto linguaggio strutturato, sono delle "regole" base di esecuzione del codice sorgente e vengono a loro volta divise in strutture di:

- Sequenza
- Selezione
- Iterazione.

La struttura di sequenza è costituita da istruzioni racchiuse da parentesi graffe: la macchina eseguirà il contenuto delle parentesi graffe in sequenza, dall'istruzione in alto a sinistra a quella in basso a destra.

Le strutture di selezione permettono di fare delle scelte in base ad un controllo, sono identificate con la parola chiave **if** o **switch** seguita dal controllo tra parentesi tonde. La parola **if** identifica l'esecuzione delle istruzioni successive racchiuse da parentesi graffe se il controllo risulta "vero"; questa inoltre può prevedere anche l'esecuzione di un'altro set di istruzioni nel caso di controllo "falso", inserendo la parola **else** alla fine del primo set di istruzioni. La struttura switch permette di fare più controlli consecutivi su una stessa variabile ed eseguire il set di istruzioni che rende "vero" il controllo.

Le strutture di iterazione permettono di esguire un set di istruzioni per un certo numero di volte, sono essenzialmente i così detti cicli **while** e **for**. La differenza che c'è tra l'uno e l'altro è la seguente: il **while** esegue il controllo di una variabile prima di eseguire le istruzioni sottostanti (racchiuse da parentesi graffe), se risulta vero le esegue, il ciclo si ripete finche il controllo non risulta falso; il ciclo **for** esegue un numero di volte specificato il

set di istruzioni ad esso assegnato.

Gli operatori sono dei simboli chiave che permetto di eseguire una operazione matematica (esempio somma), logica/relazionale (esempio verificare l'uguaglianza tra due entità) o assegnazione di un valore ad una entità. I diversi operatori hanno un preciso ordine di esecuzione sia rispetto ai suoi membri che nei confronti degli altri operatori (vedi (Brian W. Kernighan, 2004)). I simboli più usati sono: "*", "+" e "-" per le operazioni matematiche; "==" (uguagliaza), "!=" (diversità), "&&" (operazione logica AND) e "||" (operazione logica OR) per le operazioni logiche e relazionali; "=" (assegnazione), "+=" (addizione e assegnazione) per le operazioni di assegnazione.

Le variabili sono stringhe di caratteri a cui è associata una serie di bit in memoria (registri), il cui contenuto è sempre un numero. Una variabile prima di essere utilizzata deve essere dichiarata associando ad essa un tipo per consentire al sistema di riconoscerla, successivamente potrà essere richiamata per prelevarne il valore o cambiarlo semplicemente digitando la stringa di testo associata.

I tipi di variabile sono molteplici e si differenziano per dimensione occupata, struttura e accessibilità, questi sono o identificati da parole chiave standard (int, char, char*, ecc) o dichiarate dall'utente.Particolare attenzione meritano le variabili di tipo puntatore. Una variabile puntatore contiene l'indirizzo della cella di memoria associata ad un'altra variabile, questo permette ad esempio di andare a cambiare il valore della variabile puntata indipendentemente dal nome a lei assegnato in una parte di codice. Anche la variabile di tipo struct e molto utilizzata in quanto permette di costruire variabili che al loro interno sono costituite da altre variabili di diversi tipi.

Le funzioni servono a raggruppare un certo numero di operazioni che assolvono un determinato compito, restituendo infine il risultato tipicamente nel parametro di ritorno, sono per questo molto importanti ai fini della leggibilità del codice. Per far riconoscere una funzione al compilatore essa deve essere scritta fuori da altre eventuali strutture di sequenza e deve riportare nell'ordine: il tipo di parametro di ritorno, il nome identificativo, gli eventuali parametri di ingresso tra parentesi tonde e il set di istruzioni racchiuse tra parentesi graffe. Le funzioni così definite devono inoltre avere una dichiarazione in testa allo script (prima del main) chiamato prototipo Un codice C è sempre costituito da una funzione **main** che indica al compilatore qual è l'inizio del programma da eseguire. Altre funzioni possono essere definite dall'utente nello stesso script fuori dal main o in un altro file esterno.
Data la possibilità di costruire funzione esternamente allo script principale si costruiscono di solito librerie, sono degli script che contengono le funzioni che vogliono essere richiamate all'interno del proprio codice. Una libreria può essere accompagnata da un file header, cioè un file di testo con estensione ".h" che contiene eventuali direttive utilizzate nello script, le funzioni e i prototipi delle funzioni.

Per utilizzare una libreria è sufficiente indicare al compilatore dove trovare lo script della libreria (file.c) utilizzata e includere nel proprio script il file .h a essa associato, attraverso la direttiva **#include** spiegata qui di seguito.

Le direttive sono delle stringhe precedute dal carattere "#" utili solo nella fase di precompilazione, costituiscono infatti ordini di compilazione. Esistono diverse direttive, quelle più utilizzate sono **include**, **define** e **ifdef.**

La direttiva **include** dice al compilatore di "includere" il contenuto del file specificato a destra nella posizione dove tale direttiva è scritta (in testa al programma fuori dal main). **Define** dice al compilatore di sostituire ogni occorrenza nel codice della stringa che si trova a destra di "**#define**", con quella che si trova all'estrema destra della parola in oggetto (esempio **#define** time 50: sostituisce la stringa time con la stringa 50); o semplicemente definisce una stringa di testo da utilizzare ad esempio nella direttiva **#ifdef**.

Ifdef dice al compilatore di compilare la parte di codice sottostante fino a **#endif** solo se la stringa a destra di **#ifdef** è stata definita con la direttiva **#define**.

Per agevolare la scrittura dei programmi esistono vari ambienti di sviluppo (IDE: integrated design environment) dove è possibile fare ad esempio operazioni di ricerca e visualizzazioni di dipendenza, inoltre viene facilitata la scrittura corretta attraverso il riconoscimento delle parole chiave, delle chiusure parentesi e molto altro. Nel caso specifico di questo progetto di tesi l'ambiente aveva anche la così detta modalità di "debug on line" compatibile con la Discovery F4. Questa modalità consiste nel visualizzare sull' IDE l'esecuzione in tempo reale del proprio script, con eventuale monitoraggio dei singoli registri interni e delle variabili (scelte dall'utente) durante l'esecuzione del programma, si è cosi in grado di individuare con facilità eventuali errori.

