UNIVERSITÀ
DEGLI STUDI
FIRENZE

DOTTORATO DI RICERCA IN
TECNLOLOGIE ELETTRONICHE PER
L'INGEGNERIA DELL'INFORMAZIONE

CICLO XXVIII

COORDINATORE Prof. Tortoli Piero

SVILUPPO DI SOFTWARE E FIRMWARE REAL TIME PER UNA NUOVA PIATTAFORMA DI RICERCA AD ULTRASUONI

Settore Scientifico Disciplinare ING-INF/01

Dottorando
Dott. Giannini Gabriele

Tutore
Prof. Tortoli Piero

Coordinatore
Prof. Tortoli Piero

Anni 2012/2015

Indice

Introduzione 1

Capitolo 1 – Il sistema ULA-OP 256 3

1.1 ULA-OP: una piattaforma aperta 3
1.2 Miglioramenti introdotti nel sistema ULA-OP 256 5
1.3 Architettura del sistema ULA-OP 256 7
1.3.1 Scheda di interfaccia 8
1.3.2 Schede front-end a 32 canali 8
1.3.3 Scheda di trasmissione in alta potenza 10
1.3.4 Connettore sonda 11

Capitolo 2 – Sviluppo del software "Configuration Editor" 13

2.1 Configurazione del sistema ULA-OP 13
2.2 Sintassi dei file di configurazione 14
2.3 Il software "Configuration Editor" 18
2.3.1 Interfaccia Utente 19
2.3.1.1 General 20
2.3.1.2 Sequencer 22
2.3.1.3 Acquire 23
2.3.1.4 Blocksequencer 24
2.3.1.5 Modules 25
2.3.1.6 Item Editor 26
2.3.2 Modularità 27
2.3.3 Funzionamento del software 30
2.3.4 Salvataggio e lettura del file di configurazione 33

2.4 Configuration editor con funzionalità drag & drop 35
2.4.1 Interfaccia utente 36
2.4.1.1 General 37
2.4.1.2 Modules 38
2.4.1.3 Acquire 39
2.4.1.4 Sequencer 40
2.4.2 Implementazione della funzionalità “Drag & drop” 41

Capitolo 3 – Sviluppo di firmware per DSP 45
3.1 Scelte progettuali e requisiti 45
3.2 Il firmware di boot 47
3.2.1 Inizializzazione del DSP 48
3.2.1.1 PLL 49
3.2.1.2 Cache 49
3.2.1.3 GPIO 49
3.2.1.4 SPI 50
3.2.1.5 I2C 50
3.2.1.6 DDR3 51
3.2.1.7 SRIO 51
3.2.1.8 Rimappatura della memoria condivisa 52
3.2.2 Programmazione delle tabelle di routing 52
3.2.3 Caricamento della configurazione della FPGA e del firmware applicativo 53
3.3 Il firmware applicativo 54
3.3.1 Inizializzazioni per l’applicazione 55
3.3.1.1 DMA 55
3.3.1.2 Interrupt 56
3.3.1.3 Timer hardware 56
3.3.2 Kernel, eventi e task 56
3.3.2.1 Eventi 59
3.3.2.2 Task 60
3.3.2.3 Kernel 61
3.3.3 Il meccanismo di cambio di contesto 62

3.3.4 Il gestore della memoria 64
3.3.5 Il gestore delle risorse condivise 65
3.3.6 Il task per la gestione dei timer 65
3.3.7 Il task per la gestione dell'interfaccia SRIO 67
3.3.8 Il gestore dei canali di comunicazione 69
3.3.8.1 Slot di comunicazione 69
3.3.8.2 Identificatore di canale 70
3.3.8.3 Creazione del canale 70
3.3.8.4 Comunicazione DSP-DSP 72
3.3.8.5 Comunicazione Core-Core 73
3.3.8.6 Comunicazione Task-Task 74

Capitolo 4 – Applicazione: metodi di indagine multi-beam 75

4.1 Introduzione alle tecniche di beamforming parallelo 75
4.2 La tecnica OFDM 76
4.2.1 Setup per le acquisizioni in vitro 78
4.2.2 Risultati delle acquisizioni in vitro 81
4.2.3 Acquisizioni in vivo 84
4.3 OFDM multifocus 85
4.3.1 Setup per le acquisizioni in vitro 85
4.3.2 Risultati delle acquisizioni in vitro 86
4.4 Conclusioni 89

Capitolo 5 – Implementazione della modalità "Doppler angle tracking 91

5.1 Stima dell'angolo Doppler 91
5.2 Il metodo "Doppler angle tracking" 92
5.3 Implementazione ottimizzata del metodo su ULA-OP 256 95
5.4 Funzionamento dell'algoritmo 96
5.5 Implementazione dell'algoritmo su DSP 98
5.5.1 Sezione di elaborazione 98
5.5.2 Sezione decisionale 100
5.6 Applicazione alla stima della velocità di flusso 102

Conclusioni 103

Bibliografia 105

Pubblicazioni 109

Introduzione

Le tecniche ecografiche sono ideali per impieghi diagnostici in ambito medico in quanto sfruttano una forma di energia, gli ultrasuoni, non dannosa sui tessuti biologici dei pazienti, sono implementabili in strumenti di costi e dimensioni relativamente contenuti e possono essere utilizzate per effettuare analisi in tempo reale direttamente al letto del paziente o in sala operatoria.

La ricerca in questo campo è particolarmente attiva e cerca di individuare nuovi metodi di indagine, o di perfezionare quelli già esistenti, in modo da fornire strumenti di analisi e diagnosi sempre più accurati.

Negli ultimi anni, la ricerca si è concentrata anche sul lato hardware delle piattaforme ecografiche "aperte", allo scopo di realizzare sistemi in grado di implementare le tecniche di indagine più arbitrarie e complesse, non implementabili sugli apparecchi commerciali.

Il Laboratorio di Progettazione Sistemi Microelettronici (MSDLab) è da anni attivo in entrambi i campi e offre il suo contributo alla ricerca lavorando, in particolare, alla realizzazione di nuove piattaforme ecografiche in grado di permettere lo sviluppo e la validazione di nuove tecniche di indagine ultrasonica.

Nel corso degli anni sono state sviluppate alcune piattaforme, associate sia a sonde mono-elemento che ad array di traduttori. La più recente di queste piattaforme, ULA-OP (Ultrasound Advanced Open Platform), è stata un concepita come sistema estremamente flessibile e aperto, interamente dedicato alla ricerca nel campo degli ultrasuoni.

La prima versione del sistema permette il controllo indipendente di 64 elementi, (sistema a "64 canali") e contiene a bordo 1GB di memoria RAM, 1 DSP e 4 FPGA. Per estendere il possibile campo di applicazione di questo strumento, è stata recentemente progettato il sistema ULA-OP 256, in grado di pilotare sonde ad array fino a 256 elementi e dotato di risorse di calcolo molto più potenti in termini di DSP e FPGA.

Nel sistema possono essere alloggiate fino ad otto schede front-end, connesse tramite un backplane sfruttando il bus ad alta velocità Serial Rapid IO. Tali schede sono dedicate alla gestione delle fasi di trasmissione, ricezione ed elaborazione, mentre una scheda di interfaccia ha lo scopo di gestire la comunicazione tra le schede front-end ed il pc nelle due direzioni, gestendo il transito di comandi e dati elaborati.

In questa piattaforma, e le strategie di trasmissione e ricezione sono completamente configurabili e i dati acquisiti possono essere memorizzati a vari stadi dell'elaborazione, per poter essere esportati su pc e consentire la sperimentazione di nuovi algoritmi.

La mia attività di dottorato è stata mirata a contribuire alla realizzazione e al test del sistema ULA-OP 256, attraverso lo sviluppo di software e firmware e l'applicazione in due tecniche di indagine innovative.

In particolare, i miei contributi hanno riguardato: lo sviluppo di un software utilizzato per la configurazione del sistema, lo sviluppo del firmware per i DSP presenti nelle schede front-end e interfaccia, la sperimentazione di modalità di indagini basate su una tecnica di beamforming parallelo in trasmissione e l'implementazione dell'algoritmo di "Doppler angle tracking" per la soluzione del problema dell'angolo Doppler nella flussimetria ad ultrasuoni..

In questo elaborato il resoconto della mia attività è suddiviso in cinque capitoli.

Il primo capitolo introduce la piattaforma ULA-OP 256, descrivendone l'architettura generale, i dettagli hardware più importanti e le innovazioni introdotte rispetto al precedente sistema sviluppato da MSDLab.

Il secondo capitolo descrive il software sviluppato per la generazione dei file di configurazione, necessari per impostare il sistema in base al tipo di indagine che si vuole svolgere e ai dati che si vogliono ottenere. Viene quindi illustrata la struttura generale e l'interfaccia utente di tale applicativo e i vantaggi che esso offre nella programmazione delle modalità di lavoro del sistema.

Il terzo capitolo descrive i firmware DSP sviluppati per il sistema differenziandoli in base allo scopo e alle loro caratteristiche, descrivendo le scelte progettuali effettuate, il loro funzionamento e le potenzialità che offrono.

Il quarto capitolo descrive l'applicazione della piattaforma ULA-OP per imaging ad alto frame rate. In particolare, viene descritta la realizzazione ed il test sperimentale della tecnica "OFDM" che consente di triplicare il frame rate attraverso un beamforming parallelo in trasmissione.

Il quinto capitolo descrive l'implementazione nel nuovo sistema dell'algoritmo di Doppler angle tracking, mirato alla stima accurata del cosiddetto angolo Doppler. In particolare, viene discussa realizzazione di tale algoritmo nel nuovo sistema a 256 canali, e come questa consenta di superare alcune limitazioni, in termini di PRF e di rapporto segnale/rumore, riscontrate nel sistema precedente.

Capitolo 1

Il Sistema ULA-OP 256

1.1 ULA-OP: una piattaforma aperta

I sistemi commerciali realizzati per l'acquisizione e l'elaborazione di segnali ad ultrasuoni consentono di effettuare numerose tipologie di indagine: dal B-Mode (brightness mode), in cui un fascio di ultrasuoni trasmesso da un trasduttore lineare è in grado di effettuare la scansione lungo un piano mostrando a video un immagine bidimensionale, al Color Doppler, utilizzato per visualizzare la velocità del flusso sanguigno mostrata a video secondo un codice basato su colore e ampiezza del segnale.

Sebbene questi strumenti, usati soprattutto in ambito diagnostico, siano in grado di effettuare numerose tipologie di analisi, la ricerca nel campo degli ultrasuoni richiede piattaforme maggiormente aperte e flessibili in grado non solo di implementare strategie di trasmissione e ricezione non convenzionali basate su particolari forme d'onda in trasmissione o focalizzazioni dei fasci, ma anche di disporre di un hardware in grado di memorizzare e processare dati in varie sezioni della catena di elaborazione, al fine di sperimentare nuovi metodi di indagine.

Allo scopo di realizzare una piattaforma aperta da utilizzare nel campo della ricerca delle indagini ad ultrasuoni, il laboratorio MSDLab dell'Università degli studi di Firenze ha realizzato la piattaforma ULA-OP (Ultrasound Advanced Open Platform).

Tale sistema è stato realizzato con lo scopo di essere appunto un sistema aperto, ossia di consentire la massima flessibilità e programmabilità al fine di sperimentare nuove metodologie di indagine, basate su nuove tecniche di trasmissione e beamforming.

ULA-OP infatti è un sistema programmabile da pc, che può essere configurato per il tipo di acquisizione desiderata impostando le strategie di trasmissione e ricezione e mostrando in tempo reale a video il risultato delle elaborazioni selezionate; consente inoltre di accedere direttamente ai campioni salvati in memoria, permettendo di salvare su pc i dati presenti a vari stadi dell'elaborazione, per sottoporli poi a fasi di post processing tramite software.

Il primo sistema ULA-OP è stato realizzato per essere un sistema multicanale in grado di offrire la massima potenza di calcolo e flessibilità possibile, mantenendo al contempo contenuti gli ingombri e i costi; definito ULA-OP 64 per la sua capacità di trasmettere e ricevere contemporaneamente su 64 canali, utilizzando forme d'onda arbitrarie e indipendenti, era in grado di gestire sonde ad array da 192 elementi.

La configurazione hardware era stata realizzata per essere modulare, in quanto basata su un backplane in grado di accogliere più schede, selezionabili in base all'esigenza e al tipo di analisi da effettuare; in particolare nella configurazione tipica di tale sistema erano utilizzate una scheda di alimentazione, una scheda dedicata alla parte digitale, basata su un microcontrollore DSP e 5 FPGA, ed una dedicata alla parte analogica.

Figura 1.1 - Il sistema ULA-OP 64

Il sistema ULA-OP 64 è correntemente impiegato in svariate applicazioni, dall'imaging B-mode al Doppler vettoriale, in oltre trenta laboratori nel mondo. Esso è anche utilizzato, sia pure con alcune limitazioni, nell'ambito di tecniche emerse recentemente, come ad esempio il cosiddetto "plane wave imaging".

Al fine di rimuovere tali limitazioni, in MSDLab è stato quindi progettato e realizzato un nuovo sistema da affiancare ad ULA-OP in applicazioni molto impegnative, quali le acquisizioni ad alto frame rate o le ecografie tridimensionali, dotato di maggiore potenza di calcolo ed in grado di gestire un maggior numero di canali indipendenti (256).

1.2 Miglioramenti introdotti nel sistema ULA-OP 256

Il sistema ULA-OP 256 è stato realizzato per ottenere il massimo in termini di flessibilità e prestazioni.

E' un sistema modulare costituito da un backplane su cui possono essere alloggiate fino a 8 schede Front-End, ciascuna dedicata alla gestione di 32 canali di trasmissione/ricezione, una scheda Interfaccia per la comunicazione con il pc ed una scheda di alimentazione; in base al tipo di acquisizione da effettuare e al tipo di sonda possono essere inserite tutte le schede Front-End o può esserne utilizzato un numero minore.

Anche il connettore sonda posto sul retro è rimuovibile dal backplane e possono esserne utilizzati diversi in base alla sonda da connettere al sistema.

Figura 1.2 - Il prototipo del sistema ULA-OP 256

Numerose sono le migliorie rispetto al sistema precedente.

La potenza di calcolo è stata incrementata, a fronte dei 16 processori DSP disponibili nella configurazione a 8 schede, e la memoria per il salvataggio dei dati acquisiti o elaborati è stata aumentata fino alla quantità di 80 GB.

L'elevata memoria disponibile consente il salvataggio di una significativa quantità di dati grezzi a radiofrequenza, dati beamformati e dati demodulati in banda base; questo permette, in base al Pulse Repetition Interval (PRI) impostato, il salvataggio di fino a 30 secondi di dati grezzi.

La potenza di calcolo complessiva del sistema è di 5000 GMAC o 2500 GFLOP.

Poiché questa potenza è distribuita su più schede e su più core operanti all'interno di queste, per questo sistema è stato studiato un firmware estremamente flessibile strutturato in modo da suddividere il carico di lavoro tra le risorse disponibili; ogni core è in grado di ricevere dati da tutti gli altri core presenti nel sistema, ed all'interno di ognuno di essi il firmware è organizzato in modo modulare suddividendo le varie parti di gestione ed elaborazione in task dedicati.

Il sistema è in grado di effettuare il beamforming in real-time e questo viene eseguito in due step: il primo all'interno delle FPGA presenti sulle schede front-end dove gli echi ricevuti dai 32 canali gestiti dalla scheda sono opportunamente ritardati e sommati, il secondo nel DSP della scheda interfaccia dove vengono sommati i risultati dei beamforming parziali.

Una differenza importante con il precedente sistema è il fatto che, mentre in precedenza la presenza di soli 64 canali imponeva un multiplexaggio per la connessione con le sonde a 192 elementi, nel nuovo sistema sono disponibili 256 canali e questi sono mappati ciascuno con un elemento della sonda, che può essere anche di 256 elementi.

Questo, unito all'hardware più potente di cui dispone la piattaforma, consente di implementare strategie di ricezione quali l'acquisizione multilinea o l'implementazione del beamforming parallelo in ricezione; è inoltre possibile impostare aperture arbitrarie per la trasmissione o la ricezione, mentre in ULA-OP 64 queste erano limitate a causa degli effetti del multiplexer dei canali.

Ulteriori caratteristiche del nuovo sistema sono la possibilità di effettuare analisi ad alto framerate, grazie alle acquisizioni multilinea, e la possibilità di utilizzare per le elaborazioni real-time anche i dati a radiofrequenza pre-demodulazione e post-beamforming, grazie alla presenza di bus di comunicazione ad alta velocità.

1.3 Architettura del sistema ULA-OP 256

Per realizzare il sistema è stato scelto un approccio modulare.

L'architettura è basata sull'utilizzo di multiple schede Front-End, ciascuna in grado di gestire 32 canali indipendenti; la configurazione completa con 8 schede permette quindi di gestire 256 canali, ma è possibile utilizzarne un numero inferiore in base al tipo di sonda, ad esempio sono necessarie solamente 6 schede per la gestione di una sonda a 192 canali.

E' presente poi una scheda interfaccia che si occupa della comunicazione tra il sistema ed il PC, utilizzando una connessione USB 3.0 SuperSpeed.

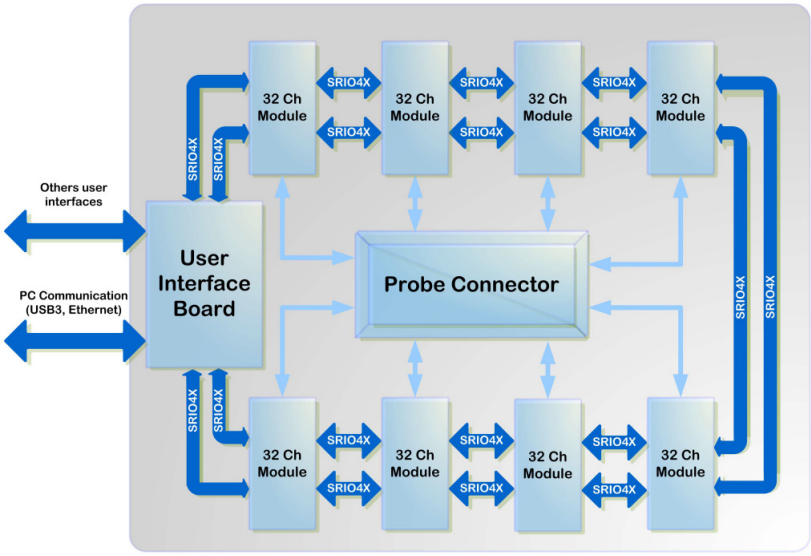

Figura 1.3 - Architettura generale del sistema ULA-OP 256

La comunicazione tra le schede del sistema è realizzata utilizzando il bus ad alta velocità Serial Rapid IO (SRIO) configurato in modo da operare in modalità a 4 lane, operanti a 5 Gbit/s ciascuna.

Ogni scheda dispone di 4 interfacce SRIO per la comunicazione con le altre, connesse tramite il backplane in modo da formare un anello chiuso; questo comporta per ogni scheda una banda totale di 80 Gbit/s full duplex.

1.3.1 Scheda di interfaccia

La scheda di interfaccia gestisce la comunicazione tra il sistema ed il pc attraverso una connessione USB; questa scheda è quindi deputata sia alla ricezione di dati e comandi dal software Real Time in esecuzione nell PC e al loro inoltro verso la scheda Front-End di destinazione, sia alla ricezione dei dati parzialmente elaborati dalle altre schede, alla loro combinazione e successiva trasmissione verso il PC.

A livello di hardware utilizza un DSP multicore della famiglia 320C6678 prodotto dalla Texas Instruments, dotato di 8 core operanti ciascuno alla frequenza di 1.2 GHz, e a cui è associato un banco di memoria DDR3 fino a 8 GB.

E' poi presente una FPGA della famiglia Cyclone V prodotta dalla Altera, che si occupa della comunicazione tra l'interfaccia USB e il resto del sistema.

Uno switch CPS-1432 della IDT si occupa del routing dei pacchetti SRIO tra DSP, FPGA e backplane.

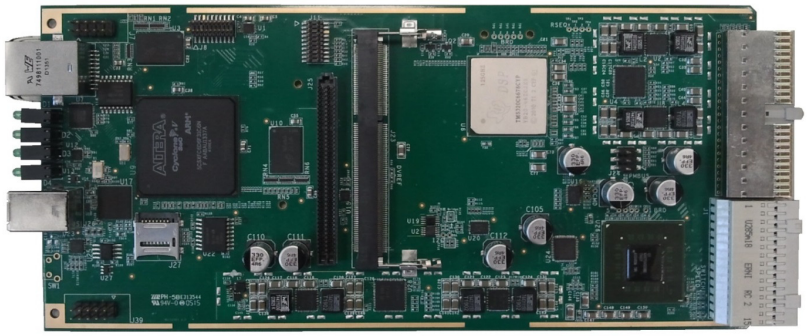

Figura 1.4 - Scheda di interfaccia

1.3.2 Schede front-end a 32 canali

Ognuna delle schede Front-End presenti nel sistema ospita a bordo l'elettronica necessaria per la gestione della trasmissione, ricezione ed elaborazione dei segnali ad ultrasuoni su 32 canali indipendenti.

Su ognuna di queste schede è presente una FPGA ARRIA V GX prodotta dalla Altera, che si occupa della generazione di 32 segnali analogici in trasmissione, con banda fino a 20 MHz, utilizzando un approccio sigma-delta che produce bitstream fino a 500 Mb/s.

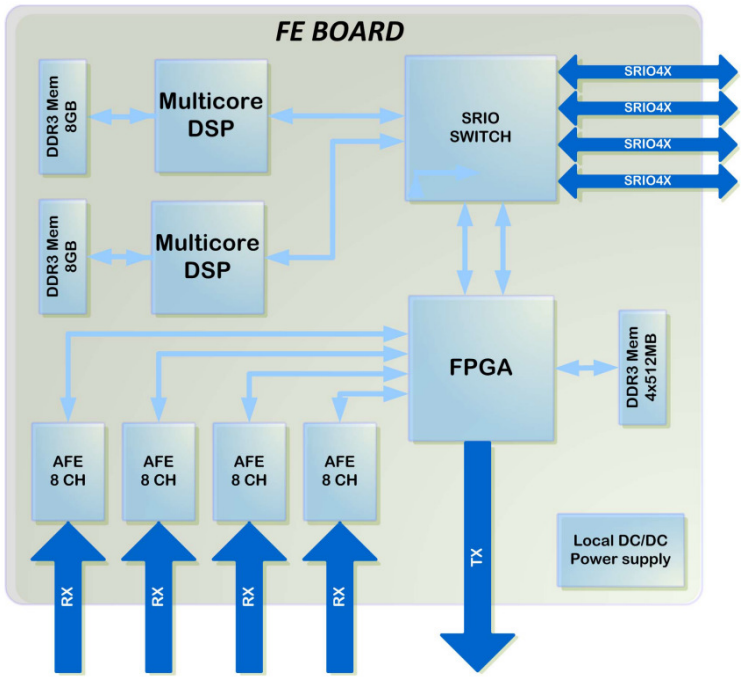

Figura 1.5 - Diagramma a blocchi delle schede front-end

L'eco ultrasonico ricevuto dai 32 elementi corrispondenti della sonda viene amplificato e digitalizzato a 80 MSPS con risoluzione a 12 bit da quattro front-end a ultrasuoni ad 8 canali AFE5807 della Texas Instruments.

Dopo l'amplificazione l' FPGA si occupa delle operazioni di somma e di applicare i ritardi per effettuare il beamforming dei segnali ricevuti.

Nella scheda sono presenti anche due DSP 320C6678 della Texas Instrument, ciascuno dei quali dotato di 8 GB di memoria DDR3; i due DSP hanno il compito di effettuare le operazioni di demodulazione, filtraggio ed elaborazione real-time.

Sopra la scheda di trasmissione un connettore permette la connessione di diverse schede di trasmissione, intercambiabili in base all'esigenza, permettendo di scegliere ad esempio una scheda in alta potenza rispetto a una che privilegia la linearità.

Uno switch CPS-1432 della IDT si occupa del routing dei pacchetti SRIO tra i due DSP, l'FPGA ed il backplane.

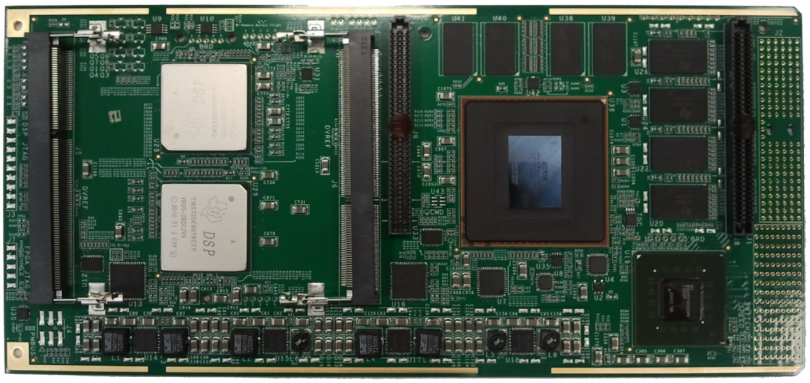

Figura 1.6 - Scheda front-end

1.3.3 Scheda di trasmissione in alta potenza

La scheda di trasmissione in alta potenza serve come interfaccia tra la scheda front-end e la sonda ad ultrasuoni ed ha il duplice scopo di generare il segnale in alta tensione per la trasmissione, disaccoppiando poi da questo il debole segnale ottenuto dagli echi ricevuti dalla sonda.

In trasmissione, 4 pulsatori ultrasonici HDL6V5583 prodotti dalla HITACHI, ciascuno in gradi di gestire 8 dei 32 canali, si occupano di generare segnali con tensioni fino a 180 Vpp.

In ricezione 4 switch per ultrasuoni TX810 della Texas Instruments disaccoppiano il segnale ricevuto dalla sonda da quello generato per la trasmissione; questo è necessario in quanto le linee di trasmissione e ricezione da e verso la sonda sono comuni.

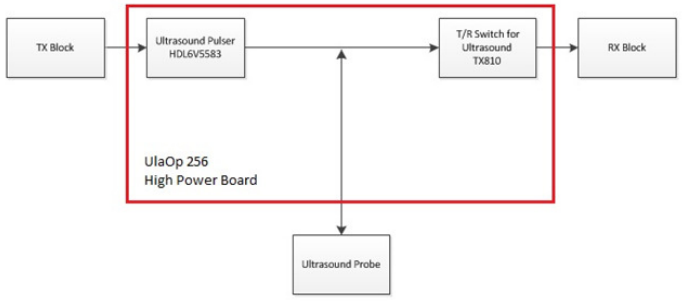

Figura 1.7 - Diagramma semplificato della scheda di trasmissione in alta potenza

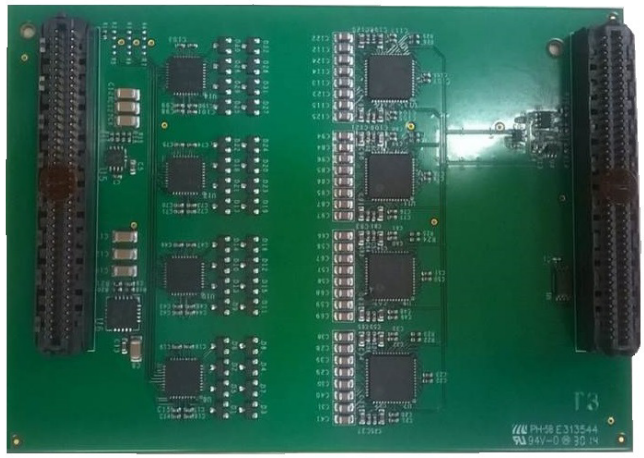

Figura 1.8 - Scheda di trasmissione in alta potenza

1.3.4 Connettore sonda

Al fine di poter utilizzare sonde diverse, che utilizzano connettori diversi, sono disponibili più schede che operano come interfaccia tra il backplane e la sonda, ciascuna con un connettore diverso in base alla sonda per cui è progettato.

Questa soluzione oltre ad aumentare la flessibilità del sistema, permette di mantenere alto il rapporto segnale/rumore, rispetto ad altre soluzioni quali l'impiego di un adattatore.

Capitolo 2

Sviluppo del software “Configuration Editor”

2.1 Configurazione del sistema ULA-OP

ULA-OP è una piattaforma realizzata per la ricerca e per questo motivo in fase di progetto è stato scelto un approccio che privilegiasse la flessibilità e la programmabilità del sistema, dando all’operatore la facoltà di scegliere come configurarlo in base al tipo di acquisizione ed elaborazione desiderate.

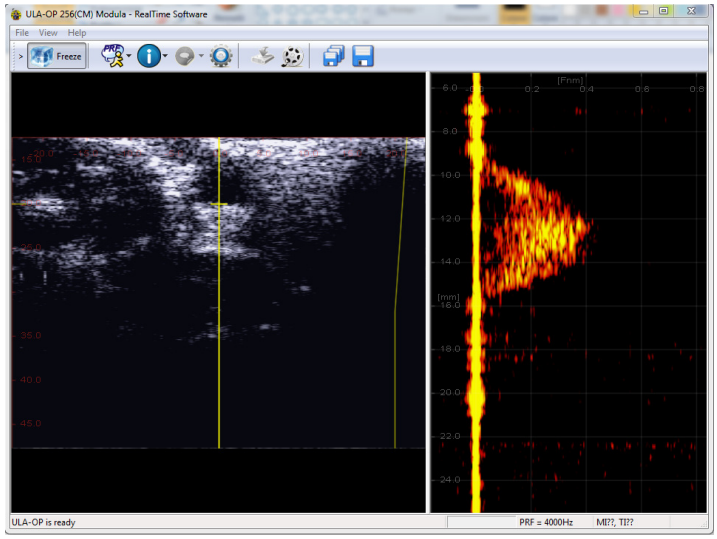

Figura 2.1 - Schermata visualizzata dal software real-time con il sistema configurato per acquisire un’ immagine B-Mode e un profilo Doppler

Sono diversi i parametri che possono essere impostati e riguardano sia il tipo di indagine che l'operatore vuole svolgere, ad esempio una scansione B-Mode o una Doppler, sia le caratteristiche di trasmissione, ricezione e acquisizione con cui impostare il sistema, quali la profondità di focalizzazione dei fasci e la quantità di memoria da riservare alle slice, ossia i buffer che contengono i dati relativi agli echi ricevuti, in corrispondenza dei vari stadi della loro elaborazione.

Per ottenere il massimo della flessibilità questi parametri non sono fissati all'interno del firmware o del software, ma sono inseriti in file esterni, detti "file di configurazione", che sono letti dal software real-time utilizzato per interfacciare il sistema al PC; all'avvio dell'applicazione questi file vengono letti e interpretati e usati per configurare sia l'interfaccia di visualizzazione su PC sia la parte di acquisizione e elaborazione nel sistema.

