UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Biomedica

Implementazione della trasformata di Fourier veloce su un microcontrollore PIC18F per la fonocardiografia interpretativa

Fast Fourier transform implementation on a PIC18F microcontroller for interpretative phonocardiography

Relatore:
Prof. **Piva Francesco**

Tesi di Laurea di:
Paolo Veri

A.A. 2020 / 2021

INDICE

1. INTRODUZIONE

1.1. FONOCARDIOGRAFIA (FCG)

L'auscultazione è una componente fondamentale della diagnosi cardiaca, fornisce infatti informazioni sull'integrità e la funzione del cuore, delle valvole e delle prestazioni emodinamiche.

Per tale indagine si utilizza, solitamente, il classico stetoscopio e, nonostante l'orecchio umano sia un apparato prodigioso, è uno strumento molto povero per l'auscultazione cardiaca. Per questo tipo di misurazioni, infatti, è preferibile utilizzare uno strumento che sia in grado di registrare suoni, salvare i dati in memoria per eventuali analisi successive e rappresentare graficamente un tracciato che possa essere facilmente interpretato. Questo migliorerebbe l'accuratezza di questa tecnica indispensabile e offrirebbe un supporto maggiore a chi deve eseguire tali misurazioni. In risposta a ciò è stata sviluppata nel corso degli anni la fonocardiografia: una tecnica diagnostica non invasiva che prevede la registrazione dei toni cardiaci volta a migliorare i risultati ottenuti con il tradizionale stetoscopio acustico.

In sostanza, si acquisiscono i suoni delle vibrazioni della superficie toracica che vengono generate nelle strutture cardiache e trasmesse alla parete toracica stessa.

1.2. STUDI SULLA FONOCARDIOGRAFIA DIGITALE

Secondo uno studio condotto dalla University of Texas, riguardo la fonocardiografia digitale, una delle applicazioni più specifiche per questo tipo di strumentazione riguarda il poter individuare malfunzionamenti delle valvole cardiache.

Di seguito vengono riportati due casi: un tono cardiaco normale e un suono cardiaco anomalo dovuto ad un "rigurgito mitralico", che si verifica quando la valvola mitrale non si chiude completamente ed il sangue refluisce nell'atrio sinistro. Per entrambi abbiamo il segnale nel tempo e il corrispettivo spettro di frequenza.

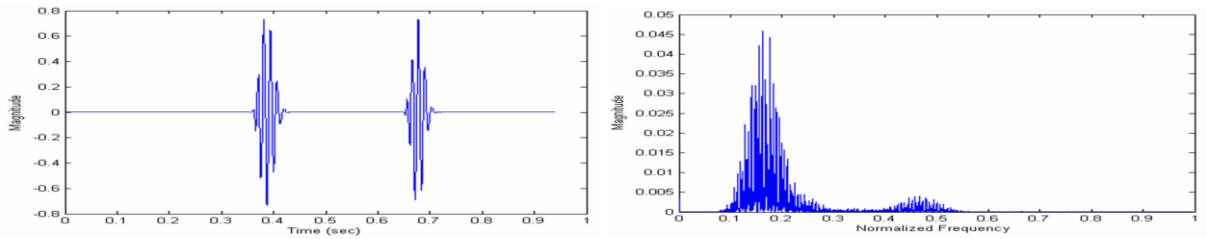

Figura 1: Tono cardiaco di un soggetto rappresentato sano nel tempo e in frequenza.

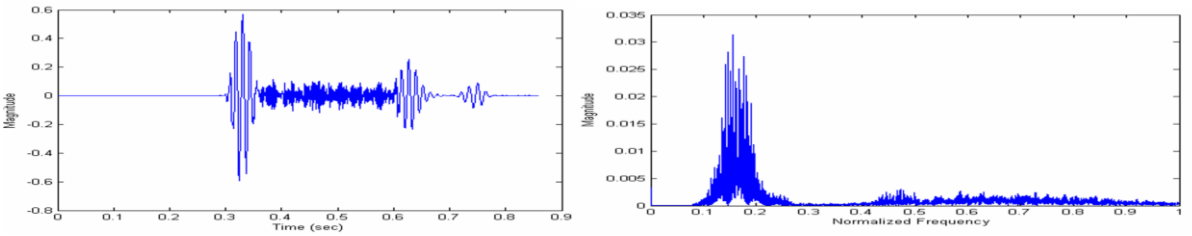

Figura 2: Tono cardiaco di un soggetto malato rappresentato nel tempo e in frequenza.

Nei grafici viene riportato un intero periodo temporale del segnale, il primo suono nel tempo è dovuto alla chiusura delle valvole mitrale e della valvola tricuspide, mentre il secondo è dovuto alla chiusura delle valvole aortica e polmonare.

I toni cardiaci sono segnali periodici di periodo temporale pari a circa 0,8 s in condizioni fisiologiche. Si nota che nel caso del tono normale si hanno componenti in frequenza nei range di 30-150Hz e 200-280Hz. In caso di anomalie, specialmente come quella presentata, le componenti in frequenza arrivano anche oltre i 500Hz. Per una più corretta analisi gli stessi dati sono stati rappresentati in due spettrogrammi tridimensionali dove negli assi sono riportati: il tempo, la frequenza e l'ampiezza.

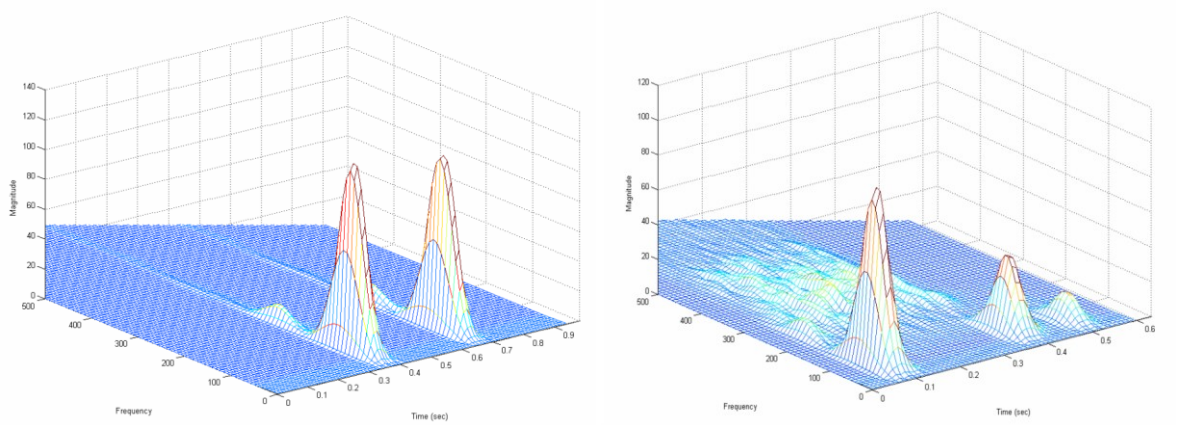

Figura 3: Spettrogrammi a confronto di due toni cardiaci, uno sano e uno malato.

Da tali spettrogrammi si evince chiaramente che il range 0-250Hz è quello per le frequenze di un tono cardiaco normale, componenti oltre questa soglia descrivono un comportamento

anomalo. Un fonocardiografo digitale permette quindi, confrontando i due segnali nel tempo e quello nella frequenza, non solo di individuare un'anomalia nei toni cardiaci ma anche di quantificare l'entità di tale anomalia e l'istante di tempo preciso in cui questa avviene.

2. SCOPO

Gli ultimi studi a favore della fonocardiografia risalgono ormai agli anni '80 circa. Nel corso degli anni successivi si sono riscontrate innumerevoli innovazione nell'ambito della ricerca, sia in campo medico che in campo ingegneristico. In attesa che la fonocardiografia venga migliorata con l'implementazione di queste nuove scoperte fatte in ambito medico, la bioingegneria dovrebbe preoccuparsi di offrire un miglior supporto con strumentazioni sempre più accurate. L'obbiettivo principale di questo lavoro di tesi è cercare di offrire proprio il supporto ingegneristico di cui necessita questa tecnica di diagnostica con grandissimo potenziale. Su tali necessità verrà sviluppato e realizzato un dispositivo di rilevazione ad hoc impiegando una componentistica con adeguate caratteristiche tecnologiche. Lo strumento permetterà di rilevare ad ogni istante di tempo i valori dei suoni cardiaci, il segnale di interesse sarà poi campionato, convertito in frequenza e rappresentato su un display grafico, come risultato si avrà due tracciati istante per istante, uno del segnale originale nel tempo e l'altro dello spettro di frequenza del segnale. In un'eventuale misurazione, quindi, si avrà il tracciato fonocardiografico del paziente sia nel dominio del tempo che in quello della frequenza.

3. MATERIALI E METODI

3.1. MICROCONTROLLORE PIC

Un microcontrollore (MCU ovvero MicroController Unit) è un dispositivo elettronico integrato su un singolo circuito elettronico. È composto da:

- CPU (Central Processing Unit)
- Memoria di programma: ROM, EPROM, FLASH

- Memoria dati: RAM e EEPROM
- Oscillatore interno o esterno
- Porte di I/O configurabili
- Gestione Interrupt
- Moduli aggiuntivi (interfacce o moduli di comunicazione)

I PIC sono una famiglia di microcontrollori prodotti dalla Microchip Technology, sono progettati per interagire tramite un programma residente nella propria memoria interna e mediante l'uso di pin configurabili dal programmatore.
Un microcontrollore rappresenta, di fatto, un computer vero e proprio dalle dimensioni alquanto ridotte e con la possibilità di aggiungere display, sensori, pulsanti o altri componenti hardware. È semplice intuire che per un dispositivo di questo tipo dunque le applicazioni di tipo pratico sono quasi infinite.
La scelta accurata delle componenti aggiuntive e il corretto settaggio del microcontrollore stesso permettono di realizzare uno qualunque dei dispositivi di ambito biomedico.