Utilizzare il linguaggio C su microcontrollori come quello a bordo della Discovery F4necessita di una approfondita conoscenza del dispositivo da programmare. Infatti, mentre nei PC ci sono architetture standard che consentono l'implementazione univoca di funzioni di alto livello (esempio printf), nei microcontrollori questo non avviene. Sebbene infatti, gran parte del lavoro di basso livello sia fatto dalla casa costruttrice nel fornire le librerie o drivers, resta comunque la necessità di conoscere il funzionamento hardware di medio/basso livello per far funzionare in modo corretto il proprio codice. In conclusione per usare una proprietà hardware è necessaria una conoscenza dei relativi registri interni e dell'impostazione degli eventuali pin ad essa associati (STMicroelectronics, 2013).

4.3 Programma

Il requisito principale del programma è quello di trasferire il pacchetto ricevuto nel modulo USART collegato al PC al modulo USART collegato alla Tau. Questa operazione però dovrà avvenire solo quando sarà la GUI ad essere collegata alla Discovery e non un qualsiasi altro software dal PC. Infatti il modulo USART della Discovery collegato al PC è lo stesso utilizzato

dal codice già esistente, esso quindi può potenzialmente ricevere anche altri comandi oltre a quelli provenienti dalla GUI.

Il primo passo è quindi quello di individuare un discriminante in grado di separare le due situazioni possibili provenienti dal PC. Le stringhe uscenti dalla GUI hanno una caratteristica invariante che è il primo byte inviato, il manuale Tau infatti prevede che ogni stringa inizi con il byte (process code) 0x6E. Quindi, supponendo che eventuali comunicazioni precedenti non abbiano usato tale byte, controllando se il byte ricevuto è uguale a 0x6E si può concludere che il primo comando dalla GUI, e solo da essa, sta arrivando e si intende entrare nella modalità di settaggio della camera.

Dato che il controllore sarà impegnato nella rutine già esistente (che essenzialmente acquisisce ed elabora immagini) è necessario che alla ricezione di un byte il controllore fermi tale routine, esamini il byte ricevuto ed entri eventualmente in modalità di set (se il byte ricevuto è 0x6E). Per fare questo si utilizza una speciale caratteristica dei microcontrollori programmabili: gli interrupt.
L'interrupt è una opzione attivabile su alcune parti hardware del microcontrollore, specificate nello user manual. L'interrupt , se attivo, genera l'interruzione del codice in conseguenza di specifici avvenimenti proprio sulla parte hardware a cui è collegato (esempio ricezione di un byte al modulo USART). Dopo l'interruzione il microcontrollore manda in esecuzione la funzione specifica collegata a quell'interrupt, esaurita, torna ad eseguire il codice precedente da dove si era fermato.

Una volta entrati nella modalità di set è necessario inoltre rimanere in tale condizione finché si decide di terminare la sessione di set. Per fare questo si sfrutta la caratteristica della GUI che prova la connessione ogni 800ms, in particolare se nessun comando o richiesta arriva entro un determinato tempo (circa 4 sec), allora tale situazione è interpretata come volontà di uscire dalla modalità di set. Sarà quindi prevista una struttura **while**, con controllo su una variabile apposita di tipo booleano, che viene eseguito finche tale variabile non assume un valore falso all'interno del ciclo stesso. In questa operazione la variabile addottata è denominata **do_flag_debug** e viene impostata per l'uscita quando scade il tempo massimo di attesa del comando. All'interno del ciclo **while** saranno messe le funzioni che realizzano il "ponti dati".

Il programma da realizzare può essere sintetizzato efficacemente con il diagramma a blocchi riportato in Figura 18.

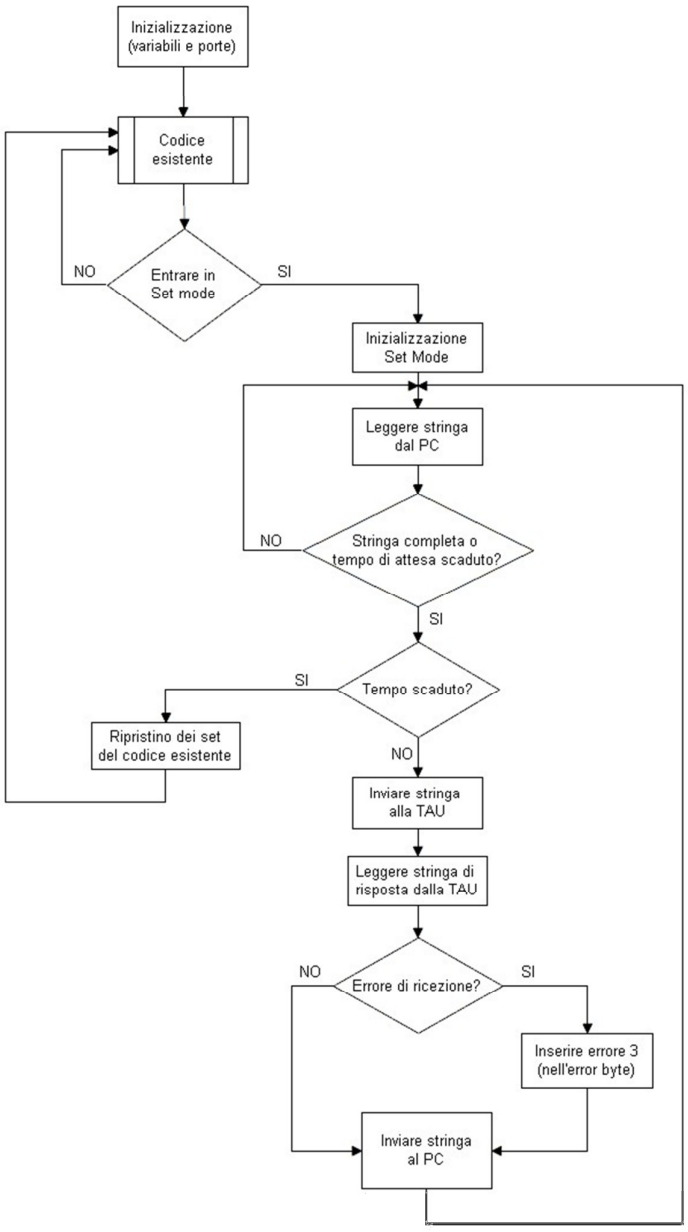

Figura 18: Schema a blocchi del codice caricato sulla Discovery

Nella scrittura del programma si è cercato di utilizzare funzioni già scritte e testate per una maggiore leggibilità del codice e per una maggiore affidabilità. In particolare si è fatto uso dei driver USART messi a diposizione dell'azienda costruttrice (stm32f40x_usart.c) e di driver di alto livello scritti per i programmi già presenti nel pacchetto della discovery (stm32f40x_usart_ HL.c).