I file di configurazione sono file testuali, in modo da poter essere letti e modificati con un qualsiasi editor di testo presente nel PC, e quindi essere aperti anche senza la necessità di utilizzare o sviluppare programmi appositi.

Il sistema usa due tipologie di file diversi: i file di configurazione del sistema (.cfg) che contengono i parametri legati alla fase di acquisizione ed elaborazione, quali la sequenza di trasmissione, la memoria da allocare e i moduli di elaborazione da istanziare, ed i file di configurazione degli item (.ula) che contengono le impostazioni riguardo alle strategie di ricezione e trasmissione degli impulsi ultrasonici.

L'utilizzo dei file di configurazione consente di passare rapidamente da un setup all'altro in modo da effettuare acquisizioni diverse, semplicemente indicando al software di caricare un altro file tra quelli esistenti; ulteriore vantaggio è che per preparare una nuova configurazione è necessario occuparsi solamente della scrittura di uno di questi file, senza necessità di cambiamenti nel software o nel firmware.

2.2 Sintassi dei file di configurazione

I file di configurazione sono organizzati al loro interno con una struttura fatta da sezioni, chiavi e valori.

Le sezioni, identificate in stampatello tra parentesi quadre, servono per offrire un raggruppamento logico ai gruppi di chiavi organizzandole appunto in sezioni logiche in base alla funzionalità del sistema a cui si riferiscono e servono fondamentalmente per migliorare la leggibilità del file e la struttura del codice del software real-time che deve interpretarlo.

Le chiavi e i relativi valori rappresentano i parametri impostabili.

Ogni sezione può avere una o più chiavi e queste vengono dichiarate ad inizio riga e seguite dal simbolo di uguale "="; successivamente sono indicati separati da virgole tutti i valori, numerici e non, da associare alla chiave, inseriti secondo un ordine prestabilito dal software che interpreta il file.

Al termine di ogni riga è inoltre possibile aggiungere dei commenti utilizzando la sintassi "//" in modo da fornire una rapida descrizione dei parametri associati a ciascuna chiave.

Il manuale tecnico del sistema ULA-OP riporta la grammatica necessaria per scrivere i file di configurazione, specificando i nomi di sezioni e chiavi e cosa rappresentano i valori da associarvi, e può essere utilizzato per preparare la configurazione desiderata.

Si riporta a titolo di esempio un possibile file di configurazione del sistema, utilizzato per acquisire un immagine B-Mode ed un profilo Doppler.

```
[NAME]
Name = B-Mode + Doppler

[COMMENT]
CommentMode = Two window with B-mode image and Doppler profile.

[TREE]
aspect = 2H000
ratio = 60,40

[SEQUENCER]
item0 = immagine.ula
item1 = doppler.ula
DefItemKey0 = 0,Focus,a,q,2
DefItemKey1 = 0,FBurst,w,s,1
DefItemKey2 = 0,TgcA,PgUp,PgDown,1
DefItemKey3 = 0,TgcB,Home,End,5

[WORKINGSET]
SoundSpeed = 1540

[SSG]
PrfsMap=1000,2000,3000,4000,5000,6000,7000,8000,9000,
Prf = 2000,Off,z,x,c
```

[SAVEOPT]
Auto = F1, -1
Movie = m

[ACQUIREIQ]
slice0 = 512, 81920, 0, BModeSliceIQ, 2, 0
slice1 = 512, 8192, 0, DopplerSliceIQ, 2, 0
slice2 = 256, 8192, 0, s1, 2, 0
slice3 = 256, 8192, 0, s2, 2, 0

[ACQUIRERF]
RfType = Post
BfLoss = 3

[ACQUIRERF_POST]
Slice0 = 4856, 1024, 0, BModeSliceRFPost, 2, 0
Slice1 = 4856, 1024, 0, DopplerSliceRFPost, 2, 0

[BLOCKSEQUENCER]
pri0 = 0, 1, 0, 0, 0
pri1 = 1, 0, 1, 0, 1
[MODULES]
module0 = IMAGE1
module1 = DOPPLER1

[IMAGE1]
ModuleName = CModBMode
Threshold = 4,Up,Down,1
Dynamic = 10,Right,Left,1
Window = 0
Slice = Slice0
ViewItem = Item0
SpecialFilters = Off, Off
SizeX = 192
SizeY = 512
VideoFilter = 0
CursorItem0 = 1

[DOPPLER1]
ModuleName = CModProfile
SizeY = 256
FFTSize = 256
FFTWindow = Hanning
Threshold = 11
Dynamic = 4
VideoFilter = 4
Window = 1
Slice = 2
RatioDivFactor = 30, 2
ViewItem = 1

Analizzando questo file si identifica ad esempio la sezione *[ACQUIREIQ]* la quale raggruppa le chiavi utilizzate per configurare le slice di memoria in cui salvare i dati demodulati; tramite la seguente sintassi si impostano due slice, una in cui saranno salvati i dati relativi all'immagine B-Mode e l'altra in cui saranno inseriti i dati della scansione Doppler.

[ACQUIREIQ]
slice0 = 512, 8192, 0, BModeSliceIQ
slice1 = 256, 8192, 0, DopplerSliceIQ

Si identificano immediatamente il nome della sezione, "[ACQUIREIQ]" e le due chiavi relative alle due slice da allocare, "slice0" e "slice1.

I primi due valori numerici servono per definire le dimensioni delle slice che dovranno contenere rispettivamente 512 e 256 gates, relativi alle profondità, e 8192 PRI ossia intervalli di trasmissione/ ricezione ultrasonica; il terzo valore è infine relativo alla profondità iniziale da cui iniziare a salvare i dati.

E' inoltre presente un quarto campo, opzionale nella configurazione, in cui viene specificato il nome da associare alla slice che può essere utilizzato come mnemonico per riconoscere rapidamente le slice durante le opzioni di esportazione del loro contenuto nel PC.

E' intuibile capire come, soprattutto per utenti non esperti, possa essere difficile ricordarsi i nomi di sezioni e chiavi e soprattutto quali numeri associare a queste e in che ordine, rendendo quindi necessario un ricorso continuo al manuale tecnico del sistema ULA-OP per la scrittura di un nuovo file di configurazione o per l'interpretazione di un file già esistente.

Per quanto sia possibile aggiungere commenti al termine di ciascuna riga per descrivere brevemente le associazioni parametro-valore, al fine di semplificare

la fase di impostazione del sistema da parte degli operatori, si è scelto di sviluppare un software che offra un'interfaccia maggiormente intuitiva ed in grado di generare automaticamente il file con la configurazione desiderata.

2.3 Il software "Configuration Editor"

Il software Configuration Editor, realizzato tramite Visual C++, consente l'interpretazione, la manipolazione e la creazione dei file di configurazione necessari al sistema.

Tale software è stato pensato per aiutare l'operatore del sistema ULA-OP durante la fase di impostazione, fornendogli un'interfaccia completa e intuitiva per la generazione della configurazione voluta.

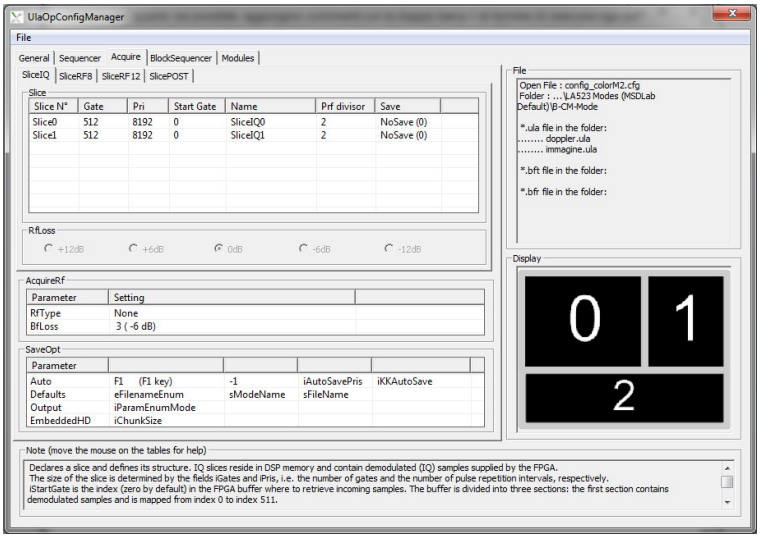

Figura 2.2 - Interfaccia del software "Configuration editor"

L'interfaccia è organizzata in modo da suddividere la programmazione in diverse sezioni logiche, ad esempio la configurazione dei buffer di memoria e la gestione dei moduli di elaborazione, e presenta diverse tabelle in cui le intestazioni di righe e colonne identificano le chiavi e i diversi parametri che le compongono.

E' evidente come tale sistema renda più facile associare il corretto valore ai parametri di sistema, rispetto alla scrittura di chiavi e liste di numeri in un file testuale; nel caso in cui un parametro possa assumere solo determinati valori è

inoltre possibile permettere la scelta di questi tramite liste a comparsa, limitando la scelta di valori possibili ed evitando così eventuali errori di configurazione.

Questo software è in grado quindi di permettere l'inserimento da parte dell'utente di tutti i valori e parametri necessari per effettuare l'acquisizione desiderata, generando al termine un file testuale contenente la corretta sintassi fatta di sezioni chiavi e valori, corrispondente alle impostazioni selezionate. Il Configuration Editor è inoltre in grado di aprire file di configurazione creati in precedenza, riempiendo le varie tabelle con i corretti valori e consentendo la lettura della configurazione precedentemente creata e la sua modifica.

Tra gli altri benefici offerti da questo software vi è inoltre un ulteriore ausilio all'utente, costituito da una funzione di aiuto in tempo reale integrata nel programma, in grado di fornire in un'apposita casella di testo una breve descrizione della chiave selezionata e dei parametri che la compongono.

E' bene specificare che l'interfaccia principale di questo software è dedicata solamente alla realizzazione della configurazione del sistema e la generazione del relativo file .cfg.

La parte di software dedicata alla manipolazione e creazione dei file relativi agli item (.ula) può essere richiamata opzionalmente ed ha un proprio form Visual Studio dedicato che sarà descritto nel sottoparagrafo 2.3.1.6, mentre la descrizione dell'interfaccia che segue sarà riferita esclusivamente alla parte di manipolazione del file di configurazione principale.

Queste parti di programma sono separate in quanto devono operare su file di tipo diverso ma si basano sulle stesse funzioni per quanto riguarda l'interpretazione dei dati inseriti nelle tabelle e il salvataggio e l'apertura dei relativi file di testo.

2.3.1 Interfaccia utente

L'interfaccia utente è stata organizzata in modo da essere suddivisa in 5 schede così da raggruppare i parametri della configurazione in sezioni logiche:

- General: Impostazioni generali.
- Sequencer: gestione degli item.
- Acquire: creazione delle slice di memoria.
- Blocksequencer: ordinamento delle sequenze di trasmissione e ricezione
- Modules: configurazione dei moduli di elaborazione

Al centro dell'interfaccia è presente il frame con il selettore delle cinque schede e tra queste viene mostrata a video solo quella selezionata.

In alto a destra è presente una casella testuale che mostra informazioni utili nel caso in cui sia stato aperto un file di configurazione già esistente, relative alla presenza nella cartella di progetto di altri file riconosciuti dal software, che siano file di configurazione degli item (.ula) o pattern di beamforming in ricezione e trasmissione (.bft e .bfr).

Al di sotto di questa casella è presente un'anteprima del "Display" ossia viene mostrato come nel software real-time saranno suddivise le finestre in cui saranno mostrati i risultati delle varie elaborazioni; le finestre sono numerate in modo da facilitarne l'associazione con i moduli di elaborazione.

In basso si trova infine la casella testuale con l'aiuto sensibile al contesto che mostra, in base alla chiave, ossia alla riga selezionata nelle diverse tabelle, una breve descrizione dei parametri che la compongono.

2.3.1.1 General

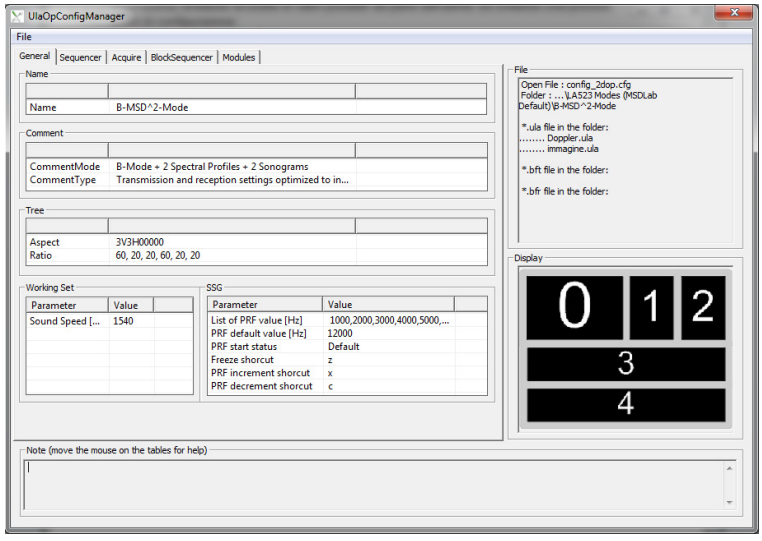

Figura 2.3 - Scheda di configurazione "General"

La prima scheda che viene visualizzata è la scheda "General" che come suggerisce il nome contiene informazioni generali sul progetto quali il nome e dei commenti che identifichino i tipi di analisi che questo permette.

Sono inoltre impostabili altri parametri generici quali la velocità del suono nel mezzo in esame e la Pulse Repetition Frequency (PRF) che stabilisce l'intervallo con cui devono essere trasmessi gli impulsi ultrasonici.

Da questa schermata si può infine definire l'impostazione del display, selezionando una delle possibili configurazioni predefinite o impostandolo manualmente, in modo da definire quante finestre dovranno essere mostrate nel software Real Time e come dovranno essere proporzionate tra di loro.

Per capire il funzionamento di questa selezione è necessario accennare brevemente al meccanismo con cui il software real-time prepara la configurazione del display.

Le impostazioni "Aspect" e "Ratio" della tabella "Tree" stabiliscono l'aspetto che dovrà avere la schermata real-time in cui visualizzare i risultati dell'acqusizione, definendo rispettivamente quante finestre creare e come suddividerle, e le proporzioni che queste dovranno avere.

Per quanto sia possibile riempire manualmente questi campi è possibile selezionare una delle configurazioni predefinite scegliendo l'immagine relativa ad essa da una cartella; dopo la selezione vengono automaticamente riempite le due celle con i valori necessari per impostare il display per ottenere quella visualizzazione, valori che sono codificati nel nome dell'immagine.

Si consideri come esempio la seguente immagine il cui nome è il codice 3V3H00000_60p20p20p60p20p20:

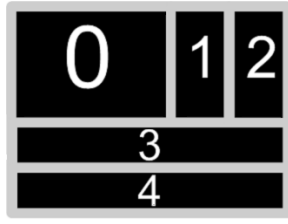

Figura 2.4 - Configurazione del display associata al codice 3V3H00000_60p20p20p60p20p20p

La prima parte è relativa al modo in cui deve essere suddivisa la schermata e la seconda alle proporzioni che le sottofinestre create devono avere.

L'aspetto viene impostato utilizzando un grafo ad albero in cui i rami rappresentano le suddivisioni in sottofinestre e le foglie le finestre create dove visualizzare i risultati; questo albero viene creato in base al codice inserito in "Aspect" che definisce le suddivisioni da effettuare ricorsivamente in orizzontale o verticale e le terminazioni, a partire dalla radice che rappresenta una schermata di visualizzazione a schermo intero.

Considerando la configurazione 3V3H00000 e leggendola da sinistra verso destra 3H indica che la schermata deve essere suddivisa in verticale in tre sottofinestre, mentre il 3V successivo indica che la prima finestra ottenuta dalla divisione deve a sua volta essere suddivisa in orizzontale in tre sottofinestre; gli zeri rappresentano le terminazioni e indicano che non è necessario effettuare ulteriori suddivisioni.

In "Ratio" le proporzioni sono associate alle sottofinestre nell'ordine con cui queste sono state suddivise; il primo 60p20p20p rappresenta quindi la proporzione in percentuale nella suddivisione verticale effettuata all'inizio, mentre il successivo 60p20p20p indica le proporzioni della suddivisione orizzontale della prima sottofinestra ottenuta.

2.3.1.2 Sequencer

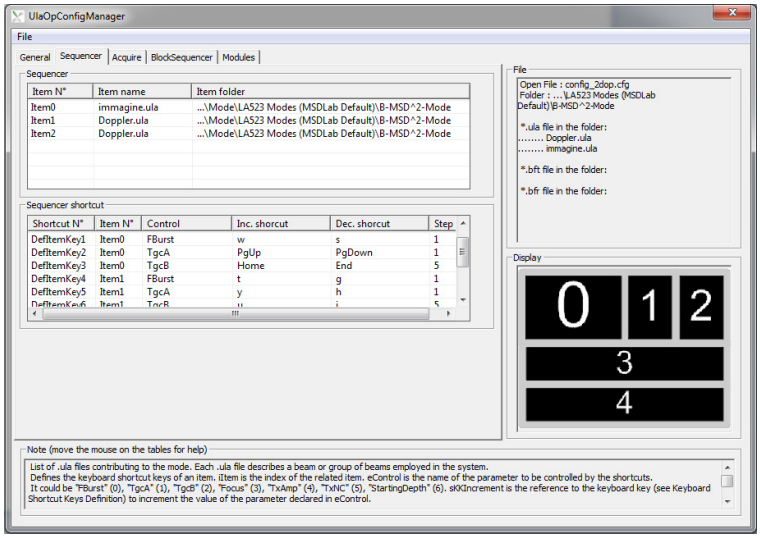

Figura 2.5 - Scheda di configurazione "Sequencer"

La seconda scheda è definita "Sequencer" e permette di selezionare gli Item da utilizzare nella configurazione.

Gli item definiscono le strategie di generazione e ricezione degli impulsi ultrasonici, permettendo di impostare parametri quali la focalizzazione, il numero di linee da trasmettere-ricevere e i parametri per la demodulazione.

La tabella degli item consente di crearne di nuovi da aggiungere al progetto, aprendo automaticamente l'interfaccia dedicata alla loro configurazione, o di importare file .ula già esistenti selezionandoli da altre cartelle di progetto ed in

questo caso il software provvede automaticamente a creare una copia locale nella cartella del progetto attivo; gli item inseriti nella tabella possono in ogni momento essere modificati aprendo l'Item editor.

Tramite il tasto destro del mouse premuto sulla tabella si apre un menù contestuale che permette l'aggiunta di nuove righe con cui importare/creare nuovi item, o la rimozione di quelli già esistenti.

Agli item presenti possono inoltre essere associate delle scorciatoie da tastiera, consentendo così al software real-time di modificare in tempo reale durante l'acquisizione alcuni loro parametri, ad esempio la profondità iniziale da cui iniziare ad salvare dati, senza la necessità di spengere il sistema ed editare il file di configurazione.

2.3.1.3 Acquire

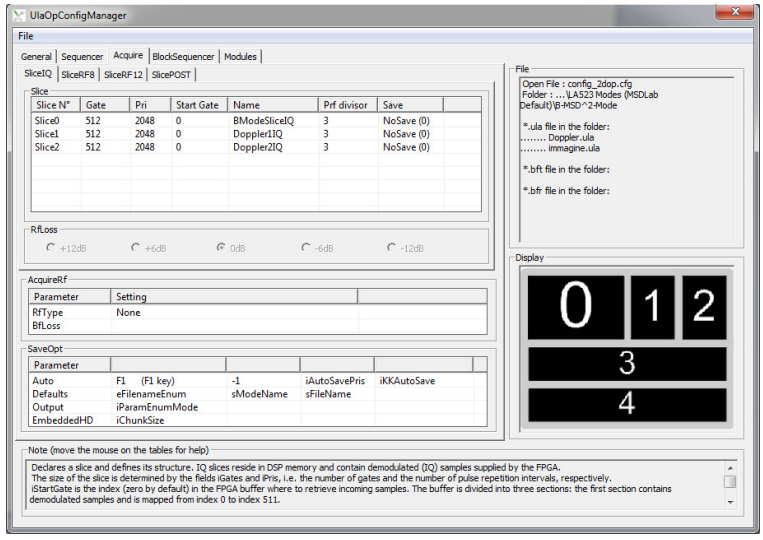

Figura 2.6 - Scheda di configurazione "Acquire"

La terza scheda, definita "Acquire" permette la configurazione delle slice di memoria in cui salvare i dati presenti a vari stadi dell'elaborazione impostandone la dimensione nello spazio, ossia quante profondità da acquisire, e nel tempo, ovvero il numero di PRI (Pulse Repetition Interval) da memorizzare, dove ogni PRI rappresenta un diverso intervallo di ricezione e elaborazione.

Sono presenti 4 sottoschede per separare la configurazione tra le slice in banda base, quelle a radiofrequenza e quelle in uscita dal beamformer e per queste ultime possono essere impostate le perdite.

Il menù contestuale associato al tasto destro del mouse consente anche in questo caso l'aggiunta di righe, corrispondenti a nuove slice da allocare, o la rimozione di quelle già esistenti.

E' possibile inoltre da questa schermata impostare la funzione di Autosave che consente il salvataggio automatico del contenuto della slice selezionata su PC.

2.3.1.4 Blocksequencer

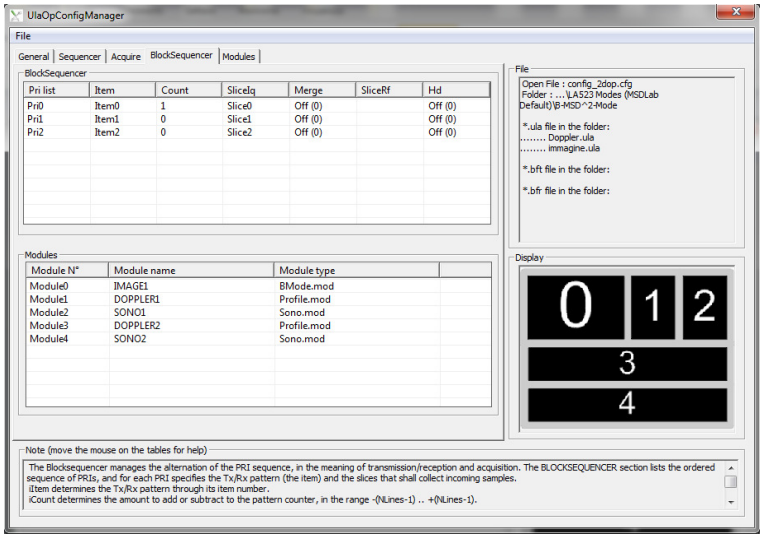

Figura 2.7 - Scheda di configurazione "Blocksequencer"

La scheda "Blocksequencer" permette di impostare l'ordine con cui gli Item devono essere utilizzati, ossia come devono essere alternate le varie sequenze di trasmissione/ricezione, impostando quante linee di ciascun item devono essere trasmesse prima di passare al successivo e se impostare eventuali ripetizioni di gruppi di item.

Questo consente a parità di item e moduli di elaborazione di variare la strategia con cui i dati son generati, ad esempio alternando una linea di scansione B-Mode ad una Doppler oppure generando tutte le linee di un frame B-Mode prima di acquisire l'unica linea Doppler.

Come ausilio all'utente e per evitare di utilizzare item o slice non inizializzate questi possono essere scelti esclusivamente tramite liste a comparsa in cui sono indicati tutti gli Item e le Slice attualmente impostati nelle schede "Sequencer" e "Acquire".

Da questa scheda è inoltre possibile stabilire quali moduli di elaborazione devono essere utilizzati dal sistema scegliendoli da una lista che mostra tutti i moduli attualmente disponibili nel software.

Anche in questo caso l'aggiunta di linee tramite tasto destro del mouse consente rispettivamente di aggiungere nuovi intervalli di trasmissione e nuovi moduli.

2.3.1.5 Modules

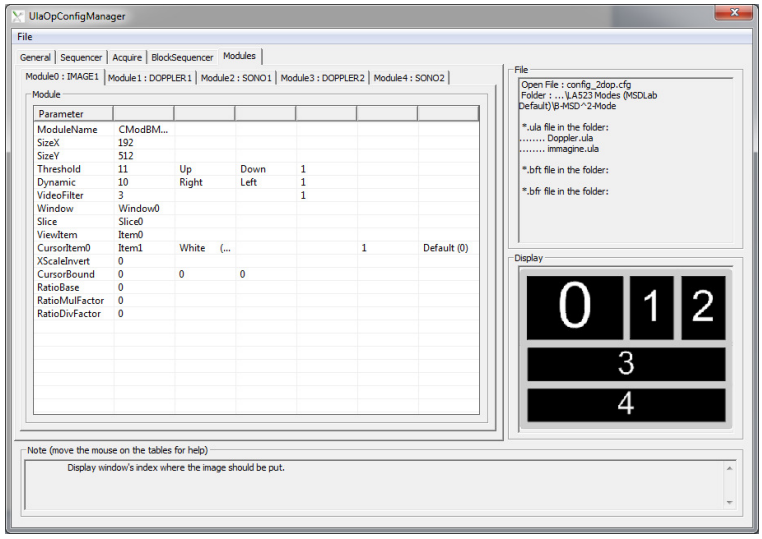

Figura 2.8 - Scheda di configurazione "Modules"

La scheda "Modules" presenta un numero variabile di sottoschede, pari al numero di moduli impostati nell'apposita tabella nella sezione "BlockSequencer".

Ad ogni modulo è associata una tabella contenente un numero variabile di parametri che dipendono dal modulo di elaborazione attualmente selezionato.

Queste tabelle sono infatti impostate tramite dei file esterni interpretati dal Configuration Editor, in modo da permettere dopo la compilazione e il rilascio del software l'aggiunta di nuovi moduli di elaborazione che siano automaticamente interpretati dal software senza la necessità di modificarlo e compilarlo nuovamente.

2.3.1.6 Item editor

L’item editor appare come un form separato richiamato dalla tabella di item nella scheda sequencer del Configuration editor; similmente a quest’ultimo questa parte del programma presenta un interfaccia organizzata in tabelle, offre una funzione di aiuto mostrata in una casella di testo e permette la manipolazione di file testuali contenenti la sintassi sezioni/chiavi/valori.

L’editor degli item presenta 5 schede con cui si possono impostare i parametri relativi a trasmissione, ricezione e demodulazione.

Sono presenti una scheda relativa ai parametri generali, quali tipo e passo della scansione e numero di linee che compongono l’item e due schede relative rispettivamente ai parametri di trasmissione e ricezione (focalizzazione, apodizzazione, frequenza utilizzata, utilizzo di file di beamforming esterni); le ultime due schede riguardano la parte di elaborazione analogica, con il parametro TGC (Time Gain Compensation) che serve per compensare la maggiore attenuazione degli echi provenienti dalle profondità maggiori, e digitale, con una tabella in cui selezionare i parametri del demodulatore.

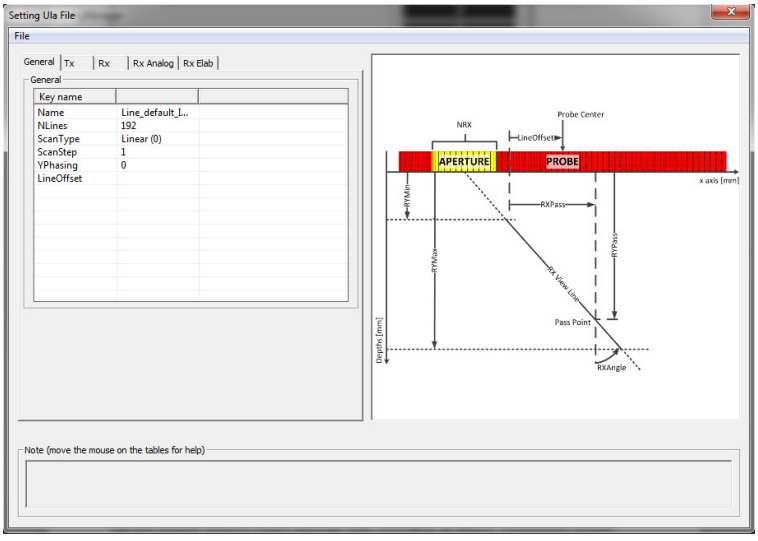

Figura 2.9 - Interfaccia per la configurazione degli Item

E’ importante far notare come, similmente a quanto detto per la schermata di editing dei moduli, anche nell’Item Editor le tabelle non vengono impostate in termini di righe e colonne secondo una struttura fissa ma sono compilate a partire dalle indicazioni di un file esterno dedicato.

La scelta di generare le tabelle dell'Item Editor non in modo fisso ma tramite file esterni è stata fatta poiché con il proseguire degli sviluppi del sistema può essere necessario variare il modo in cui gli item devono essere generati e i parametri che questi devono avere; per avere il massimo della flessibilità e non dover ricorrere a modifiche nel codice del software queste tabelle vengono quindi compilate da file esterni facilmente modificabili.

Questo ci introduce ad una delle caratteristiche principali del software ossia la sua modularità e dipendenza da file esterni.

2.3.2 Modularità

Una delle caratteristiche più importanti richieste al software Configuration Editor è stata la struttura flessibile adattabile a modifiche del firmware e del software real-time, mediante un numero minimo di modifiche al codice.

Ad esempio, per poter aggiungere e configurare nuovi moduli di elaborazione, mantenendo la coerenza tra il software real-time e il software Configuration Editor sarebbe stato necessario modificare il codice di quest'ultimo per implementare la generazione e l'interpretazione delle tabelle relative alla configurazione del nuovo modulo.

Per evitare questa necessità e rendere più flessibile e ottimizzato il codice è stato quindi scelto di basare alcune parti del software su file esterni che possano essere aggiunti o modificati anche dall'utente senza la necessità di modificare il codice sorgente; l'aggiunta di un nuovo modulo di elaborazione ad esempio viene effettuata aggiungendo un file testuale con estensione .mod con all'interno un'apposita sintassi che consente di definire i parametri che lo caratterizzano.

All'interno della cartella del programma sono quindi presenti 4 sottocartelle contenenti file che possono essere aggiunti o modificati.

- Display Template
- Help
- ModulesLib
- UlaTemplate

Display Template contiene al suo interno file di tipo immagine .png che rappresentano possibili impostazioni per il Display che, come spiegato in precedenza, da indicazioni su come i risultati delle elaborazioni dovranno essere visualizzate durante l'analisi Real-Time.