3.2. REGISTRI E MODULI

Tra i modelli più recenti e più diffusi c'è sicuramente la famiglia dei PIC 18F. Hanno una CPU a 8bit, dispone di un moltiplicatore hardware per un calcolo più rapido degli algoritmi di controllo e interfacce di comunicazione: USART, SPI, I^2C, CAN e USB.
Il modello utilizzato è il 18F4550 a 40pin disponibili.
Ad ogni pin è associata una porta I/O che sono quindi programmabili, a meno di alcune porte specifiche. Tolte, infatti, le porte destinate all'oscillatore esterno e quelle destinate all'alimentazione Vdd e Vss, sono tutte programmabili come ingresso o uscita grazie a degli appositi registri. Altri pin invece sono multiplexati con una funzione alternativa, per cui quando una periferica è abilitata quel pin non può essere usato come pin I/O generico.
Ogni porta ha tre registri per il suo funzionamento:

- TRISregister
- PORTregister
- LATregister

Il registro TRIS organizza la direzione dei dati, l'impostazione di un bit a 1 di questo registro renderà il pin corrispondente un ingresso, al contrario impostare tale bit a 0 corrisponde a

rendere il pin un'uscita. I registri PORT e LAT, invece, in scrittura sono la stessa cosa, è infatti possibile scrivere un dato indifferentemente sul registro LAT o sul registro PORT.
Mentre, la lettura del registro PORT restituisce il dato letto sul pin del microcontrollore, e quindi influenzato da fattori esterni, la lettura del registro LAT restituisce il dato presente sul latch di scrittura.
Altri moduli di rilevante importanza ai fini dell'utilizzo di questo microcontrollore sono:

• *Modulo SPI*

SPI, interfaccia periferica seriale, è la modalità che consente di sincronizzare 8bit di dati trasmessi e ricevuti simultaneamente. Alcuni modelli di PIC hanno più di una interfaccia seriale, il 18F4550 ne ha solo una e per questo non c'è necessità di specificare al compilatore a quale SPI si fa riferimento. I pin utilizzati per questa configurazione sono solitamente quattro: SDO che rappresenta l'uscita seriale, SDI che è l'ingresso seriale e SCK orologio seriale e per finire la linea SS conosciuta anche come CS è utilizzata per selezionare lo slave a cui trasferire i dati.
I registri che ne permettono il controllo sono:

- SSPCON1
- SSPSTAT
- SSPBUF
- SSPSR

Il primo e il secondo sono rispettivamente il registro di controllo e il registro di stato riferiti a tale interfaccia, permettono la configurazione del modulo per il trasferimento dei dati in modalità SPI. SSPSR, registro a scorrimento, è il registro utilizzato per spostare i dati. SSPBUF è il registro del buffer, contiene il byte ricevuto o da trasmettere. Quando un byte di dati è ricevuto da uno dei due dispositivi, è copiato nel registro SSPBUF. Se dei dati devono essere inviati, allora questo registro è caricato dal programma. Dopo essere stato caricato con il byte da inviare il registro SSPBUF trasferisce il suo contenuto sul registro SSPSR. Quest'ultimo non è quindi direttamente accessibile ma può essere letto e scritto solo attraverso il registro SSPBUF.

• *Modulo I2C*

I2C, circuito inter-integrato, permette la comunicazione di dati tra due o più dispositivi I2C utilizzando un bus a due fili.

Per il trasferimento dei dati vengono utilizzati due pin:

- SDA
- SCL

Il primo è la linea dati seriale che viene quindi usata per l'effettivo passaggio di informazioni, il secondo è l'orologio seriale. Sia in modalità SPI che I2C bisogna impostare il controllore come master o come slave.

Il dispositivo master è semplicemente il dispositivo che controlla il bus in un certo istante; tale dispositivo controlla il segnale di Clock e genera i segnali di START e di STOP. I dispositivi slave semplicemente "ascoltano" il bus ricevendo dati dal master o inviandone qualora questo ne faccia loro richiesta.

• *Clock*

Il Clock è l'orologio di sistema, dà la base dei tempi necessaria per mantenere il sincronismo fra le operazioni imponendo che l'esecuzione di ogni operazione impieghi un tempo al massimo uguale al periodo di clock. Il segnale di clock è un'onda quadra con una certa frequenza che viene generata da un oscillatore al quarzo piezoelettrico. Ogni oscillatore si differenzia da un altro per la massima frequenza raggiungibile. Il PIC in questione ha a disposizione due diverse sorgenti di clock: una è l'oscillatore interno con frequenza di 8MHz, e l'altra è data dalla possibilità di aggiungere oscillatori esterni con frequenze appositamente scelte. Un eventuale oscillatore aggiuntivo va collegato a specifici pin che sono OSC1 e OSC2. È sempre possibile scegliere il clock e la frequenza desiderata grazie al settaggio di appositi registri, in particolare OSCCON e OSCTUNE, che applicano operazioni di moltiplicazione o divisione, a seconda dei casi, per coefficienti standard. Grazie a ciò è possibile abbassare o alzare la frequenza, che altrimenti sarebbe sempre uguale a quella dell'oscillatore stesso.

• *Timer*

Timer, dal punto di vista circuitale è un semplice contatore binario. Il conteggio riguarda però i periodi di clock del processore o multipli di questa quantità. Il timer è dotato di un registro di configurazione, che permette al programmatore di fissare, come minimo, la frequenza del conteggio, ma spesso anche altri particolari relativi alla modalità di conteggio. Al raggiungimento del valore impostato il timer può generare una richiesta di interruzione oppure fissare direttamente lo stato di un bit su una porta di uscita, senza richiedere l'intervento del processore.

• *Modulo PWM*

PWM, modulazione di larghezza d'impulso, è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dal rapporto tra la durata dell'impulso positivo e dell'intero periodo. Utilizza un treno di impulsi rettangolare la cui modulazione si traduce, quindi, nel valore medio della sequenza di impulsi.

I dispositivi PIC hanno tutti due moduli CCP (Capture / Compare / PWM). Ogni modulo contiene un registro a 16 bit, che può funzionare come un registro di cattura a 16 bit, un registro di confronto a 16 bit o un registro del ciclo di lavoro master o slave PWM che è principalmente controllato dai moduli timer.

• *Modulo ADC*

ADC, convertitore da analogico a digitale, è il mezzo principale con cui un segnale analogico viene convertito in dati digitali che possono essere elaborati facilmente utilizzando un computer digitale.

Il PIC18F4550 ha 13 canali, il che significa che 13 segnali di ingresso analogici possono essere convertiti simultaneamente utilizzando il modulo. Sono disponibili 8 ingressi di clock selezionabili per la conversione e il modulo può essere configurato in modalità di attivazione automatica. Inoltre, questo PIC ha un modulo ADC ad approssimazione successiva con una risoluzione di 10 bit. L'approssimazione successiva ADC genera una serie di codici digitali ciascuno corrispondente ad un livello analogico fisso con un contatore interno da confrontare con il segnale analogico in conversione. La generazione viene interrotta quando il livello analogico diventa maggiore del segnale analogico. Il codice digitale corrispondente al livello analogico è la rappresentazione digitale desiderata del segnale analogico. Per un ADC con risoluzione a 10 bit, è possibile dividere fino a 1024 (2^{10}) tensioni.

Questo PIC ha due ingressi di tensione di riferimento, Vref- e Vref +. Il primo è la tensione di ingresso minima dell'ingresso analogico, mentre il secondo è la tensione massima.

Ciò significa che se Vref- viene applicato al pin per la conversione, il valore digitale dopo la conversione sarà 0 e per Vref +, questo sarà il valore massimo (1023 per risoluzione a 10 bit).

In casi normali, impostiamo Vref- come 0 e Vref + come 5V.

Il modulo viene configurato scrivendo in determinati registri di funzioni speciali. Sono disponibili tre registri di controllo per il modulo ADC e due registri dei risultati:

- ADCON0

- ADCON1
- ADCON2
- ADRESH
- ADRESL

Il primo registro viene utilizzato per selezionare i canali di ingresso e controllare il processo di conversione, il secondo ed il terzo si utilizzano rispettivamente uno per selezionare il riferimento di tensione e la selezione dei pin della porta per gli ingressi analogici, e l'altro per selezionare il formato dei dati ADC, cioè le opzioni dell'orologio e il tempo di acquisizione. Il risultato della conversione che è un valore digitale a 10 bit viene memorizzato nei registri ADRESH e ADRESL nel formato prescritto specificato dalla configurazione.

• *Modulo USB*

USB, i PIC18F4550 sono dotati di controller USB compatibile con lo standard 2.0. Supportano le velocità Low Speed (1.5Mbits/s) e Full Speed (12Mbits/s).

Sono caratterizzati da una frequenza di lavoro massima 48 MHz, ottenuta tramite un moltiplicatore di frequenza a PLL interno. I dati USB si spostano attraverso uno spazio di memoria noto come USB RAM. Questa è una speciale memoria a doppia porta mappato nel normale spazio di memoria dati nei banchi da 4 fino a 7 per un totale di 1Kbyte.

Il funzionamento del modulo USB è configurato e gestito attraverso innumerevoli registri, i principali sono:

- UCON
- UCFG
- USTAT
- UADDR
- UFRMH e UFRML

Il registro di controllo USB, UCON, contiene i bit necessari per controllare il comportamento del modulo durante i trasferimenti, il secondo registro, UCFG, è il registro della configurazione USB Prima di comunicare tramite USB, il modulo l'hardware interno e / o esterno associato deve essere configurato. La maggior parte della configurazione viene eseguita con l'utilizzo di tale registro.

USTAT è il registro di stato USB, riporta lo stato dei trasferimenti cioè contiene valore e direzione dell'informazione e rappresenta, di fatto, una finestra di lettura. Il registro UADDR è il registro degli indirizzi USB, contiene l'indirizzo a 7 bit del dispositivo USB. Tale indirizzo è

assegnato al momento dell'enumerazione del dispositivo da parte dell'host ed è univoco tra tutti i dispositivi connessi all'host in quel momento.
Gli ultimi due registri UFRMH e UFRML sono i registri del numero di frame, contengono rispettivamente la parte alta e bassa del numero di frame ricevuti o trasmessi dopo il segnale di Start Of Frame.

3.3. COMPILATORE E PROGRAMMATORE HARDWARE

Per programmare un PIC è necessario l'utilizzo di un compilatore e di un programmatore hardware. Il compilatore è un programma informatico che traduce una serie di informazioni scritte in un determinato linguaggio di programmazione, chiamato codice sorgente, in istruzioni di un altro linguaggio, chiamato codice oggetto, che generalmente sono codici eseguibili dalla macchina.
Il programmatore invece, o più correttamente programmatore USB, è un dispositivo che facendo da ponte tra il computer e il PIC fa passare il codice esadecimale in uscita dal compilatore direttamente alla memoria interna del microcontrollore, salvando tale codice nei registri di memoria permanente appositi.
La Microchip Technology ha sviluppato a tal proposito un'applicazione software chiamata MPLAB, si tratta di un ambiente di sviluppo integrato (IDE) per la programmazione dei PIC a 8bit, 16bit e 32bit. Oggi disponibile per macOS e Linux oltre che per Windows.
MPLAB supporta diversi compilatori in continuo sviluppo, alcuni disponibili per la programmazione con linguaggio Assembly e altri in C, tra i più usati troviamo:
- MPLAB XC8
- MPLAB XC16
- MPLAB XC32

Il linguaggio di programmazione è il C per tutti i compilatori, e la versione XC32 dispone anche del C++.
In alternativa per la programmazione in C dei PIC è disponibile anche il compilatore MikroC Pro (MikroElektronika). Il vantaggio di questo compilatore è la semplicità di utilizzo e le tante librerie già pronte a disposizione, vantaggi che rendono però questo software a pagamento, mentre MPLAB è completamente gratuito. MikroC Pro è disponibile solo per Windows. Per quanto riguarda i programmatori la società produttrice di PIC offre la famiglia dei PICkit utilizzati per la programmazione di microcontrollori e memorie EEPROM.