Sarà ora spiegata nel dettaglio la scrittura del codice, che è in particolare la traduzione dello

schema a blocchi in Figura 18.

La parte di definizione e dichiarazione è la prima parte del codice e si trova nello stesso script in cui si trova la funzione **main**(). In questa sezione vengono incluse le librerie utilizzate, definite stringe di testo utili con l'apposita direttiva **#define** e dichiarate variabili (non necessariamente richiamate nel main direttamente).

Le librerie utilizzate oltre a quelle stesse del codice pre-esistente è la sola "debug.h". Nello script debug.c (associato a debug.h) sono infatti presenti tutte le funzioni costruite appositamente per realizzare il programma. Per quanto riguarda la direttiva **#defin**e è stata utilizzata per abilitare i due moduli USART necessari: USART2 per il collegamento discovery-camera e USART3 per il collegamento PC-discovery. Infatti nel driver di alto livello stm32f40x_usart_HL.c la parte di inizializzazione dei moduli suddetti viene compilata e quindi eseguita solo se presente una **#define** che riporti USART2_ENABLE e USART3_ENABLE. Entrambe le porte sono in questo modo abilitate avendo avuto cura di impostare prima i relativi pin (vedi 4.1) nel modo opportuno, i parametri di comunicazione (entrambe le porte secondo lo standard della tau) e abilitare gli interrupt nel file stm32f40x_usart_HL.c.
Riguardo le variabili dichiarate nello script in oggetto vi è solo la variabile **do_falg_debug**, di tipo booleano utilizzata per entrare ed uscire dalla funzione di set dei parametri (funzione **debug**).

Prima di entrare nel particolare funzionamento della modalità di set conviene evidenziare il funzionamento della ricezione dei byte ai moduli USART, in quanto gestito da interrupt. Nel driver di alto livello stm32f40x_usart_HL.c sono dichiarati degli array con elementi da 8bit visibili anche da altri script, essi hanno la funzione di buffer USART, cioè di memorizzare tutti i byte ricevuti al modulo in oggetto. Ogni volta che si genera un interrupt perché arriva un byte all'USART il codice momentaneamente in esecuzione si interrompe, viene letto il byte alla porta e immagazzinato nel buffer USART. Questo avviene nelle funzioni di gestione degli interrupt dei moduli USART2 e USART3. In particolare le funzioni già fornite sono **USART2_IRQHandler()** e **USART3_IRQHandler()**. Dato che tali funzioni vengono eseguite per qualsiasi evento accaduto alla porta (ricezione byte, errore di overflow del registro, ecc.), si necessita di un controllo sull'origine dell'interrupt, si è infatti interessati al solo generato dall'arrivo di un byte alla porta; pertanto all'interno di tali funzioni viene fatta per prima cosa questa verifica. Se la fonte è quella voluta (byte alla porta) si hanno due gestioni diverse per la porta 2 e la porta 3. Riguardo USART3 si controlla il byte ricevuto e lo stato di **do_flag_debug**, se quest'ultima ha valore "falso" e il btye ricevuto è diverso da 0x6E si esegue il vecchio codice, altrimenti si immagazzina direttamente il dato nel buffer e si imposta la variabile **do_flag_debug** a valore "vero". A questo punto si entrerà o si rimarrà (se si è già entrati) nella funzione **debug**(). Riguardo USART2 si controlla invece solo la variabile **do_flag_debug** che, se di valore "vero" (cioè si è all'interno della funzione **debug**), fa immagazzinare direttamente il byte ricevuto nel buffer, altrimenti esegue la funzione del

codice preesistente.

All'interno del **main()** il punto di accesso alla funzione **debug()** è unico e avviene all'interno del loop infinito dove è situata la funzione di acquisizione ed elaborazione dell'immagine **Load_image()**. In particolare ad ogni iterazione del ciclo in oggetto il programma tenta di accedere alla funzione debug, ma il tentativo ha successo solo se la variabile **do_flag_debug** ha valore logico "vero".

La variabile **do_flag_debug** è visibile da diversi script e può essere settata in due soli punti esterni al main: all'interno della funzione che gestisce l'interrupt del modulo USART3 (**USART3_IRQHandler()**) o all'interno della funzione **debug()**.

La funzione **debug()** è quella che effettivamente assolve il compito di "ponte" ed è contenuta nel rispettivo file debug.c appositamente scritto. Questa funzione è costituita da una parte iniziale di dichiarazione e inizializzazione di variabili interne, da una sezione di inizializzazione della modalità di set, da una sezione centrale e da una sezione di uscita o riconfigurazione.
Nella prima sezione si dichiarano e inizializzano due array con 34 elementi di 8bit chiamati **data_bridge** e **data_bridgeBackup** che sono destinati a contenere la stringa da trasferire. Vengono inoltre dichiarate altre tre variabili da 8bit che sono **string_lengthCmd**, **string_lengthReply** ed **i**; queste sono destinate rispettivamente: a contenere la lunghezza della stringa ricevuta dal PC, a contenere la lunghezza della stringa ricevuta dalla camera e ad essere usata come variabile ausiliaria; l'ultima è semplicemente da utilizzare per accedere ai vari elementi dei vettori dichiarati.

La sezione di inizializzazione della modalità di set disabilita semplicemente l'acquisizione dell'immagini da parte del microcontrollore e il segnale di sincronia di acquisizione. In questo modo il microcontrollore non sarà contemporaneamente occupato nell'acquisizione dell'immagine dalla camera durante la variazione dei parametri.

La sezione centrale è costituita da un ciclo while che viene eseguito finche la variabile discriminante **do_flag_debag** non viene portata a valore "falso". All'interno del **while** viene per prima cosa eseguita una attesa per aspettare che tutta la stringa sia ricevuta dal PC. L'attesa è gestita dalla funzione **ReceiveStr()** posizionata all'interno di un altro ciclo **while** da cui si esce quando si riceve una stringa o scade il tempo massimo di attesa. Successivamente, se non è scaduto il tempo massimo, si passa ad un controllo di integrità della stringa, quindi si procede all'invio della stringa alla camera attraverso la porta USART2; questo passo è gestito dalla funzione apposita **SendStr()**. Effettuato l'invio si compie un backup della stringa inviata in **data_bridgeBackup** per non perderla, infatti la variabile **data_bridge** sarà sovrascritta con la risposta della camera. Quindi si entra in attesa della risposta della camera sempre attraverso la funzione **ReceiveStr(),** in questo caso la stringa e la sua lunghezza sono immagazzinate rispettivamente in **data_bridge** e **string_lengthReply**. Ricevuta la risposta dalla camera o scaduto il relativo tempo di attesa (che è fissato ad un

massimo di 500ms) si procede all'invio al PC di una risposta. La stringa inviata al PC è quella ricevuta dalla camera nel caso in cui il tempo non scada prima della ricezione completa della stringa, altrimenti viene inviata la stringa contenuta nella variabile data_bridgeBackup con la sostituzione del byte di errore previsto dal formato della stringa. L'errore che in questo caso viene inviato è quello corrispondente all'errore "Cam busy" (FLIR, 2010). A questo punto il ciclo **while** principale si ripete e riinizia l'attesa della stringa dal PC.