Figura 2.10 - Esempi di configurazioni display selezionabili

Questi file immagine servono in realtà solamente come ausilio all’utente per capire come verrà organizzata la visuale, in quanto come spiegato nel paragrafo 1.3.1.1 ciò che effettivamente il software Real-Time interpreterà per la preparazione delle finestre sarà proprio il nome di tale file.

Per quanto sia sempre possibile inserire manualmente il codice nella tabella “Tree” della scheda con le configurazioni generali, sono stati realizzati alcuni immagini riferite alle configurazioni più comuni come ausilio per l’utente.

L’aggiunta di ulteriori setting preimpostati può essere fatta semplicemente aggiungendo una nuova immagine rappresentante l’anteprima del display con sotto il relativo codice.

La cartella Help contiene un file testuale help.txt al cui interno sono definite le definizioni che dovranno essere mostrate a video dalla casella di testo di aiuto, in base alla chiave selezionata nelle varie tabelle; questo file può essere modificato, per cambiare la lingua con cui deve essere fornito l’aiuto.

La cartella ModulesLib contiene i template che permettono di generare le tabelle per la configurazione dei moduli; questi sono file testuali che possono quindi essere creati o modificati con un qualsiasi editor di testo.

Come detto in precedenza ciascun modulo necessita di parametri diversi per cui le tabelle nella sezione “Modules” del configuration editor non possono essere compilate a priori; i campi di tali tabelle sono quindi riempiti grazie alle indicazioni di questi file, ciascuno riferito ad un modulo, in cui viene specificato quanti parametri sono necessari, il loro tipo (ad esempio se numeri a scelta

libera o da selezionare tra un range di valori), e viene associato ad ognuno di essi una breve descrizione da mostrare nella casella testuale dell'aiuto.

In questi file la sintassi è costituita da sezioni "[SECTION]" che in questo caso rappresentano il nome del parametro da mostrare nella colonna sinistra della tabella, da uno o più campi "field", che rappresentano i valori che devono essere associati a tale parametro nelle varie celle della tabela e da una o più sezioni "help" in cui viene inserita una breve descrizione di ogni valore.

E' infine presente un campo "list" che indica, nel caso in cui il valore non sia a scelta libera ma vada scelto da una lista, come questa deve essere creata, ad esempio se deve basarsi su oggetti già definiti nel progetto, come le slice dichiarate nell'apposita scheda, o se deve basarsi su un intervallo numerico, o ancora se deve assumere solo valori predefiniti.

Si riporta come esempio la definizione di un campo "Window" per un modulo di elaborazione B-Mode che permetta la scelta della finestra in cui visualizzare il risultato dell'elaborazione del modulo a partire dal numero di finestre di visualizzazione disponibili

[Window]
Field0 = 0, TYPECELL_COMBO, LIST
Help0 = Display window's index where the image should be put.
List0 = Display

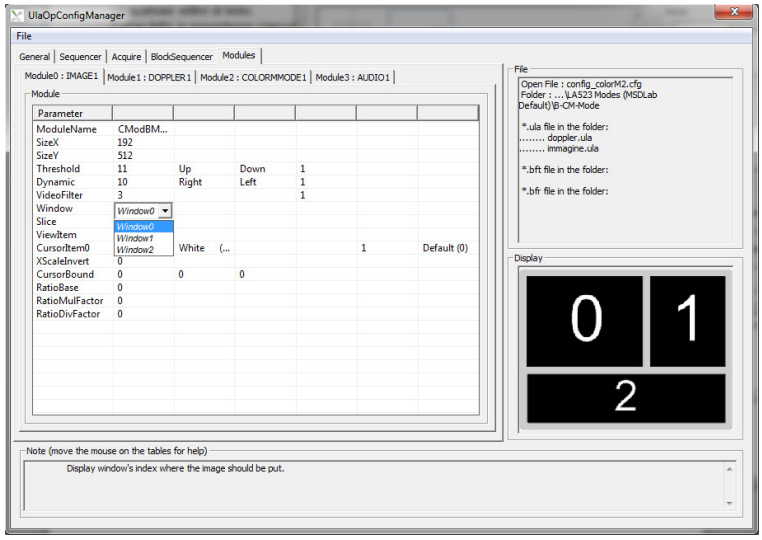

Figura 2.11 – Selezione del display tramite menù a tendina definita del template del modulo

Come si può vedere nella figura 2.11 tramite questa sintassi viene definita una riga della tabella, nella cui prima colonna sarà riportata la dicitura "Window" e che nella cella adiacente permetterà la scelta di un singolo parametro tramite una lista, generata a partire delle possibili finestre di visualizzazione disponibili nella configurazione.

Poiché all'avvio del software Configuration Editor il contenuto della cartella ModulesLib viene scansionato e i moduli presenti sono aggiunti automaticamente alla lista di quelli selezionabili per la configurazione, l'aggiunta di un nuovo modulo può essere fatta semplicemente scrivendo l'apposito template specificando quali parametri sono necessari per la sua configurazione e questo sarà automaticamente riconosciuto dal software.

Infine la cartella UlaTemplate contiene i template per l'Item editor; questi, similmente a quanto visto con i template per i moduli, permettono la generazione delle tabelle per impostare gli item dato che contengono indicazioni sui parametri impostabili, i valori che possono assumere e la descrizione per il testo di aiuto.

2.3.3 Funzionamento del software

Il funzionamento del software è basato su una struttura dati interna che mantiene le impostazioni di configurazione inserite nelle varie tabelle presenti nelle schede dell'interfaccia; questa struttura dati serve per memorizzare la configurazione, ossia tutte le impostazioni inserite mano a mano dall'utente, in modo da mantenere ordinate e catalogate tutte le variabili impostate per renderle pronte al salvataggio su un file esterno

La struttura dati interna è una classe C++ al cui interno vengono salvati tutti i parametri impostati durante la configurazione; questa è stata scritta in modo da rispettare la suddivisione in tabelle realizzata nell'interfaccia grafica.

Le variabili di questa classe, ossia le varie "key" che saranno impostate nel file di configurazione, possono essere rappresentate da variabili di tipo numerico o stringhe, nel caso di chiavi a cui deve essere associato un singolo valore, oppure possono essere rappresentate da array contenenti a loro volta classi C++, nel caso di chiavi multiple che necessitano di più parametri.

```
typedef struct TConfig{ //General
                        CString Directory;
                        CString Name;
                        CString CommentMode;
                        CString CommentType;
                        CString TreeAspect;
                        CString TreeRatio;
                        int     TreeNWin;
                        CString TreeFile;
                        int     SoundSpeed;
                        TPrf    Prf;

                        //Sequencer
                        std::vector<TItem >Item;
                        std::vector<TItemShortCut >ItemSC;

                        //Acquire
                        TSliceSection  SliceIQ;
                        TSliceSection  SliceRF8;
                        TSliceSection  SliceRF12;
                        TSliceSection  SliceRFPOST;
                        CString rftype;
                        CString loss;
                        CString bfloss;
                        TSaveOpt    SaveOpt;

                        //Blocksequencer
                        std::vector<TPri>   Pri;
                        std::vector<CString> Modules;

                        //Modules
                        std::vector<CString> ModulesFileName;
                        std::vector<TModule> Module;
                        }
```

Figura 2.12 - Classe relativa alla struttura dati

Durante la creazione di una nuova configurazione, questa struttura dati viene progressivamente aggiornata mano a mano che i parametri sono inseriti nelle diverse tabelle dell’interfaccia; ogni scheda del Configuration Editor ha associata una classe C++ che si occupa di accedere a questa struttura trasferendo i valori impostati nelle proprie tabelle in essa e viceversa.

```
//BlockSequencer PRI
typedef struct TPri{    int     RefItem;
                        int     Count;
                        CString SliceIQ;
                        int     Merge;
                        CString SliceRF;
                        int     HdStorage;
                        int     Rip;
                        }
```

Figura 2.13 - Classe relativa alla definizione degli oggetti di tipo PRI nella scheda blocksequencer

```
//Acquire -> Slice
typedef struct TSlice{  CString SliceName;
                        int     Gate;
                        int     nPri;
                        CString StartGate;
                        CString PrfDivisor;
                        CString Save;
                        }
```

Figura 2.14 - Classe relativa alla definizione degli oggetti di tipo slice nella scheda Acquire

Tutte le classi relative alle schede mostrate a video sono derivate dalla medesima classe madre che ne definisce le funzioni di interfaccia, in modo da sfruttare la caratteristica di polimorfismo offerta dalle classi C++.

In particolare si individuano tre funzioni principali, ridefinite in ogni classe figlia per adattarsi le tabelle e ai parametri presenti nella scheda associata.

La funzione *Init* serve come inizializzazione, viene chiamata subito dopo l’avvio del software e serve per impostare le tabelle da disegnare a video, definendone la dimensione, i testi da inserire nella prima riga e prima colonna e le caratteristiche delle celle, ad esempio se devono essere modificabili o meno oppure basate su liste di elementi.

La funzione *UpDateConfigFileToTable* viene chiamata quando viene aperta una scheda e consente di riempire i vari campi delle tabelle con i valori presenti nella struttura dati interna.

La funzione *UpDateTableToConfigFile* viene chiamata quando una scheda viene chiusa e durante questa operazione vengono letti i valori inseriti nelle tabelle, una riga ossia una chiave alla volta, e vengono allocate nella struttura dati le rispettive classi per memorizzarle.

Le tabelle presenti nelle varie schede sono definite tramite l’override degli oggetti di tipo lista in Visual C++ e sono associate a classi che presentano funzioni di controllo per creare la tabella, aggiungendo righe e colonne, interpretare i messaggi relativi alla pressione dei tasti del mouse e identificare la riga o la cella selezionate al momento della pressione di un tasto del mouse.

Un click con il tasto sinistro sulla tabella viene rilevato tramite il corrispondente messaggio di Visual Studio: viene quindi identificata la cella su cui si trova il cursore e questa viene “attivata”.

Anche alle singole celle sono infatti associate classi C++ e viene sfruttato il polimorfismo in modo che tramite un’unica funzione di interfaccia all’apertura di una cella questa mostri il comportamento desiderato.

In particolare si distinguono le celle contenenti testo non modificabile, tipicamente quelle della prima colonna a sinistra che indicano i nomi delle chiavi,

le celle che accettano inserimenti di testo da tastiera e le celle che mostrano una lista da cui è possibile selezionare il valore desiderato; nelle celle testuali modificabile possono inoltre essere inseriti controlli che ad esempio limitano i caratteri accettabili ai soli numeri.

Come ausilio per l’utente vengono infine create per ogni scheda le liste di oggetti creati, che siano le slice oppure gli item; queste liste vengono aggiornate ogni volta che la corrispondente scheda viene chiusa e sono utilizzate in altre tabelle in modo da limitare all’utente la scelta tra i soli oggetti effettivamente definiti; si consideri ad esempio la definizione dei parametri per un modulo di elaborazione: in questo deve essere dichiarata la slice da cui prelevare i dati da elaborare e per evitare configurazioni incorrette all’utente viene permesso di scegliere solamente tra quelle attualmente dichiarate

2.3.4 Salvataggio e lettura del file di configurazione

Completata la configurazione questa può essere salvata su un file esterno da utilizzare con il software Real-Time.

L’operazione di salvataggio si basa sulla lettura sequenziale della struttura dati: per ogni sezione da generare vengono lette le variabili da associare a questa, generando nel file di testo le rispettive chiavi e associandovi i rispettivi valori contenuti nella struttura dati.

```
//Salvataggio Slices IQ
for(i=0;i<(int)_Config.SliceIQ.Slice.size();i++)
{
    str.Format(_T("Slice%d"),i);;
    if(TestSlice(&_Config.SliceIQ.Slice[i]))
        filemanager["ACQUIREIQ"][str]<<_Config.SliceIQ.Slice[i].Gate
                                     <<_Config.SliceIQ.Slice[i].nPri
                                     <<_Config.SliceIQ.Slice[i].StartGate
                                     <<_Config.SliceIQ.Slice[i].SliceName
                                     <<_Config.SliceIQ.Slice[i].PrfDivisor
                                     <<RemoveBracket(_Config.SliceIQ.Slice[i].Save);
}
```

Figura 2.15 – Codice relativo al salvataggio su file di configurazione delle slice iq

Si consideri ad esempio la parte di codice riguardante la creazione della sezione [ACQUIREIQ] nel file di configurazione e le relative chiavi riportata in figura 2.15

All’interno di un ciclo for, che ripete l’inserimento per ognuna delle slice iq create dall’utente, viene manipolata una variabile di tipo stringa grazie alla funzione format, che permette di generare una stringa a partire da del testo e da

variabili numeriche; in questo caso permette la generazione delle chiavi Slice1, Slice2, Slice3, incrementando l'indice ad ogni iterazione del ciclo. Questa stringa definirà in questo caso il nome delle chiavi da inserire nella sezione.

E' presente come controllo preliminare una funzione di test, che effettua una verifica sui dati inseriti, per evitare il salvataggio di chiavi impostate in maniera incorretta o incompleta.

Superato il controllo viene preparato localmente il testo da salvare nel file; la sintassi *filemanager["ACQUIREIQ"][str]*, dove str assume sequenzialmente il nome di tutte le chiavi da inserire, genera (se ancora non esiste) la sezione e aggiunge a questa la chiave seguita dall'elenco dei parametri associati, inseriti utilizzando l'operatore di inserimento su stream "<<".

Queste operazioni sono ripetute per tutte le variabili e classi della struttura dati interna; al termine dell'inserimento delle sezioni viene effettuata la scrittura file di testo esterno.

Nel caso di apertura di un file creato in precedenza viene fatta l'operazione opposta, ossia il file esterno viene letto e da esso vengono identificate le sezioni e le chiavi, permettendo così di associare alle rispettive classi nella struttura dati interna i valori corretti.

La lettura di un file esterno comporta anche un controllo di eventuali errori, segnalando all'utente la presenza di sezioni o chiavi duplicate o la mancata definizione di alcune di esse.

Si consideri ad esempio la parte di codice relativa alla sezione [ACQUIREIQ] del file di configurazione riportata nella figura 2.16; viene utilizzata una funzione *CompileAcquire* che riceve come parametri di ingresso il puntatore al file di configurazione caricato in memoria, la stringa che definisce il nome della sezione da caricare, nel caso in esempio ACQUIREIQ e il puntatore alla struttura che definisce le slices nella struttura dati interna.

Viene inizialmente controllato se la sezione esiste nel file di configurazione e in caso positivo vengono ricercate all'interno di questa le key che potrebbero essere presenti, in modo incrementale (Slice1, Slice2, Slice3…); per ogni key esistente i valori che questa possiede vengono letti sequenzialmente tramite l'operatore di estrazione ">>" ed inseriti nelle rispettive variabili all'interno della struttura dati.

```
void CConfFile::CompileAcquire(NCfg::CConfigFile *pCfgfile,CString str, TSliceSection *pSlice)
{
    int nkey;
    bool go=true;

    TSlice SliceApp;
    CString key;

    if(pCfgfile->SectionExists(str))
    {
        const CSection &sectionname=(*pCfgfile)[str];

        nkey=0;

        while(go)
        {
            key.Format(_T("Slice%d"),nkey);
            if(sectionname.KeyExists(key))
            {
                sectionname[key]>>SliceApp.Gate
                                >>SliceApp.nPri
                                >>SliceApp.StartGate
                                >>eQUIET>>SliceApp.SliceName
                                >>SliceApp.PrfDivisor
                                >>SliceApp.Save;
                pSlice->Slice.push_back(SliceApp);
            }
            else
                go=false;

            nkey++;
        }

        if(sectionname.KeyExists(_T("RfLoss")))
            sectionname[_T("RfLoss")]>>eQUIET>>pSlice->RfLoss;
    }
}
```

Figura 2.16 - Codice relativo alla lettura dal file di configurazione delle impostazioni delle slice iq

2.4 Configuration editor con funzionalità drag & drop

Il “Configuration editor drag & drop” è un’ulteriore sviluppo del software volto a offrire un interfaccia ulteriormente più facile e intuitiva da utilizzare.

Questa versione presenta modifiche sostanziali soprattutto a livello di interfaccia mentre mantiene a livello di codice la stessa architettura basata sulla struttura dati interna, e le medesime funzioni che ne gestiscono l’aggiornamento e la sua scrittura o lettura verso file di testo esterni.

La variazione fondamentale è a livello di interfaccia, in particolare nelle schede dedicate alla scelta e configurazione dei moduli e alla scelta e ordinamento delle sequenze di trasmissione ricezione; le modifiche fatte non

influiscono sulla generazione del file di configurazione e nella sua sintassi, ma permettono una configurazione più intuitiva.

Rispetto alla versione precedente del software in cui tutti i parametri erano impostati tramite tabelle, scrivendone il valore o selezionandoli da liste, in questa versione la funzione di trascinamento consente di selezionare gli "oggetti" da associare al progetto, trascinandoli appunto nella posizione e nell'ordine voluto.

Si consideri ad esempio la scelta dei moduli di elaborazione: per rendere più immediata e comprensibile l'associazione tra la configurazione realizzata tramite "configuration editor" e la schermata mostrata dal software real time in cui vengono visualizzate le varie finestre con i risultati delle elaborazioni, in questa versione del software viene creata una struttura in grado di accogliere oggetti trascinati di tipo "modulo" che è suddivisa in sottofinestre basandosi sulle impostazioni del display cosi come l'interfaccia real-time.

In questo modo è possibile dopo la selezione del "Display" trascinare direttamente un modulo nella finestra in cui si vuole che vengano mostrati i risultati, provvedendo successivamente a configurarlo, a vantaggio della semplicità di utilizzo da parte dell'operatore.

Questa funzionalità viene anche utilizzata anche per la selezione e il riordino delle fasi di ricezione/trasmissione, consentendo di prelevare gli item da una lista e di inserirli nella lista "Blocksequencer" nell'ordine desiderato.

Il drag & drop consente inoltre di riordinare gli elementi inseriti in questa lista, consentendo di riposizionare gli elementi semplicemente trascinandoli nella posizione voluta.

2.4.1 Interfaccia Utente

Come nella versione descritta in precedenza l'interfaccia utente è realizzata utilizzando oggetti Visual Studio, a cui sono associate classi C++ che ne implementano i controlli.

L'interfaccia è suddivisa in 4 schede, selezionabili tramite l'apposito selettore:

- General: Impostazioni generali.
- Modules: scelta e configurazione dei moduli di elaborazione
- Acquire: creazione delle slice di memoria.
- Sequencer: sezione e ordinamento degli item.

2.4.1.1 General

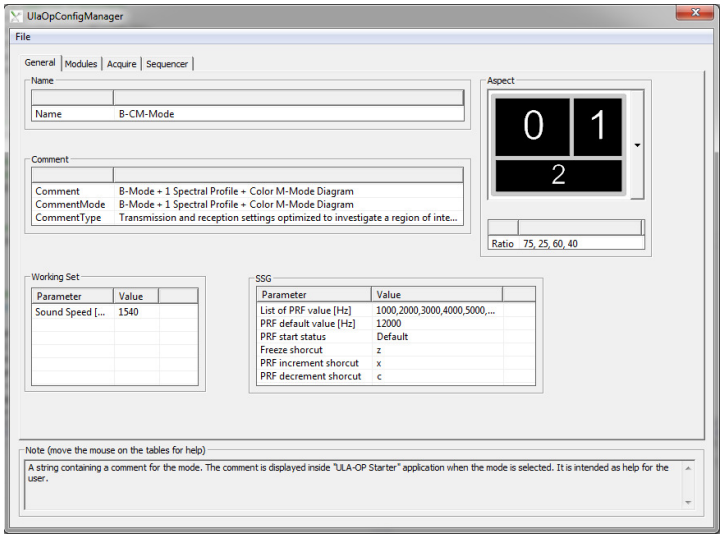

Figura 2.17 - Scheda di configurazione “general” del configuration editor drag & drop

La prima scheda, è sostanzialmente immutata rispetto a quella descritta nella precedente versione del configuration editor e presenta delle tabelle per impostare i parametri generali della configurazione.

E’ stato però modificato il meccanismo per la scelta dell’organizzazione del display in quanto è stata implementata una combobox che mostra una lista a comparsa, in cui sono contenute le immagini di tutte le possibili configurazioni selezionabili.

Le immagini presenti nella cartella DisplayTemplate sono automaticamente riconosciute all’avvio del software e aggiunte a questa lista.

E’ stata rimossa la casella di testo che mostra i file presenti nella cartella contenente il file di configurazione in quanto questa informazione viene mostrata solo tramite una message-box all’apertura della configurazione stessa

2.4.1.2 Modules

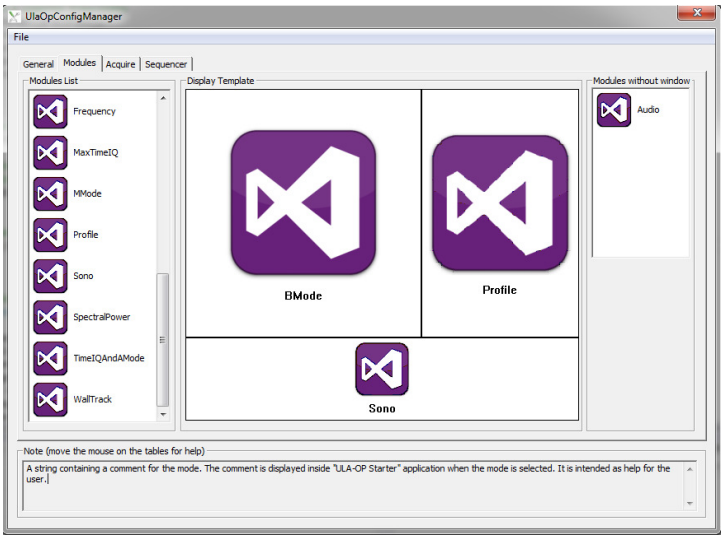

Figura 2.18 - Scheda di configurazione “modules” del configuration editor drag & drop

La seconda scheda permette la scelta dei moduli e implementa la nuova funzionalità di Drag & Drop.

A sinistra dell’interfaccia è presente una lista in cui sono presenti tutti i moduli di elaborazione riconosciuti dal sistema, e questi sono elementi che possono essere trascinati mantenendo il tasto sinistro del mouse premuto su di essi.

Al centro viene realizzato un frame di oggetti che accettano il rilascio degli elementi trascinati, il cui aspetto complessivo richiama quello del display che sarà mostrato dal software real-time; il numero e le proporzioni di questi oggetti contenitori è stabilito infatti dalle impostazioni “display” nella prima scheda e rilasciando un modulo su una di queste “celle” questi sarà automaticamente associato a quella finestra di visualizzazione e aggiunto alla configurazione.

A destra è infine presente una lista, dedicata ad accogliere quei moduli che non necessitano di una finestra di visualizzazione; un controllo effettuato durante il rilascio dei moduli impedisce di inserire in questa lista moduli che necessitano di una visualizzazione a video.

E’ possibile invertire di posizione due moduli già posizionati semplicemente trascinandone uno al posto dell’altro, così come possono essere eliminati tramite il menù contestuale aperto dal tasto destro del mouse.

Tale menù permette inoltre di accedere alla schermata di modifica del modulo in cui viene mostrata una tabella che permette di impostare i parametri che lo caratterizzano (fig. 2.19).

Nonostante attualmente siano impiegate immagini generiche per rappresentare i moduli, future modifiche di natura puramente estetica faranno si che l'interfaccia mostri per ciascun modulo una diversa immagine a rappresentarne l'anteprima, in quanto il codice realizzato è in grado di associare ad ogni file template dei moduli una diversa immagine con lo stesso nome presente nella cartella

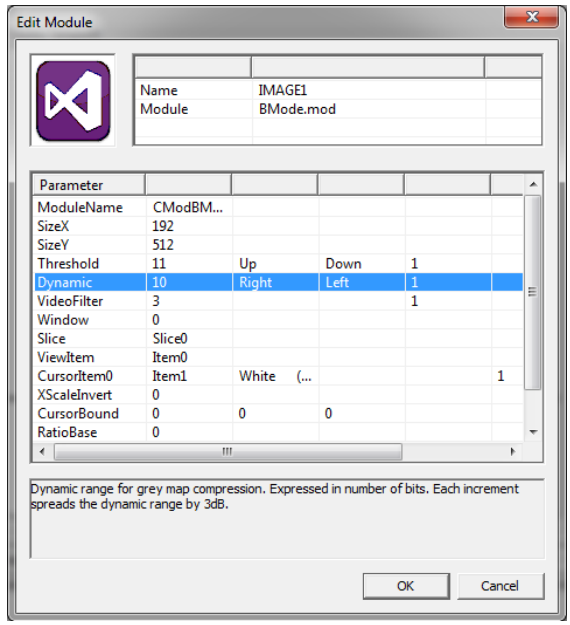

Figura 2.19 - Interfaccia per la configurazione di un modulo

2.4.1.3 Acquire

La terza scheda è sostanzialmente invariata rispetto a quella vista nel Configuration Editor precedente e consente sempre l'impostazione delle slice di memoria.

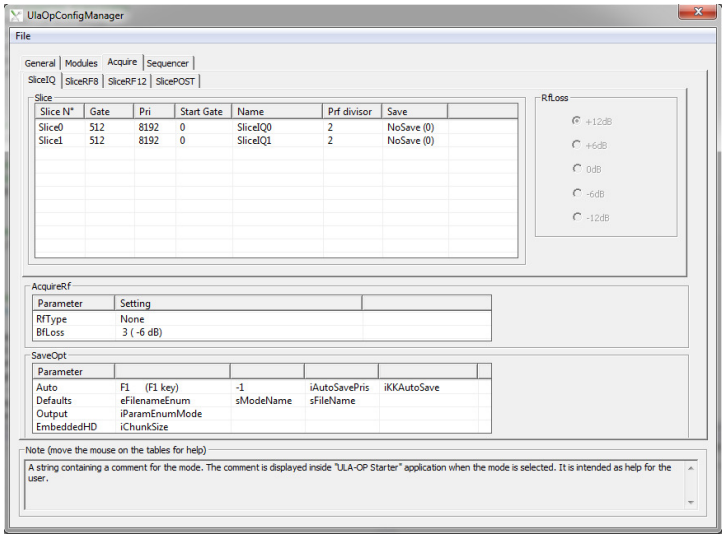

Figura 2.20 - Scheda di configurazione “acquire” del configuration editor drag & drop

2.4.1.4 Sequencer

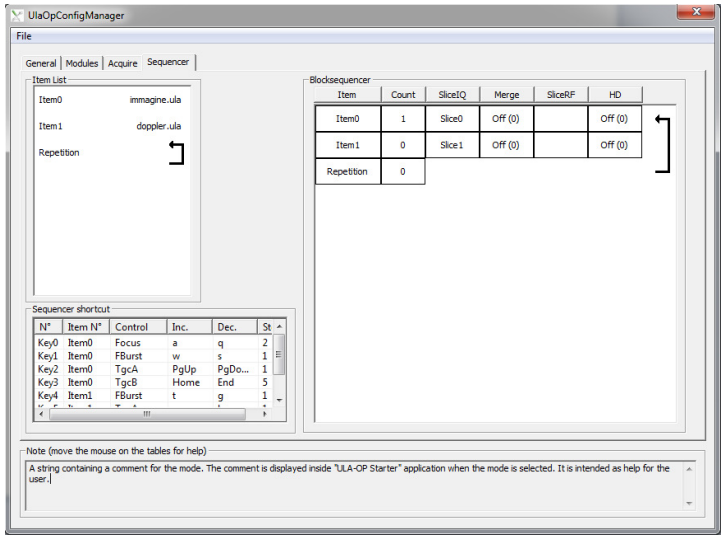

Figura 2.21 - Scheda di configurazione “sequencer” del configuration editor drag & drop

La quarta e ultima scheda invece consente le impostazioni degli item e del blocksequencer e presenta anche essa funzionalità di Drag & Drop.

A sinistra è presente una lista in cui sono inseriti gli item attualmente disponibili per la configurazione, che possono ancora una volta essere aggiunti a partire da item già esistenti nel computer, oppure modificati o creati da zero tramite l'Item Editor, richiamabile tramite il menù aperto dalla pressione del tasto del mouse.

La lista sulla destra rappresenta il blocksequencer e gli item possono esservi trascinati, per poi essere successivamente modificati utilizzando un doppio click del mouse per renderli editabili.

Gli elementi di questa lista possono essere eliminati, utilizzando il menù aperto dal tasto destro del mouse, oppure riordinati, selezionandoli e trascinandoli nella posizione desiderata.

Come ausilio grafico l'aggiunta di un blocco di ripetizione, che serve per far ripetere più volte i PRI impostati in precedenza, provoca la comparsa a video di una freccia che mostra quali sono gli elementi coinvolti dalla ripetizione.

2.4.2 Implementazione della funzionalità "Drag & drop"

Per implementare il Drag & Drop sono stati ridefiniti alcuni oggetti visual studio, in particolare le liste di oggetti e gli elementi di tipo static, per far si che ricevano i messaggi del mouse relativi alla pressione dei tasti e al movimento e permettano di implementare il trascinamento degli oggetti.

Tenendo premuto il tasto sinistro del mouse su un oggetto trascinabile, ad esempio un modulo presente nella lista, e spostandolo, il movimento del mouse attiva il trascinamento e viene disegnata una copia dell'oggetto centrato sotto il puntatore; in fase di trascinamento infatti spostandolo il cursore lungo lo schermo l'oggetto verrà ridisegnato seguendone i movimenti;

Rilasciando il tasto sinistro l'oggetto viene rilasciato e viene verificato se il cursore si trova in quel momento sopra un oggetto che può accettare il Drop di quel particolare elemento, accettandone l'inserimento o respingendolo.

Il Drag & Drop è basato su una macchina a stati che identifica la condizione dell'oggetto trascinato, gestita dall'oggetto da cui ha origine il trascinamento; nello stato iniziale, chiamato NONE, non sta accadendo nulla e non è on corso nessuna operazione di trascinamento.

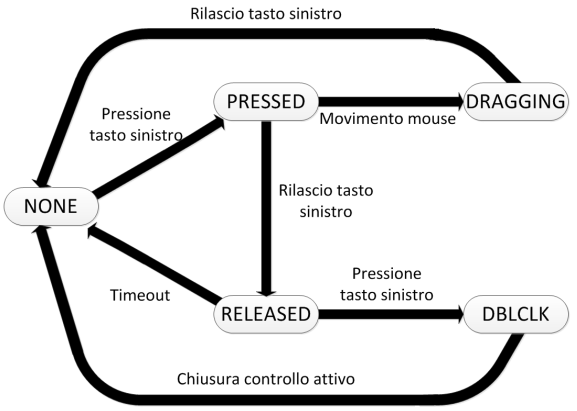

Figura 22 - Diagramma della macchina a stati che implementa il drag & drop

Premendo il tasto del mouse lo stato diventa PRESSED e notifica che è stata rilevata la pressione del tasto sinistro del mouse ma ancora non è stato avviato il trascinamento dell’oggetto.