I modelli esistenti vanno dal PICkit 1, oramai completamente inutilizzato, al PICkit 4 il quale è stato rilasciato solo da qualche anno e offre una completa compatibilità con le versioni più recenti di MPLAB.
Un secondo programmatore altrettanto valido è il MikroProg for PIC, i cui driver sono disponibili solo per Windows. In ogni caso, l'utilizzo dei programmatori USB è molto semplice. Nella schermata di apertura bisogna sempre impostare il tipo di PIC che si utilizza, il programmatore poi permette di selezionare il file esadecimale di interesse poi il programmatore lo scrive nella memoria EEPROM del microcontrollore. Inoltre, tramite il programmatore si può fornire corrente alla scheda del microcontrollore per testarne il funzionamento. Per questo lavoro sono stati utilizzati entrambi i compilatori sfruttando per ognuno di essi il rispettivo programmatore hardware, il programma è stato scritto inizialmente utilizzando MikroC, poi è stato riadattato, in termini di sintassi, per poter essere compilato anche da XC8 all'interno di MPLAB.

3.4. DISPLAY GRAFICO TFT

I TFT sono una variante dei display a cristalli liquidi (LCD) che utilizzano, appunto, la tecnologia TFT (thin film transistor, cioè transistor a pellicola sottile) per migliorare le qualità dell'immagine. Sono quindi display ad alta risoluzione, di solito a colori e alcuni anche touch screen. Fortunatamente sono a basso costo e ormai disponibili in moltissimi siti online.
I display TFT possono essere pilotati tramite interfaccia SPI, precedentemente descritta, o tramite interfaccia parallela a 8bit o a 16bit.
Quest'ultima consente una maggiore velocità di trasmissione dati rispetto alla prima, per fare questo però sono necessari più collegamenti e ciò significa avere più pin occupati.
Il vantaggio, infatti, di utilizzare un'interfaccia di tipo seriale è proprio il fatto che per comunicare sono sufficienti meno pin.
Abbiamo lavorato inizialmente con un display TFT 2.4" con risoluzione grafica 320x240 con un controllore di tipo ILI9341 ad interfaccia SPI.
I pin da collegare sono: Vcc, Gnd, CS, RST, D/C, SDO, SCK, SDI
I primi due pin sono per l'alimentazione, quello di CS (chip select) è un valore emesso dal master per indicare con quale dispositivo slave si vuole comunicare, normalmente il valore è 0, mentre il valore alto 1 indica la comunicazione.

Il terzo è il pin di reset, gli altri sono rispettivamente: data/command, serial data output, serial clock che ovviamente è emesso dal master e serial data input.
Tutti i pin sono configurati come uscite, eccetto quello relativo a SDI che deve essere necessariamente un ingresso, attraverso gli appositi registri di porta precedentemente descritti.
Vista però la possibilità di poter sfruttare più pin per la connessione è stato, in secondo tempo, scelto un display TFT 3.2" con risoluzione grafica 480x320 con interfaccia parallela, questo per velocizzare il passaggio di dati ed informazioni tra il microcontrollore e il display stesso.
I pin questa volta sono: RST, WR, RS, RD, CS, Vcc_3.3V, Vcc_5V, Gnd e la porta dati formata da 8 pin.
I due Vcc e Gnd sono per l'alimentazione, con la possibilità di poter pilotare lo schermo a 3.3V o 5V. Il primo è il pin di reset, poi rispettivamente abbiamo il write operation, register select, read operation e il chip select.

4. RISULTATI

4.1. LIBRERIA GRAFICA

Un aspetto positivo di questi display è che esistono diverse librerie già pronte da includere nel proprio progetto. D'altro canto, però le librerie a disposizione sono per un limitato numero di versioni sia di display TFT che di microcontrollori. Quindi, è stato necessario riscrivere tali librerie per poterle adattare sia al PIC utilizzato sia ai driver del display. Sono stati analizzati tutti i registri, confrontandoli tra le varie schede dati dei PIC, e le variabili citate nella libreria e sono stati sostituiti così da poter sfruttare le funzioni già presenti.
Inoltre, questi display prima di poter funzionare correttamente devono essere inizializzati, cioè è necessario scrivere le impostazioni base nei registri del display prima di poter effettivamente interagire con questo. I codici di inizializzazione, ovvero il valore delle le istruzioni e gli specifici registri in cui queste vanno scritte sono disponibili nei manuali di istruzione del display. Con tale inizializzazione si impostano ad esempio il numero di righe e di colonne composte da pixel, l'intensità di illuminazione, la rotazione dello schermo e, oltre alle impostazioni grafiche si accede ai registri di accensione e spegnimento, di controllo e

molti altri ancora. È stato però necessario riscrivere le funzioni che permettono di fare questo, cioè quelle che si occupano di scrivere dati e comandi nei registri corretti. Lo schema seguente illustra la sequenza che deve essere eseguita per poter scrivere comandi e dati dal microcontrollore al controllore del display.

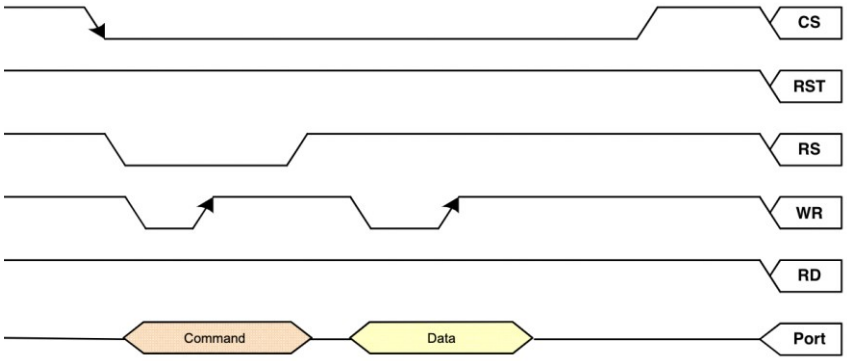

Figura 4: Sequenza di scrittura per display TFT con interfaccia parallela ad 8 bit.

Inoltre, sono state riscritte le funzioni che consentono di poter scrivere un testo sul display, quelle necessarie a puntare una finestra o un singolo pixel da dover accendere, quella che permette proprio l'accensione del pixel stesso ed è stato aggiunto successivamente un font di dimensioni 8x16 che avesse maggior leggibilità, lasciando però all'utente la possibilità di poter scrivere anche con quello precedente di dimensioni 7x5.

Di seguito sono riportati alcuni pezzi di codici della libreria grafica in questione e la sequenza di inizializzazione del display compilato in MikroC Pro.

```
/*Includo le librerie in C necessarie per poter utilizzare le
funzioni base*/

#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>

/*Associo ad ogni porta del PIC usata per la connessione con il
display in parallelo il rispettivo pin per motivi di leggibilità. I
pin in questione sono quelli per l'interfaccia parallela, viene
omessa la definizione per la porta dei dati in quanto gli 8 pin in
questione occupano l'intera porta B, per cui sarà sufficiente citare
questa.*/

#define LCD_RST          RD6_bit
#define LCD_CS           RD5_bit
#define LCD_RS           RD4_bit
#define LCD_WR           RC7_bit
```

```
#define LCD_RD            RC6_bit

/Associo ad ogni bit del registro di porta del PIC il rispettivo
pin*/

#define LCD_WR_Direction          TRISC7_bit
#define LCD_RD_Direction          TRISC6_bit
#define LCD_CS_Direction          TRISD5_bit
#define LCD_RS_Direction          TRISD4_bit
#define LCD_RST_Direction         TRISD6_bit

//Definisco le dimensioni del display
#define TFTWIDTH   320
#define TFTHEIGHT  480

//Dichiaro le variabili globali per la libreria grafica
unsigned _width = 320, _height = 480,
         cursor_x = 0, cursor_y = 0;

unsigned short rotation = 0,  txt_r = 0, txt_g = 0, txt_b = 0,
               bg_r = 0, bg_g = 0, bg_b = 0, textsize = 1,
               font_dim_X = 5, font_dim_Y = 7, N_font = 1;

/*Dichiaro la funzione necessaria ad inviare un comando ai registri
del tft, la variabile in ingresso alla funzione è un valore in
esadecimale corrispondente al numero del registro da interrogare. La
sequenza di valori assegnati ai pin del display in questo preciso
ordine è la combinazione necessaria alla scrittura di un comando sul
registro del tft.*/

void Send_Command_8bit(unsigned char Com)
{
 LCD_RS = 0;
 LCD_RD = 1;
 LCD_WR = 1;
/*Assegno alla porta dei dati per la comunicazione parallela il
valore dell'istruzione*/
 PORTB = Com;
 LCD_WR = 0;
/*Aspetto un microsecondo per permettere al display di leggere WR
come zero*/
 delay_us(1);
 LCD_WR = 1;
}

/*Dichiaro la funzione che serve ad inviare dati al registro citato
con la funzione Send_Command_8bit, la variabile in ingresso è un
valore esadecimale contenente i dati da trasmettere, questi variano
a seconda delle impostazioni del registro nel quale si vuole
scrivere. Anche in questo caso la combinazione presentata è la
sequenza che serve per poter scrivere tale dato nel registro del
display.*/

void Send_Data_8bit(unsigned char Dataa)
{
 LCD_RS = 1;
/*Assegno alla porta dei dati per la connessione parallela il valore
che voglio scrivere*/
```

```
 PORTB = Dataa;
 LCD_WR = 0;
 LCD_RD = 1;
 LCD_WR = 1;
}
```

*/*Prima di ogni funzione per l'invio dei comandi o dei dati CS viene alzato ad uno e a fine funzione viene riportato a zero, questo sempre per rispettare lo schema per la scrittura su registri del display.*/*