L'ultima sezione della funzione **debug**, quella di riconfigurazione, è eseguita solo se scade il tempo di attesa massimo della stringa dal PC (circa 4 sec) e consiste nel ripristino delle due funzioni che consento l'acquisizione dell'immagini e la sincronia di acquisizione. In questo modo il codice in esecuzione prima della funzione **debug** potrà tornare a funzionare. Una volta ripristinate le due funzioni si esce dalla modalità di set e verrà ripreso il ciclo while situato nel main che prevede l'acquisizione, l'elaborazione dell'immagine e il controllo per l'entrata in modalità set.

In Figura 19 viene riportata la parte di codice relativa alla funzione **debug()** appena spiegata.

```
void debug(void){

  unsigned char data_bridge[34], data_bridgeBackup[34],i=0;
  unsigned char string_lengthCmd=0, string_lengthReply=0;
  memset(data_bridge,0x00,34); //init variable to 0
  memset(data_bridgeBackup,0x00,34);
  //bool Byte_at_USART2_flag;

  /*access debug mode: stop image acquisition*/
  if(do_flag_debug == true){
    DCMI_CaptureCmd(DISABLE);
    TAU320_Sync_Enable(SYNC_DISABLE);
  }

  while(do_flag_debug){
  /*read first string command*/
    while((string_lengthCmd==0) && i<LongWait){//wait more time then the receive function
for a comand from pc

      string_lengthCmd = ReceiveStr(USART3, data_bridge);
      i++;
    }
    if(i<LongWait){ //transfer the received byte

    STM_EVAL_LEDOn(LED3);
    /*send string command*/
      if(string_lengthCmd=data_bridge[5]+10)//check if the string command is complete
        SendStr(USART2, data_bridge, string_lengthCmd);

    /*wait Tau reply for ReplyWait_ms at last */
      for(i=0;i<34;i++){
        data_bridgeBackup[i]=data_bridge[i]; //backup data_bridge
        data_bridge[i]=0;
      }
      Delay_mS(3);

      string_lengthReply = ReceiveStr(USART2, data_bridge); //wait raply for a maximum time
(see ReceiveStr)

    /* get reply*/

    /*check errors*/
      if(string_lengthReply == 0){/*reply with command string but insert error case 1 (cam
busy)*/

        for(i=0;i<34;i++){
          data_bridge[i]=data_bridgeBackup[i];
```

```
      }
      data_bridge[1]=0x01; //inseret error
      string_lengthReply = string_lengthCmd;
    }
   /* send reply string*/
    SendStr(USART3, data_bridge, string_lengthReply);

   }else{ //exit debug mode because the computer is not connected anymore

  /*check for exit condition*/
     do_flag_debug=false;//exit debug mode
     /*Exit from debug mode*/
     DCMI_CaptureCmd(ENABLE);
     TAU320_Sync_Enable(SYNC_ENABLE);
   }

   STM_EVAL_LEDOff(LED3);
   /*reset variables*/
   i=0;
   string_lengthReply = 0;
   string_lengthCmd=0;
   memset(data_bridge,0x00,34);
   memset(data_bridgeBackup,0x00,34);
  }

}
```

Figura 19: Funzione debug()

Si spiega ora il funzionamento delle singole funzioni richiamate in **debug()**, costruite appositamente per realizzare del programma.

La funzione **ReceiveStr()** realizza un ciclo di attesa, per una specificata porta, per la ricezione della stringa completa. I sui parametri di ingresso sono quindi: il nome della porta su cui attendere e il puntatore all'array dove mettere la stringa. Questa funzione fornisce inoltre come output la lunghezza della stringa ricevuta. L'attesa è realizzata con un ciclo **while** che controlla il riempimento del già citato buffer e il tempo trascorso dall'inizio dell'attesa. Se nelle iterazioni successive alla prima i dati contenuti nel buffer non aumentano, ed inoltre esso non rimane vuoto, allora si esce dal ciclo e la stringa completa viene trasferita dal buffer all'array specificato in ingresso. Se invece scade il tempo massimo si esce ugualmente dal ciclo ma la lunghezza fornita come output della funzione è 0. Come ultima operazione questa funzione azzera l'indirizzo di riempimento del buffer della specificata porta.

La funzione **SendStr()** consente di inviare una generica stringa di byte stringa. Per fare ciò necessita in ingresso della porta su cui trasmettere, del puntatore alla stringa da trasmettere e della lunghezza di tale stringa. La funzione è realizzata in modo molto

semplice dichiarando una variabile ausiliaria "i" che viene utilizzata per fare un ciclo **for**. All'interno di questo ciclo, che si ripete per ogni elemento della stringa da inviare, viene richiamata la funzione **USART_SendData(),** fornita dalla casa costruttrice nel file stm32f40x_usart.c. Questa permette infatti di inviare un singolo byte ad una specificata porta. Successivamente, sempre all'interno del ciclo **for**, vi è un ciclo **while** che attende che il byte sia stato inviato. Infatti quando un byte viene inviato, un flag (bit) collegato al registro del modulo USART specificato passa allo stato logico 0 ad indicare che il registro di trasmissione è stato svuotato.

4.4 Risultati

Il programma riportato sopra è il risultato finale di una serie di correzioni apportate durante il processo di verifica, qui di seguito verranno elencati gli errori trovati rispetto alla prima versione ed eventuali problemi risolti o meno durante la validazione.

I due requisiti specificati all'inizio del capitolo sono stati validati in due sedi separate, avvalendosi della modalità di debug dell'ambiente di sviluppo. In primo luogo è stato validato con successo il requisito di trasferimento della stringa con il solo programma fatto dal sottoscritto presente sulla Discovery (commentando le righe di codice già esistenti). Il codice in oggetto è stato validato mediante prove a step di riga successivi della funzione debug. In particolare, avvalendosi dei breackpoint di esecuzione, si imponeva lo stop del codice a valle della riga da verificare, in questo modo se la riga veniva eseguita il programma si bloccava e potevano essere controllati i risultati confrontandoli con quelli attesi. In caso contrario il problema era in una sottoparte del programma , allora il procedimento andava ripetuto posizionando il brackpoint nella parte di sottocodice interessata da malfunzionamento.