Quando il mouse viene premuto infatti viene salvata la posizione attuale del cursore ed è necessario che questi si allontani oltre una certa distanza per avviare l’operazione di trascinamento; questa soglia è necessaria in quanto si vogliono distinguere i casi in cui si vuole avviare lo spostamento dell’oggetto da quelli in cui si vuole effettuare un singolo click, o doppio click, su di esso e lo stato non deve cambiare in quello che riflette il trascinamento.

Se il movimento del mouse è tale da superare la soglia viene memorizzata una struttura dati che contiene i parametri dell’oggetto trascinato e lo stato passa in DRAGGING, provocando il disegno a video dell’oggetto trascinato sotto al puntatore.

Rilasciando il tasto sinistro del mouse viene effettuato il Drop dell’oggetto: questa operazione consiste nel verificare se al momento del rilascio il puntatore del mouse si trova sopra un elemento che può ricevere oggetti tramite trascinamento, informazione ottenibile poiché Visual Studio permette di risalire alla classe associata all’oggetto su cui si trova il puntatore del mouse, e in caso positivo l’oggetto trascinato viene inserito nell’elemento di destinazione; nel caso di Drop su una lista ad esempio a questa viene aggiunto un nuovo elemento in base ai parametri dell’ oggetto memorizzato.

Sono inoltre presenti altri 2 stati necessari per implementare la funzione di doppio click.
Dallo stato PRESSED infatti, se il puntatore del mouse non viene spostato oltre la soglia impostata per attivare il trascinamento, rilasciando il tasto sinistro del

mouse si raggiunge lo stato RELEASED; da questo stato viene attivato un timer che opera come timeout per attendere il secondo click, e se il tasto sinistro del mouse viene premuto nuovamente in questo intervallo viene effettuato il passaggio allo stato DBLCLK e attivate le relative funzioni relative a questo evento, ad esempio viene abilitata la modifica dell'elemento selezionato nella lista blocksequencer.

La chiusura del controllo attivato dal doppio click, ad esempio premendo il tasto sinistro mentre il cursore si trova fuori dall'oggetto, provoca il ritorno allo stato NONE.

Capitolo 3

Sviluppo di firmware per DSP

3.1 Scelte progettuali e requisiti

I compiti dei DSP a bordo del sistema ULA-OP sono principalmente due: occuparsi della programmazione delle FPGA e elaborare opportunamente i dati ricevuti da questa.

La fase di programmazione consiste nella scrittura dei registri interni alla FPGA per predisporla a operazioni quali la gestione delle forme d'onda da utilizzare in trasmissione, la focalizzazione dei fasci e le operazioni di somma e pesatura dei campioni ricevuti per effettuare il beamforming.

La fase di elaborazione consiste nelle operazioni di filtraggio ed elaborazione digitale sui dati beamformati ricevuti dalla FPGA in modo da convertire tale flusso dati in immagini visibili da pc in base al tipo di elaborazioni richieste.

Una delle caratteristiche principali ricercate nel firmware sviluppato è la massima flessibilità: trattandosi ULA-OP di un sistema di ricerca infatti è necessario che il firmware sia facilmente espandibile e modificabile, al fine di aggiungere con la massima semplicità ulteriori moduli per svolgere nuovi tipi di acquisizione e analisi; per rispondere a queste esigenze la struttura base del firmware deve essere quindi organizzata nel modo più modulare e aperta possibile.

Per i DSP presenti nel sistema sono stati realizzati due firmware ben distinti: un firmware di boot e un firmware applicativo.

Il firmware di boot è un firmware molto compatto in quanto deve essere precaricato su memorie EEPROM, una su ciascuna delle schede del sistema,

dotate di soli 8kB di memoria, ed ha il compito di effettuare le semplici inizializzazioni base del sistema, preparandolo ad accogliere i firmware applicativi per DSP e relativa FPGA ricevuti da PC; il firmware applicativo invece svolge le funzioni di analisi e elaborazione dati per cui è stato realizzato il sistema ULA-OP.

La scelta di utilizzare due firmware distinti è stata fatta per ragioni di efficienza e flessibilità: avere un unico firmware contenente inizializzazioni e moduli di acquisizione e elaborazione dati avrebbe richiesto innanzitutto memorie EEPROM di dimensioni maggiori per la sua memorizzazione e si sarebbe inoltre rivelata una soluzione poco efficiente in quanto l'aggiunta di ulteriori moduli di elaborazione per nuovi tipi di indagine o la modifica di quelli esistenti avrebbe costretto a una riprogrammazione della memoria di boot, tramite un apposito programmatore, poco pratica in quanto il sistema ULA-OP è pensato per essere distribuito in tutto il mondo.

Utilizzando due firmware invece, la EEPROM di boot viene precaricata con un firmware che non necessita aggiornamenti o modifiche, mentre il firmware applicativo venendo caricato da PC può essere facilmente modificato e aggiornato in quanto questo non è permanentemente situato all'interno della memoria della scheda, ma viene ogni volta distribuito nella sua versione più aggiornata, assieme al relativo software di interfaccia e visualizzazione real-time delle immagini acquisite su PC.

Ulteriore vantaggio risultante da questa scelta è la possibilità di avere anche più firmware applicativi diversi, in base a esigenze particolari, potendo scegliere tramite PC quale utilizzare di volta in volta, caratteristica impossibile da ottenere nel caso in cui il firmware fosse caricato permanentemente nella memoria del sistema.

Riguardo al firmware applicativo una scelta progettuale fatta durante la programmazione è stata quella di utilizzare un firmware univoco da caricare su tutti i core di tutti i DSP presenti nelle schede front-end e nella scheda interfaccia e differenziandone il comportamento solo in base a comandi ricevuti dall'esterno.

Il vantaggio più evidente di questa soluzione è la semplicità progettuale, in quanto non è necessario realizzare firmware diversi ed estremamente specificati in base al core e al DSP di destinazione ma è sufficiente svilupparne uno il più generico possibile; ricordiamo che nel sistema sono utilizzati 17 DSP con 8 core ciascuno e sarebbe stato molto oneroso sviluppare un firmware diverso per ciascuno di essi.

Un ulteriore vantaggio è il fatto che essendo uguali i firmware le risorse allocate sono le medesime e posizionate agli stessi indirizzi ovunque, per cui

ogni core può identificare l'indirizzo logico di un buffer o una tabella presenti in un DSP qualsiasi nel sistema, semplicemente a partire dalla conoscenza dell''indirizzo locale di quella particolare risorsa.

Ovviamente questa scelta porta anche ad alcuni svantaggi, in particolare una ridotta ottimizzazione del codice, in quanto il firmware deve essere appunto generico per adattarsi a tutti i possibili dispositivi, e uno spreco di memoria in quanto vengono comunque inserite nel codice classi e funzioni e allocate variabili, a prescindere dal fatto che quel particolare core possa avere bisogno di utilizzarle.

Un ulteriore scelta progettuale effettuata allo scopo di aumentare la modularità del firmware è stata quella di organizzare le varie parti funzionali come dei singoli task indipendenti, molto specifici come tipo di operazioni da eseguire, gestiti da un microkernel che ciclicamente richiama in esecuzione solamente i task che hanno effettivamente dei dati da elaborare o dei compiti da svolgere; il vantaggio immediato dato da questa scelta è un ottimizzazione nell'utilizzo della CPU in quanto sono richiamati per l'esecuzione solo i task che hanno bisogno di eseguire le proprie operazioni, evitando i tempi morti tipici invece delle soluzioni che effettuano controlli a polling.

Un ulteriore vantaggio dato da questa scelta progettuale è dato dal fatto che l'aggiunta di nuove funzioni può essere effettuata semplicemente scrivendo codice per realizzare task che le implementano, senza necessità di modifiche sostanziali nel codice se non nei singoli task che devono comunicare con quelli appena aggiunti.

Utilizzare task separati consente inoltre di aumentare ulteriormente la flessibilità del sistema, in quanto è possibile tramite il software di interfaccia dare all'utente la facoltà di decidere in quale core e in quale DSP allocare i task disponibili, a vantaggio dell'ottimizzazione e della suddivisione del carico di lavoro.

Per sfruttare appieno questa struttura e ottenere il massimo della flessibilità è stato infine sviluppato un sistema, che astrae dalla posizione dei task consente di metterli in comunicazione tra di loro per lo scambio dati, indipendentemente dalla scheda, dal DSP e dal core su cui si trovano in esecuzione.

3.2 Il Firmware di boot

Il firmware di boot viene caricato solamente nel core 0 del DSP 0; i restanti core e l'altro DSP saranno invece attivati eventualmente solo dal firmware applicativo, in base alle richieste effettuate da PC.

Il firmware di boot viene caricato automaticamente nel DSP dalla memoria EEPROM che lo contiene attraverso l'interfaccia I2C, appena viene rilasciato il reset; questo è un firmware estremamente semplice e compatto, in quanto risiede in una memoria da soli 8kB, ed ha fondamentalmente 4 funzioni:

- Inizializzazione primaria del DSP
- Programmazione delle tabelle di routing locale dello switch SRIO
- Caricamento del firmware nella FPGA
- Caricamento del firmware applicativo per il DSP stesso

Per effettuare queste operazioni, nel firware di boot vengono definite alcune classi, ciascuna associata ad un diverso modulo hardware del DSP, quali CGpio, CSPI, CI2C e CSRIO.
Viene inoltre definita una classe CSwitch contenente le funzioni di interfaccia per la configurazione dello switch Serial Rapid IO.

La programmazione dell'hardware di sistema per la configurazione delle varie periferiche e lo sfruttamento delle loro funzionalità è basata sulla scrittura dei rispettivi set di registri; la definizioni di classi di controllo per i vari moduli hardware consente di mettere a disposizione funzioni di interfaccia per avviare la programmazione della periferica, tramite una funzione Setup(), o per sfruttarne le varie funzioni offerte, quali la lettura dello stato dei pin GPIO o la trasmissione di dati via SPI, offrendo un set di funzione di interfaccia sfruttabili dalle altre classi o moduli a vantaggio della pulizia e della leggibilità del codice.

Una volta effettuate tutte le operazioni viene automaticamente lanciato in esecuzione il firmware applicativo notificando al pc l'avvenuto caricamento.

3.2.1 Inizializzazione del DSP

Immediatamente subito dopo il boot il firmware si occupa di programmare i registri per configurare le seguenti periferiche:

- PLL
- Cache
- GPIO
- Interfaccia SPI
- Interfaccia I2C
- Gestore memorie DDR3
- Interfaccia SRIO

Questa fase di mini–inizializzazione è necessaria per rendere operativo il DSP nelle sue funzionalità base, preparandolo poi ad accogliere i dati da Pc: in particolare sono necessari i bus di comunicazione SPI, I2C e SRIO rispettivamente per la programmazione dell' FPGA, per la lettura della Serial Presence Detect (SPD) EEPROM nei moduli DDR3 contenente i parametri da impostare per la loro configurazione e per la comunicazione DSP-DSP e DSP-FPGA sulla stessa scheda o su schede diverse nel sistema.

3.2.1.1 PLL

Il clock principale del core da 1GHz viene ottenuto a partire dal clock in ingresso da 156,25 MHz; gli altri clock interni utilizzati dalle varie periferiche sono frazioni del clock principale

Nome Clock	Fattore divisivo	Frequenza	Periferiche
SYSCLK1	-	1 GHz	DSP Core
SYSCLK3	1 / 2	500 MHz	MSMC, HyperLink, TeraNet, DDR, EDMA
SYSCLK4	1 / 3	333,33 MHz	Fast Peripherals
SYSCLK7		166,66 MHz	Slow Peripherals
SYSCLK10	1 / 3	333,33 MHz	SRIO

Tabella 3.1- Valori di clock interni al DSP

3.2.1.2 Cache

Cache dati e cache programma sono impostate al massimo del valore ossia 32 kB ciascuna.

La cache programma è a mappatura diretta (unica via), mentre la cache dati è di tipo associativo a 2 vie. La politica di allocazione/disallocazione della cache dati è LRU (Last Recently Used).

3.2.1.3 GPIO

La configurazione dei pin GPIO prevede l'impostazione della direzione che deve assumere ogni singolo pin, specificando se si tratta di una linea di uscita o di ingresso.

I seguenti gruppi di pin sono impostati come ingressi:

- I pin GP4, GP5, GP6, GP7, GP8 sono collegati al backplane e sono utilizzati principalmente per identificare la posizione in cui è stata inserita la scheda.
- I pin GP9, GP10, GP13, GP14, GP15 sono collegati alla FPGA e sono utilizzati come sorgente di interrupt per la comunicazione con quest'ultima.
- Il pin GP3 è collegato alla FPGA è serve per ricevere una conferma del corretto avvio di quest'ultima dopo aver caricato il suo firmware.

I seguenti gruppi di pin sono impostati come uscite:

- I pin GP11, GP12 sono collegati ai led frontali della scheda.
- Il pin GP1 è collegato al bus transceiver è viene utilizzato come Output Enable.
- Il pin GP2 è collegato alla FPGA e serve come segnale di reset per prepararla a ricevere la propria configurazione.
- Il pin GP0 è inutilizzato ma viene comunque impostato come uscita e collegato ad un resistore di pull-up per ridurre il consumo di corrente da parte del dispositivo.

3.2.1.4 SPI

L'interfaccia SPI viene utilizzata per caricare il firmware nella FPGA Tale periferica riceve un clock di 166,66 MHz (SYSCLK7) e può trasmettere a una frequenza massima di SYSCLK7/3 = 55,55 MHz; il prescaler viene quindi impostato per dividere per 3 il clock in ingresso; con questo vincolo viene abbondantemente rispettata anche la frequenza massima ammessa dall'interfaccia SPI della FPGA pari a 125 MHz.

La periferica viene inoltre impostata in modalità master mode a 3 pin

3.2.1.5 I2C

L'interfaccia I2C viene utilizzata per la lettura della Serial Presence Detect (SPD) EEPROM nei moduli di memoria DDR3.

Tale periferica riceve un clock di 166,66 MHz (SYSCLK7) e può operare ad una frequenza massima di 12 MHz; il prescaler viene quindi impostato in modo tale da dividere il clock in ingresso per 14 ottenendo un clock di periferica di 11,9 MHz.

La frequenza di trasmissione viene ottenuta dividendo il clock di periferica per un ulteriore prescaler; poiché la memoria EEPROM a bordo del modulo DDR3 richiede un clock massimo di 400 kHz il prescaler viene impostato in modo da dividere per 18 il clock di periferica ottenendo così una frequenza di trasmissione pari a 396,66 kHz.

3.2.1.6 DDR3

Le memorie DDR3 installate sono moduli SO-DIMM da 8 GB sfruttabili con una velocità fino a 1333MT/s (Clock da 666,5 MHz).

Il controllore DDR3 viene impostato utilizzando i parametri di timing forniti dal produttore di memorie.

Per una maggiore flessibilità del sistema è possibile effettuare l'autoconfigurazione delle memorie, ossia leggendo in automatico i parametri di configurazione e eliminando la necessità di aggiornamenti al firmware di BOOT in casi di utilizzo di moduli con parametri molto differenti: lo standard Jedec infatti prevede all'interno dei moduli di memoria la presenza di una piccola memoria EEPROM accessibile tramite I2C contenente tutti i parametri hardware e di timing necessari per la configurazione del modulo; è quindi possibile accedere a tale memoria e ottenere i valori di configurazione corretti, rendendo così il firmware molto più flessibile in caso di sostituzione dei moduli di memoria.

3.2.1.7 SRIO

Il firmware associa alla propria periferica SRIO un ID dipendente dall'identificatore del DSP e dall'identificatore della posizione della scheda nel backplane; entrambe le informazioni sono ottenute leggendo tramite GPIO delle linee collegate a pull-up.

La periferica Serial Rapid IO viene configurata in modo da operare in modalità a 4 Lane al datarate di 5 Gbps ciascuna.

Viene anche impostato uno swap dei dati ricevuti a 64 bit in modo da adattarsi all'endianità dei dati all'interno dell'architettura di indirizzamento della FPGA.

3.2.1.8 Rimappatura della memoria condivisa

Durante le operazioni di caricamento firmware durante il boot e durante il normale funzionamento a run-time viene sfruttata la memoria condivisa del DSP da 4 MB.

Poiché il DSP non si occupa di mantenere la coerenza della cache con questa memoria, ogni volta che è necessario utilizzarla sarebbe necessario effettuare un'operazione di invalidazione della cache; come alternativa più rapida e efficiente è stato quindi scelto di rimappare l'indirizzo logico utilizzato per accedere alla memoria condivisa tramite i registri del controllore di memoria, utilizzando un range di indirizzi per cui i dati non vengono inseriti nella cache.

3.2.2 Programmazione delle tabelle di routing

La seconda fase consiste nella programmazione dello switch Serial Rapid IO presente nella scheda, in modo da abilitare le porte necessarie alla comunicazione e impostare le tabelle di routing locale per instradare i pacchetti verso i due DSP o verso la FPGA presenti nella scheda.

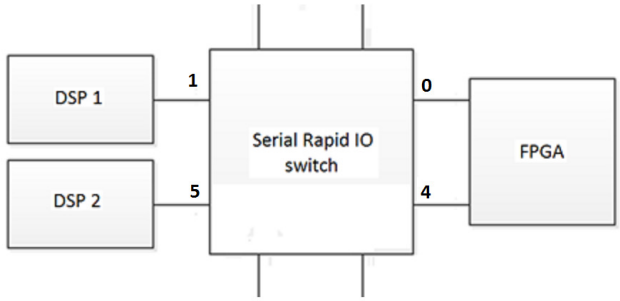

Figura 3.1 - Connessione dei dispositivi alle porte dello switch SRIO

Il routing dei pacchetti SRIO è basato su ID a 16 bit e viene effettuato su due livelli distinti: gli 8 bit più significativi rappresentano il "domain ID", utilizzato per lo scambio di pacchetti tra una scheda e l'altra, mentre gli 8 bit meno significativi rappresentano il "device ID", utilizzato per lo scambio di pacchetti tra dispositivi presenti sulla stessa scheda.

Il firmware di BOOT si occupa di programmare la tabella di routing "device", permettendo così ai pacchetti destinati verso la scheda di essere instradati ciascuno al dispositivo giusto, in quanto la presenza e la connessione dei DSP e della FPGA alle porte dello switch sono fisse e immutabili in ciascuna scheda

del sistema; viene inoltre impostato il dominio dello switch, corrispondente alla posizione della scheda nel backplane letta tramite alcuni resistori di pull-up.

Viceversa le tabelle di routing di DOMAIN saranno programmate da pc, uno switch alla volta, in quanto dipendenti dal numero di schede inserite nel sistema e dalla loro posizione.

La classe CSwitch definisce quindi una serie di funzioni di basso livello per avviare transazioni SRIO in grado di effettuare letture o scritture nei registri di sistema dello switch o per effettuare il set o il clear di un singolo bit di un registro; queste sono richiamate da funzioni di più alto livello che permettono la configurazione voluta dello switch andando a modificare la tabella di routing interna e l'abilitazione delle porte.

3.2.3 Caricamento della configurazione della FPGA e del firmware applicativo

Completata l'inizializzazione il dispositivo rimarrà in attesa di comandi da pc per le successive fasi di inizializzazione: in particolare il PC può inviare due tipi di comandi, uno relativo alla programmazione della FPGA e uno per avviare il caricamento del firmware applicativo e la sua esecuzione, inviando contemporaneamente il relativo binario al DSP tramite Serial Rapid IO.

Per la comunicazione viene sfruttata la memoria condivisa del DSP da 4 MB, utilizzando un header da 64 bit in cui viene specificato il tipo di operazione richiesta e la dimensione del payload inviato da PC.

La ricezione di nuovi dati e di un nuovo comando da PC vengono notificati al dispositivo tramite un Doorbell inviato tramite Serial Rapid IO che consente la sua uscita dalla fase di attesa e l'esecuzione del relativo comando.

Inizialmente viene ricevuto da pc il firmware relativo alla configurazione della FPGA; per le stesse ragioni di flessibilità ed espandibilità del sistema enunciate in precedenza il firmware della FPGA non è situato su una memoria installata nella scheda ma viene caricato all'avvio del sistema.
Prima della fase di attesa la FPGA era stata preparata a ricevere la nuova configurazione inviando un segnale di reset (nConfig) su una linea GPIO.

Alla ricezione del comando di caricamento il DSP inizia a trasmettere il codice ricevuto, utilizzando l'interfaccia SPI precedentemente inizializzata, verso la FPGA.

Dopo il boot della FPGA da pc viene ricevuto il firmware applicativo del DSP; per il caricamento e l'avvio di questo viene utilizzata una apposita funzione

assembly situata al termine della memoria L2 del Core, qui posizionata per evitare la sovrascrittura durante la copia del programma.

Tale funzione si occupa di copiare il firmware ricevuto nella memoria condivisa verso l'indirizzo di memoria 0x00800000 (Inizio memoria L2 del core) andando così a sovrascrivere il firmware di boot.

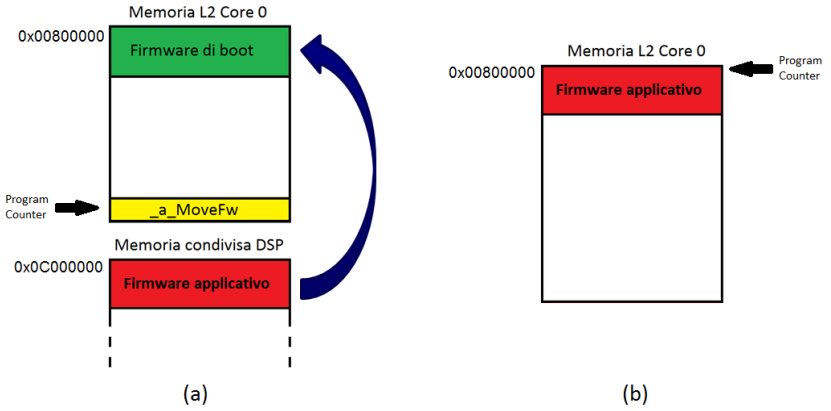

Figura 2.2 - Rappresentazione della procedura di caricamento del firmware applicativo nella memoria L2, mostrata prima (a) e dopo (b) l'esecuzione della funzione assembly

Terminata la copia vengono invalidate la cache programma e la cache dati e il fetching delle istruzioni viene fatto saltare all'indirizzo 0x00800000, iniziando così l'esecuzione del firmware applicativo.

3.3 Il firmware applicativo

Il firmware applicativo è quello principale per le operazioni del sistema ULA-OP in quanto si occupa di comunicare con il PC e con gli altri dispositivi del sistema, ricevendo comandi o trasmettendo dati, e di gestire la parte di acquisizione ed elaborazione durante le indagini ecografiche, generando i segnali necessari per stimolare gli elementi della sonda e elaborando opportunamente gli echi ricevuti in base al tipo di analisi richiesto.

In questo firmware viene svolta un'ulteriore fase di inizializzazione e di definizione di classi per la gestione dell'hardware, vengono istanziati e inizializzati i vari moduli e i vari task interni e viene infine avviato il kernel che si occupa della gestione dei task e del meccanismo di context switch che consente al processore di passare dall'esecuzione di un task all'altro.

Nei paragrafi che seguono sarà descritto il funzionamento del firmware, in particolare le classi e le funzioni che implementano il kernel e i task ed il meccanismo del cambio di contesto, descrivendo il comportamento di due task legati all'hardware del sistema, in particolare ai timer hardware e all'interfaccia Serial Rapid IO.

Saranno descritti anche i meccanismi che consentono la sincronia e lo scambio dati tra i task, ossia gli eventi e i canali di comunicazione, e altre classi necessarie per la gestione della memoria e delle risorse condivise in un contesto multicore.

3.3.1 Inizializzazioni per l'applicazione

Le prime operazioni svolte dal firmware applicativo sono le restanti inizializzazioni necessarie al DSP e la definizione di ulteriori classi necessarie per sfruttare l'hardware del DSP:

- DMA
- Interrupt
- Timer hardware

Queste inizializzazioni sono state escluse dal firmware di boot per due motivi: la prima è per non appesantirlo eccessivamente, visto che queste inizializzazioni riguardano funzionalità non utilizzate dal boot, la seconda è che queste inizializzazioni possono dover essere modificate col proseguire dello sviluppo del firmware e l'aggiunta di nuovi moduli (si pensi alla necessità di abilitare un nuovo interrupt) per cui è preferibile lasciarle nel firmware applicativo come detto facilmente modificabile.

3.3.1.1 DMA

E' stata creata una classe CDma per sfruttare il modulo DMA all'interno del DSP, definendo funzioni per preparare canali per il trasferimento dati e per avviarli e per sfruttare le funzionalità di tale modulo, quale il linking, che permette di caricare un canale al termine di un trasferimento con i parametri per un altro trasferimento, e il chaining che permette di utilizzare l'evento di termine trasferimento per avviarne un altro.

3.3.1.2 Interrupt

La classe CInterrupt si occupa di gestire il Chip Interrupt Controller (un multiplexer di interrupt che instrada gli eventi di sistema verso il core) gli interrupt del core e gli interrupt intercore.

Viene inizializzata la Interrupt Service Table e vengono inizializzati e instradati gli interrupt base ossia relativi ai task sempre attivi nel sistema quali:

- Interrupt intercore.
- Eventi timer
- Eventi completamento trasferimenti DMA
- Eventi completamento transazioni SRIO
- Ricezione doorbell

Sono poi fornite le funzioni per manipolare il CIC, abilitando e instradando gli interrupt verso il core desiderato, per gestire gli interrupt di sistema, scegliendo quale abilitare e a quale interrupt hardware associare un evento, e per il trigger degli interrupt intercore.

3.3.1.3 Timer hardware

Il DSP dispone di 8 timer dedicati ciascuno a un core e 8 timer condivisi. La classe CTimer si occupa di inizializzare un timer condiviso che sarà utilizzato come riferimento da tutti i core e relativi task.

Il modulo clock dei DSP riceve un timer di 166,66 MHz ed essendo il timer impostato come contatore a 32 bit è possibile impostare un timer con periodo fino a 25 secondi.

La classe CTimer offre poi le funzioni per manipolare i timer locali di ciascun core, consentendo il loro avvio con un intervallo voluto, il loro arresto e il controllo del tempo rimanente.

3.3.2 Kernel, Eventi e Task

L'obiettivo del firmware applicativo è quello di realizzare una struttura simile al kernel di un sistema operativo dove una serie di task si contendono l'uso del processore, ottenendone il controllo a turno in base alle proprie esigenze e rilasciandolo una volta terminati i propri compiti; il concetto alla base del sistema è il permettere l'esecuzione ai soli task che hanno del codice da eseguire,

lasciando in attesa quelli che devono attendere nuovi comandi o nuovi dati da processare.

La struttura del firmware di ULA-OP si sposa bene con questa architettura in quanto sono necessari diversi task, ossia parti di codice associate delle fasi di gestione hardware ed elaborazione, e ciascuno dei quali ha bisogno di essere eseguito per poco tempo, rimanendo in attesa di dati o richieste dei task a monte e a valle della catena di elaborazione per la maggior parte del tempo.

Per la loro natura infatti i task che si occupano del funzionamento del sistema non hanno bisogno di essere permanentemente in esecuzione: la maggior parte di essi deve essere richiamata in esecuzione solo quando riceve dei dati da processare o dei comandi da eseguire.

Per queste motivazioni per il firmware applicativo è stato realizzato un sistema composto da un "micro kernel" e da tanti task in cui sono implementate le operazioni elementari del DSP.

Il kernel è una classe che opera da gestore dei task inserendo in una lista quelli pronti per l'esecuzione e prelevandoli sequenzialmente per mandarli in esecuzione appena la CPU è libera.

La coda è gestita secondo una logica di tipo FIFO e vengono inseriti solamente i task che hanno codice da eseguire e richiedono l'uso della CPU

Lo scheduling dei task viene effettuato utilizzando una strategia non-preemptive, ossia questi hanno tutti la medesima priorità e non possono venire interrotti dal kernel durante la loro esecuzione; questa soluzione consente di ottenere un codice più semplice in quanto non è necessario uno scheduler che gestisca l'avvio o l'interruzione dei singoli task.

Come meccanismo di sincronia sono infine stati definiti dei particolari oggetti, gli "eventi" che servono per notificare appunto particolari eventi che possono accadere durante l'esecuzione, quali l'arrivo di un interrupt o lo svuotamento di un buffer.

I task basano la loro esecuzione sugli eventi, infatti questi sono utilizzati per capire quando sono verificate le condizioni che i task stanno attendendo per poter riprendere la loro esecuzione.

Gli eventi regolano l'ingresso dei task nella coda e la loro interruzione: all'arrivo di un evento il kernel valuta se un task lo stava attendendo e in caso positivo lo inserisce nella FIFO, ma allo stesso modo un task in esecuzione nella CPU può decidere di autosospendersi perché per proseguire nella sua esecuzione deve attendere un particolare evento.

Nel precedente sistema ULA-OP 64 il firmware era basato su un main loop in cui una serie di processi erano mandati in esecuzione ciclicamente e all'infinito,

dove i processi erano operazioni di varia complessità relative alle varie fasi di acquisizione/elaborazione che il DSP doveva eseguire.

Ad ogni iterazione del loop i vari processi erano interrogati secondo un meccanismo di polling, ossia venivano richiamati dalla CPU e venivano controllati dei flag per verificare se tale processo doveva essere mandato in esecuzione per eseguire il suo compito; tali flag di controllo venivano settati da altri processi e servivano per notificare ad esempio la presenza di dati da elaborare o l'arrivo di nuove istruzioni.

Completata l'esecuzione del processo questo tornava inattivo, in attesa di essere eventualmente svegliato alla successiva iterazione del ciclo.

Con questo meccanismo ogni processo era richiamato e il suo set di flag controllato ad ogni iterazione, indipendentemente dal fatto che la sua esecuzione fosse necessaria o meno; è evidente come questo meccanismo si presenti poco efficace nei momenti in cui siano attivi solo pochi processi alla volta poiché si vanno a risvegliare e controllare anche quelli attualmente non operativi.