*/*Definisco la funzione che serve per selezionare le coordinate del singolo pixel, i valori in ingresso sono le coordinate lungo x e lungo y della posizione del pixel rispetto all'origine del sistema di riferimento che coincide con uno dei 4 spigoli dello schermo. Vengono definite le variabili che conterranno i valori in ingresso, questo perché il valore delle coordinate possono valore anche 2 byte, ma il registro può ricevere valori solo fino ad un byte. Le variabili contrassegnate con "t" riceveranno i primi 8bit, grazie all'operatore >>8 che permette lo scorrimento dei bit della variabile in ingresso di 8 posti a destra, le variabili contrassegnate con "b" riceveranno gli ultimi 8bit. Seleziono il registro con indirizzo 0x2A (set_column_andress) al quale invierò i dati per regolare il cursore_x, i primi due dati sono i due byte contenenti il valore di x da cui partirà la colonna, gli ultimi due invece sono i byte contenenti il valore di x in cui finirà la colonna. In questa funzione i due valori (inizio e fine colonna) coincidono. Seleziono il registro con indirizzo 0x2B (set_page_andress) al quale invierò i dati per regolare il cursore_y, i primi due dati sono i due byte contenenti il valore di y da cui partirà la riga, gli ultimi due invece sono i byte contenenti il valore di y in cui finirà la riga. In questa funzione i due valori (inizio e fine riga) coincidono. A fine blocco di istruzioni se, come in questo caso, è stato citato un registro per la regolazione delle dimensioni dello schermo bisogna interrogare il registro 0x00.*/*

```
void set_address(unsigned x, unsigned y) {
unsigned short xt, xb, yt, yb;
 xt = (x >> 8);
 xb = x;
 yt = (y >> 8);
 yb = y;
   LCD_CS = 0;
     Send_Command_8bit(0x2A);
     Send_Data_8bit(xt);
     Send_Data_8bit(xb);
     Send_Data_8bit(xt);
     Send_Data_8bit(xb);
     Send_Command_8bit(0x00);
   LCD_CS = 1;
     delay_us(1);

   LCD_CS = 0;
     Send_Command_8bit(0x2B);
     Send_Data_8bit(yt);
     Send_Data_8bit(yb);
     Send_Data_8bit(yt);
```

```
    Send_Data_8bit(yb);
    Send_Command_8bit(0x00);
  LCD_CS = 1;
    delay_us(1);
}
```

*/*Definisco la funzione che serve a selezionare una finestra di pixel, i valori in ingresso sono le coordinate x e y da cui far partire la finestra e le dimensioni da assegnare a questa. Le variabili x2 e y2 rappresentano il valore delle coordinate dello spigolo opposto a quello di partenza. Anche questa volta vengono selezionati gli indirizzi dei registri a in cui scrivere, questa volta le variabili contengono ciascuna 2 byte per cui lo scorrimento per permettere al registro la lettura di un byte alla volta è inclusa all'interno dell'argomento della funzione per il trasferimento dei dati.*/*

```
void setAddrWindow(unsigned x1, unsigned y1, unsigned w, unsigned h)
{

    unsigned x2 = (x1 + w - 1),
                  y2 = (y1 + h - 1);

  LCD_CS = 0;
    Send_Command_8bit(0x2A);
    Send_Data_8bit(x1 >> 8);
    Send_Data_8bit(x1);
    Send_Data_8bit(x2 >> 8);
    Send_Data_8bit(x2);
    Send_Command_8bit(0x00);
  LCD_CS = 1;
    delay_us(1);

  LCD_CS = 0;
    Send_Command_8bit(0x2B);
    Send_Data_8bit(y1 >> 8);
    Send_Data_8bit(y1);
    Send_Data_8bit(y2 >> 8);
    Send_Data_8bit(y2);
    Send_Command_8bit(0x00);
  LCD_CS = 1;
    delay_us(1);
}
```

*/*Definisco la funzione che permette di colorare un pixel, i valori in ingresso sono le coordinate e il colore, separato in tre byte diversi ognuno contenente rispettivamente la tonalità di rosso, verde e blu. I valori che questi possono assumere vanno da 0 a 255 saranno indici dell'intensità per ognuno dei tre colori. Prima viene chiamata la funzione necessaria a puntare il singolo pixel, poi al registro di indirizzo 0x2C, incaricato della scrittura in RAM del display, vengono inviati i dati contenenti i valori che determineranno il colore.*/*

```
void write_pixel(unsigned x, unsigned y, unsigned short r, unsigned
short g, unsigned short b)  {
   set_address(x, y);
 LCD_CS = 0;
```

```
   Send_Command_8bit(0x2C);
   Send_Data_8bit(r);
   Send_Data_8bit(g);
   Send_Data_8bit(b);
 LCD_CS = 1;
}

/*Vengono riportati solo due righe, coincidenti con due caratteri,
per ogni font per motivi di spazio. La matrice contenente i valori
dei caratteri viene dichiarata come costante in modo che possa
essere salvata sulla ROM e non sulla RAM, per non occupare spazio
visto che per queste matrici è sufficiente la singola lettura. Le
righe partono dal valore ASCII 32 che coincide con lo spazio. La
prima dimensione della matrice rappresenta il numero di caratteri
disponibili, mentre la seconda i byte che servono per definirlo.*/
const unsigned short font5X7[96][5] = {
                                …
    0x7e, 0x11, 0x11, 0x11, 0x7e,   //Riga n.33  "A"
                                …
    0x7c, 0x14, 0x14, 0x14, 0x08,   //Riga n.80  "p"
                                …
};

const unsigned short font8X16[94][16] ={

…
0xF0,0x1F,0xF8,0x1F,0x0C,0x01,0x06,0x01,0x06,0x01,0x0C,0x01,0xF8,0x1
F,0xF0,0x1F,   //Riga n.33 "A"

…
0x20,0x80,0xE0,0xFF,0xC0,0xFF,0x20,0x90,0x20,0x10,0x20,0x10,0xE0,0x1
F,0xC0,0x0F,  //Riga n.80 "p"

…
};

/*Viene dichiarata la funzione che serve a scrivere un carattere sul
display. Il concetto di base per la
scrittura di un carattere sul display è quello che ad ogni carattere
scritto da tastiera corrisponde un numero della tabella ASCII, se
scriviamo ad esempio "A" il compilatore assocerà a questo carattere
il valore 65, da qui scegliamo quindi la riga della nostra matrice
"font" sottraendo però il valore 32 per motivi precedentemente
spiegati, per cui il valore di riferimento sarà 33. Una volta scelta
la riga vengono accesi (in questo caso colorati) i pixel
corrispondenti ai bit con valore 1 e lasciati spenti quelli con
valore 0. Questa funzione è regolata in base ai valori impostati
prima di scrivere, cioè dimensione del testo, numero del font che si
vuole usare, colore del testo e dello sfondo. In base a ciò che si è
scelto verrà scritto il carattere li dove è stato puntato il cursore
utilizzando la funzione "write_pixel" precedentemente definita*/

void display_putc(unsigned short c) {
  unsigned short i, j,line, line1, line2;

  if (c == ' ' && cursor_x == 0 && wrap)
    return;
  if(c == '\r') {
```

```
    cursor_x = 0;
    return;
  }
  if(c == '\n') {
    cursor_y += textsize * (font_dim_Y+1);
    return;
  }

if (N_font == 1)
{
  for(i = 0; i < font_dim_X; i++ )
  {
    line = font5x7[c-32][i];
    for(j = 0; j < font_dim_Y; j++, line >>= 1)
    {
      if(line & 1)
      {
        if(textsize == 1)
          write_pixel(cursor_x + i, cursor_y + j, txt_r, txt_g,
txt_b);
        else
          display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, txt_r, txt_g, txt_b);
      }
      else
        if((bg_r != txt_r) | (bg_g != txt_g) | (bg_r != txt_b))
          {
          if(textsize == 1)
            write_pixel(cursor_x + i, cursor_y + j, bg_r, bg_g,
bg_b);
          else
            display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, bg_r, bg_g, bg_b);
          }
     }  //end for
  }  //end for
}

else if(N_font == 2)
{
  for(i = 0; i < font_dim_X; i++)
  {
    line1 = font8x16[c-32][i*2];
    line2 = font8x16[c-32][i*2+1];
    for(j = 0; j < font_dim_Y; j++, line1 >>= 1)
    {

      if(line1 & 1)
      {
        if(textsize == 1)
          write_pixel(cursor_x + i, cursor_y + j, txt_r, txt_g,
txt_b);
        else
          display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, rand(), rand(), rand());
      }
      else
        if((bg_r != txt_r) | (bg_g != txt_g) | (bg_r != txt_b))
```

```
          {
          if(textsize == 1)
            write_pixel(cursor_x + i, cursor_y + j, bg_r, bg_g,
bg_b);
          else
            display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, bg_r, bg_g, bg_b);
          }
     }  //end for

    for(j = 8; j < 15; j++, line2 >>= 1)
    {
      if(line2 & 1)
      {
        if(textsize == 1)
          write_pixel(cursor_x + i, cursor_y + j, txt_r, txt_g,
txt_b);
        else
          display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, rand(), rand(), rand());
          //display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, txt_r, txt_g, txt_b);
      }
      else
        if((bg_r != txt_r) | (bg_g != txt_g) | (bg_r != txt_b))
          {
          if(textsize == 1)
            write_pixel(cursor_x + i, cursor_y + j , bg_r, bg_g,
bg_b);
          else
            display_fillRect(cursor_x + i * textsize, cursor_y + j *
textsize, textsize, textsize, bg_r, bg_g, bg_b);
          }
     }  //end for
  }  //end for
} //end elseif

  if((bg_r != txt_r) | (bg_g != txt_g) | (bg_r != txt_b))
    {
    if(textsize == 1)  display_drawVLine(cursor_x + font_dim_X,
cursor_y, (font_dim_Y+1), bg_r, bg_g, bg_b);
    else               display_fillRect(cursor_x + font_dim_X *
textsize, cursor_y, textsize, (font_dim_Y+1) * textsize, bg_r, bg_g,
bg_b);
    }

  cursor_x += textsize * (font_dim_X+1);

  if( cursor_x > ((unsigned)display_width + textsize *
(font_dim_X+1)) )
    cursor_x = display_width;

  if (wrap && (cursor_x + (textsize * font_dim_X)) > display_width)
  {
    cursor_x = 0;
    cursor_y += textsize * (font_dim_Y+1);
    if( cursor_y > ((unsigned)display_height + textsize *
(font_dim_Y+1)) )
```

```
      cursor_y = display_height;
  }
}

/*Variabile che permette di assegnare il valore delle coordinate x e
y in ingresso alle variabili globali "cursor" che serviranno per la
funzione "Display_putc".*/

void display_setCursor(unsigned x, unsigned y) {
    cursor_x = x;
    cursor_y = y;
}

/*Funzione che permette di selezionare il font con il quale si vuole
scrivere, i valori in ingresso alla funzione sono le dimensioni del
font rappresentate da x e da y e il numero del font che può essere 1
o 2, per tutte e tre le variabili è stata usata la funzione "switch"
per permettere un valore di default nel caso la funzione venisse
omessa o uno dei valori inseriti fosse sbagliato.*/

void display_setTextFont(unsigned short x, unsigned short y,
unsigned short f) {
  switch (x) {
    case 5: font_dim_X = 5;   break;
    case 8: font_dim_X = 8;   break;
    default: font_dim_X = 5;
             }

  switch (y) {
    case 7: font_dim_Y = 7;   break;
    case 16: font_dim_Y = 16;  break;
    default: font_dim_Y = 7;
             }

  switch (f) {
    case 1: N_font = 1;   break;
    case 2: N_font = 2;   break;
    default: N_font = 1;
             }
}

/*Funzione che permette di scegliere il colore del testo e lo sfondo
relativo ad esso, i valori in ingresso sono i byte contenenti
l'intensità dei colori rosso, verde e blu per poter comporre in
colore desiderato. I primi 3 valori sono riferiti al testo mentre
gli ultimi allo sfondo. Questo permette di avere colori a 24bit.*/

void display_setTextColor(unsigned short a, unsigned short b,
unsigned short c, unsigned short d, unsigned short e, unsigned short
f) {
    txt_r = a;
    txt_g = b;
    txt_b = c;
    bg_r  = d;
    bg_g  = e;
    bg_b  = f;
}
```

```
/*Sequenza di inizializzazione. Prima è necessario un reset fisico
abbassando e alzando il pin RST display e dando qualche ms di tempo
al controllore del tft di percepire la variazione, poi vengono
inviati dati agli indirizzi dei registri specifici per poter
regolare le impostazioni di base del display. Anche in questo caso
ogni volta che viene chiamato un registro e si vuole scrivere su
questo bisogna abbassare CS a inizio istruzione e rialzarlo alla
fine, dando sempre qualche ms di tempo per permettere al display di
percepire il gradino prima che CS venga nuovamente riabbassato per
l'inizio dell'istruzione successiva. La sequenza di inizializzazione
deve essere collocata all'interno del main, e non nella libreria
grafica, prima di voler utilizzare il display; se venisse chiamata
una qualche funzione grafica prima di aver inizializzato lo schermo
questa non produrrebbe alcun risultato grafico.*/

//imposto tutti i pin come uscite
LCD_WR_Direction = 0;
LCD_RS_Direction = 0;
LCD_RD_Direction = 0;
LCD_CS_Direction = 0;
LCD_RST_Direction = 0;

LCD_RST = 0;                    //Reset fisico del display
     delay_ms(50);
LCD_RST = 1;

LCD_CS = 0;
     Send_Command_8bit(0x01);  //Software reset
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x11);  //Sleep out
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x36);  //Memory access control
     Send_Data_8bit(0xE8);
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x3A);  //Interface parallel format
     Send_Data_8bit(0x76);
LCD_CS = 1;
     delay_ms(100);

LCD_CS = 0;
     Send_Command_8bit(0x12);  //Partial mode ON
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x29);  //Display ON
LCD_CS = 1;
     delay_ms(50);
```

```
LCD_CS = 0;
     Send_Command_8bit(0x2A);  //Set cursor X
     Send_Data_8bit(0x00);
     Send_Data_8bit(0x00);
     Send_Data_8bit(0x01);
     Send_Data_8bit(0xDF);
     Send_Command_8bit(0x00);   //NOP
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x2B);  //Set cursorY
     Send_Data_8bit(0x00);
     Send_Data_8bit(0x00);
     Send_Data_8bit(0x01);
     Send_Data_8bit(0x3F);
     Send_Command_8bit(0x00);   //NOP
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x51);  // Set brightness
     Send_Data_8bit(0xFF);
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0x53);  // Set brightness control
     Send_Data_8bit(0x2C);
LCD_CS = 1;
     delay_ms(50);

LCD_CS = 0;
     Send_Command_8bit(0xB1);  //Set framerate
     Send_Data_8bit(0xB0);
     Send_Data_8bit(0x11);
LCD_CS = 1;
     delay_ms(50);

//End of Init
```