Durante questo primo processo di verifica, nella funzione **ReceiveStr()** si è scoperta la mancata gestione, nel ciclo di attesa, del caso in cui non venisse ricevuto niente alla porta connessa al PC (per un tempo inferiore a quello massimo stabilito). Questo risultava in una interpretazione da parete del codice di una stringa di lunghezza nulla come completa, il che non era ammissibile. Nello specifico succedeva che, anche se il tempo non fosse scaduto, il **while** sarebbe terminato in quanto il controllo sul riempimento sarebbe risultato falso (cioè stringa completamente ricevuta), infatti l'indice di riempimento del buffer rimane a 0 per due iterazioni successive se appunto non si riceve niente. Il problema è stato quindi risolto includendo il caso nel while come specificato di seguito:

while((USART2_bytes_read==0 || USART2_bytes_read!=i) && timer<ReplyWait_ms){

E' stata poi aggiunto un controllo di integrità della stringa come riportato sotto al fine di assicurare ulteriormente l'invio di una stringa completa, evitando così possibili compromissioni della camera.

If(string_lengthCmd=data_bridge[5]+10)//check if the string command is complete

Con le suddette correzioni il codice funzionava in modo corretto per quanto riguarda il primo requisito.

Problemi più gravi sono stati invece incontrati nella compatibilità tra i codici, quello del sottoscritto e quello già esistente nella Discovery. In particolare durante la seconda parte della validazione mentre si provava l'assieme dei due codici sono emersi due errori difficilmente individuabili. Il primo comportava il mancato funzionamento di una parte del codice preesistente, questo molto probabilmente era dovuto all'utilizzo di variabili in comune tra il codice del sottoscritto e quello già esistente.
Il secondo errore riguardava invece un hardware fault che avveniva al tentativo di uscita dalla funzione debug. In particolare al settaggio della variabile **do_flag_debug** della prima versione del codice si entrava in un ciclo infinito il cui commento riportato affianco indicava appunto l'avvenuto hardware fault. Questo errore era quasi sicuramente dovuto ad una sovrascrittura di un registro destinato contemporaneamente ad un'altra variabile rispetto a **do_flag_debug**, risolvere tale problema in modo diretto sarebbe stato molto difficile in quanto era ncessaria l'individuazione e la gestione del singolo registro fonte dell'errore. Si è deciso quindi di risolvere il problema in modo indiretto togliendo tutte le variabili in comune con il programma preesistente e di cambiare la modalità di uscita dalla funzione **debug()**. In particolare la prima versione del codice prevedeva l'uscita dalla funzione a seguito di uno specifico comando inviato dalla GUI (Reset camera) che è stato sostituito con il funzionamento sopra riportato.

Dopo questa modifica il codice che implementa la modalità di set funzionava correttamente e poteva essere dichiarato valido.

5 Conclusioni

Nella prima parte del lavoro è stata fatta la validazione della graphical user interface attraverso l'utilizzo di specifici strumenti informatici e hardware precedentemente citati. Si è pertanto resa necessaria una approfondita conoscenza dei protocolli e delle funzioni adottate dai dispositivi stessi.

In questa prima validazione si sono quindi corretti errori e indecisioni sul codice presenti dal primo sviluppo, eseguito tra l'altro dal sottoscritto durante il periodo di tirocinio. Alla fine di questo processo la GUI è risultata valida seppur con minime differenze dal software originale. Tali differenze sono di seguito elencate e giustificate:

1. Mancata implementazione dello "scratch pad" (nella scheda setup) e del controllo "video color" (scheda video): I comandi non erano presenti sul manuale ed inoltre non sono stati giudicati indispensabili per l'attività futura da svolgere.
2. Implementazione di una scheda in più ("Avanzate"): attraverso la quale si ha accesso diretto a comandi specifici importanti (ad esempio il comando diretto dello shutter) citati sul manuale, ma che non rientrano nel contesto delle altre schede.
3. Impossibilità discattare e visualizzare foto di prova: anche se la GUI originale lo permette, questa non è una funzione necessaria per la GUI riprodotta in quanto le foto di prova vengono acquisite ed elaborate dalla Discovery F4.

Nella seconda fase del progetto ci si è occupati della programmazione della scheda discovery F4 allo scopo di farle trasferire alla camera i comandi provenienti dalla GUI. Il codice doveva essere anche in grado di convivere con uno già esistente che si occupava dell'acquisizione ed elaborazione dell'immagine. Il processo ha richiesto una approfondita conoscenza dell'hardware da programmare e del linguaggio di programmazione C.

Dopo aver sviluppato il codice si è passati alla sua validazione evidenziando anche qui gli errori di seguito riportati:

1. Riconoscimento di una stringa nulla come completa.
2. Mancato controllo sulla integrità della stringa (per scungiurare compromissioni del firmware della Tau).
3. Mancato funzionamento dei due codici (preesistente e del sottoscritto) se presenti simultaneamente.

I primi due errori sono stati individuati e risolti, mentre l'ultimo è stato risolto in modo indiretto con una modifica alla modalità di uscita dalla funzione **debug()** e con una separazione delle variabili utilizzate dai due codici.

Per quanto riguarda l'aspetto formativo personale, oltre che tecnico, dell'esperienza si può dire che sono state acquisite importanti nozioni sullo svolgimento di un progetto e sulla sua

organizzazione, sia in ambito generico che in quello specifico dello spazio. Durante lo svolgimento del lavoro la pianificazione ha svolto un ruolo cruciale, nonché l'organizzazione delle singole fasi di lavoro e la loro tracciabilità. Quest'ultima è una caratteristica fondamentale di un progetto ed agevola molto sia le operazioni dell'esecutore che l'interfacciamento con altri operatori. Nell'ambiente spazio un buon interfacciamento è fondamentale, infatti in un satellite, ad esempio come ESEO, prendono parte al progetto molti team ed è necessario che ognuno riesca a capire e verificare rapidamente il lavoro dell'altro, questo è possibile attraverso appunto la tracciabilità delle operazioni. Nel caso specifico di questo progetto tale caratteristica è soddisfatta ad esempio dalla checklist e dai file di log per la validazione della GUI, nel caso del programma della Discovery è invece costituita ad esempio dai vari commenti di sintesi presenti sul codice, nonché dal presente documento.