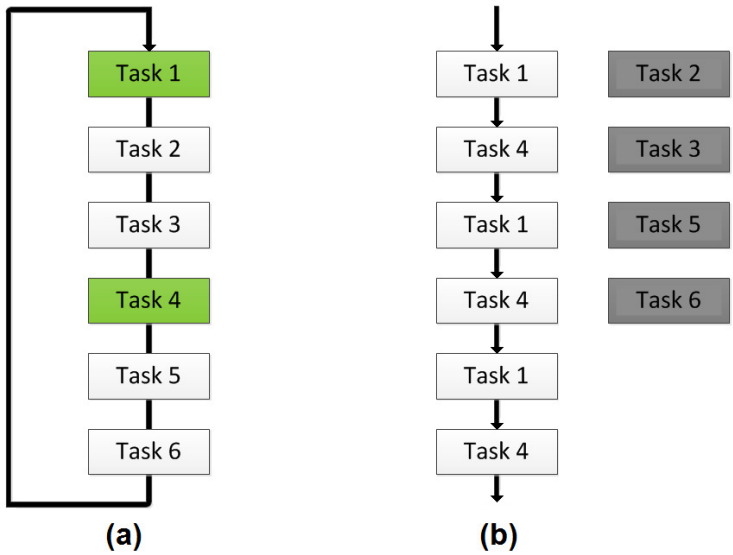

Figura 3.3 - Confronto tra strategia di gestione a polling (a) e con micro kernel (b) nel caso di un sistema con 6 task nel caso in cui solo due devono essere eseguiti

La nuova struttura del firmware consente di rimediare a questo problema: invece dei processi si considerano solamente i singoli task che rappresentano operazioni o compiti più elementari e questi task vengono risvegliati e eseguiti

dalla CPU solamente nel momento in cui hanno effettivamente bisogno di essere richiamati e eseguiti, rimanendo inattivi nel resto del tempo.

Si eliminano cosi i tempi morti della CPU in cui venivano interrogati i processi non necessari a fronte di una piccola spesa in termini di cicli macchina nel passaggio da un task all'altro.

E' evidente come, contrariamente al main loop e alla gestione a polling dei processi, in questo caso si ha il difetto opposto, ossia una perdita di efficienza nei momenti in cui si hanno un gran numero di task in esecuzione contemporaneamente all'interno della CPU, ma poiché questa situazione è meno frequente tale debolezza rimane trascurabile.

Un ulteriore vantaggio sta in una maggiore pulizia del codice e nella maggiore facilità di inserire nuovi task nel firmware.

Nel precedente sistema il main loop era costituito da una serie di chiamate ai vari processi e ed aggiungere un nuovo processo consisteva nell'aggiungere la chiamata alle sue funzioni di inizializzazione e di esecuzione.

Nel nuovo firmware aggiungere un nuovo task comporta solo l'aggiunta dei relativi file .h e .cpp nella cartella del progetto e nel dichiarare il suo #include nei task che devono comunicare con lui e usare le sue funzioni di interfaccia.

Ogni task è poi scritto e visto dalla CPU come se fosse l'unico in esecuzione, in quanto presenta un main loop infinito da cui si può uscire solamente tramite cambio di contesto, rendendo a livello di codice più leggibile e chiara la sua funzione, in quanto gli altri task sono raggiunti solo tramite la notifica di eventi o le loro funzioni di interfaccia.

3.3.2.1 Eventi

Gli eventi rappresentano il meccanismo che permette ai task di essere mandati in esecuzione e come suggerisce il nome rappresentano simbolicamente i vari eventi che possono verificarsi durante l'esecuzione del firmware, quali il termine di un trasferimento DMA o la notifica di passaggio dati da un task ad un altro.

La classe CEvent contiene un flag, che indica se l'evento è scattato o meno, e un puntatore al task in attesa di tale evento; è bene precisare che per ragioni di semplicità ogni evento è sempre associato a un unico task che può richiederne l'attesa.

Le funzioni di interfaccia sono:

- *IsSet()* che verifica se il flag è settato e quindi l'evento si è verificato.
- *Reset()* che rimuove la connessione task-evento e resetta il flag.

- *Detach()* che rimuove solamente la connessione task-evento
- *Set()* che setta il flag e nel caso in cui vi sia un task in attesa di tale evento chiama la funzione per aggiungerlo alla coda di esecuzione.
- *Wait()* che associa un task all'evento e nel caso in cui questo sia già scattato chiama immediatamente la funzione per aggiungere il task alla coda di esecuzione

3.3.2.2 Task

Ogni task è derivato da una classe base contenente informazioni come il suo stato attuale ed il puntatore allo stack associato ed una funziona virtuale pura *Run()* che rappresenta il cuore del task e contiene le istruzioni da eseguire ad ogni iterazione.

Lo stato del task può assumere 4 valori:

- CREATED: il task è stato creato, il suo stack allocato ma ancora non preparato e ancora non è stato inserito nella coda dei task. A run time un task in questo stato rappresenta una condizione di errore poiché indica che non è stato avviato correttamente.
- SUSPENDED: il task è attualmente sospeso in attesa di ricevere un evento che lo risvegli.
- RUNNING il task è attualmente attivo e in esecuzione nel core.
- QUEUED il task è pronto per la prossima esecuzione e inserito nella coda di esecuzione. Questo stato indica che un task è già schedulato per la prossima esecuzione e evita che sia nuovamente reinserito in coda.

La Funzione *Run()* viene ridefinita da ogni task derivato dalla classe base e tipicamente contiene una parte di inizializzazione e un main loop principale che, similmente a quanto avveniva con il firmware di ULA-OP 64, ripete continuamente le stesse istruzioni; possono comunque essere realizzati task che richiedono un numero limitato di esecuzioni prima di essere sospesi e che quindi non necessitano di un loop infinito.

All'interno del main loop si possono trovare oltre alle funzioni tipiche del task delle funzioni di attesa evento, per sospendere il task e lasciare la CPU libera ad altri, e delle funzioni di notifica per risvegliare altri task.

Ogni task durante il normale funzionamento esegue il suo codice fino ad arrivare alla funzione di attesa evento, che può ad esempio rappresentare l'attesa che un altro task che effettua un passo precedente dell'elaborazione

fornisca dei dati o che un buffer sia libero; tramite context switch quindi il kernel sospende il task e passa al prossimo nella lista di attesa.

Una volta che l'evento si verifica il kernel provvederà a inserire nuovamente il task in coda di esecuzione.

3.3.2.3 Kernel

Il kernel è il cuore del firmware in quanto si occupa di gestire l'inizializzazione dei task e la loro prima esecuzione, di gestire l'attesa degli eventi e il cambio di contesto e di gestire la coda dei task pronti per l'esecuzione.

Le funzioni della classe kernel sono:

- *CreateStack():* chiamata dal costruttore dei task serve per allocare uno spazio di memoria che sarà utilizzato come stack dal task stesso.
- *PrepareTask():*chiamata anch'essa dal costruttore inserisce il task in coda per la sua prima esecuzione e chiama la funzione assembler *_a_CtxPrepare* che riserva nello stack del task lo spazio per i registri da salvare durante il context switch e pone come indirizzo di prima esecuzione quello della funzione *Execute()* che avvia il task.
- *Execute():*viene chiamata solo durante la prima esecuzione del task e serve per salvare il puntatore allo stack del kernel e chiamare la funzione Run() che avvia il task stesso. E' presente una chiamata alla funzione di context switch nel caso in cui il task non debba essere eseguito indeterminatamente e possa uscire dal suo Run().
- *AddToQueue():*serve per aggiungere un task pronto per l'esecuzione nella FIFO gestita dal kernel, a run time viene chiamata ogni volta che si verifica un evento atteso da un task, in modo da inserirlo in coda e farlo eseguire nuovamente. Tale funzione rappresenta una sezione critica in cui gli interrupt devono essere disattivati, poiché può essere chiamata contemporaneamente da un task e da una funzione interrupt, cosa che potrebbe portare ad una perdita di coerenza tra i puntatori di testa e coda compromettendo l'esecuzione del firmware.
- Reschedule():questa funzione viene chiamata da un task per chiedere al kernel di sospenderlo e reinserirlo immediatamente nella coda di esecuzione. Può essere utilizzata da task particolarmente pesanti per interrompere l'esecuzione e restituire il controllo al kernel in modo da non utilizzare la CPU per troppo tempo.

- KillTask():serve per terminare l'esecuzione di un task, che può essere sia il chiamante della funzione oppure un altro tra quelli presenti.
- WaitForEvent():è la funzione che viene chiamata per attendere il verificarsi di un evento (esiste l'override per attendere eventi multipli). Sono possibili due tipi di comportamento:
  Attesa normale: viene notificato all'evento che un task lo sta attendendo tramite la sua funzione *Wait()* e viene poi sospeso il task corrente effettuando un context switch; questa procedura, nel caso in cui l'evento sia già avvenuto, pone immediatamente il task in coda alla FIFO di attesa.
  Attesa rapida: viene preventivamente controllato se l'evento da attendere è già avvenuto e in questo caso il task non viene sospeso ma prosegue con la sua normale esecuzione.
  In breve l'attesa rapida sospende il task solo se il suo evento non è ancora avvenuto, l'attesa normale lo sospende in ogni caso
- *Start():*avvia il main loop del kernel; questo è un ciclo infinito che ha il compito di prelevare i task dalla testa della coda di attesa e effettuare un context switch verso di loro, riprendendo l'esecuzione appena tale task si pone in attesa di un evento.
  Nel caso in cui il kernel arrivi alla condizione per cui la coda dei task è vuota, ossia non vi sono task da mandare in esecuzione, questo entra in modalità IDLE – LOW POWER al fine di ridurre i consumi; la CPU esce automaticamente da questa modalità non appena viene ricevuto un interrupt, tipicamente associato a un evento, e riprende la valutazione della coda

3.3.3 Il meccanismo di cambio di contesto

Come detto in precedenza ogni task è costituito da un main loop e la CPU lo vede come se fosse l'unica parte di codice disponibile. Per passare da un task all'altro sono state quindi realizzate due funzioni assembler, la prima di preparazione chiamata durante il setup dei task e la seconda di esecuzione, che permettono di passare da un contesto all'altro ossia dall'esecuzione di un task all'altro nel caso in cui il task sia messo in attesa di un evento o decida di autosospendersi.

Come contesto si intende lo stato dei registri della CPU così come si trovano al momento in cui il task viene interrotto; in particolare si considerano i registri da **A10** a **A15**, da **B10** a **B13**, in quanto questi sono i registri "children" ossia associati alla funzione chiamata (come il *Run()* del task), il Data pointer (registro **B14**) e l'indirizzo di ritorno (registro **B3**); per questa operazione viene inoltre utilizzato il puntatore allo stack (registro **B15**).

Il cambio di contesto consiste nel salvataggio di questi registri nello stack del task sospeso e nella lettura dei medesimi registri dallo stack del task da riattivare, facendo riprendere l'esecuzione di quest'ultimo dal punto in cui si era interrotto grazie all'indirizzo di ritorno precedentemente salvato.

La funzione *_a_CtxPrepare* prepara lo stack del task per la prima esecuzione.

Il cambio di contesto si basa sul salvataggio e la lettura di 6 Double Word in cui viene memorizzato lo stato dei registri interni del core; poiché al primo avvio dei task il loro stack è vuoto effettuando il primo context switch si verificherebbe un underflow che comprometterebbe il funzionamento del firmware, per cui questa funzione si occupa di effettuare 6 STORE di double word in modo da portare lo stack pointer alla posizione corretta.

L'ultima scrittura inoltre memorizza nello stack il Data Pointer del task e l'indirizzo della funzione *Execute()* come indirizzo di ritorno, in modo che questa sia lanciata in modo automatico dopo il context switch permettendo il primo avvio del task.

La funzione assembly *_a_CtxSwitch* si occupa del cambio di contesto ossia di passare dall'esecuzione di un task all'altro ripristinando lo stato dei registri di sistema e riprendendo l'esecuzione dal punto in cui si era interrotta.

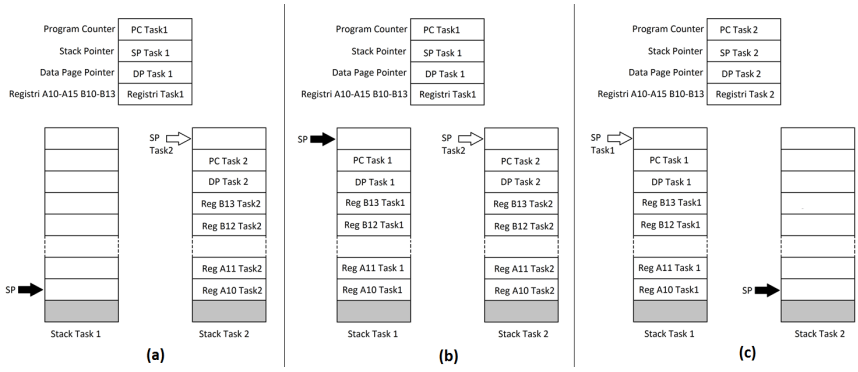

Figura 3.4 - Stato dello stack e dei registri durante il context switch. (a) rappresenta la situazione prima del cambio di contesto, (b) dopo il salvataggio del contesto attuale e (c) dopo che è avvenuto il cambio di contesto e il nuovo task è in esecuzione

La prima fase è il salvataggio del contesto attuale che consiste in una serie di STORE, per salvare nello stack del task in esecuzione lo stato dei registri interni, il Data Pointer e l'indirizzo di ritorno da cui riprendere l'esecuzione.

Viene poi memorizzato in una funzione lo stack pointer che sarà restituito dalla funzione al termine delle sue operazioni.

La seconda fase è il caricamento del nuovo contesto: la funzione riceve come parametro il puntatore allo stack del nuovo task da caricare che viene salvato nell'apposito registro e questo viene utilizzato per effettuare una serie di LOAD per leggere dallo stack l'indirizzo di ritorno e il Data Pointer e i registri precedentemente salvati nello stack.

E' importante puntualizzare come il cambio di contesto non avvenga direttamente da un task al successivo nella coda di esecuzione, ma venga sempre effettuato un passaggio intermedio verso il kernel, visto a tutti gli effetti come se fosse un task, il quale si occupa di prelevare dalla coda il task successivo e effettuare un context switch verso di esso.

Questa soluzione, per quanto più onerosa in termini di tempo computazionale in quanto richiede un context switch supplementare rispetto al salto diretto da un task all'altro, è preferibile perché offre vantaggi in termini di spazio utilizzato, in quanto effettuare il salto diretto avrebbe richiesto a ogni task l'impiego di una maggiore quantità di stack per memorizzare anche il contesto del kernel.

3.3.4 Il gestore della memoria

Nel DSP sono utilizzabili tre tipologie di memorie:

- Memoria L2 interna a ciascun core da 512 kB ciascuno.
- Memoria condivisa tra i vari core, da 4 MB
- Memoria esterna DDR3 da 4GB

Per poter gestire in maniera ottimale queste memorie, permettendo ai task e alle classi di allocare dinamicamente dei buffer di memoria, verificando la disponibilità di spazio e effettuando l'allocazione evitando la frammentazione della memoria stessa, è stata creata una classe definita CMemory; questa viene istanziata localmente nella L2 di ciascun core e singolarmente nella memoria condivisa e DDR3 in modo sia unica tra tutti i core.

CMemory si basa su un buffer allocato nella memoria controllata dal gestore, che rappresenta il totale disponibile e allocabile, e mantiene come variabili interne il puntatore al buffer, lo spazio disponibile e lo spazio attualmente

utilizzato; allocando i gestori di memoria condivisa e DDR3 direttamente in quella memoria ogni core accede alla medesima istanza che mantiene quindi lo stato di utilizzo della memoria sincronizzato tra di loro.

Come interfaccia offre una funzione *Alloc* con cui il richiedente specifica la quantità di spazio necessaria e, se disponibile, il gestore la riserva nel buffer restituendo il puntatore. Queste richieste sono allocate sequenzialmente fino all'esaurimento dello spazio disponibile.

La disallocazione invece viene fatta tramite una funzione *Rewind* che disalloca tutti i buffer attualmente creati e ripristina lo stato iniziale delle variabili.

Questa scelta è stata fatta per ragioni di efficienza e semplicità, in quanto disallocare un singolo buffer avrebbe lasciato dei buchi nello spazio di allocazione provocandone la frammentazione e sarebbe stato necessario un gestore più complesso per tenerne traccia e ottimizzare le allocazioni successive.

La funzione di allocazione implementa un semaforo utilizzato per bloccare l'accesso durante l'allocazione, per evitare conflitti tra i core nel caso di allocazione di memorie condivise.

3.3.5 Il gestore delle risorse condivise

I moduli hardware del DSP mettono a disposizione risorse in numero limitato che devono essere suddivise tra i vari utilizzatori all'interno dei vari core; queste risorse comprendono ad esempio i Doorbell e i canali DMA.

Per la loro gestione sono state realizzate delle classi che, protette da un semaforo per evitare conflitti da accessi condivisi, si occupano di fornire al richiedente la prima risorsa richiesta disponibile, ad esempio il numero del canale DMA da utilizzare.

Lo stato attuale del numero di risorse attualmente utilizzato viene memorizzato in una struttura all'interno della memoria condivisa.

3.3.6 Il task per la gestione dei timer

Questo task si occupa di istanziare e gestire multipli timer software basandosi su due timer hardware per il conteggio.

Il suo funzionamento è basato su oggetti di tipo CSingleTimer che rappresentano i timer software; questa classe memorizza il numero di colpi di

clock rimanenti per far scattare il timer e il valore a cui ricaricarlo dopo la scadenza, oltre a un evento di attesa ed offre funzioni di interfaccia per impostare il periodo, verificare se è già scattato, decrementare il conteggio, facendo scattare l'evento associato se questo raggiunge lo 0, o rimanere in attesa fino alla prossima scadenza.

Il task offre funzioni di interfaccia per registrare o cancellare questi timer, ossia aggiungerli o rimuoverli dalla lista di quelli attualmente attivi; i soft timer sono organizzati sfruttando una lista doppiamente concatenata in modo da rendere più semplici le operazioni di rimozione di essi.

I timer registrati sono quelli che vengono aggiornati periodicamente dal task e di cui viene controllata la scadenza.

I soft timer registrati vengono aggiornati tutti contemporaneamente a ogni iterazione del task, decrementandone i contatori interni e facendo così eventualmente scattare gli eventi dei timer che hanno esaurito il conteggio; il task è sincronizzato e attivato da un proprio evento basato sul timer hardware dedicato del core la cui scadenza periodica permette una nuova iterazione del task.

Per non sovraccaricare la CPU con continue richieste di context switch a discapito dell'efficienza è stato scelto di non impostare il timer hardware in modo da avere scadenze a intervalli fissi, ad esempio ogni millisecondo, ma di impostarlo nuovamente ad ogni iterazione del task in base alla scadenza del timer software più vicina; in questo modo si ha il massimo dell'efficienza in quanto il task viene risvegliato effettivamente solo quando deve gestire almeno una scadenza.

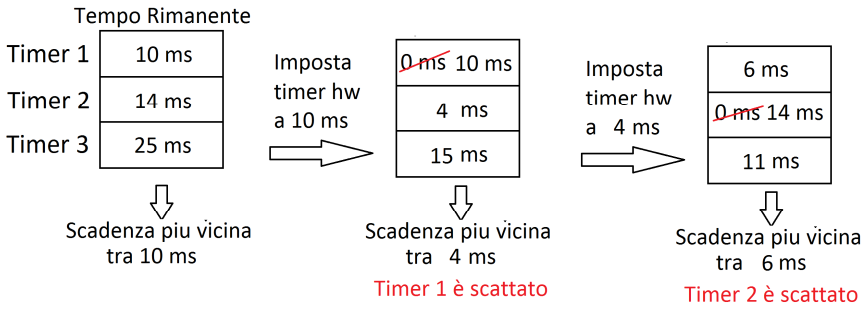

Figura 3.5 - Esempio di iterazioni successive del task dei timer, con tre timer software registrati

Il main loop del task consiste quindi nella scansione della lista dei timer registrati, decrementando ciascuno dell'intervallo trascorso dall'ultima iterazione

del task, memorizzato come variabile membro della classe, facendo così scattare quelli giunti al termine del conteggio.

Durante la scansione della lista viene inoltre per ogni timer letto il tempo rimanente alla prossima scadenza ricercando il valore minimo; questi rappresenta l'intervallo rispetto alla scadenza timer software più vicina e viene quindi memorizzato per il decremento durante la prossima iterazione del task e utilizzato per impostare la prossima scadenza del timer hardware che dovrà risvegliare il task.

Per ottimizzare le prestazioni dei timer in realtà, all'intervallo del timer hardware viene prima effettuata una correzione.

Poiché per il meccanismo di context switch e della variabilità di carico di lavoro vi può essere un ritardo tra il trigger dell'interrupt e l'effettiva esecuzione del task, il ciclo di aggiornamento dei timer software avviene regolarmente in ritardo rispetto al momento previsto e si ha nel lungo termine una deriva del comportamento dei timer rispetto al comportamento ideale; per correggere questo problema è stato implementato un meccanismo di correzione che si basa su un ulteriore timer hardware.

Come espresso in precedenza durante il setup iniziale del firmware applicativo viene impostato un timer general pourpose comune a tutti i core; questo opera come contatore hardware e viene utilizzato come parametro di riferimento confrontandolo con un contatore software mantenuto dal task stesso.

Ad ogni ciclo infatti il contatore software viene incrementato dell'intervallo trascorso rispetto all'ultima iterazione del task e il valore ottenuto viene confrontato con il valore del timer hardware; la differenza tra i due rappresenta il ritardo del task rispetto al caso ideale e questo delta viene quindi sottratto dall'intervallo di attivazione del timer hardware, anticipando la prossima scadenza per compensare volta per volta il ritardo.

3.3.7 Il task per la gestione dell'interfaccia SRIO

Questo task si occupa di gestire le richieste di accesso alla periferica SRIO, avviando le transazioni e notificando ai Task che le hanno richieste quando queste son terminate, segnalando il rispettivo evento.

Per funzionare sfrutta oggetti di una classe di tipo CSrioTicket che rappresentano le richieste di transazioni effettuate dagli altri task; gli oggetti di questa classe oltre ai parametri della transazione, quali indirizzi di partenza e destinazione, identificativo del dispositivo ricevente e bytecount, integrano un

evento che indica se il trasferimento è terminato e offrono funzioni di interfaccia per impostare la transazione desiderata.

Il funzionamento del task è basato su due code, una di attesa, contenente le richieste di nuove transazioni fatte dagli altri task e una attiva, limitata a 4 elementi, contenente le transazioni attualmente avviate e in attesa di completamento; il limite a 4 è dovuto al fatto che ogni core ha a disposizione solo 4 shadow register per l'accesso alla periferica SRIO.

Per le caratteristiche hardware del DSP e della periferica SRIO non è possibile monitorare il completamento delle singole transazioni ma solo ricevere un generico interrupt di "transazione completata" senza informazione su quale effettivamente sia stata portata a termine; questo comporta che avviando transazioni multiple non c'è modo di ricevere singolarmente notifiche di completamento per ciascuna di esse rendendo impossibile associare all'interrupt uno specifico evento da settare.

Per questo motivo il task è stato strutturato in modo da avviare fino a 4 transazioni alla volta, in così da sfruttare tutti e 4 gli shadow register, ricevendo un unico interrupt di completamento associato all'ultima transazione del gruppo, che identifica che sono state tutte eseguite e gli eventi di ciascuna di esse possono essere settati per avvisare i rispettivi task del completamento.

Questo task è normalmente inattivo e viene risvegliato da un unico evento che può identificare due situazioni: l'arrivo di nuove richieste di transazione o il termine di transazioni già avviate.

Questo evento può essere settato o dalla funzione di interfaccia StartTransaction che riceve la transazione, ossia l'oggetto CSrioTicket, da avviare e la aggiunge alla coda di attesa, o dall'interrupt di completamento transazione.

Per evitare inserimenti multipli del task in coda StartTransaction verifica che la coda attiva sia vuota, poiché in caso contrario è previsto un interrupt che effettuerà il set dell'evento e riattiverà il task.

Il main loop consiste per prima cosa nel controllo della coda attiva: se questa non è vuota e il task è stato risvegliato significa che le transazioni attive sono terminate e questa viene quindi svuotata prelevando i singoli ticket e settando i relativi eventi.

Successivamente viene controllata la coda di attesa e se non è vuota vengono prelevati fino a 4 ticket per riempire la coda attiva, avviando ogni volta la relativa transazione e abilitando solo per l'ultima l'interrupt di completamento.

Questo task si occupa anche della gestione dei doorbell, offrendo il relativo gestore in quanto questi sono una risorsa condivisa tra i vari core e fornendo una funzione di interfaccia che permette di associare ad ognuno di essi uno

specifico un evento; la relativa routine di interrupt si occuperà di controllare i registri di stato per identificare quale o quali doorbell sono stati ricevuti andando a settare i relativi eventi.

3.3.8 Il gestore dei canali di comunicazione

Un task può avere la necessità di effettuare uno scambio dati con un altro task che può trovarsi in esecuzione all'interno dello stesso core, su un core diverso nello stesso dsp oppure su un diverso DSP all'interno del sistema; ovviamente nei 3 casi saranno necessari meccanismi di comunicazione diversi ossia rispettivamente un buffer condiviso all'interno del core, un trasferimento DMA e un trasferimento SRIO.

Per raggiungere il massimo livello di astrazione e offrire un'interfaccia generica, in cui si richiede un generico canale di comunicazione tra due task, indipendentemente dal core e DSP su cui sono in esecuzione, sono stati sviluppate due classi COutChannel e CInChannel e relative classi derivate per ogni tipo di transazione; in particolare COutChannel è il gestore di uscita, ossia associato al task che vuole inviare dati e CInChannel il gestore di ingresso, utilizzato dal task che vuole riceverli.

I canali per poter essere inizializzati correttamente e funzionare hanno bisogno di dei parametri di inizializzazione, che indicano dove si trovano i task che devono comunicare e quindi il tipo di canale necessario, e di un sistema che permetta ai due gestori di canali di comunicare scambiandosi le informazioni necessarie per il loro funzionamento.

3.3.8.1 Slot di comunicazione

Il meccanismo che permette ai due gestori di comunicare tra di loro è stato definito come "Comunicazione tramite slot".

In ogni core viene istanziato un array di interi a 64 bit ed ogni elemento del vettore rappresenta uno slot che può essere utilizzato come una sorta di mailbox per ricevere dati dagli altri gestori dei canali; ogni gestore legge quindi la tabella slot locale, ossia istanziata nel suo stesso core, e scrive nelle tabelle remote, ossia istanziate nel core in cui si trova l'altro gestore del canale con cui si è instaurata la comunicazione.

Le informazioni inserite nei 64 bit dei vari slot non sono fisse ma dipendono dal tipo di canale a cui sono associati.

Ogni canale, di ingresso o uscita, per comunicare con il suo omologo sfrutta queste tabelle, andando a scrivere nello slot dell'altro e leggendo dal proprio; durante la lettura dello slot locale è sufficiente utilizzare il campo "slot" presente nell'identificatore del canale come indice nel vettore, durante la scrittura invece, poiché ogni core dispone del medesimo firmware il puntatore alla base di questo vettore è lo stesso in tutti i core e l'identificatore del canale permette di risalire al DSP e core in cui la tabella di destinazione si trova, permettendo di utilizzare il campo "slot" come indice per scrivere in questa.

3.3.8.2 Identificatore di canale

Gli identificatori di canale sono delle variabili a 64 bit utilizzate in fase di inizializzazione per istanziare il tipo di canale corretto, in base al DSP e al core in cui si trovano i task che devono comunicare.

Di questi identificatori, 32 bit sono dedicati all' OutChannel e 32 all' InChannel; ogni gestore, indipendentemente che sia di ingresso o uscita, legge i 32 bit che lo riguardano per istanziare il tipo di canale corretto e sfrutta i restanti 32 bit come parametri per la comunicazione

Out Channel info			In Channel Info		
SRIO ID	Core	Slot	SRIO ID	Core	Slot
16b	8b	8b	16b	8b	8b

Tabella 3.2 - Campi contenuti nell'identificatore del canale a 64 bit

- SRIO ID è l'indirizzo Serial Rapid IO che serve per identificare un qualsiasi DSP all'interno della scheda.
- Core indica su quale degli 8 core del sistema è in esecuzione il Task che richiede il canale.
- Slot rappresenta l'indice nel vettore degli slot per indicare quello associato al gestore del canale.

3.3.8.3 Creazione del canale

Per ottenere il massimo della flessibilità viene sfruttato il polimorfismo tra classi C++; le classi COutChannel e CInChannel, definite classi madri, presentano ciascuna tre classi derivate ognuna relativa ad una diversa tipologia di canale di comunicazione e con le medesime funzioni di interfaccia.

Il polimorfismo consente di utilizzare nel codice solamente le classi madri e le funzioni di interfaccia che queste offrono, le cui implementazioni dipenderanno dal tipo di oggetto effettivamente istanziato.

Il tipo di classe derivata da istanziare dipende dal canale che è necessario creare e questa informazione è contenuta nell'identificatore del canale.

Quando viene istanziato un gestore di un canale CInChannel o COutChannel il costruttore controlla i campi SRIO ID e Core che identificano dove sono in esecuzione i task che devono essere messi in comunicazione e in base a questi istanzia il gestore corretto.

L'identificatore del canale permette infatti di immediatamente di capire che tipo di comunicazione è richiesta:

- Se i due SRIO ID sono diversi i due task sono in esecuzione su due distinti DSP per cui è necessaria una comunicazione di tipo Serial Rapid IO.
- Se i campi SRIO ID sono uguali vengono controllati i campi Core e se questi sono diversi deve essere impostato un canale DMA.
- Se i controlli precedenti falliscono i due Task sono in esecuzione nello stesso core e quindi già condividono la stessa memoria.

Identificato il gestore necessario viene chiamato il relativo costruttore e il puntatore viene infine restituito al chiamante.

Ovviamente ogni costruttore ha bisogno di altri parametri che non dipendono dal tipo di canale ma sono relativi ai buffer di memoria da utilizzare per accogliere i dati da trasmettere o quelli ricevuti.

Per accedere ai due lati del canale vengono utilizzate delle funzioni di interfaccia, virtuali pure nelle classi madri e ridefinite in ogni implementazione in quanto sono necessari meccanismi diversi per lo scambio dati in base al tipo di canale.

I gestori di uscita utilizzano *Lock()* e *Commit()* che consentono rispettivamente di ottenere l'accesso al buffer di trasmissione per poterlo riempire e di rilasciare tale buffer attivando nel contempo il trasferimento dati.

I gestori di ingresso utilizzano *Get()* e *Release()* che consentono rispettivamente di ottenere l'accesso al buffer di ricezione per leggerne i contenuti e di rilasciarlo abilitandolo alla ricezione di nuovi dati.