4.2. FAST FOURIER TRANSFORM (FFT)

4.2.1. Discrete Fourier Transform (DFT)

La trasformata discreta di Fourier, comunemente nota con l'acronimo DFT (Discrete Fourier Transform), risponde all'esigenza di implementare al calcolatore la trasformata di Fourier di una funzione nel tempo, la quale generalmente è descritta dalla relazione:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-i\omega t}\, dt$$

Non è, infatti, possibile far lavorare un qualunque calcolatore con un segnale continuo, ma sarà necessario prelevare un numero finito di campioni a diversi istanti temporali. L'implementazione numerica impone di modificare la definizione convenzionale di trasformata (e antitrasformata) di Fourier, al fine di ottenere una procedura di calcolo efficiente. Ciò consente il calcolo degli N campioni nel dominio della frequenza a partire dalla conoscenza degli N campioni nel dominio del tempo. La teoria che c'è alla base della conversione tempo-frequenza deve quindi essere concretizzata in un algoritmo che sia realmente in grado di trattare il segnale considerato per poterlo poi convertire nel dominio della frequenza. Diciamo che il segnale da analizzare viene prima campionato nel tempo, ciò corrisponde a moltiplicare il segnale per una sequenza campionante rappresentata da successione di delta di Dirac, o impulsi matematici. La trasformata del risultato del campionamento nel tempo è un segnale che si ottiene replicando infinite volte la trasformata di Fourier del segnale originale, ciascuna replica centrata su un multiplo della frequenza di campionamento. Per tale ragione è necessario che la frequenza di campionamento rispetti le condizioni di Nyquist, cioè che sia almeno il doppio della banda passante del segnale da campionare. Quella di Nyquist è una condizione che deve essere rispettata per evitare l'Aliasing, questo è un fenomeno di distorsione dovuto alla sovrapposizione delle repliche del segnale originale e deve essere evitato in quanto tale fenomeno porterebbe ad una irreversibile perdita di informazioni.
L'insieme dei campioni temporali definiscono una finestra di campionamento. La finestra di campionamento dovrebbe coincidere con un periodo del segnale da campionare. Ad esempio, se il periodo cardiaco dura 0,8 secondi, la finestra di campionamento dovrebbe essere di questa esatta durata. Se non lo fosse si avrebbe uno spettro di forma errata. Negli oscilloscopi questo fenomeno viene affrontato introducendo finestre di vario tipo, ad

esempio quella Hamming. Nel nostro caso è importante che sia l'operatore a stabilire esattamente l'inizio e la fine di un periodo cardiaco.
La funzione originale del tempo è approssimata con N campioni e, allo stesso modo, anche la trasformata di Fourier è approssimata con N campioni. Questi N campioni definiscono la coppia segnale-trasformata di Fourier discreta e, nel senso precisato, costituiscono un'approssimazione della coppia segnale-trasformata originale.
La DFT è descritta dalla formula:

$$A_k = \sum_{n=0}^{N-1} W_N^{kn}\, a_n$$

Dove

$$W_N^{kn} = e^{-i\frac{kn2\pi}{N}}, \quad k = 0, 1, \dots, N-1$$

Cioè ogni ampiezza in frequenza A_k è data dalla sommatoria dei valori di tutti i campioni a_n ciascuno pesato da un preciso coefficiente complesso dipendente dall'indice del campione considerato rispetto al numero totale di campioni N.

4.2.2. Algoritmo di Cooley-Tukey

La trasformata di Fourier veloce o FFT (Fast Fourier Transform) è un algoritmo che permette di concretizzare la DFT riducendone la quantità di operazioni aritmetiche. Infatti, la DFT richiede una quantità di operazioni O(N^2) mentre un algoritmo FFT ottiene, di solito, lo stesso risultato con un numero di operazioni O(N$\log(N)$).
L'algoritmo FFT più diffuso è l'algoritmo di Cooley-Tukey, il quale si basa sul principio di "divide et impera", permettendo una riduzione del numero di operazioni O($^N/_2 \log_2(N)$).
Questo principio, in informatica, indica un approccio per la risoluzione di problemi computazionali, consiste nel dividere ricorsivamente un problema in due o più "sotto-problemi" più elementari, risolti questi, poi, si combinano le soluzioni al fine di ottenere la soluzione del problema dato.
La FFT opera scomponendo un segnale nel dominio del tempo composto da N campioni in N segnali ciascuno composto da un singolo campione. Il secondo passo consiste nel calcolare le N intensità di frequenza corrispondenti a questi N segnali. Infine, gli spettri N sono uniti in un

singolo spettro di frequenza. La scomposizione avviene dividendo il segnale in due parti: una contenente solo campioni di indici pari e una con soli indici dispari. Ognuno dei due segnali sarà poi suddiviso allo stesso modo, formando così un segnale contenente campioni con indici pari ed uno composto solo da indici dispari, per entrambi i segnali; questo schema viene ripetuto finché non si ottengono N segnali costituiti tutti da un solo campione, per questo tipo di scomposizioni sono necessarie $O(\log_2(N))$ operazioni.
Il seguente schema mostra un esempio di scomposizione per la FFT nel caso di 16 campioni temporali.

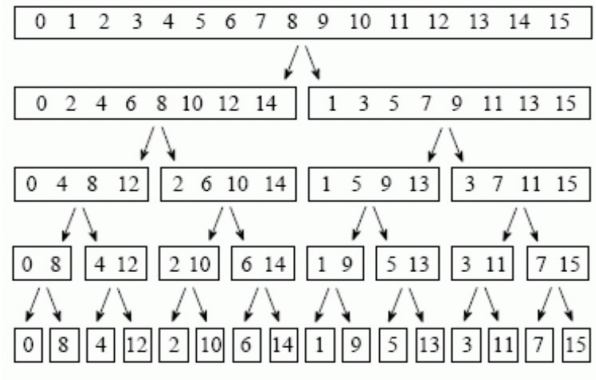

Figura 5: Schema della scomposizione per la FFT con 16 campioni.