In ultima conclusione quindi il progetto è stato molto formativo sia dal punto di vista personale che tecnico, ha infatti permesso, oltre allo sviluppo delle capacità di gestione di un progetto e di un codice informatico, anche lo sviluppo di un oggetto fisico come l'RS232 Tau adapter.

Bibliografia

Brian W. Kernighan, D. M. (2004). *Il linguaggio C. Principi di programmazione e manuale di riferimento.* Pearson - Prentice Hall.

Curzi, G. (2014). *Interfaccia software in labview per il controllo dei parametri della fotocamera FLIR Tau-D.*

FLIR. (2010, January). *Tau Camera - User's Manual.* Tratto da www.flir.com: http://www.flir.com/uploadedfiles/tau_320_users_guiderev120.pdf

STMicroelectronics. (2013, June). *STM32F407xx - Datasheet.* Tratto da www.st.com: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf

Appendice A

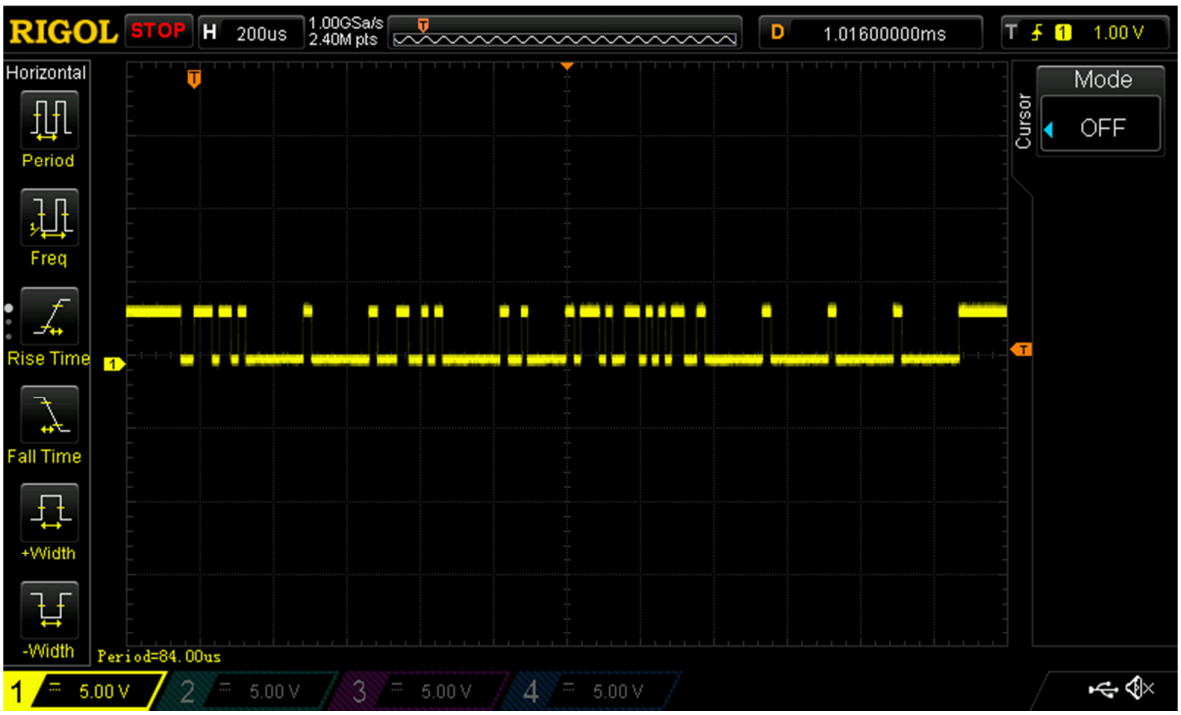

Figura 20: Segnale di tensione al punto di controllo T2 a seguito del comando "richiesta ROI"

Sopra è rappresentato il segnale di tensione al punto di controllo T2 (vedi Figura 6: Schema RS232 Tau Adapter) provocato dall'invio del comando di richiesta della ROI. In riferimento al manuale della Tau e notando che in figura il tempo incrementa verso destra, possono essere individuati i singoli bit trasmessi. A tale scopo va anche considerato che ogni pacchetto da 8 bit è preceduto da uno start bit e seguito da uno stop bit.

Appendice B

Il **Cyclic Redundancy Check** o **CRC** è un modo efficiente ed ampiamente utilizzato per verificare se le informazioni ricevute sono affette da corruzioni di tipo accidentale. Esso è rappresentato da una funzione il cui ingresso è un insieme di byte (array di byte o una stringa di carattere ASCII) detto anche byte stream e la cui uscita è un valore numerico, solitamente un intero a 8, 16 o 32 bit. Il termine CRC viene utilizzato per riferirsi sia alla funzione che al valore da essa calcolato.

Il valore CRC è calcolato come il resto della divisione in aritmetica modulo 2 tra il polinomio associato al dato traslato (p) e il polinomio generatore associato(g). Il dispositivo che riceve il dato, per controllarne la correttezza, sottrae il resto (valore del CRC) e ricalcola il CRC, se quest'ultimo ha valore 0 il dato è corretto altrimenti no.

La funzione che identifica il CRC è associata ad un polinomio generatore (g) i cui coefficenti (a) sono dati dai singoli bit di un numero binario, ad esempio:

$$\mathrm{g(x)} = \sum_{k=0}^{\infty} a_k x^k \qquad \Longrightarrow \qquad \mathrm{g(x)} = x^5 + x^3 + 1$$

$$\ldots 00101001$$

In modo analogo il polinomio associato al dato (b) è dato dal numero binario identificato da tutto il byte stream, questo però deve essere traslato di c posizioni, con c uguale al grado del polinomio (g) (c=5 nell'esempio corrente), andando a formare il polinomio associato al dato e traslato (p), ad esempio:

$$\mathrm{b(x)} = x^7 + x^2 + 1 \qquad\qquad \mathrm{p(x)} = x^{12} + x^7 + x^5$$

Traslare il polinomio (b) originato direttamente dal dato, permette di renderlo divisibile per il polinomio (g) qualunque sia la sua lunghezza. Questo introduce la possibilità di inizializzare i bit corrispondenti di traslazione con valori diversi tra 0 e 1, si parla quindi di inizializzazione del CRC; solitamente però tali bit sono inizializzati al valore 0.

A questo punto facendo uso dell'aritmetica in modulo 2 si effettua la divisione tra (p) e (g), in particolare si effettua la tipica divisione in colonna tra polinomi. Si può notare che nel ricavare il resto ad ogni passaggio della divisione, fino a quello finale, non si fa altro che l'operazione logica EXOR (proprio perché si è in aritmetica modulare 2) tra i due numeri binari associati ai polinomi sovrastanti. Questa è facilmente implementabile sui calcolatori, quindi gli algoritmi di calcolo ricavano direttamente il resto della divisione con semplici operazioni EXOR ricorsive sul dato traslato di c posizioni.