All'interno di queste funzioni sono utilizzati meccanismi di sincronizzazione, dipendenti dal tipo di comunicazione usato, in modo da mantenere la coerenza dei dati, ossia evitare che OutChannel sovrascriva buffer non ancora letti e InChannel legga buffer che ancora non hanno ricevuto dati.

3.3.8.4 Comunicazione DSP-DSP

Le due classi CDsp2DspOutChannel e CDsp2DspInChannel consentono la comunicazione tra due DSP distinti sfruttando l'interfaccia Serial Rapid IO; in questo caso lo slot di comunicazione serve all'OutChannel per sapere a quale indirizzo di memoria mandare i dati e quale doorbell inviare come conferma trasmissione, mentre all'InChannel per sapere quale doorbell mandare come conferma di lettura.

La sincronizzazione degli accessi al buffer è ottenuta utilizzando due doorbell, uno inviato da OutChannel verso InChannel per avvisarlo che sono arrivati nuovi dati e un altro che procede in direzione opposta utilizzato come conferma che i dati ricevuti son stati utilizzati e il buffer di ricezione è pronto ad accoglierne di nuovi; lato trasmissione viene inoltre sfruttato l'interrupt di completamento trasferimento SRIO in modo da avere una notifica su quando il buffer di trasmissione è pronto a essere nuovamente utilizzato.

Il task che utilizza OutChannel ottiene l'accesso al buffer di trasmissione con la funzione *Lock()*. Nel caso in cui il buffer sia a singolo livello viene atteso il termine della transazione precedente per evitare di sovrascrivere i dati.

Una volta riempito il buffer il task chiama la funzione *Commit()* la quale deve attendere il doorbell di conferma lettura relativo alla transazione precedente nel caso in cui non sia ancora arrivato; questo doorbell è associato a una transazione con cui l'InChannel comunica il nuovo indirizzo a cui inviare i dati, necessario per buffer a più livelli, e il doorbell da utilizzare.

Aggiornati i parametri della transazione questa viene inserita nella coda delle transazioni in attesa del task Serial Rapid IO.

Il task che utilizza InChannel ottiene l'accesso al buffer di ricezione con la funzione *Get()*, che si occupa di attendere l'arrivo del doorbell di fine trasmissione prima di restituire un puntatore al buffer.

Al termine dell'utilizzo del buffer questo viene rilasciato dalla funzione *Release()* che provvede a inviare verso lo slot di OutChannel i parametri aggiornati per la prossima transazione oltre al doorbell di conferma lettura.

La sincronia e il mantenimento della coerenza dei dati è garantita dai Doorbell che i due gestori si scambiano in quanto OutChannel può ottenere l'accesso a una sezione del buffer di trasmissione ma non può avviare la transazione prima di aver ricevuto un Doorbell di conferma lettura, mentre InChannel non può ottenere l'accesso al buffer prima di aver ricevuto un Doorbell di conferma transazione.

3.3.8.5 Comunicazione Core-Core

Le due classi CCore2Core2OutChannel e CCore2Core2InChannel consentono la comunicazione tra due Core all'interno dello stesso DSP, sfruttando il modulo DMA; in questo caso lo slot di comunicazione serve solo in fase di inizializzazione in quanto serve per comunicare quale canale di comunicazione è stato scelto e quale interrupt intracore utilizza l'OutChannel come conferma di lettura dall' InChannel

La sincronizzazione degli accessi al buffer è ottenuta utilizzando due interrupt intercore, uno attivato automaticamente tramite un trasferimento DMA, per avvisare InChannel della ricezione di nuovi dati e l'altro attivato da quest'ultimo per avvisare OutChannel che i dati ricevuti son stati utilizzati e il buffer di ricezione è pronto ad accoglierne di nuovi.

L'attivazione automatica dell' interrupt intercore è necessaria per ovviare al limite che solo un core può gestire l'interrupt di completamento del trasferimento; è stato quindi scelto di lasciare la gestione di questo interrupt all'OutChannel in modo da avere una conferma che il buffer di trasmissione sia nuovamente utilizzabile, e di sfruttare le possibilità di linking e chaining offerte dal modulo DMA per caricare un nuovo set di parametri di trasferimento e attivarlo in automatico per trasferire il codice di attivazione interrupt nell'opportuno registro mappato in memoria, ricaricando poi il set relativo al trasferimento dati tra i due core.

Il task che utilizza OutChannel ottiene l'accesso al buffer di trasmissione con la funzione *Lock()*. Nel caso in cui il buffer sia a singolo livello viene atteso il termine del trasferimento DMA precedente per evitare di sovrascrivere i dati.

Una volta riempito il buffer viene chiamata la funzione *Commit()* la quale deve attendere l'interrupt di conferma lettura relativo alla transazione precedente nel caso in cui non sia ancora arrivato; questo interrupt notifica anche che InChannel ha aggiornato i parametri del trasferimento DMA con il nuovo indirizzo di destinazione, necessario per buffer a più livelli, OutChannel quindi provvede a aggiornare l'indirizzo sorgente e attiva il trasferimento DMA.

InChannel ottiene l'accesso al buffer di ricezione con la funzione *Get()*, che si occupa di attendere l'arrivo dell' interrupt intercore di fine trasmissione prima di restituire un puntatore al buffer.

Al termine dell'utilizzo del buffer questo viene rilasciato dalla funzione *Release()* che provvede a aggiornare i parametri di trasferimento con il nuovo indirizzo destinazione e a inviare l'interrupt intercore di conferma lettura.

La sincronia e il mantenimento della coerenza dei dati è garantita dagli interrupt intercore che i due gestori si scambiano in quanto OutChannel può

ottenere l'accesso a una sezione del buffer di trasmissione ma non può avviare la transazione prima di aver ricevuto un interrupt di conferma lettura, mentre InChannel non può ottenere l'accesso al buffer prima di aver ricevuto un interrupt di conferma transazione.

3.3.8.6 Comunicazione Task-Task

E' il tipo di comunicazione più semplice in quanto non è richiesto alcun trasferimento di dati poiché i due task condividono la medesima memoria all'interno dello stesso core; viene utilizzata una particolare classe CMultiBuffer che serve per l'accesso a buffer di memoria condivisi e le due classi CTask2TaskOutChannel e CTask2TaskInChannel servono solo come interfacce verso questa classe. Lo slot di comunicazione serve solo in fase di setup per comunicare il puntatore al multibuffer da utilizzare.

La sincronizzazione degli accessi al buffer è garantita dalla stessa classe CMultiBuffer che ha integrato un evento di attesa; questo viene sfruttato per mettere in attesa task che provano a scrivere nel buffer se questo è pieno o provano a leggere dati se è vuoto, venendo poi rispettivamente settato nel caso di una nuova lettura o una scrittura nel buffer.

Il task che utilizza OutChannel ottiene l'accesso al buffer di trasmissione con la funzione *Lock().* Nel caso in cui il buffer sia riempito in tutti i suoi livelli viene impostato l'evento di attesa sospendendo il task che ha richiesto la scrittura.

Una volta riempito il buffer viene chiamata la funzione *Commit()* la quale incrementa il contatore di sezioni riempite, imposta il puntatore alla sezione successiva e nel caso vi siano in ricezione task in attesa di nuovi dati setta l'evento di attesa per risvegliarli.

Il task che utilizza InChannel ottiene l'accesso al buffer di ricezione con la funzione *Get().* Nel caso in cui il buffer sia completamente vuoto viene impostato l'evento di attesa sospendendo il task che ha richiesto la scrittura.

Al termine dell'utilizzo del buffer questo viene rilasciato dalla funzione *Release()* che provvede a decrementare il contatore di sezioni riempite, imposta il puntatore alla sezione successiva e nel caso vi siano in trasmissione task in attesa di spazio libero nel buffer setta l'evento di attesa per risvegliarli.

La sincronia e il mantenimento della coerenza dei dati è garantita dall'evento di sincronizzazione all'interno della classe CMultiBuffer, in quanto OutChannel può ottenere l'accesso a una sezione del buffer di trasmissione fino a che ve ne sono a disposizione ma deve rimanere in attesa che se ne liberi una se queste sono esaurite, mentre InChannel non può ottenere l'accesso al buffer finché questo è vuoto e non son arrivati nuovi dati.

Capitolo 4

Applicazione: Metodi di indagine multi-beam

4.1 Introduzione alle tecniche di beamforming parallelo

Nelle classiche tecniche di imaging ad ultrasuoni la velocità di acquisizione, è limitata dalla velocità del suono; questo comporta un notevole rallentamento nelle applicazioni che richiederebbero un elevato framerate quali le acquisizioni ecografiche tridimensionali.

Per superare questo limite e incrementare la velocità di acquisizione sono state sviluppate tecniche di beamforming parallelo che basandosi sull'utilizzo di fasci multipli riescono ad acquisire più linee contemporaneamente; queste tecniche possono essere implementate durante la fase di ricezione o quella di trasmissione e sono definite rispettivamente PBR (Parallel Beamforming in Reception) e PBT (Parallel Beamforming in Transmission).

Questo capitolo è basato sui seguenti articoli:
L. Demi, J. Viti, G. Giannini, A. Ramalli, P. Tortoli and M. Mischi, "Tissue harmonic images obtained with parallel transmit beamforming by means of orthogonal frequency division multiplexing" Ultrasonics Symposium Proceedings (IUS), 2014 IEEE International, Pages: 1213 - 1216, September 2014

L. Demi, A. Ramalli, G. Giannini and M. Mischi, "In Vitro and in Vivo tissue harmonic images obtained with parallel transmit beamforming by means of orthogonal frequency division multiplexing [Correspondence]" Ultrasonics, Ferroelectrics, and Frequency Control, IEEE Transactions on, Volume: 62, Issue: 1 Pages: 230 - 235, : January 2015

L. Demi, G. Giannini, A. Ramalli, P. Tortoli, and M. Mischi "Multi-Focus Tissue Harmonic Images Obtained with Parallel Transmit Beamforming by means of Orthogonal Frequency Division Multiplexing" Ultrasonics Symposium Proceedings (IUS), 2015 IEEE International, October 2015

Le tecniche di beamforming parallelo in ricezione sono implementate trasmettendo un onda piana che interessi una vasta sezione del tessuto da analizzare, acquisendo durante la fase di ricezione più linee contemporaneamente da utilizzare per formare l'immagine; questa tecnica ha lo svantaggio di non essere applicabile nel campo dell'imaging armonico in quanto l'utilizzo di un onda piana riduce l'ampiezza delle onde di pressione che si generano nel tessuto rispetto a quelle ottenute da un fascio focalizzato e questa caratteristica va in conflitto con la necessita di onde di pressione di ampiezza elevata, necessarie per la formazione delle componenti armoniche.

Le tecniche di beamforming parallelo in trasmissione al contrario sono implementate trasmettendo fasci ultrasonici multipli, differenziandoli in fase di ricezione tramite filtraggio e altre elaborazioni; poiché questi fasci sono focalizzati le onde di pressione che si generano hanno ampiezza elevata e possono essere utilizzate per migliorare la velocità di acquisizione nel campo dell'imaging armonico.

L'imaging armonico, rispetto all'imaging basato sull'analisi della componente fondamentale, ha i vantaggi di migliorare la risoluzione dell'immagine, assiale e laterale, riducendo al contempo gli effetti del clutter e dei lobi secondari, migliorando di conseguenza la risoluzione dell'immagine; è evidente quindi come le tecniche di beamforming parallelo in trasmissione si integrino bene con l'imaging armonico, unendo i vantaggi in termini di qualità dell'immagine con quelli di un maggiore framerate di acquisizione.

In questo capitolo viene presentata una nuova tecnica di beamforming parallelo implementata nella piattaforma ULA_OP e vengono presentate due sue possibili applicazioni nel campo dell'imaging armonico orientate rispettivamente al miglioramento del framerate di acquisizione e al miglioramento del rapporto segnale/rumore, riportando i risultati ottenuti dalle acquisizioni in vitro e in vivo.

4.2 La tecnica OFDM

La tecnica OFDM (Ortogonal Frequency Division Multiplexing) fa parte delle tecniche per implementare il beamforming parallelo in trasmissione.

L'approccio consiste nell'utilizzare in trasmissione fasci multipli, separati temporalmente ed in frequenza per poter differenziare gli echi ricevuti, generati da ciascuno di essi.

In trasmissione la banda disponibile del trasduttore viene suddivisa equamente tra i fasci da utilizzare, separandoli in modo da minimizzare le interferenze reciproche tra di essi, mentre in ricezione vengono utilizzati filtri passabanda, ciascuno centrato sulla frequenza centrale di un diverso fascio

Per minimizzare ulteriormente la sovrapposizione tra i fasci, che potrebbe provocare la nascita di frequenze spurie e effetti di intermodulazione, questi vengono separati anche temporalmente e trasmessi consecutivamente in istanti diversi.

L'utilizzo della tecnica OFDM consente di ridurre l'ampiezza dei lobi laterali, mantenendo al contempo elevata l'ampiezza del lobo principale rispetto alle altre tecniche di beamforming parallelo in trasmissione; questo consente di incrementare la profondità di penetrazione degli ultrasuoni.

Questa tecnica ha però degli svantaggi, in particolare la riduzione della banda degli impulsi trasmessi porta a una riduzione della risoluzione assiale; questo effetto può comunque essere compensato implementando questa tecnica assieme all'imaging armonico che ha l'effetto di migliorare la risoluzione assiale.

Il secondo problema è dato dal fatto che il ritardo temporale tra le trasmissioni dei fasci provoca un incremento della durata della fase di trasmissione e questo provoca l'aumento della profondità minima da cui è possibile iniziare a acquisire gli echi ultrasonici, in quanto la fase di ricezione non può iniziare prima che sia terminata la trasmissione; l'effetto risultante è l'aumento della regione cieca sottostante la sonda, rispetto ad una classica acquisizione B-mode.

La tecnica OFDM è stata implementata nel sistema ULA-OP al fine di migliorare il framerate di acquisizione rispetto ad un immagine ottenuta con la tecnica B-mode standard.

In questa implementazione sono utilizzati fino a tre fasci paralleli in trasmissione focalizzati su punti diversi del tessuto sotto indagine in modo da acquisire simultaneamente fino a tre linee di immagine.

Per validare questo metodo di indagine sono state svolte diverse prove in vitro, confrontando i risultati ottenuti con quelli risultanti da un immagine B-mode in modo da verificare la qualità dell'immagine OFDM rispetto ad un immagine di riferimento.

4.2.1 Setup per le acquisizioni in vitro

Per le acquisizioni è stata utilizzata una sonda ad array lineare LA533 prodotta dalla Esaote.

Questa consiste in un array di 192 elementi di dimensione 0.215mm per 6mm con passo di 0.245mm. La sonda è stata connessa al sistema ULA-OP e durante le acquisizioni è stata sfruttata un'apertura attiva di 62 elementi sia in trasmissione che ricezione.

Come soggetto di test è stato utilizzato un phantom di agarose, in cui sono state ricavate 3 cavità cilindriche di diverso raggio, approssimativamente pari a 1, 2.5 e 7 mm rispettivamente.

In trasmissione il fuoco è stato impostato a 25mm ed è stata utilizzata una apodizzazione di Hanning; non è stato applicato nessuno steering ai fasci.

In ricezione è stata utilizzata focalizzazione dinamica e apodizzazione con f-number = 2.

Per generare l'immagine a 192 linee l'apertura attiva è stata fatta scorrere lungo tutto l'array di elementi della sonda.

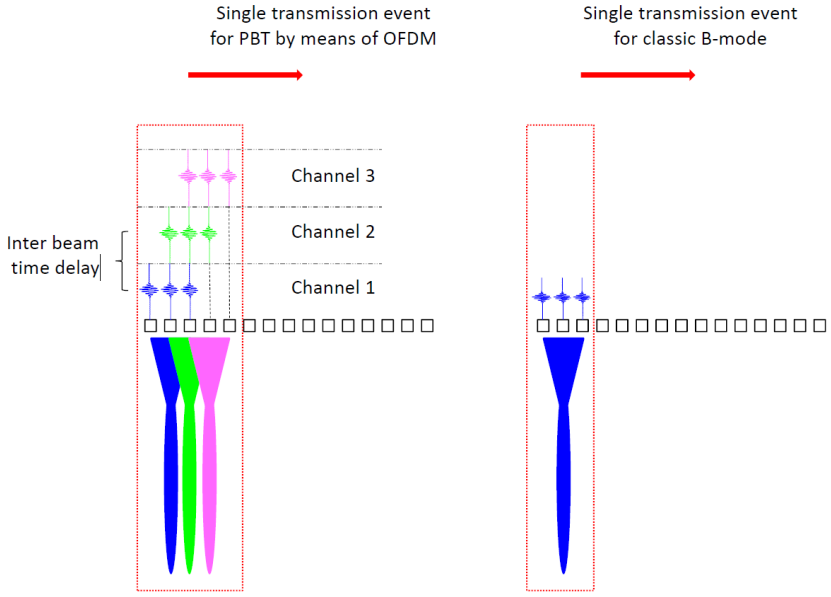

Figura 4.1 - Confronto tra le fasi di trasmissione delle acquisizioni OFDM e B-mode

Le prove effettuate sono state fatte acquisendo un immagine b-mode, da utilizzare come confronto, e 3 immagini realizzate tramite OFDM, rispettivamente con uno, due e tre fasci distinti.

Per quanto la configurazione con un singolo fascio sia inutile ai fini dell'applicazione, in quanto non comporta nessun incremento nel framerate, questa acquisizione è stata fatta in modo da verificare il peggioramento dell'immagine a causa dell'utilizzo di impulsi a banda più stretta, rispetto agli impulsi a banda larga che formano l'immagine B-mode.

Durante le acquisizioni B-mode le linee sono state formate sequenzialmente utilizzando impulsi da 0.8 μs con finestratura flat-top utilizzando una frequenza centrale di 5.5MHz; le immagini così ottenute sono utilizzate come riferimento nella valutazione dei risultati ottenuti dalle acquisizioni tramite OFDM.

Durante le acquisizioni OFDM sono stati utilizzati impulsi gaussiani da 3 μs utilizzando fino ad un massimo di tre fasci trasmessi in parallelo, in modo da sfruttare tutta la banda disponibile del trasduttore senza avere effetti significativi di sovrapposizione tra le bande.

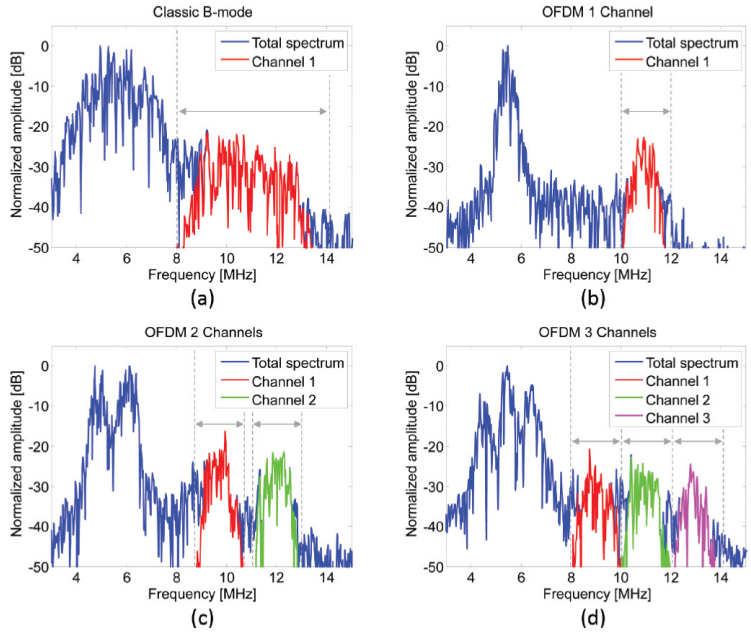

Figura 4.2 - Confronto tra i profili spettrali ottenuti in ricezione nel caso di trasmissioni B-mode (a) e OFDM con uno (b), due (c) e tre (d) fasci

Nei casi in cui sono stati utilizzati due o tre fasci simultanei questi hanno permesso la formazione di due o tre linee di immagine in parallelo rispettivamente.

Durante le acquisizioni in vivo la frequenza centrale utilizzata è stata di 5.5 MHz nel caso di un singolo fascio, 5 e 6 utilizzando 2 fasci e 4.5, 5.5 e 6.5 trasmettendone 3.

La frequenze centrali sono state stabilite al fine di distribuire equamente le bande dei fasci all'interno della banda disponibile del trasduttore, mentre la durata degli impulsi è stata scelta in modo tale da avere una larghezza di banda di ogni singolo impulso tale da poterne utilizzare tre simultaneamente.

Per evitare sgraditi effetti di intermodulazione, i fasci sono stati trasmessi ritardati di 3 μs l'uno dall'altro. A causa dell'effetto di allungamento del tempo di trasmissione questo ha portato all'estensione della regione cieca fino a 15 mm, nel caso di un acquisizione con 3 fasci.

In ricezione è stato impostato un fattore di TGC (Time Gain Compensation) pari a 5.5 dB/cm per entrambi i tipi di immagine e per estrarre gli echi relativi alla componente di seconda armonica sono stati utilizzati filtri passa-banda di Butterworth del settimo ordine, la cui frequenza centrale è il doppio della frequenza di centro banda utilizzata in trasmissione e la cui larghezza di banda è pari a 3.6 MHz e 1.4 MHz rispettivamente per ricevere il segnale B-mode o il segnale di un singolo fascio durante l'acquisizione OFDM.

Dopo il filtraggio è stato calcolato l'inviluppo tramite trasformata di Hilbert.

La tabella sottostante riassume le configurazioni utilizzate durante le prove

Modalità	Numero di linee	Frequenze centrali [MHz]	Lunghezza degli impulsi [μs]	Banda filtro passa-banda [MHz]
B-mode (In vitro e in vivo)	1	5.5	0.8	3.6
OFDM (in vitro)	1	5.5	3	1.4
OFDM (in vitro)	2	5 e 6	3	1.4
OFDM (in vitro)	3	4.5, 5.5 e 6.5	3	1.4

Tabella 4.1 - Impostazioni utilizzate per le acquisizioni

4.2.2 Risultati delle acquisizioni in vitro

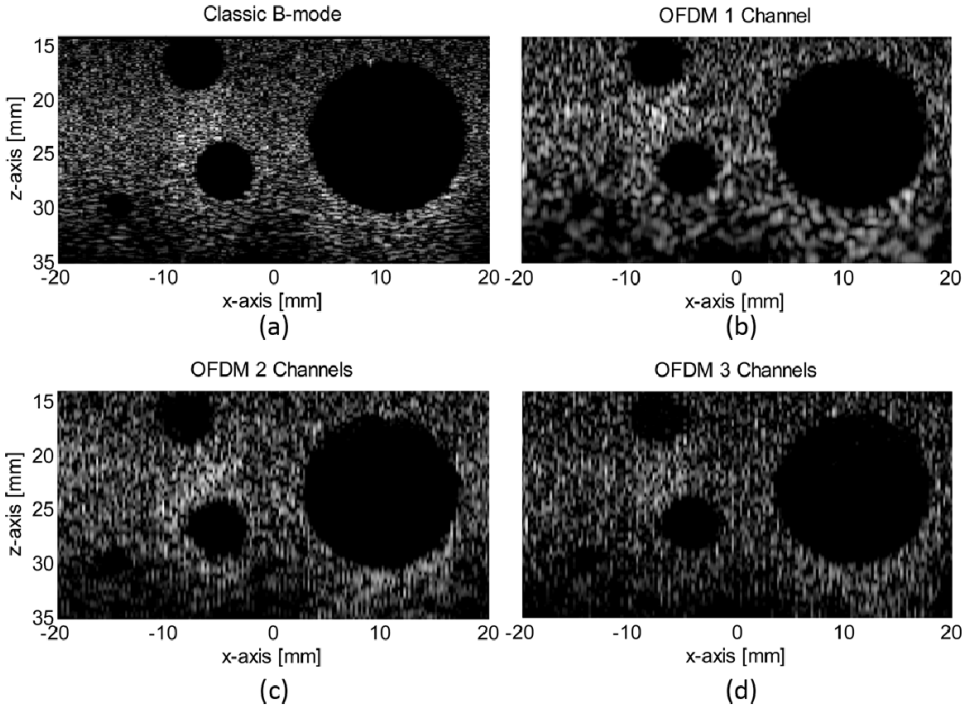

Figura 4.3 - Risultati in vitro delle acquisizioni con imaging di seconda armonica

I dati acquisiti sono stati esportati su PC dove gli echi dei fasci multipli sono stati combinati per ottenere le immagini

La figura riporta i risultati delle acquisizioni su phantom.

Si possono notare due differenze tra l'immagine ottenuta tramite B-mode e quelle tramite PBT: innanzitutto una perdita di risoluzione assiale, secondariamente una deformazione dello spekle pattern che all'aumentare dei canali viene deformato fino ad assumere un andamento a strisce.

Questo effetto è dovuto all'utilizzo di impulsi a banda stretta, fattore che provoca una riduzione della risoluzione assiale dell'immagine e per questo lo spekle viene deformato nella direzione assiale; inoltre, poiché sono utilizzati più impulsi a banda stretta con diverse frequenze centrali questa deformazione avviene anche in direzione laterale.

Come indice di performance della tecnica, dopo la formazione dell'immagine è stato valutato il rapporto contrasto-rumore (Contrast To Noise ratio, CNR) calcolato con la seguente formula:

$$\mathrm{CNR} = 10\log_{10}\left[\frac{\left(\frac{1}{N_{\mathrm{out}}}\sum a_{\mathrm{out}} - \frac{1}{N_{\mathrm{in}}}\sum a_{\mathrm{in}}\right)^2}{\frac{\sigma_{\mathrm{out}}^2+\sigma_{\mathrm{in}}^2}{2}}\right] \qquad (4.1)$$

Dove a_{out} e a_{in} sono i valori di ampiezza in scala lineare e N_{out} e N_{in} il numero di punti e σ_{out} e σ_{in} le deviazioni standard, riferiti rispettivamente all'esterno e all'interno della regione di cui si desidera valutare il contrasto.

Come regioni di interesse sono state considerate le cavità di diverse dimensioni presenti nel phantom, valutando il contrasto tra la zona scura rappresentante la cavità e la zona luminosa circostante rappresentante il tessuto.

In particolare è stato scelto di considerare regioni di tessuto comprese tra due cerchi concentrici, uno sovrapposto al bordo della cavità e l'altro esterno di raggio pari a 2, 5 e 9 mm rispettivamente per cavità piccole medie e grandi.

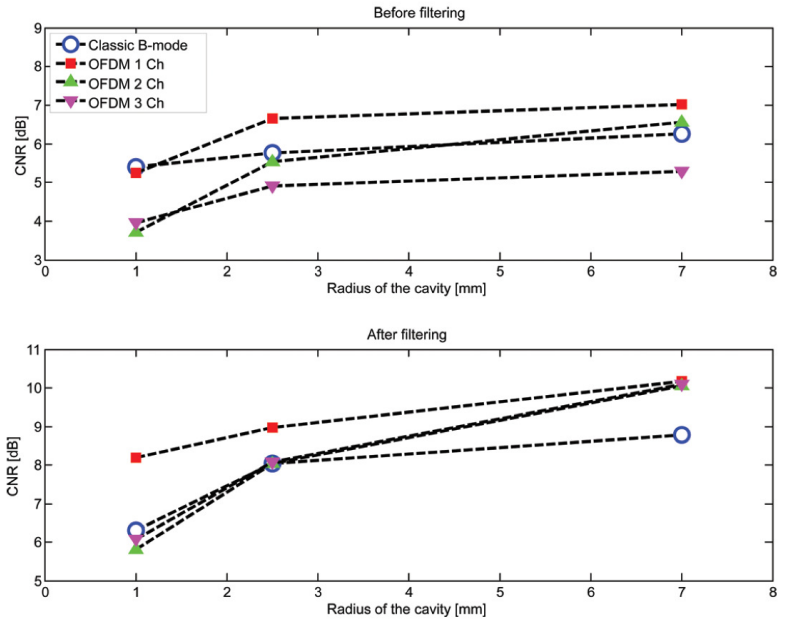

Figura 4.4 - Confronto tra i valori di Contrast-to-noise ratio ottenuti da acquisizioni B-mode e OFTM valutati prima e dopo l'applicazione del filtro spaziale

Questo rapporto è stato considerando sia l'immagine l'immagine ottenuta dopo l'acquisizione, sia la stessa immagine ottenuta dopo l'utilizzo di un filtro gaussiano spaziale bidimensionale rotazionalmente simmetrico di 1.4 per 1.4 mm.

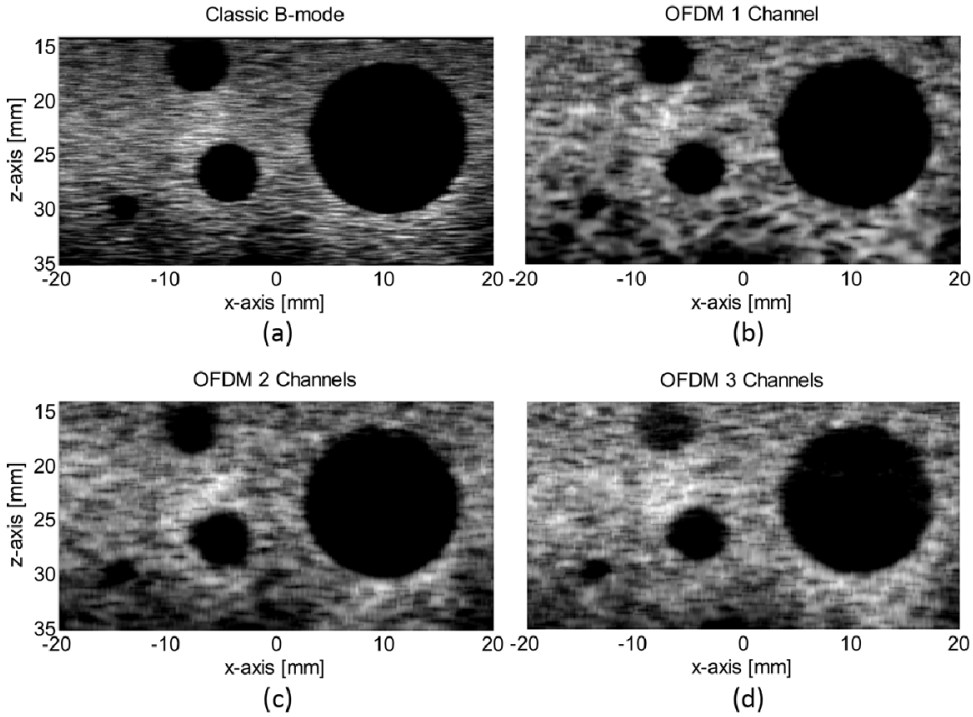

Figura 4.5 - Risultati in vitro delle acquisizioni con imaging di seconda armoica dopo il filtraggio

Si può notare come l'utilizzo della tecnica OFDM provochi una riduzione di circa 1dB rispetto alla stessa immagine ottenuta tramite B-Mode ed è anche stato valutato un incremento di 10 dB nella luminosità dell'immagine.