Questa decomposizione può essere notevolmente semplificata in quanto rappresenta solamente un "riordino" dei campioni all'interno del segnale stesso, e corrisponde ad una inversione binaria degli indici, ad esempio se consideriamo un segnale di partenza composto da 16 campioni a fine schema risulterà che il campione di indice 3 (0011) viene scambiato con il campione di indice 12 (1100).
Il passo successivo nell'algoritmo FFT è trovare gli spettri di frequenza degli N campioni nel dominio del tempo e dato che ogni segnale è costituito da un solo campione questo passaggio è implicito nell'algoritmo in quanto la trasformata di una costante è un impulso di area pari al valore della costante, cioè l'ampiezza in frequenza per ogni segnale corrisponde a quello del campione nel tempo. L'ultimo passaggio della FFT consiste nel combinare gli N spettri di frequenza nell'esatto ordine inverso rispetto alla decomposizione nel dominio del tempo. Questo viene fatto associando ad ogni spettro un coefficiente dipendente sia dal numero di campioni N, sia dall'indice dello spettro. Gli N coefficienti corrispondono ai vertici di un poligono regolare inscritti nella circonferenza unitaria del piano complesso, con uno dei vertici in (1,0) e le altre radici sono, quindi, equidistanti tra loro.

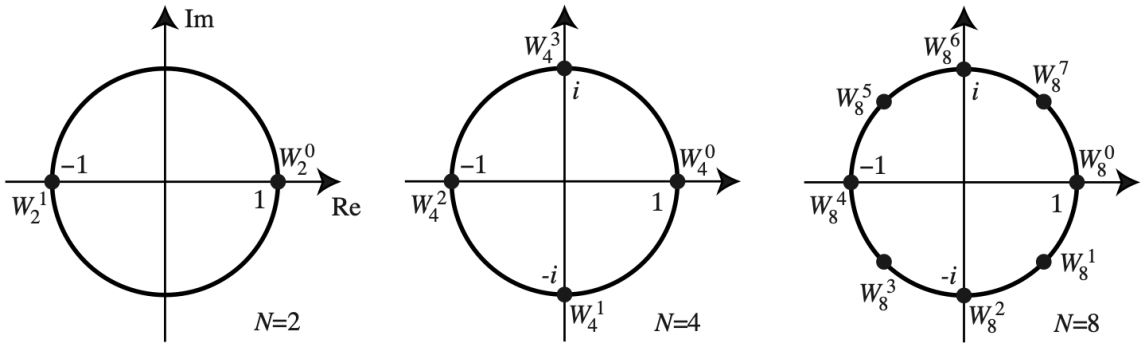

Figura 6: Rappresentazione dei coefficienti per algoritmo FFT sul piano complesso. Dove a_0 è l'ampiezza del primo campione temporale. 1, -1, i e -i sono i coefficienti riferiti alla figura 6.

Per implementare questo schema si utilizza il diagramma "a farfalla", il quale permette di capire nel dettaglio come sono combinati i vari spettri ad ogni passaggio per la costruzione dello spettro finale. I coefficienti ordinati con i nuovi indici vengono sommati a coppie, il secondo termine di ogni coppia è moltiplicato per una delle radici complesse appena trattate; i risultati saranno a loro volta sommati a coppie di due, anche questa volta il secondo termine sarà moltiplicato per il rispettivo coefficiente. Di seguito viene riportato un esempio di diagramma con N=4.

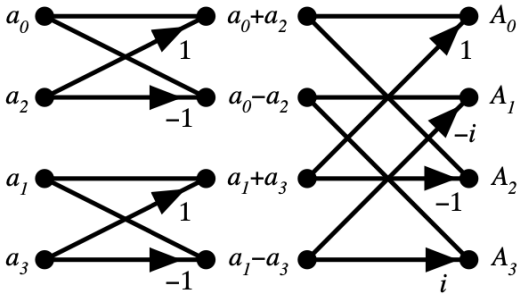

Figura 7: Diagramma a farfalla nel caso di 4 campioni.

Di seguito viene meglio spiegata una transizione del diagramma in figura 7:

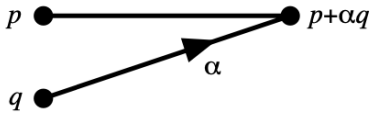

Figura 8: Struttura di base del diagramma a farfalla.

Nel caso in cui N=8 avremo, invece, uno schema più complesso che mantiene però la stessa struttura.

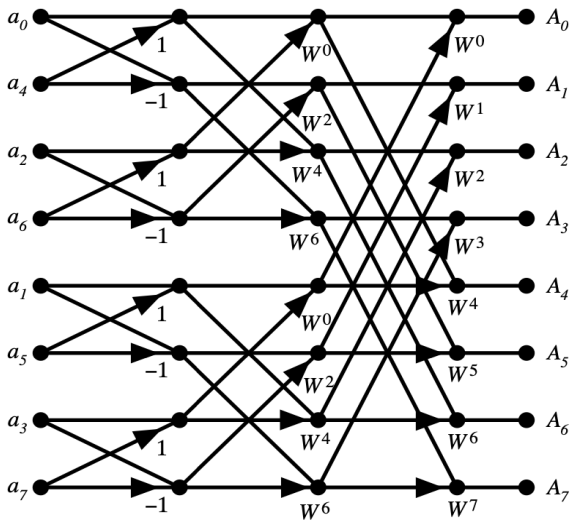

Figura 9: Diagramma a farfalla nel caso di 8 campioni.

Una volta ottenuti gli N valori complessi $(A_0, \dots, A_k)$, di questi viene ricavato il modulo tra parte reale ed immaginaria per ottenere le ampiezze delle componenti in frequenza. Con tale algoritmo siamo in grado, quindi, di ottenere lo spettro di frequenza di un segnale discreto nel tempo riducendo il numero di operazioni da svolgere rispetto a quelle necessarie per la DFT.

4.2.3. Implementazione algoritmo FFT su compilatore

In internet è possibile trovare parti di codice di programmi che implementano un algoritmo di questo tipo. Purtroppo, codici che dovrebbero assolvere alla stessa funzione, sono molto diversi tra loro, sono difficili da comprendere perché poco commentati e alcuni non funzionano affatto. Inoltre, la maggior parte dei codici non sono ottimizzati per minimizzare l'occupazione di memoria e, a volte, nemmeno per avere rapidità di calcoli.
In questo progetto specifico è stato necessario innanzitutto riscrivere il codice trovato da un linguaggio di programmazione C++ [6] ad un linguaggio C, correggendo tutti i vari errori di sintassi riscontrati. È stato necessario poi adattare il codice alle caratteristiche del compilatore, infatti, MikroC non include, nel caso della famiglia dei PIC 18f, librerie che

consentano di lavorare con i numeri complessi. Per cui tutti i valori complessi sono stati separati in parte reale e parte immaginaria. I vettori di dimensione N, quindi, sono stati trasformati in matrici di dimensioni Nx2, in cui su una colonna si trovano i valori reali e sull'altra quelli immaginari. Tale programma calcola solo metà dei coefficienti, in particolare quelli con parte immaginaria negativa. È possibile questa semplificazione grazie al fatto che i coefficienti considerati hanno il loro rispettivo opposto in posizione simmetrica rispetto al centro degli assi. Per cui basterà calcolare la metà dei coefficienti, cambiando poi di segno si ottiene il coefficiente opposto. Avendo occupato un maggior spazio di archiviazione dovuto ad un aumento del numero delle variabili e, soprattutto, a questa separazione tra parte reale e immaginaria è stato necessario ottimizzare ulteriormente il programma. Una di queste ottimizzazioni è la possibilità di calcolare solo un quarto dei coefficienti grazie alle proprietà di simmetria. Cioè nel caso di N=8 verranno calcolati solo W_0, W_1 e W_2. Il programma acquisisce in ingresso un vettore contenente i campioni del segnale nel tempo e restituisce lo stesso vettore con all'interno i moduli dei valori delle ampiezze ordinati dello spettro di frequenza.

Inoltre, per testare l'efficienza reale del programma realizzato, sono state eseguite simulazioni medianteMicrosoft Excel. Nel foglio sono stati inseriti i valori dei campioni ai diversi istanti temporali, mentre i valori in frequenza sono stati calcolati prima con la formula classica della DFT, poi implementando l'algoritmo scritto nel programma. Questa simulazione è stata ripetuta per diversi tipi di segnale in ingresso: segnale sinusoidale, onda quadra e segnale triangolare, per ogni segnale è stata fatta una simulazione a N=4, N=8 ed N=16.

Nel caso della classica DFT prima sono stati definiti i valori nel tempo, il numero dei campioni ed è stata fatta una tabella nella quale sono stati calcolati, per ogni sommatoria, i singoli valori dei campioni temporali moltiplicati per il corrispettivo coefficiente, in cui ogni colonna corrisponde all'indice del valore in frequenza da calcolare ed ogni riga all'indice del campione considerato. Un esempio esplicativo delle funzioni utilizzate per calcolare i valori della sommatoria:

=COMP.PRODOTTO(COMPLESSO($C3;0);COMP.EXP(COMPLESSO(0;
-2*PI.GRECO()*E3*$D5/$D$2)))

Questa è una delle equazioni presenti nelle celle della tabella costruita, i valori reali nel tempo sono prima convertiti in numeri complessi con parte immaginaria uguale a zero grazie alla funzione "COMPLESSO", operazione necessaria dovuta al fatto che le funzioni per i

numeri complessi accettano solo valori complessi, con la stessa funzione si definisce l'esponente del rispettivo coefficiente indicando, stavolta, come parte reale zero e come parte immaginaria $-\frac{kn2\pi}{N}i$, utilizzando la funzione "COMP.EXP" viene costruito l'esponenziale e quindi il coefficiente vero e proprio, e viene moltiplicato per il valore temporale utilizzando la funzione "COMP.PRODOTTO". Gli elementi di ogni colonna vengono sommati grazie alla funzione "COMP.SOMMA" e viene poi fatto il modulo del risultato usando la funzione "COMP.MODULO", con i valori finali viene poi costruito un grafico.
Per il caso dell'implementazione dell'algoritmo, invece, sono stati calcolati prima i coefficienti sfruttando il fatto che ogni coefficiente di indice n vale proprio W_1^n, i valori in ampiezza vengono calcolati un passaggio alla volta sfruttando le stesse funzioni per lavorare con i numeri complessi riferite alla DFT e implementando lo schema "a farfalla" precedentemente visto. Ad ogni passaggio, quindi, i valori nelle n posizioni vengono continuamente aggiornati e sommati con quelli delle altre posizioni moltiplicati per uno dei coefficienti calcolati in partenza, alla fine dello schema viene calcolato il modulo dei valori ottenuti per avere le ampiezze in frequenza; queste sono poi inserite in un grafico che possa descrivere lo spettro di frequenza del segnale.
Di seguito sono riportati alcuni esempi grafici ottenuti nelle simulazioni fatte con 16 campioni, nella realtà servono molti più campioni per una trasformata come questa ma ai fini didattici è un numero sufficiente per valutare la validità e l'accuratezza di questo algoritmo.