Il più comune CRC nelle applicazioni di comunicazione (bluetooth, Schede SD, Tau320) è il CRC-16 CCITT, il cui polinomio associato normale (esistono infatti i polinomi invertito e invertito reciproco) è identificato dal numero esadecimale 0x1021.

Appendice C

Esempio della check list utilizzata: nei “checkbox” veniva annotato il corretto funzionamento o eventuali modifiche da apportare.

Scheda GUI	Comandi				Checkbox GUI originale	Checkbox GUI
1. STATUS	Part number					
	Serial number					
	FPA temperature					
2. SETUP	FFC	External				
		Auto				
		FFC interval				
		Low gain FFC interval				
		Temperature change				
		Low gain temperature change				
		Manual				
		Do FFC				
	Op. mode	Frozen				
		Real time				
	Test pat.	Ramp	Ascending			
			Big vertical			
			Horizontal shade			
			Color bars			
			Ramp with steps			
		Off				
	Ext. Sync.	Disable				
		Slave				
		Master				
	Gain mode	Auto				
		Low				
		High				
		Manual				
	Save setting					
	Factory default					
	Reset Camera					
	Scratch pad					
	ADVANCED	Get spot meter data				
		Shutter				
		DDE Trashold				
		Spatial Trashold	Manual			
			Auto			
		DDE gain				
3. VIDEO	Analog	Orientation	Invert			

			Revert			
			Invert + Revert			
			None			
		Polarity/LUT	W-H			
			B-H			
			F			
			R			
			G			
			I-1			
			I-2			
			S			
			C-1			
			C-2			
			I-F			
			Rain			
			Custom (R-H)			
		FFC				
		DDE Gain				
		Pan & zoom	Zoom 2x			
			Zoom 4x			
			Unzoom			
			Tilt	2x		
				4x		
			Pan	2x		
				4x		
			Center			
			Pan			
			Fine			
		Video	On			
			Off			
		Video standard	NTSC			
			PAL			
	Digital	Camera type				
		Camera dig. out	Off			
			14 bit Filtered			
			8-bit Filtered			
			14-bit Row			
		Xp bux out	None			
			BT656			
			CMOS			
4. AGC	Algorithm	Automatic				
		Once bright				
		Auto bright				
		Manual				
		Linear Histogram				

	Lin. Param.	Contrast		
		Brigthness		
		Brigthness bias		
	Auto Param.	Plateau value		
		ITT Mean		
		Max gain		
	ROI	4x		
		2x		
		No zoom		
5. Thermal	Spot meter mode	Fahrenheit		
		Celsius		
		Therm.		
		Dig. & Therm		
		Dig.		
		No Dig. & No Therm		
		Off		
	Isotherm	Upper		
		Lower		
		%		
		°C		
		Enable		
	Gain switch value	H-L Thrash		
		H-L POP		
		L-H Thrash		
		L-H POP		

Appendice D

Qui sotto è riportato un pezzo del file di log in uscita dal punto uno della validazione della GUI (correttezza delle stringhe inviate). In particolare è relativo alla ad una parte di controllo sulla scheda 2 della check-list. Come si può notare l'ordine dei comandi è esattamente quello riportato nella check-list.

6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs		OK
6E-00-00-0B-00-02-0F-08-00-01-10-21	B	02	1021	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0D-00-02-BD-A8-00-02-20-42	D	02	2042	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0D-00-04-DD-6E-00-02-00-04-2E-E4	D	04	2EE4	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0E-00-02-E4-F8-00-01-10-21	E	02	1021	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0E-00-04-84-3E-00-01-00-03-07-53	E	04	753	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0B-00-02-0F-08-00-00-00-00	B	02	00	crcs OK		
6E-00-00-20-00-02-79-3F-00-00-00-00	20	02	00	crcs OK		
6E-00-00-0C-00-02-8A-98-00-00-00-00	C	02	00	crcs OK		

Di seguito sono inoltre riportati due estratti dei file di log della comunicazione GUI-Tau relativi ad una parte della scheda 4 della check-list, in questi file è riportata solo la risposta della camera.

A sinistra le risposte ai comandi della GUI della casa costruttrice, a destra quella della GUI validata.

Come si può notare non c'è una precisa corrispondenza delle stringhe sebbene sia stata usata la check-list, questo è dovuto al fatto che ad ogni passo della check-list si facevano anche altre prove per vedere le interazioni e le risposte dei comandi, anch' esse venivano ovviamente registrate.