Questo effetto è causato dalle interferenze tra i fasci trasmessi che si manifestano nell'immagine attraverso la comparsa di scatteratori fantasma, traslati spazialmente in base al ritardo temporale tra i fasci trasmessi.

E' stato anche osservato come il CNR tenda a ridursi in base alla diminuzione del raggio della cavità e all'aumentare del numero di fasci trasmessi in parellelo; anche questi effetti sono correlato alla presenza di scatteratori fantasma dovuti alle interferenze interbeam.

Questi echi fantasma sono dovuti per ciascun fascio ai contributi di frequenza degli altri fasci che rientrano nella banda del filtro dedicato; poiché all'aumentare

del numero di fasci utilizzato si ridue la loro distanza in frequenza, la presenza di queste frequenze indesiderate aumenta.

Riguardo alla dimensione delle cavità quelle dal raggio minore sono influenzate maggiormente dal contributo degli scatteratori fantasma, in quanto questi si estendono su una superficie proporzionalmente maggiore e riducono il contrasto rispetto al tessuto circostante.

L'utilizzo del filtraggio migliora il CNR portandolo a valori comparabili con quelli ottenuti da un immagine B-Mode.

4.2.3 Acquisizioni in vivo

Le prove in vivo sono state effettuate effettuando acquisizioni sull'arteria carotidea di volontari sani, anche in questo caso effettuando il confronto con un acquisizione B-mode come riferimento.

Rispetto alle prove in vitro è stato scelto di limitare il numero di fasci paralleli a due, in quanto i risultati delle acquisizioni con tre fasci hanno portato a immagini di qualità troppo bassa a causa dell'elevata interferenza tra di essi.

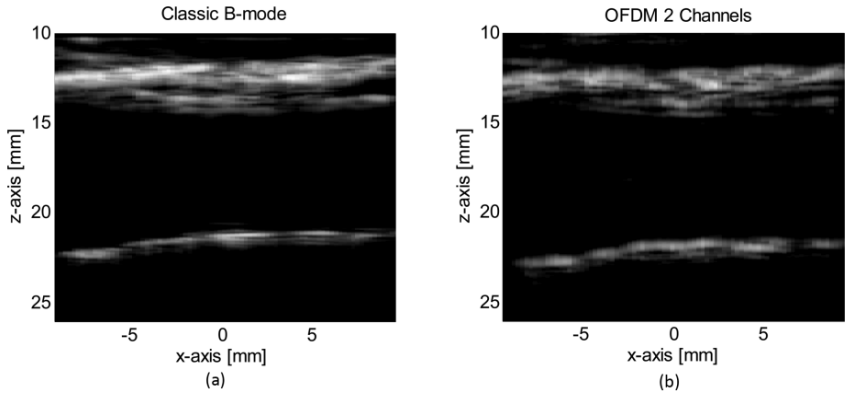

Figura 4.6 - Risultati in vivo delle acquisizioni con imaging di seconda armoica

L'utilizzo di soli due fasci ha portato all'acquisizione di sole due linee in parallelo ma ha consentito di utilizzare impulsi più brevi, pari a 2,5us, con l'effetto di ottenere una migliore risoluzione assiale e la riduzione della regione cieca a soli 10mm.

Come si può vedere dalle immagini la qualità delle acquisizioni OFDM è comparabile con quella ottenuta tramite B-mode.

4.3 OFDM multifocus

Un altro possibile campo applicativo della tecnica OFDM consiste nell'utilizzare fasci multipli non per aumentare il framerate di acquisizione ma per migliorare il rapporto segnale-rumore delle immagini acquisite.

L'incremento del rapporto segnale rumore è particolarmente importante nel campo dell'imaging armonico, in quanto le componenti di seconda armonica presentano ampiezze minori se confrontate con la fondamentale.

In questa implementazione sono utilizzati tre fasci trasmessi nella stessa direzione ma focalizzati a profondità diverse, acquisendo quindi una linea con tre diverse focalizzazioni e ricostruendo l'immagine data dai tre fasci in fase di post-elaborazione su PC.

Per validare questo metodo di indagine sono state svolte alcune prove in vitro, confrontando i risultati ottenuti con quelli risultanti da un immagine B-mode in modo da verificare la qualità dell'immagine OFDM rispetto ad un immagine di riferimento.

4.3.1 Setup per le acquisizioni in vitro

Per le acquisizioni è stata utilizzata una sonda ad array lineare LA533 prodotta dalla Esaote.

Questa consiste in un array di 192 elementi di dimensione 0.215mm per 6mm con passo di 0.245mm. La sonda è stata connessa al sistema ULA-OP e durante le acquisizioni è stata sfruttata un'apertura attiva di 64 elementi sia in trasmissione che ricezione.

Come soggetto di test è stato utilizzato il phantom 404GSE prodotto dalla Gammex, contenente fili iperecoici e cisti anecoiche.

Durante l'acquisizione è stata prodotta un'immagine B-mode standard, da utilizzare come termine di paragone, e un'immagine prodotta tramite la tecnica OFDM a 3 fasci.

In trasmissione per la generazione dell'immagine B-mode sono stati generati impulsi con banda 3 MHz e frequenza centrale di 5.5 MHz, focalizzati a 20 mm; per l'acquisizione OFDM sono stati utilizzati 3 impulsi gaussiani di banda 1 MHz con frequenze centrali di 4.5, 5.5 e 6.6 MHz trasmessi nella stessa direzione con focalizzazioni rispettivamente a 40, 30 e 20 mm, ossia a profondità decrescente all'aumentare della frequenza centrale.

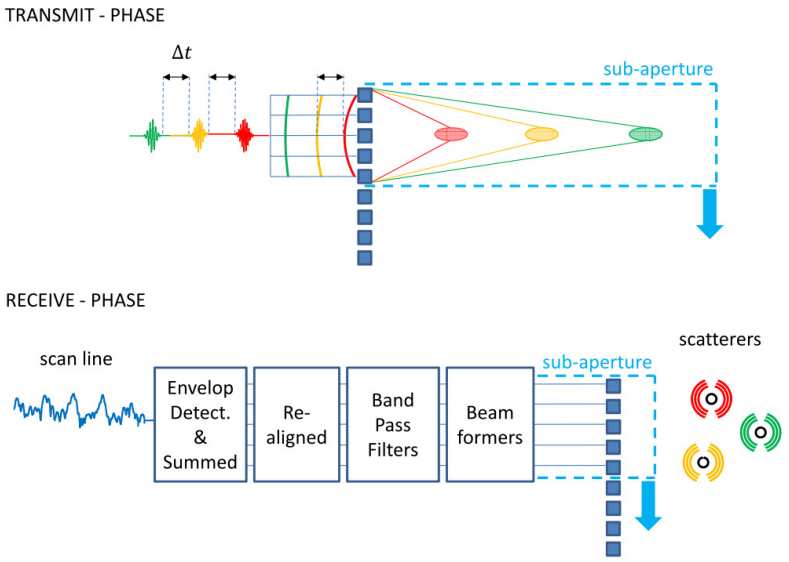

Figura 4.7 Fasi di trasmissione e ricezione per un indagine multifocus tramite tecnica OFDM

Per ridurre l'interferenza tra i fasci e la nascita di frequenze spurie è stato applicato tra di essi un ritardo di 3 μs.

Per generare l'immagine a 192 linee l'apertura attiva è stata fatta scorrere lungo tutto l'array di elementi della sonda.

In ricezione è stato impostato un fattore di TGC pari a 5.5 dB/cm per entrambi i tipi di immagine e per estrarre gli echi relativi alla componente di seconda armonica sono stati utilizzati filtri passa-banda di Butterworth del settimo ordine, applicando infine la trasformata di Hilbert per estrarre l'inviluppo.

4.3.2 Risultati delle acquisizioni in vitro

Per comparare i risultati ottenuti dalla scansione B-mode e OFDM è stata innanzitutto valutata la linea di scansione che intercetta i fili di nylon presenti nel phantom, valutando l'energia del segnale riferita a questi punti.

Il guadagno di segnale è stato definito come il rapporto delle intensità dei picchi in corrispondenza dei fili di nylon, mentre la risoluzione assiale è stata stimata come la larghezza media dei punti a -6 dB rispetto ai fili bersaglio

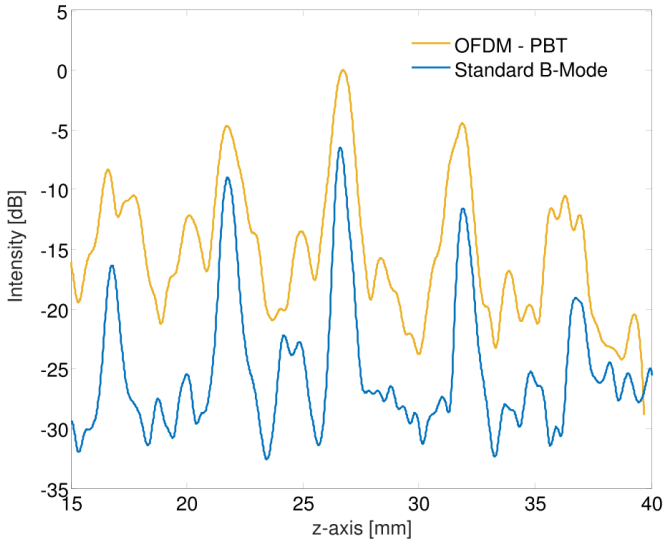

Figura 4.8 - Confronto tra i profili B-scan ottenuti in corrispondenza dei fili di nylon

Il risultato osservato è stato di un guadagno medio del segnale di 6 dB a fronte di una diminuzione della risoluzione assiale da 0.7 a 1.1 mm.

Per confrontare i risultati è stato inoltre stato calcolato il rapporto segnale-rumore dell'immagine attraverso la seguente formula:

$$\mathrm{SNR} = \frac{1}{4}\sum_{i=1}^{4} 20\log_{10}\left[\frac{\mathrm{mean}_i}{\theta_n}\right] \qquad (4.2)$$

Dove mean_i rappresenta il valore di ampiezza medio dell'immagine e θ_n la deviazione standard relativa al rumore.

Il rapporto segnale rumore è stato valutato nelle regioni di interesse mostrate in figura 4.9.

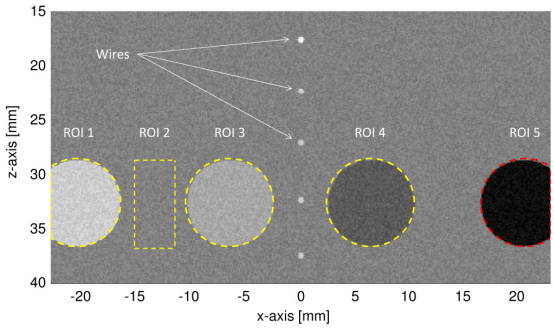

Figura 4.9 - Regioni di interesse utilizzate per il calcolo del SNR

In particolare le regioni circondate dalla linea gialla sono state utilizzate per calcolare i valori di ampiezza media del segnale, mentre la regione interna alla linea rossa è stata utilizzata per stimare la deviazione standard.

Modalità	ROI 1	ROI 2	ROI 3	ROI 4
OFDM (ampiezza)	75	60	70	63
OFDM (SNR)	32	22	27	19
B-mode (Ampiezza)	63	57	59	56
B-mode (SNR)	23	17	20	16

Tabella 4.2 - Confronto tra i valori di ampiezza media e SNR ottenuti con le due modalità

La tabella mostra come rispetto alla tecnica B-mode standard si sia ottenuto un incremento medio relativo al SNR.

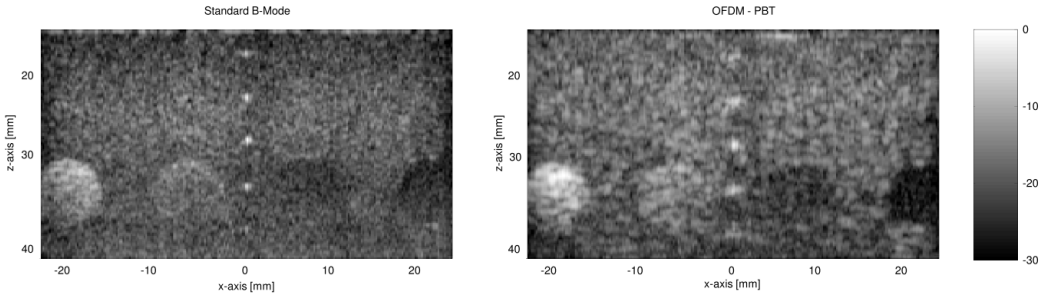

Figura 4.10 - Confronto tra le immagini ottenute con le due modalità di scansione

La figura 4.10 mostra la comparazione tra le immagini ottenute nei due casi e mostra come OFDM consenta di ottenere un miglior rapporto segnale rumore e una maggiore profondità di penetrazione, a fronte di una riduzione nella risoluzione laterale.

4.4 Conclusioni

L'analisi dei risultati ottenuti in vivo ha dimostrato l'applicabilità della tecnica OFDM-PBT nel campo dell'imaging armonico.

Le prove effettuate hanno mostrato come la qualità dell'immagine di un acquisizione OFDM sia comparabile a quella di un acquisizione B-mode standard e come questa tecnica sia utilizzata per migliorare il framerate di acquisizione o il rapporto segnale-rumore, al costo di una riduzione della risoluzione assiale dovuta all'impiego di fasci a banda stretta.

In particolare nell'implementazione multi-focus è stato misurato un incremento del rapporto segnale/rumore di 6 dB a fronte di una perdita di risoluzione assiale dell'ordine di 0.4 mm.

Gli ulteriori sviluppi di questa tecnica saranno focalizzati sul confronto con altre tecniche di beamforming parallelo in trasmissione e sull'incremento del framerate di acquisizione, con particolare interesse al campo dell'ultrasonografia cardiaca, dove OFDM potrebbe potenzialmente portare ad acquisizioni, tramite imaging armonico focalizzato, con framerate fino a 900 Hz.

Capitolo 5

Implementazione della modalità “Doppler angle tracking”

5.1 Stima dell’angolo Doppler

Uno dei problemi principali durante le indagine flussimetriche ad ultrasuoni è la difficoltà di stimare l’angolo Doppler θ, ossia l’angolo tra la direzione del vettore velocità e la direzione di propagazione del fascio ultrasonico. Questa incertezza comporta un’analoga incertezze nella stima della velocità di flusso in quanto θ è un parametro della formula Doppler standard:

$$v = \frac{c\, f_d}{2 f_t \cos\theta} \tag{5.1}$$

Senza la conoscenza dell’angolo Doppler è possibile misurare solamente la componente “assiale” di velocità, ossia la proiezione del vettore velocità lungo la direzione del fascio ultrasonico.

Per ovviare al problema e risalire a una stima dell’angolo Doppler più accurata possibile, nel laboratorio MSDLab è stato messo a punto un originale metodo di indagine che impiega contemporaneamente due fasci ultrasonici (Dual-Beam). Tale metodo presuppone la capacità di posizionare un fascio ultrasonico a 90° rispetto alla direzione del flusso. Se questo avviene, la direzione del flusso può considerarsi nota, e un secondo fascio, inclinato di una

quantità nota rispetto al primo, può permettere una misura di velocità basata sull'equazione Doppler. Nel sistema ULA-OP usato per un primo test del metodo tali fasci sono stati prodotti da due differenti aperture di una sonda ad array lineare utilizzando steering differenti in modo da far intersecare ad entrambi lo stesso sample volume di analisi.

5.2 Il metodo "Doppler angle tracking"

L'algoritmo di Doppler angle tracking implementato nel sistema ULA-OP è basato sull'impiego di due fasci ultrasonici generati da due distinte aperture nella sonda: il primo, definito "fascio di riferimento", viene orientato variandone automaticamente lo steering in modo da posizionarlo il più ortogonalmente possibile rispetto al flusso, mentre il secondo, definito "fascio di misura" viene orientato in modo da formare l'angolo maggiore possibile rispetto al fascio di riferimento.

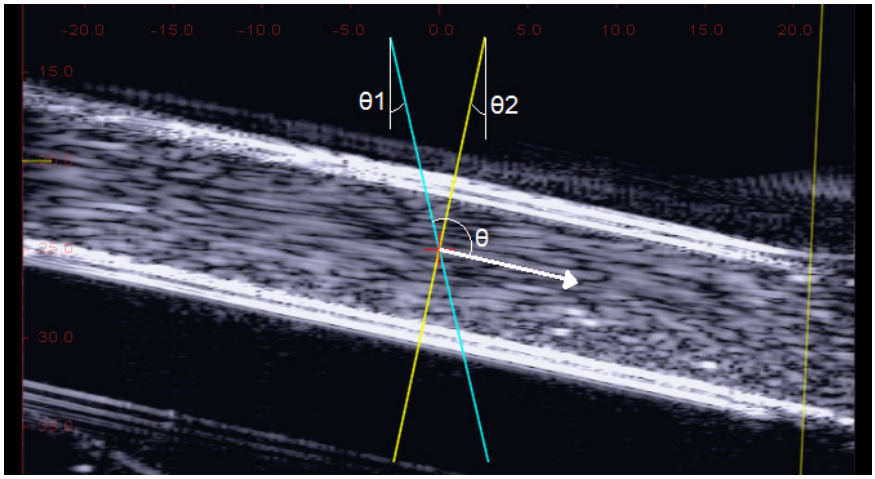

Figura 5.1 - Geometria di posizionamento dei fasci

Poiché è noto l'angolo relativo tra i due fasci, data la condizione di ortogonalità tra il fascio di misura e il vettore velocità del flusso è possibile ottenere una stima precisa dell'angolo Doppler risultante per il fascio di misura.

La possibilità di posizionare accuratamente il fascio di riferimento a 90° rispetto al flusso è garantita dal fatto che in tale condizione lo spettro del fascio di riferimento risulta simmetrico, per via del fenomeno detto "Intrinsecal Spectral Broadening" (ISB).

A 90°, infatti, l'equazione Doppler prevede uno shift pari a zero, ma questo vale per la frequenza media del segnale, il quale occupa un intero spettro di frequenze attorno a tale valore medio.

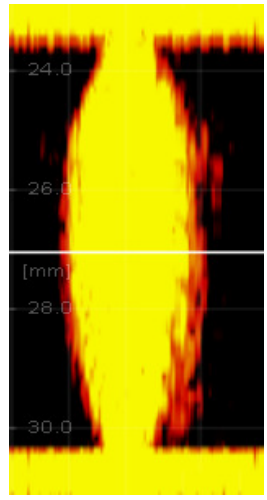

Figura 5.2 - Profilo doppler simmetrico di un flusso stazionario

La condizione di simmetria dello spettro viene utilizzata come feedback e rappresenta lo stato in cui si è raggiunta l'ortogonalità, permettendo così il calcolo dell'angolo Doppler; se gli angoli che i fasci formano sulla sonda, grazie allo steering elettronico, sono rispettivamente θ_1 e θ_2, come mostrato in figura 5.1, l'angolo Doppler che il fascio di misura forma con il flusso è pari a:

$$\theta = 90° + (\theta_1 + \theta_2) \tag{5.2}$$

Per valutare con precisione la simmetria dello spettro l'algoritmo utilizza come parametro di riferimento l'indice di simmetria spettrale, Spectral Symmetry Index (SSI):

$$SSI = 100 \min\left\{\frac{P_-}{P_+}; \frac{P_+}{P_-}\right\} \% \tag{5.3}$$

dove P- e P+ rappresentano rispettivamente le potenze associate alle frequenze negative e positive dello spettro, calcolate a partire da

$$P_+ = \sum_{k=1}^{\frac{N}{2}-1} \|S_k\|^2 \qquad , \qquad P_- = \sum_{k=-\frac{N}{2}+1}^{-1} \|S_k\|^2 \tag{5.4}$$

$$S_k = \sum_{n=0}^{N-1} s_n e^{-j\frac{2\pi}{N}nk} \tag{5.5}$$

L'indice di simmetria è espresso in percentuale e quando questo è pari al 100% la simmetria è completa, ossia le energie dello spettro a frequenze negative e a frequenze positive si equivalgono.

La bontà dell'SSI come parametro utilizzato nell'algoritmo per la valutazione della simmetria risiede nella sua elevata sensibilità rispetto a possibili, anche lievi, sbilanciamenti dello spettro.

Tale percentuale infatti decresce rapidamente a fronte di piccole deviazioni del fascio di riferimento rispetto alla condizione di ortogonalità; in particolare è stato dimostrato che il fascio di riferimento si trova nella condizione di ortogonalità con un margine di ± 1° fintanto che l'indice di simmetria resta sopra all'80% ed un errore di un solo grado è ritenuto accettabile nella stima dell'angolo Doppler.

Viene inoltre utilizzata nel calcolo la frequenza Doppler media, definita :

$$f_d = \frac{PRF}{N} \cdot \frac{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}-1} k\|S_k\|^2}{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}-1} \|S_k\|^2} \tag{5.6}$$

Una volta raggiunta la condizione di simmetria tale frequenza si azzera.

L'algoritmo di Doppler angle tracking agisce quindi andando a variare lo steering del fascio di riferimento fino a portarlo nella condizione di ortogonalità rispetto al flusso di cui si desidera misurare la velocità, utilizzando come feedback del corretto orientamento la simmetria spettrale, misurata tramite l'indice di simmetria spettrale SSI.

Una volta raggiunto un SSI superiore all'80% il fascio di riferimento è considerato orientato di 90° (± 1°) rispetto al flusso; il fascio di misura viene

quindi orientato in modo da intercettare il fascio di riferimento nel sample volume di analisi e in questa condizione la formula (4.2) può essere utilizzata per ottenere una stima dell'angolo Doppler.

Come è facilmente intuibile, mentre durante un'acquisizione in vitro la sonda viene posizionata stabilmente e non vi sono variazioni nell'orientamento tra il fascio di riferimento e il flusso da analizzare, durante un acquisizione in vivo ciò non è vero e sono possibili lievi spostamenti della sonda e conseguentemente dell'orientamento dei fasci; per ovviare a tale problema il sistema continua a monitorare l'indice di simmetria spettrale e nel caso in cui questo scenda sotto l'80% viene rilevata la perdita di ortogonalità e viene quindi variata nuovamente l'orientazione del fascio di riferimento in modo da inseguire la posizione del vaso sotto analisi.

Il termine "tracking" deriva proprio dal fatto che il flusso sotto indagine viene continuamente inseguito in modo da riportare il fascio di riferimento automaticamente alla condizione di ortogonalità dopo ogni spostamento della sonda.

L'automatizzazione data dall'algoritmo porta a 2 vantaggi:

- L'operatore non deve occuparsi di orientare la linea di riferimento perpendicolarmente rispetto al flusso.
- Eventuali spostamenti della sonda e conseguente perdita della condizione di ortogonalità sono automaticamente compensati dal sistema.

5.3 Implementazione ottimizzata del metodo su ULA-OP 256

L'algoritmo di era stato originariamente sviluppato sul sistema ULA-OP 64, ma lo sviluppo della nuova piattaforma ULA-OP 256 ha consentito di implementare su quest'ultima una versione migliorata dell'algoritmo.

Nella precedente implementazione i due fasci Doppler erano trasmessi durante intervalli di TX/RX successivi, alternando la linea relativa al fascio Doppler di riferimento a quella del fascio Doppler di misura; la frequenza effettiva degli impulsi era quindi pari a PRF/2 (dove PRF è la frequenza di ripetizione degli impulsi trasmessi).

Il segnale di misura inoltre presentava echi dalla potenza ridotta in quanto questi erano generati da una trasmissione/ricezione effettuata con un angolo di steering elevato e questo, considerando la "direttività" dell'apertura acustica

utilizzata, comportava una scarsa energia trasmessa sul bersaglio e ricevuta da esso.

Il sistema ULA-OP 256 consente di effettuare il beamforming parallelo in ricezione in real-time, caratteristica che ha consentito di implementare una strategia di trasmissione/ricezione in cui viene trasmesso solamente il fascio Doppler di riferimento effettuando poi l'acquisizione simultanea delle due linee relative al riferimento e alla misura,

Questo è stato fatto allo scopo di ottenere due vantaggi:

- Aumento del framerate, in quanto le due linee Doppler sono generate nel medesimo PRI invece che in due consecutivi; si ha un aumento della velocità di acquisizione da PRF/2 a PRF.
- Aumento della potenza del segnale ricevuto dalla linea di misura, in quanto questo è dovuto all'eco prodotto dal fascio di riferimento che tipicamente presenta angoli di steering ridotti.

5.4 Funzionamento dell'algoritmo

L'algoritmo alla base della modalità tracking opera calcolando continuamente l'indice di simmetria SSI e la frequenza Doppler media f_d al fine di mantenere il fascio di riferimento orientato a 90° rispetto al flusso nella regione di interesse; tali parametri sono ottenuti dallo spettro del segnale del fascio di riferimento valutato attraverso FFT, tipicamente effettuando una media su più spettri in modo da irrobustirne la stima riducendo così gli effetti del rumore e la variabilità dello spettro dovuta allo scattering di ultrasuoni da parte dei globuli rossi.

L'indice di simmetria viene utilizzato come feedback riguardo al raggiungimento della condizione di ortogonalità: come detto in precedenza si considera l'orientamento del fascio di riferimento accettabile con un SSI superiore all'80% mentre al di sotto di tale soglia è necessario un intervento correttivo e uno spostamento di tale fascio.

La frequenza Doppler media viene utilizzata invece per stabilire in quale direzione spostare il fascio in quanto essa è nulla in condizione di ortogonalità mentre assume valori positivi o negativi in base alle frequenze verso cui è spostato lo spettro del segnale.

Poiché quando lo spettro è simmetrico, la frequenza media è nulla, si potrebbe pensare di utilizzarla come indicatore di simmetria verificando la condizione per cui essa si annulla, ma è stata preferita la misura del SSI per due motivazioni: innanzitutto la frequenza media Doppler è proporzionale alla

velocità del flusso, rendendo così difficile la scelta di una soglia utile per stabilire quando la perpendicolarità del fascio sia accettabile, in secondo luogo la stima della frequenza Doppler è particolarmente critica in prossimità della frequenza zero, perché influenzata dal clutter.

Per questi due motivi la verifica dell'ortogonalità è effettuata dall'analisi di SSI mentre la frequenza Doppler media indica la direzione verso cui orientare il fascio per riportarlo all'ortogonalità.

All'avvio del sistema il fascio di riferimento si trova in posizione arbitraria e quindi probabilmente lontano dalla posizione ottimale di perpendicolarità; a ogni iterazione quindi l'algoritmo sposta il fascio di riferimento verso la posizione di perpendicolarità aumentando o riducendo l'angolo di steering in base al segno della frequenza Doppler media.

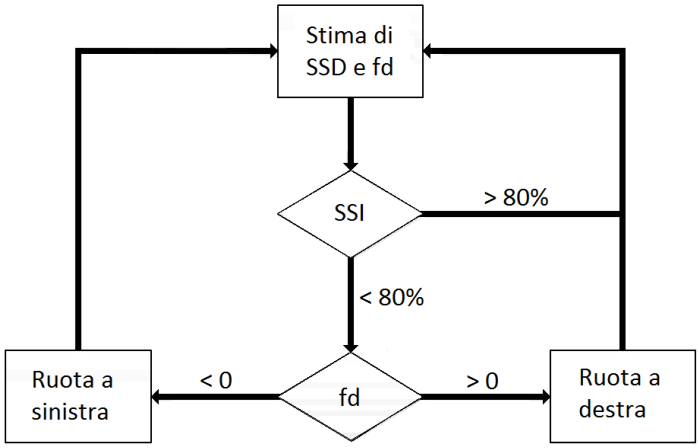

Figura 5.3 - Diagramma di flusso semplificato dell'algoritmo di tracking

Possono occorrere varie iterazioni affinché l'algoritmo converga alla posizione di ottimale indicata da un valore dell'SSI superiore alla soglia dell'80%; al di sopra di tale soglia l'algoritmo non sposta più il fascio di riferimento e angolo Doppler può essere stimato con un errore massimo di 1°.

L'indice di simmetria viene comunque monitorato continuamente e nel caso in cui cali al di sotto della soglia l'algoritmo riprende a spostare il fascio di riferimento per riportarlo nella condizione di ortogonalità

Sono presenti 61 possibili linee di riferimento, con angoli che variano da +15° e -15° e distanziate tra di loro di 0.5°. Le linee di misura sono solamente due e sono quelle con steering maggiore, ossia quelle a +15° e -15°.

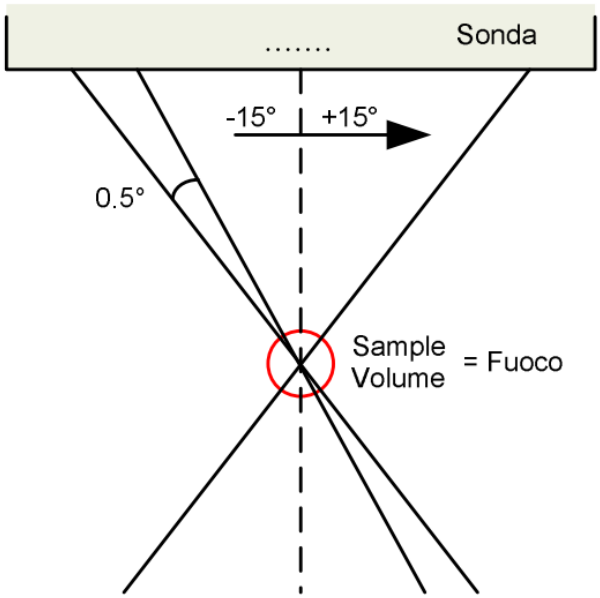

Figura 5.4 - Orientamento dei fasci utilizzati nell'algoritmo

Poiché devono essere acquisite simultaneamente le due linee, di riferimento e di misura, è necessario considerare tutte le combinazioni possibili: questo porta a dover avere in memoria 122 possibili strategie di trasmissione/ricezione.

5.5 Implementazione dell’algoritmo su DSP

A livello di codice possono essere distinte due diverse sezioni:

- La sezione di elaborazione, che si occupa di processare i dati ricevuti al fine di ottenere i parametri necessari per il tracking, ossia SSI e f_d.
- La sezione decisionale che valutando i parametri ottenuti dal processing decide se spostare i fasci di riferimento e di misura.