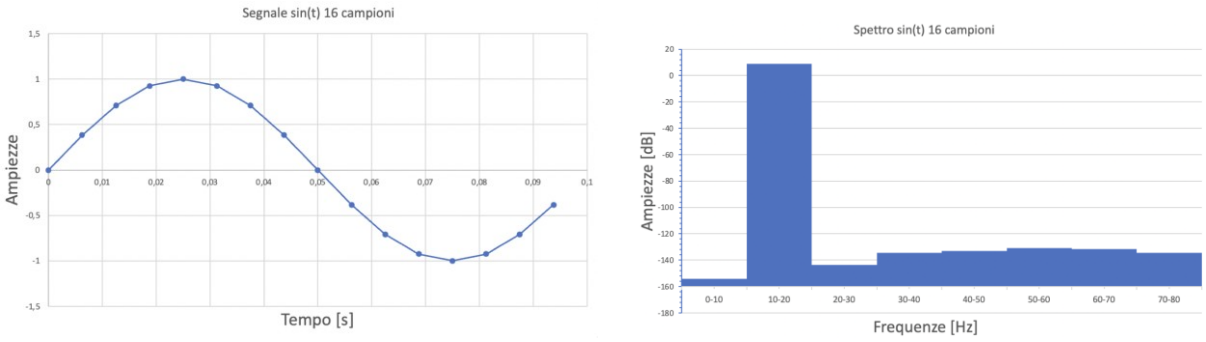

Figura 10: Sinusoide rappresentata nel tempo e corrispettivo spettro di frequenza.

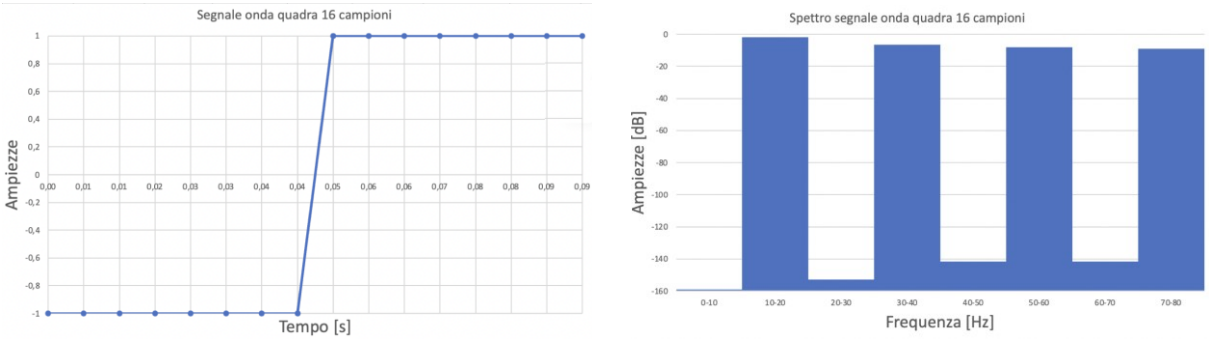

Figura 11: Segnale rettangolare rappresentato nel tempo e corrispettivo spettro di frequenza

Dimostrata l'efficienza del nuovo algoritmo è stato poi necessario controllare che quest'ultimo venisse realmente compilato da un editor con linguaggio di programmazione C, questo è stato fatto sfruttando un editor online che stampasse i valori finali del modulo delle ampiezze in frequenza a partire, sempre, da i valori dei campioni nel tempo. I risultati ottenuti sono stati identici a quelli della simulazione fatta in Excel; da tale conclusione possiamo dimostrare la correttezza di questo programma come affidabile algoritmo per la semplificazione della formula generale della DFT.

Di seguito viene riportata la libreria che permette di implementare su microcontrollore PIC la FFT.

```
/*Includo le librerie necessarie per poter chiamare funzioni utili
alla libreria*/
#include <stdio.h>
#include <math.h>

//Dichiaro le variabili globali, Nc è il numero di campioni.
const unsigned Nc = 128;
const double PI = 3.141592655358;
double h, b;

/*Dichiaro l'operatore che consente di calcolare il logaritmo in
base due del numero di campioni, l'operatore >>=1 rappresenta di
fatto una divisione per due, l'indice i fa da contatore e il
risultato dell'operazione è il numero di volte che è stato possibile
divedere per due il numero di campioni meno uno perché il contatore,
spostando ogni bit di una posizione alla volta conteggia anche il
passaggio da 1 a 0.*/

unsigned Log2()
{
  unsigned k = Nc;
  unsigned i = 0;
  while(k) {
```

```
    k >>= 1;
    i++;
  }
  return i - 1;
}

/*Dichiaro l'operatore che permette l'inversione binaria degli
indici del vettore dei campioni. Questo operatore compara log in
base due di Nc volte bit a bit l'indice n con il valore 1 spostato a
sinistra ogni volta di una posizione in più cosi da controllare ogni
bit di n, una volta individuati gli uni e gli zeri il valore p viene
aggiornato di volta in volta per costruire il binario inverso
rispetto a quello di partenza. Es:  0011 --> 1100,  1000 --> 0001.
*/

unsigned reverse(unsigned Log_N, unsigned n)
sinistra per testare le posizioni di n
{
  unsigned j, p = 0;
  unsigned limite = Log_N;
  for(j = 1; j <= limite; j++)
    {
      if(n & (1 << (limite - j)))
           p = p | 1 << (j - 1);
    }
  return p;
}

/*Definisco la funzione che permette di ordinare i campioni secondo
un ordine di inversione binaria. Il valore in ingresso alla funzione
è il puntatore della matrice contenente il valore dei campioni nel
tempo. Per ogni indice delle colonne del vettore viene creato un
nuovo indice secondo inversione binaria sfruttando l'operatore
"reverse" precedentemente definito, i due indici vengono confrontati
ad ogni ciclo e solo se i valori di questi sono diversi allora il
valore nella cella di indice "i" viene messo nella cella di indice
"nuovo_indice" e viceversa, questo viene fatto sfruttando una
variabile di appoggio per non perdere dati durante l'esecuzione. La
seconda condizione dell'if serve ad evitare che la funzione
sostituisca nuovamente i valori che sono già stati ordinati,
evitando quindi che venga restituito il vettore di partenza così
come era entrato nella funzione. Infatti, per una coppia di indici è
sufficiente che i valori vengano sostituiti una sola volta. Ad
esempio, se l'indice 3 si trova in coppia con 12 quando la funzione
arriverà all'indice 3 scambierà i valori con la cella 12 e quando
arriverà all'indice 12 non avverrà alcuno scambio.*/

void ordina(int *lettura)
{
  unsigned i, temp, nuovo_indice;
  unsigned Log_N = Log2();

for(i = 0; i < Nc; i++)
  {
    nuovo_indice = reverse(Log_N, i);
    if ((i != nuovo_indice) && (i < nuovo_indice))
     {
      temp = lettura[i];
```

```
      lettura[i] = lettura[nuovo_indice];
      lettura[nuovo_indice] = temp;
     }
  }
}
```

*/*Dichiaro la funzione che permette di calcolare il prodotto tra due numeri complessi, le variabili in ingresso alla funzione sono la parte reale e quella immaginaria dei due numeri complessi di cui si vuol fare il prodotto. Il valore della parte reale e quello della parte immaginaria del risultato sono poi assegnate rispettivamente alle variabili globali "h" e "b"*/*

```
void prodotto_c(double re1, double im1, double re2, double im2)
{
  h = (re1 * re2) - (im1 * im2);         //parte reale
  b = (re1 * im2) + (im1 * re2);        //parte immaginaria
}
```

*/*Dichiaro la funzione che permette di elevare un numero complesso ad un certo esponente "expp", se questo vale zero il risultato sarà la coppia di valori 1 e 0, se vale 1 la funzione restituisce i valori in ingresso, altrimenti verrà fatto il prodotto del numero complesso per se stesso expp-1 volte sfruttando la funzione "prodotto_c".*/*

```
//(a+ib)^expp
void pot_c(double re, double im, unsigned expp)
{
  unsigned k;

  if (expp == 0)
   {
    h = 1;
    b = 0;
   }
  else
   {
    h = re;
    b = im;
    for(k=1; k < (expp); k++)
     {
       prodotto_c(h, b, re, im);
     }
   }
}
```

*/*Dichiaro la funzione che permette di ottenere le ampiezze in frequenza a partire dai valori dei campioni temporali. Vengono inizialmente definite le variabili che serviranno per la conversione in frequenza, la matrice "W" in particolare è quella che conterrà i coefficienti, non lavoriamo più con un vettore ad una singola colonna ma con una matrice che ne contiene due perché la prima sarà destinata ai valori reali mentre la seconda a quelli immaginari. Il vettore dei campioni viene per prima cosa ordinato secondo inversione binaria, alla prima colonna di "f" vengono attribuiti i valori riordinati, essendo questi puramente reali alla seconda colonna verranno attribuiti solo valori nulli, inoltre è necessaria*