#6E #00#00#13#00#02#E5#CA#00#05#50#A5 #6E #00#00#13#00#02#E5#CA#00#05#50#A5 #6E #00#00#3F#00#02#16#6D#00#96#E3#7F #6E #00#00#55#00#02#4A#C7#00#7F#8F#78 #6E #00#00#6A#00#02#A3#53#00#07#70#E7 #6E #00#00#3E#00#02#21#5D#00#40#48#C4 #6E #00#00#15#00#02#57#6A#12#9E#07#66 #6E #00#00#14#00#02#60#5A#00#28#A5#6° #6E #00#00#18#00#02#15#3B#FF#01#13#DE #6E #00#00#00#00#00#DF#BB#00#00 #6E #00#00#3F#00#02#16#6D#00#96#E3#7F #6E #00#00#13#00#02#E5#CA#00#05#50#A5 #6E #00#00#13#00#02#E5#CA#00#05#50#A5 #6E #00#00#3F#00#02#16#6D#00#96#E3#7F #6E #00#00#55#00#02#4A#C7#00#7F#8F#78 #6E #00#00#6A#00#02#A3#53#00#07#70#E7 #6E #00#00#3E#00#02#21#5D#00#40#48#C4 #6E #00#00#15#00#02#57#6A#12#9E#07#66 #6E #00#00#14#00#02#60#5A#00#28#A5#6° #6E #00#00#18#00#02#15#3B#FF#01#13#DE #6E #00#00#67#00#02#E1#02#08#BE#CF#BC #6E #00#00#D6#00#08#89#87#FF#FF#FF#FF#FF#FF#FF#FF#FF#A6#E1 #6E #00#00#12#00#08#73#B0#FF#5E#FF#80#00#A2#00#80#78#4B #6E #00#00#4C#00#08#36#7F#FF#B0#FF#C4#00#50#00#3C#21#2B #6E #06#00#32#00#00#B9#FB#00#00 #6E #06#00#32#00#00#B9#FB#00#00	#6E #00#00#11#00#02#8B#AA#00#00#00#00 #6E #00#00#12#00#02#D2#FA#00#00#00#00 #6E #00#00#13#00#02#E5#CA#00#00#00#00 #6E #00#00#14#00#02#60#5A#00#20#24#62 #6E #00#00#15#00#02#57#6A#20#00#06#E6 #6E #00#00#18#00#02#15#3B#00#00#00#00 #6E #00#00#1F#00#02#90#AB#00#00#00#00 #6E #00#00#20#00#02#79#3F#01#40#7B#F5 #6E #00#00#21#00#02#4E#0F#00#00#00#00 #6E #00#00#22#00#02#17#5F#00#00#00#00 #6E #00#00#23#00#04#40#A9#00#5A#00#5F#32#15 #6E #00#00#25#00#02#92#CF#00#00#00#00 #6E #00#00#2B#00#02#89#CE#00#00#00#00 #6E #00#00#3C#00#02#4F#3D#00#3C#F7#DF #6E #00#00#3E#00#02#21#5D#00#40#48#C4 #6E #00#00#3F#00#02#16#6D#00#96#E3#7F #6E #00#00#4C#00#18#24#4E#FF#60#FF#80#00#A0#00#80#FF#B0#FF#C4#00#50#00#3C#FF#D8#FF#E0#00#28#00#20#87#27 #6E #00#00#55#00#02#4A#C7#00#7F#8F#78 #6E #00#00#66#00#20#D2#12#34#31#33#32#30#30#30#39#48#2D#53#50#4E#4C#58#5E#34#31#33#32#30#30#30#39#48#00#00#00#00#00#00#00#78#4A #6E #00#00#6A#00#02#A3#53#00#08#81#08 #6E #00#00#70#00#04#47#37#00#00#00#00#00#00 #6E #00#00#72#00#02#49#91#00#00#00#00 #6E #00#00#79#00#02#B9#60#00#00#00#00 #6E #00#00#2C#00#02#0C#5E#00#10#12#31 #6E #00#00#E2#00#02#31#A8#00#64#2C#22 #6E #00#00#E3#00#02#06#98#01#1B#90#6B #6E #00#00#DB#00#08#CB#D6#00#8C#00#14#00#64#00#5F#98#7F

Appendice E

Il MAX3232 è un integrato progettato per interfacciare circuiti che lavorano a livelli di tensione diversi, fornisce per questo quattro linee di interfaccia, due di trasmissione e due di ricezione. In particolare si occupa di adattare i livelli logici e di tensione RS232 standard ai rispettivi livelli utilizzati nei moduli USART (Universa Syncronous Asyncronous Receiver Trasmitter) dei microcontrollori. Questi ultimi infatti sono realizzati con tecnologie che lavorano in un rage di tensione non negativo, generalmente con tensioni di alimetazione di 3.3 o 5 volt, ma possono essere anche superiori.

Il range di alimentazione del MAX3232 va da 3 a 5.5 volt ed è proprio quello che serve per adattare i livelli di tensione della seriale a quelli della Tau320, ma non i livelli logici in quanto, come già detto in 2.3, non sono gli stessi di quelli usati dai moduli USART.

L'integrato in oggetto riesce ad adattare i livelli di tensione attraverso dei traslatori di tensione a pompa di carica. Per fare questa operazione il MAX3232 deve essere alimentato con la tensione più bassa (Figura 21), cioè quella dell'evetuale microcontrollore. Poi attraverso l'ausilio dei quattro condensatori collegati come in Figura 21, verrà creta la tensione duale simmetrica necessaria per la comunicazione seriale.

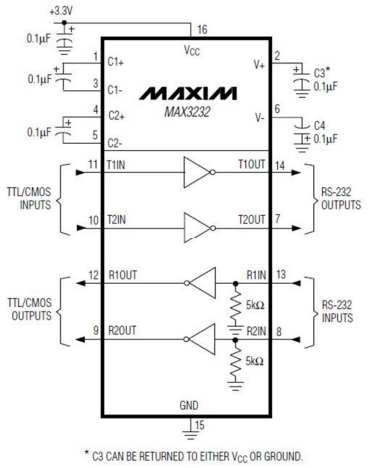

Figura 21: Tipico collegamento del MAX3232

L'utilizzo dei condensatori puo essere spiegato in linea di principio con l'ausilio della Figura 22.

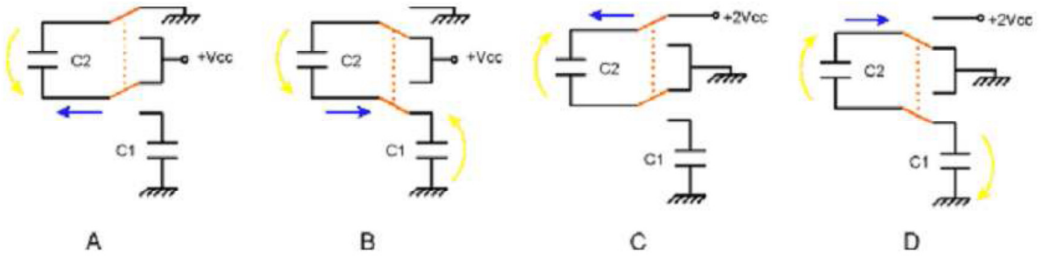

Figura 22: Schema di principio dei traslatori di tensione a pompa di carica

L'innalzamento della tensione positiva è spiegato attraverso la Figura 22 A e B in questo modo:

- Inizialmente il condensatore C2 viene connesso tra massa e Vcc; quindi la corrente (in blu) carica C2 alla tensione di alimentazione (in giallo). Si ha Vc2 = Vcc
- C2 viene successivamente connesso tra Vcc ed un secondo condensatore C1; la tensione ai capi di C1 deve essere uguale alla somma di Vcc e Vc2 e quindi C2 si scarica verso C1, che aumenta la propria tensione rispetto a massa.
- Il processo è ripetuto fino a quando la tensione ai capi di C1 è uguale a 2Vcc: in questo caso, infatti, C2 non si può più scaricare.

Analogamente le figure C e D mostrano come è possibile l'inversione di tensione partendo dalla tensione gia innalzata:

- Inizialmente C2 è caricato alla tensione di alimentazione a 2Vcc (ricavata con il precedente circuito)
- Quindi C2 è connesso tra massa e C1. In questo modo C1 si carica a -2Vcc.

In questo caso specifico, dove la tensione di alimentazione del MAX3232 è di 3.3 volt, si ottiene una tensione duale di circa ±6 volt.