5.5.1 Sezione di elaborazione

Il modulo richiede una lettura della slice di memoria in cui sono contenuti i dati ricevuti relativi al fascio di riferimento; a questi campioni viene applicata una finestra e tramite una FFT viene calcolato lo spettro del segnale.

A partire da questo sono calcolate le potenze per le frequenze positive e negative, P+ e P-, rispettivamente, e il segno della frequenza Doppler; poiché

per l'algoritmo è necessario solamente sapere se f_d assume valori positivi o negativi, viene utilizzata una versione semplificata della formula per calcolarla, in modo da ridurre il carico di lavoro del DSP.

$$f_d^* = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}-1} k \left\| S_k \right\|^2 \tag{5.7}$$

Dei valori calcolati viene poi effettuata una media su più spettri e su più profondità in modo da irrobustire il valore calcolato.

P+ e P- vengono presentati alle funzioni per il calcolo di SSI e del segno di f_d attraverso due percorsi distinti; nel caso in cui il sistema sia sganciato, ossia alla ricerca della condizione di simmetria i parametri vengono calcolati direttamente, viceversa nel caso in cui il sistema sia agganciato, ossia il fascio di riferimento sia già orientato perpendicolarmente rispetto al flusso, viene effettuata su P+ e P- un'ulteriore media su 4 valori.

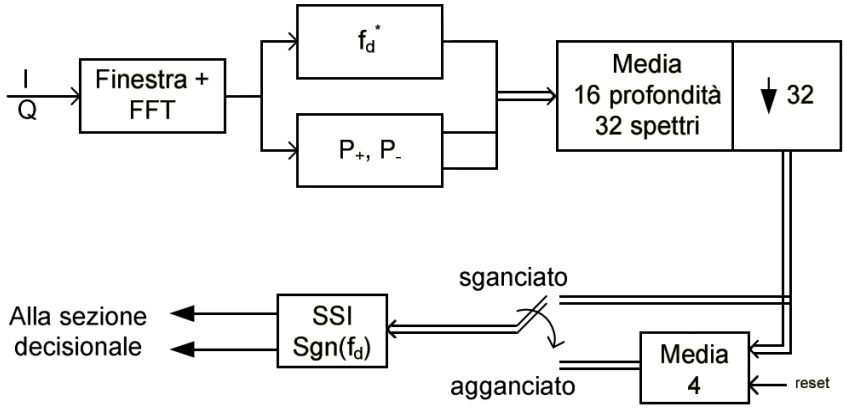

Figura 5.5 - Diagramma a blocchi della sezione di processing

L'ultimo stadio dell'elaborazione consiste quindi nel calcolo dei parametri fondamentali, ossia l'indice di simmetria SSI ed il segno della frequenza Doppler media, sgn(f_d), presentandoli poi alla sezione decisionale che li analizza per decidere se e come spostare il fascio di riferimento.

5.5.2 Sezione decisionale

Alla base della sezione decisionale c’è una macchina a 3 stati, ognuno dei quali connesso ad una delle possibili condizioni in cui può trovarsi il sistema.

- Non agganciato: SSI inferiore alla soglia dell’80%.
- In aggancio: SSI ha superato la soglia e l’algoritmo sta valutando se il fascio selezionato è il migliore possibile oppure no.
- Agganciato: SSI superiore alla soglia, condizione di perpendicolarità ottenuta.

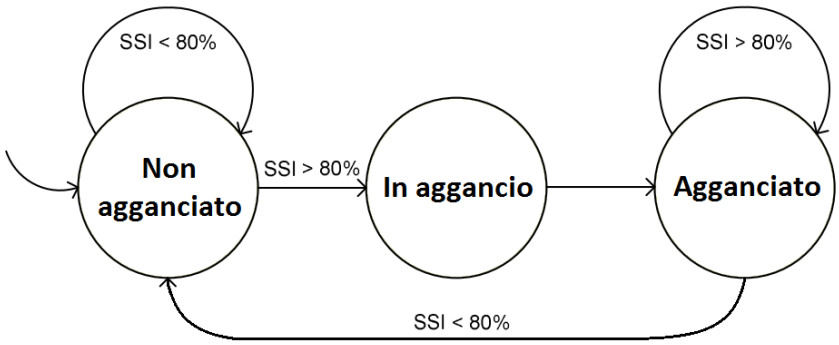

Figura 5.6 - Diagramma degli stati della macchina decisionale

Durante la ricerca della perpendicolarità il fascio viene spostato ad ogni iterazione del modulo di una quantità, chiamata “passo” che può variare tra 0.5° e 4°; il valore del passo viene calcolato attraverso una funzione inversamente proporzionale al SSI, in modo da ottenere passi tanto più lunghi tanto più il fascio è lontano dalla condizione di simmetria.

$$J(SSI) = 4 - \frac{1}{2}\left\lfloor \frac{SSI}{10} \right\rfloor \tag{5.8}$$

All’avvio del sistema il tracking si trova nello stato “non agganciato”; all’avvio delle prime stime di SSI e del modulo della frequenza Doppler media il modulo decide in base a tali valori se il fascio è lontano dalla condizione di perpendicolarità e va spostato o se entrare nello stato “in aggancio”.

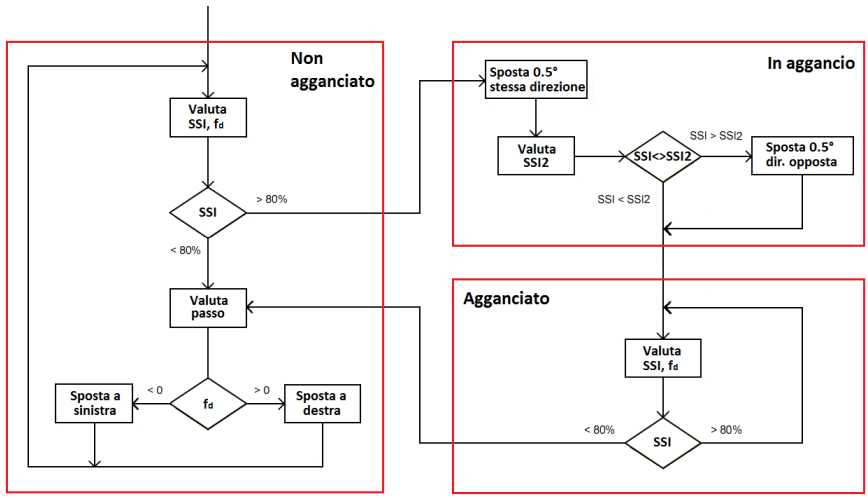

Figura 5.7 - Diagramma di flusso completo dell’algoritmo di tracking

Ad ogni iterazione dell’algoritmo la sezione decisionale valuta il valore di SSI e se questi è inferiore alla soglia dell’80% il fascio di riferimento viene spostato nella direzione indicata dal segno di f_d, verso destra se il segno è positivo verso sinistra se negativo, calcolando il passo con cui variare l’inclinazione di tale fascio.

Superata la soglia il sistema entra nello stato “in aggancio” e l’algoritmo cerca tra le possibili linee generabili quella che offre un indice di simmetria migliore.

In questa fase viene quindi memorizzato lo SSI attuale e viene spostato il fascio di 0.5° nella stessa direzione in cui era stato mosso durante l’ultima iterazione, attendendo dalla sezione di processing una nuova stima del valore di SSI relativa al nuovo fascio; alla ricezione dei nuovi dati la sezione decisionale confronta la nuova stima di SSI con la precedente e nel caso in cui questa sia più elevata l’algoritmo passa nello stato agganciato, viceversa viene prima selezionata la linea precedente che garantiva un maggiore indice di simmetria e successivamente viene cambiato lo stato in “agganciato”.

Nello stato agganciato, l’algoritmo continua a controllare il valore dell’indice di simmetria assicurandosi che resti superiore alla soglia, tornando nello stato sganciato non appena questa condizione viene meno.

La sezione decisionale ha anche lo scopo di decidere quale linea deve essere selezionata per il fascio di misura tra le due possibili, quella a +15° e quella a -15°.

Per massimizzare l’angolo Doppler questa linea deve sempre trovarsi in direzione opposta rispetto a quella di misura, per cui l’implementazione più

semplice di questa strategia consisterebbe nel selezionare il fascio a +15° quando la linea di riferimento scelta dall’algoritmo ha un angolo negativo ed il fascio a -15° nel caso opposto.

Lo svantaggio di tale scelta si verificherebbe nella situazione in cui il vaso sotto indagine si trovi praticamente parallelo alla sonda, cosa che porterebbe a selezionare la linea di riferimento centrale e quelle immediatamente adiacenti durante l’inseguimento e provocando nel fascio di misura una continua oscillazione tra i due possibili orientamenti a +15° e -15°.

Per eliminare questo problema è stata quindi introdotta un’isteresi sullo spostamento del fascio di misura: il fascio di misura viene quindi spostato solamente quando si trova nella stessa direzione del fascio di riferimento e quest’ultimo possiede un angolo in valore assoluto maggiore o uguale a 2°.

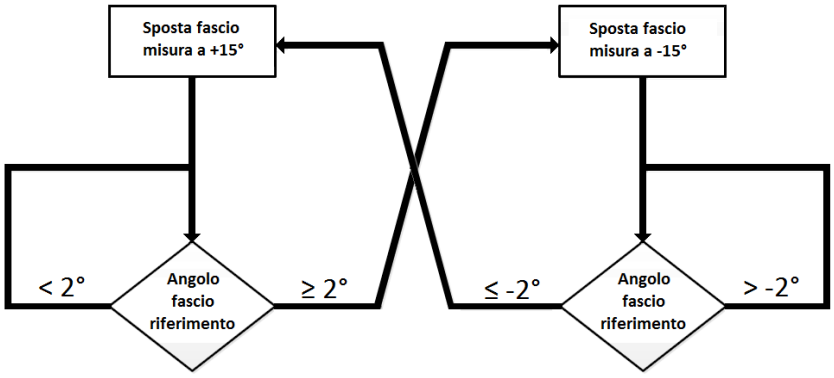

Figura 5.8 - Diagramma di flusso per la selezione del fascio di misura

5.6 Applicazione alla stima della velocità di flusso

L’algoritmo ottimizzato di Doppler angle tracking è stato sperimentato mediante test in vitro per verificare la sua capacità di posizionare correttamente il fascio di riferimento, inseguendo la posizione del vaso a diverse inclinazioni, e di fornire una stima di velocità accurata.

Per effettuare i test è stato impiegato un simulatore di flusso costituito da un tubo in Rilsan® di 8mm in cui è stato fatto scorrere un fluido realizzato in modo tale da simulare il comportamento “aucustico” del sangue.

Questo fluido è stato ottenuto a partire da una miscela di acqua, tensioattivo e Orgasol®, microsfere di plastica le cui dimensioni sono comparabili a quelle dei globuli rossi e, quando investite da un fascio ad ultrasuoni si comportano da riflettori isotropi, dando luogo ad un segnale di backscattering equivalente a quello riscontrabile nel sangue.

Questo fluido è stato fatto scorrere nel tubo tramite una pompa in grado di generare un flusso a velocità costante.

Per effettuare le misure è stata collegata al sistema ULA-OP 256 una sonda ad array lineare a 192 elementi prodotta dalla posizionata longitudinalmente rispetto al tubo, orientabile tramite un sistema di posizionamento regolabile, in modo da poter formare angoli diversi rispetto al tubo.

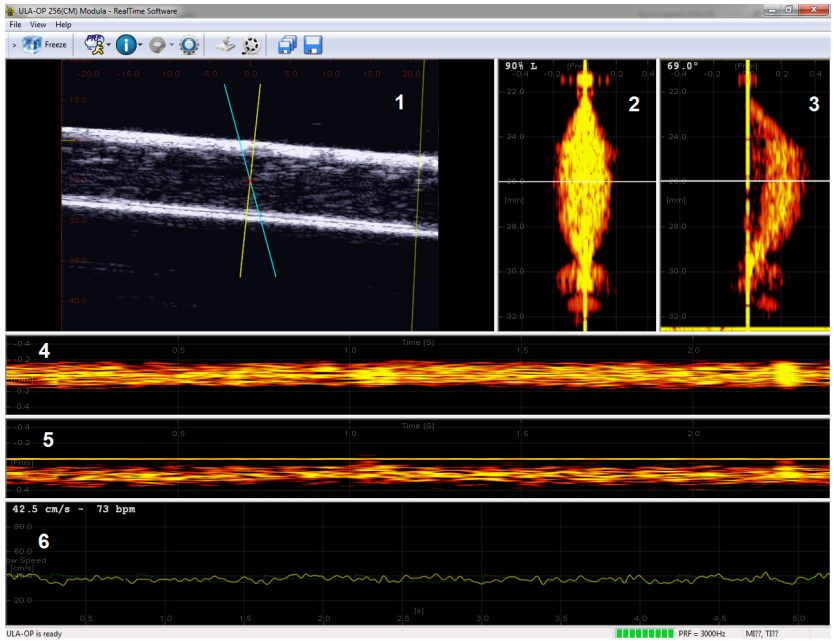

Figura 5.9 - Schermata visualizzata durante un acquisizione col metodo "Doppler angle tracking"

Un esempio di schermata fornita dal sistema ULA-OP 256 durante i test in vitro è riportato in figura 5.9.

Sono presenti una finestra in cui viene mostrata una immagine B-mode (1), in cui sono evidenziate in giallo la linea relativa al fascio di riferimento e in blu la linea relativa al fascio di misura.

Sono poi presenti due profili Doppler spettrali e due sonogrammi, rispettivamente associati al fascio di riferimento (2 e 5) e al fascio di misura (3 e 5); è infine presente una finestra che mostra l'andamento della velocità rilevata (6).

Inizialmente è stato valutato l'aspetto "tracking" ossia la capacità del sistema di inseguire la posizione del vaso e di orientare correttamente il fascio di riferimento.

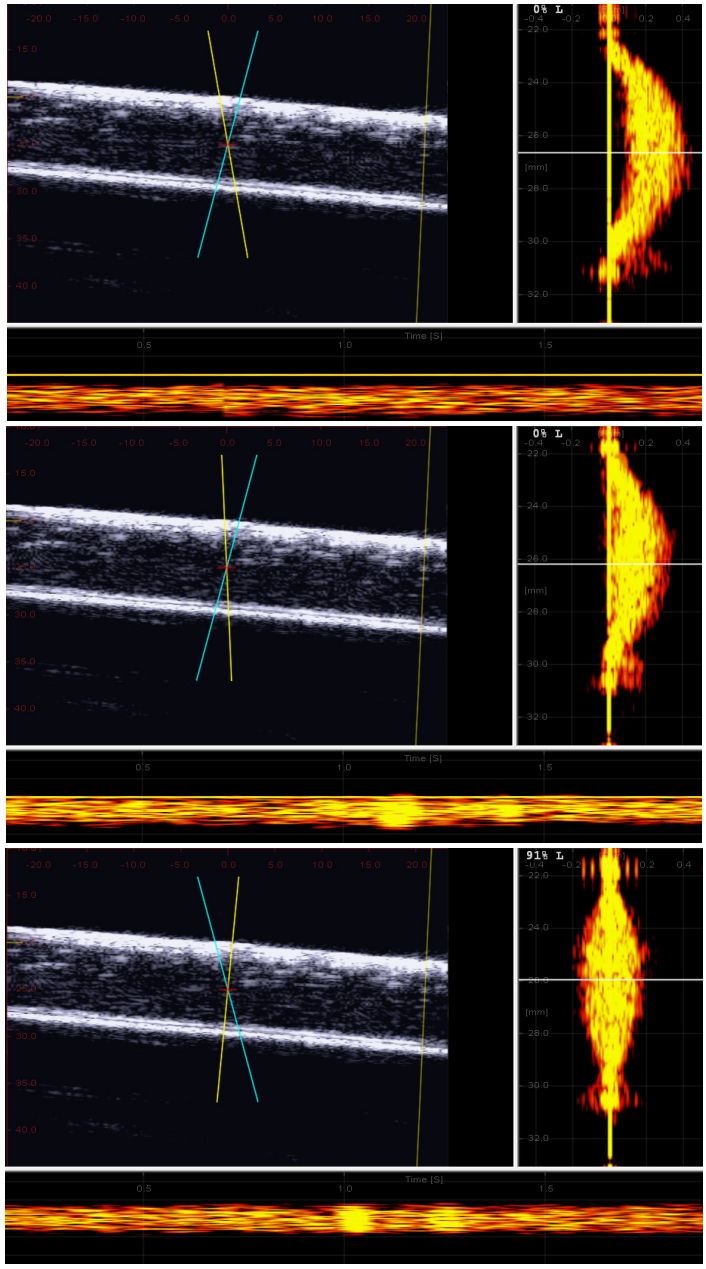

Figura 5.10 - Evoluzione temporale del posizionamento dei fasci, del profilo Doppler e del sonogramma relativi al fascio di riferimento durante la ricerca della condizione di perpendicolarità

Per fare ciò sono state utilizzati diversi orientamenti della sonda rispetto al vaso, andando a valutare graficamente l'orientamento della linea rappresentante il fascio e la simmetria dello spettro, e numericamente valutando l'indice di simmetria riportato in percentuale nella finestra Doppler della linea di riferimento.

Sono riportate in figura 5.10 tre schermate che mostrano come a partire dalla condizione iniziale (fig. in alto) la linea di riferimento tenda progressivamente a portarsi in posizione sempre più ortogonale mentre il profilo Doppler e il sonogramma relativi a tale fascio tendano a raggiungere la condizione di simmetria.

Tale procedura è stata ripetuta varie volte, ed ogni volta cambiando l'angolo tra sonda e tubo in modo da intercettare il flusso con angoli Doppler sempre diversi.

Verificata la capacità del sistema di orientare correttamente i fasci sono state effettuate prove volte a valutare la correttezza nella stima della velocità del flusso.

In particolare, più che l'errore assoluto compiuto dalla singola stima a un inclinazione nota rispetto alla reale velocità del flusso, lo scopo delle acquisizioni era verificare la variazione tra le velocità stimate a diversi angoli di inclinazione, in modo da testare la capacità del sistema di inseguire la posizione dal vaso e fornire una stima di velocità indipendente dalla sua posizione relativa rispetto alla sonda.

Il calcolo della stima della velocità è affidato ad un modulo di elaborazione presente nel pc, il quale riceve i dati relativi alla densità spettrale di potenza all'interno del sample volume sotto analisi e il valore stimato dell'angolo Doppler.

I campioni di potenza sono elaborati in modo da estrarre la frequenza Doppler media e questa è utilizzata nella formula Doppler per calcolare la velocità del flusso:

$$v = \frac{c \cdot f_{mean}}{f_t} \frac{1}{\cos\theta} \tag{5.9}$$

Nei test effettuati è stata quindi misurata la portata del tubo, con la pompa azionata in modo da produrre un flusso costante, e sono state fatte misure per ottenere una stima di velocità a diverse inclinazioni della sonda rispetto al tubo, in modo da coprire il range compreso tra la posizione di perpendicolarità e il massimo steering ottenibile dal fascio di riferimento.

A fronte di una portata di fluido costante misurata di 342 ml/minuto, pari ad una velocità stimata di 22,6 cm/s il modulo per il calcolo della velocità ha fornito i seguenti valori.

Inclinazione sonda	Velocità misurata (cm/s)
90°	23.9
95,5°	25
98,5°	25.3
103°	25.5

Tabella 5.1 – Risultati delle misure di velocità

Da questi valori è stata calcolata la deviazione standard, pari a 0,618 cm/s e il coefficiente di variazione, pari al 2,47%

Questi risultati preliminari mostrano come le variazioni nella velocità rilevata nel range di steering gestito dall'algoritmo di Doppler angle tracking siano molto contenute, testimoniando le potenzialità del metodo nel fornire una stima dell'angolo doppler che possa portare a una stima di velocità accurata indipendentemente dalla posizione relativa tra la sonda e il vaso.

Questa è comunque un implementazione ancora in fase di perfezionamento e sono tuttora in corso ulteriori ottimizzazioni volte a rendere più precisa la stima della velocità e misure atte a valutare il miglioramento del rapporto S/N dato dall'implementazione di questo metodo su ULA-OP 256 rispetto alla precedente implementazione su ULA-OP 64

Conclusioni

In questo elaborato stato descritto il mio contributo allo sviluppo della nuova piattaforma ecografica “aperta” “ULA-OP 256”, progettata per consentire il test sperimentale di nuovi metodi di indagine ad ultrasuoni.

La mia collaborazione, in particolare, si è concretizzata nello sviluppo di un software mirato ad aiutare l’utente durante la fase di configurazione, e nella realizzazione di un firmware basato su micro-kernel, il cui scopo è quello di massimizzare le caratteristiche di flessibilità del sistema stesso. Nei capitoli finali di questa tesi è inoltre descritto il mio contributo al test funzionale del sistema ULA-OP, effettuato attraverso due applicazioni sperimentali distinte.

La prima di queste è la tecnica OFDM, un nuovo metodo di indagine basato sulla trasmissione di fasci multipli, volta a migliorare il frame rate di acquisizione ed il rapporto segnale rumore. Questa tecnica è stata validata con prove in vitro ed in vivo che hanno evidenziato l’applicabilità anche nel campo dell’imaging armonico.

La seconda tecnica, definita “Doppler angle tracking” consente di ottenere la stima dell’angolo Doppler con grande grado di accuratezza, ed è quindi di notevole interesse nella flussimetria ecografica. Sviluppata originariamente per il sistema ULA-OP a 64 canali, questa tecnica è stata implementata nel nuovo sistema sfruttando le potenzialità offerte dal nuovo hardware, che consentono di usare in modo flessibile aperture diverse della sonda sia in trasmissione che in ricezione. In questo modo è stato

ottenuto un aumento del framerate di acquisizione e una maggiore qualità del segnale ricevuto.

Il lavoro svolto contribuirà ad un'applicazione intensiva del nuovo sistema in numerosi campi di indagine, dall'imaging ad alto frame rate alla rivelazione vettoriale della velocità di flusso sanguigno. Sfruttando, con il nuovo firmware, le notevoli potenzialità offerte dall'architettura hardware in termini di modularità e flessibilità, ULA-OP 256 sarà così utilizzabile nell'ambito delle numerose collaborazioni internazionali in cui MSDLab è già coivolto.

Bibliografia

[1] G. Bambi, T.Morganti, S.Ricci, E. Boni, F.Guidi, C.Palombo, P.Tortoli, "A novel ultrasound instrument for investigation of arterial mechanics", Ultrasonics, Vol 42/1-9, pp 731-737, 2004.

[2] P. Tortoli, F. Guidi, G. Bambi, S. Ricci, R. Muchada, "A Novel Instrument for Investigation of Aortic Haemodynamics", 2000 IEEE Ultrasonics Symposium Proceedings, pp.1479-1482, San Juan (USA), Ottobre 2000.

[3] S.Ricci, E. Boni, F.Guidi, T.Morganti, P.Tortoli, "A Programmable Real-Time System for Development and Test of New Ultrasound Investigation Methods", IEEE Trans. Ultrason., Ferroelect., Freq. Contr., vol. 53, no. 10, pp. 1813–1819, 2006.

[4] T. Misaridis and J. A. Jensen, "Use of modulated excitation signals in ultrasound. Part I: Basic concepts and expected benefits", IEEE Trans. Ultrason., Ferroelect., Freq. Contr., vol. 52, no. 2, pp. 192–207, 2005.

[5] Texas Instruments Incorporated, SPRS691E "TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor", March 2014. http://www.ti.com

[6] Texas Instruments Incorporated, SPRUGY9C "KeyStone Architecture Bootloader User Guide", July 2013. http://www.ti.com

[7] Texas Instruments Incorporated, SPRUGV2F "KeyStone Architecture Phase-Locked Loop (PLL) User Guide", May 2013. http://www.ti.com

[8] Texas Instruments Incorporated, SPRUGY8 "TMS320C66x DSP Cache User Guide", November 2010. http://www.ti.com

[9] Texas Instruments Incorporated, SPRUGV1 "KeyStone Architecture General Purpose Input/Output (GPIO) User Guide", November 2010. http://www.ti.com

[10] Texas Instruments Incorporated, SPRUGW0C "TMS320C66x DSP CorePac User Guide", July 2013. http://www.ti.com

[11] Texas Instruments Incorporated, SPRUGV3 "KeyStone Architecture Inter-IC Control Bus (I2C) User Guide", August 2011. http://www.ti.com

[12] Texas Instruments Incorporated, SPRUGP2A “KeyStone Architecture Serial Peripheral Interface (SPI) User Guide”, March 2012. http://www.ti.com

[13] Texas Instruments Incorporated, SPRUGW1B “KeyStone Architecture Serial Rapid IO (SRIO) User Guide”, November 2012. http://www.ti.com

[14] Texas Instruments Incorporated, SPRUGS5A “KeyStone Enhanced Direct Memory Access (EDMA3) Controller User Guide”, December 2011. http://www.ti.com

[15] Texas Instruments Incorporated, SPRUGV8C “KeyStone Architecture DDR3 Memory Controller User Guide”, November 2011. http://www.ti.com

[16] Texas Instruments Incorporated, SPRABL2A “KeyStone DDR3 Initialization Application Report” October 2012. http://www.ti.com

[17] Texas Instruments Incorporated, SPRUGV5A “KeyStone Architecture Timer64P User Guide”,March 2012. http://www.ti.com

[18] Texas Instruments Incorporated, SPRUGS3A “KeyStone Architecture Semaphore2 Hardware Module User Guide”, April 2012. http://www.ti.com

[19] Texas Instruments Incorporated, SPRUGV4B “KeyStone Architecture Power Sleep Controller (PSC) User Guide”, November 2011. http://www.ti.com

[20] Texas Instruments Incorporated, SPRUGW4A “KeyStone Architecture Chip Interrupt Controller (CIC) User Guide”, March 2012. http://www.ti.com

[21] Texas Instruments Incorporated, SPRUGH7 “TMS320C66x DSP CPU and Instruction Set Reference Guide”, November 2010. http://www.ti.com

[22] Texas Instruments Incorporated, SPRU186X “TMS320C6000 Assembly Language Tools v7.6”, March 2014. http://www.ti.com

[23] Texas Instruments Incorporated, SPRU187V “TMS320C6000 Optimizing Compiler v7.6”, March 2014. http://www.ti.com

[24] Integrated Device Technology Incorporated, “CPS-1432 Datasheet”, October 2011. http://www.idt.com

[25] Integrated Device Technology Incorporated, “CPS-1432 User Manual”, February 2012 http://www.idt.com

[26] Lieneke Kusters “The OFDM parallel transmit beamforming technique for ultrasound imaging - Analysis of possibilities and trade-offs of an implementation of this technique”, 2012

[27] L. Demi, and M.D. Verweij, K.W.A. van Dongen “Parallel Transmit Beamforming using Orthogonal Frequency Division Multiplexing for Harmonic Imaging”, 2011 IEEE Ultrasonics Symposium Proceed, pp.148-151, Orlando, October 2011.

[28] L. Demi, M.D. Verweij, K.W.A. van Dongen “Parallel Transmit Beamforming Using Orthogonal Frequency Division Multiplexing Applied to Harmonic Imaging - A Feasibility Study”, IEEE Trans. Ultrason., Ferroelect., Freq. Contr., Vol. 59, no. 11, 2012

[29] L. Demi, J. Viti, L. Kusters, F. Guidi, P. Tortoli, and M. Mischi, “Implementation of Parallel Transmit Beamforming using Orthogonal Frequency Division Multiplexing - Achievable Resolution and Interbeam Interference”, IEEE Trans. Ultrason., Ferroelect., Freq. Contr., Vol. 60, no. 11, 2013

[30] A.Dallai “Sviluppo dIl hardware e firmware per un Sistema di imaging ad ultrasuoni”, Tesi di dottorato, Università degli studi di Firenze, 2008

[31] L. Francalanci, “Problemi e soluzioni nella stima di velocità di flusso mediante Doppler ad ultrasuoni”, Tesi di laurea, Università degli studi di Firenze, 2006.

[32] L. Francalanci “Sviluppo di applicazioni innovative per imaging e Doppler ad ultrasuoni su un sistema ecografico di ricerca”, Tesi di dottorato, Università degli studi di Firenze, 2010

[33] P. Tortoli, G. Bambi, S. Ricci, “A novel dual beam approach for removing doppler angle ambiguity”, 2005 IEEE Ultrasonics Symposium Proceed, Vol. 1 N.18 pp.150-153, Rotterdam, September 2005.

[34] P. Tortoli, G. Bambi, S. Ricci, “Accurate Doppler Angle Estimation for Vector Flow Measurements”, IEEE Trans. Ultrason., Ferroelect., Freq. Contr., Vol. 53, no. 8, pp 1425-1431, 2006.

Pubblicazioni

L. Demi, J. Viti, G. Giannini, A. Ramalli, P. Tortoli and M. Mischi, “Tissue harmonic images obtained with parallel transmit beamforming by means of orthogonal frequency division multiplexing” Ultrasonics Symposium Proceedings (IUS), 2014 IEEE International, Pages: 1213 - 1216, September 2014
(DOI: 10.1109/ULTSYM.2014.0299)

E. Boni, L. Bassi,A. Dallai,G. Giannini,F. Guidi,V. Meacci,A. Ramalli, S . Ricci,P. Tortoli, “Open Platforms for the Advancement of Ultrasound Research.” In Applications in Electronics Pervading Industry, Environment and Society, In LECTURE NOTES IN ELECTRICAL ENGINEERING pp.59-64, 2014
(DOI:10.1007/978-3-319-20227-3_8)

L. Demi, A. Ramalli, G. Giannini and M. Mischi, “In Vitro and in Vivo tissue harmonic images obtained with parallel transmit beamforming by means of orthogonal frequency division multiplexing [Correspondence]” Ultrasonics, Ferroelectrics, and Frequency Control, IEEE Transactions on, Volume: 62, Issue: 1 Pages: 230 - 235, : January 2015
(DOI: 10.1109/TUFFC.2014.006599)

L. Demi, G. Giannini, A. Ramalli, P. Tortoli, and M. Mischi “Multi-Focus Tissue Harmonic Images Obtained with Parallel Transmit Beamforming by means of Orthogonal Frequency Division Multiplexing” Ultrasonics Symposium Proceedings (IUS), 2015 IEEE International, October 2015
(DOI: 10.1109/ULTSYM.2015.0501

E. Boni, L. Bassi, A. Dallai, G. Giannini, F. Guidi, V. Meacci, R. Matera, A. Ramalli, S. Ricci, M. Scaringella, P. Tortoli and J. Viti “ULA-OP 256: a portable high-performance research scanner” . Ultrasonics Symposium Proceedings (IUS), 2015 IEEE International, October 2015
(DOI: 10.1109/ULTSYM.2015.0145)