*la conversione da valori associati alla variabile di tipo "int" alla variabile di tipo "double", visto che nel processo di trasformazione quasi sicuramente i valori non rimarranno più interi. Vengono poi calcolati i coefficienti, divisi sempre in parte reale e immaginaria, il primo ha sempre come coppia di valori 1,0; il secondo può essere calcolato dando alla parte reale il valore di cos(-2 * PI / Nc) e a quella immaginaria il valore cin(-2 * PI / Nc). Gli altri saranno calcolati come esponenziali del coefficiente 1. I coefficienti calcolati saranno solo quelli del quarto quadrante perché sfruttando le dovute simmetrie si potranno calcolare gli altri a partire da questi. Per ogni passaggio di costruzione (pari ad log2Nc con Nc numero di campioni) gli indici dei campioni vengono confrontati bit a bit con un valore n, il quale parte da 1 e raddoppia ad ogni ciclo, se almeno un valore 1 coincide tra i due allora si passa al campione successivo e solo se non vengono trovati valori 1 nella stessa posizione allora i valori temporali vengono sommati con il rispettivo valore associato (secondo lo schema a farfalla presentato in precedenza) moltiplicato per il rispettivo coefficiente. I valori aggiornati vengono scritti nella matrice "f" ad ogni passaggio, alla fine di questo processo nella matrice si trovano i valori delle ampiezze in frequenza. Di ognuno di questi si calcola il modulo (essendo valori complessi), si divide per il numero di campioni per recuperare l'amplificazione che questi subiscono a causa dell'algoritmo stesso, si fa il logaritmo in base dieci del valore ottenuto e si moltiplica per 10 così da ottenere le ampiezze in dB. I singoli valori ottenuti vengono poi convertiti in interi e restituiti al vettore temporale di partenza. In questo modo la funzione prendendo in ingresso il vettore contenente i valori temporali restituisce lo stesso vettore ma con le ampiezze in dB e riordinate secondo l'algoritmo FFT, ottenendo così le ampiezze dello spettro di frequenza del segnale nel tempo.*/*

```
void transform(int *lettura)
{

  double t_re, t_im, Temp_Re, Temp_Im;
  double f[Nc][2];          //matrice f_re e f_im, campioni temporali
  double W[(Nc/4)+1][2];    //matrice dei coefficienti
  unsigned t, i, j, z, n = 1, a = Nc / 2,

/*inverte l'ordine dei bit nel byte che costituisce l'indice dei
campioni temporali*/
  ordina(lettura);

  for(i = 0; i < Nc; i++)
  {
    f[i][0] = (double) lettura[i];  //da int a double
    f[i][1] = 0;
  }

  W[0][0] = 1; //Re
  W[0][1] = 0; //Im

  W[1][0] = cos(-2 * PI / Nc);  //Re
  W[1][1] = sin(-2 * PI / Nc);  //Im

  for(i = 2; i < (Nc / 4 + 1); i++)
  {
```

```
    pot_c(W[1][0], W[1][1], i);
    W[i][0] = h;
    W[i][1] = b;
  }

  for(j = 0; j < Log_N; j++)
  {
   for(i = 0; i < Nc; i++)
    {
      if(!(i & n))
      {
        t_re = f[i][0];  //parte Re di a0
        t_im = f[i][1];  //parte Im di a0

        z = (i * a) % (n * a);   // "%" resto della divisione

        if (z > (Nc / 4))
            {
                Temp_Re = -1 * W[Nc/2-z][0];
                Temp_Im = W[Nc/2-z][1];
            }
        else
            {
                Temp_Re = W[z][0];
                Temp_Im = W[z][1];
            }

        prodotto_c(Temp_Re, Temp_Im, f[i + n][0], f[i + n][1]);
        f[i][0] = t_re + h;
        f[i][1] = t_im + b;
        f[i + n][0] = t_re - h;
        f[i + n][1] = t_im - b;
      } //if
    }  // for
    n = n * 2;
    a = a / 2;
  }  //for

for(t = 0; t < Nc; t++)
  {
    t_re = f[t][0];
    t_im = f[t][1];
    lettura[t] =  (int) 10*(log10(sqrt(pow(t_re, 2) + pow(t_im,
2))/Nc));
  }
}

/*Viene dichiarata la funzione da chiamare per eseguire la FFT,
questa prende in ingresso il vettore dei campioni temporali e un
coefficiente intero, viene chiamata la funzione transform e poi i
valori in frequenza del vettore ordinato vengono moltiplicati ognuno
per il coefficiente "x" per migliorarne la visibilità sul display
grafico qualora dovessero essere valori troppo piccoli e quindi
difficilmente individuabili sullo schermo.*/

void FFT(int *lettura, int x)
{
  int m;
```

```
  transform(lettura);

  for(m = 0; m < Nc; m++)
  {
    lettura[m] *= x;
  }
}

//End of FFT Library
```

4.3. MEMORIA ESTERNA

Da questo lavoro è emersa, principalmente, la necessità di avere una quantità di memoria RAM più consistente per poter acquisire un maggior numero di campioni temporali. È stato, infatti, stimato che per un vettore contenente, ad esempio, 1024 campioni temporali, conteggiando anche le matrici necessarie ai fini del calcolo in frequenza del segnale discreto, sono necessari almeno 12kB di memoria. Per cui mi sono reso conto della necessità di utilizzare un microcontrollore PIC che abbia maggior spazio a disposizione per contenere variabili volatili. Se non si è in grado di soddisfare questa condizione si deve scegliere un numero di campioni più basso, ovviamente a discapito dell'accuratezza dello strumento, oppure si deve utilizzare una memoria esterna.

La SRAM, acronimo di Static Random Access Memory, è un tipo di RAM volatile la quale consente di mantenere le informazioni per un tempo teoricamente infinito. Le operazioni che questa memoria esterna può eseguire sono poche e semplici. Essa è caratterizzata da tre diversi stati: (i) quello di standby, ovvero il momento in cui il circuito è inattivo, (ii) lo stato di lettura, in cui i dati sono richiesti dal microcontrollore il quale in qualità di master controlla la memoria esterna identificata come slave e (iii) lo stato di scrittura, il quale comporta un aggiornamento dei contenuti all'interno della memoria esterna. Le SRAM hanno bassi consumi, specialmente in condizioni statiche, ma a causa di queste variazioni di stato che devono essere continuamente eseguite per svolgere le funzioni richieste, l'esecuzione del programma viene rallentata. La Microchip Technology offre, ovviamente, moltissimi modelli, i quali si differenziano principalmente per la tensione di alimentazione, la frequenza di clock, la velocità di acquisizione dei dati e di lettura e, ovviamente, per quantità di memoria.

In particolare, la memoria analizzata è la 23LC1024-I/P. Ha una memoria di 1024kbits, una frequenza di clock di 20MHz, alimentazione da 2,5V a 5,5V e viene pilotata con un'interfaccia seriale SPI, per cui le connessioni e i pin sono gli stessi trattati in precedenza. Il vantaggio è sempre quello di avere un ristretto numero di pin occupati, per cui è possibile interfacciare il

microcontrollore con altri dispositivi. L'utilizzo di una memoria esterna con queste caratteristiche permette di acquisire un maggior numero di campioni temporali, il quale comporta poi uno spettro in frequenza molto più accurato. Permette poi, di poter salvare in memoria sia i valori dei campioni analizzati, sia le corrispondenti ampiezze in frequenza per eventuali analisi successive.

5. CONCLUSIONI

Grazie agli ultimi studi fatti sulla fonocardiografia e riportati nelle recenti pubblicazioni possiamo sostenere che questa tecnica è promettente a patto di intraprendere ulteriori studi clinici e progettare apparecchiature per un'accurata interpretazione dei toni cardiaci. Nonostante si tratti di una parte indispensabile per la ricerca medica, questa tecnica è stata poco curata nel tempo; tantoché nelle ultime pubblicazioni, prima che venisse di recente ripresa, per la fonocardiografia non si faceva neppure la differenza tra una misurazione effettuata su un paziente di sesso maschile da quella fatta su un paziente di sesso femminile, differenza che oggi sappiamo essere indispensabile ai fini di una diagnostica sempre più veritiera e accurata. In questi anni il progresso nella ricerca medica è stato affiancato da un sempre maggiore sviluppo della bioingegneria, con strumentazioni all'avanguardia e moderne tecniche di analisi, esistono infatti pubblicazioni più recenti che permettono di offrire nuovi punti di vista ingegneristici riguardo la fonocardiografia. In questo lavoro di tesi è stata focalizzata l'attenzione sulla possibilità di implementare queste nuove tecniche, e che attraverso l'utilizzo di un microcontrollore PIC è possibile sviluppare un dispositivo semplice ma funzionale. Con un PIC18F4550 si possono acquisire, fino a 128 campioni circa senza l'utilizzo di memorie esterne. Naturalmente questi campioni non sono sufficienti per un accurato processamento del suono cardiaco. Per questo abbiamo programmato anche un microcontrollore PIC18F47K42 utilizzando, stavolta, MPLAB; per cui il programma è stato riscritto per il compilatore XC8. Questo PIC, senza alcun supporto aggiuntivo di RAM, riesce ad arrivare fino a 1024 campioni, con le dovute ottimizzazioni. Infatti, è stata sfruttata per prima cosa la possibilità di poter ridurre i bit assegnati alle variabili double o float passando da 32 a 24 bit, inoltre è stato escluso il vettore contenente il valore dei campioni temporali scrivendo questi ultimi direttamente nella matrice utilizzata per l'algoritmo della FFT; in

appendice è riportata la libreria modificata e aggiornata per questo PIC compilata in MPLAB. Per prestazioni migliori si può fare rifermento alla famiglia PIC32, le cui caratteristiche sono di gran lunga superiori a quelle della famiglia PIC18F. In ogni modo, sfruttando un microcontrollore di questo tipo si è certamente in grado di fornire alla ricerca e alla medicina un contributo non indifferente, il quale può, in questo caso, contribuire allo sviluppo della fonocardiografia digitale interpretativa, che mira da sempre a curare uno degli aspetti principali per la diagnostica cardiaca.

6. APPENDICE

```
#include <stdio.h>
#include <math.h>
const double PI = 3.141592655358;
double h, b,
       f[Nc][2],              //matrice f_re e f_im, campioni
temporali
       W[(Nc/4)+1][2];        //matrice dei coefficienti

unsigned char Log2()
{
  unsigned int k = Nc;
  unsigned char i = 0;
  while(k) {
    k >>= 1;
    i++;
  }    i--;    return i;
}

unsigned int reverse(unsigned char Log_N, unsigned int n)
{
  unsigned char j, p = 0;
  unsigned char limite = Log_N;
  for(j = 1; j <= limite; j++)
    {
      if(n & (1 << limite - j))
```

```
          p = p | 1 << (j-1);
    }    return p;
}

void ordina(double f[][2])
{
  double temp;
  unsigned int i, nuovo_indice;
  unsigned int Log_N = Log2();

for(i = 0; i < Nc; i++)
  {
    nuovo_indice = reverse(Log_N, i);
    if ((i != nuovo_indice) && (i < nuovo_indice))
     {
      temp = f[i][0];
      f[i][0] = f[nuovo_indice][0];
      f[nuovo_indice][0] = temp;
     }
  }
}

void prodotto_c(double re1, double im1, double re2, double im2)
{
  h = (re1 * re2) - (im1 * im2);  //parte reale
  b = (re1 * im2) + (im1 * re2);  //parte immaginaria
}

void pot_c(double re, double im, unsigned int expp)   //(a+ib)^expp
{
  unsigned int k;
  if (expp == 0)
   {
    h = 1;
    b = 0;
   }
  else
   {
    h = re;
```

```
    b = im;
    for(k=1; k < (expp); k++)
     {
       prodotto_c(h, b, re, im);
     }
   }
}


void transform(double f[][2])
{
  double t_re, t_im, Temp_Re, Temp_Im;
  unsigned int t, i, j, z, n = 1, a = Nc / 2, Log_N = Log2();

  ordina(f);
  for(i = 0; i < Nc; i++)
  {
    f[i][1] = 0;
  }
  W[0][0] = 1; //Re
  W[0][1] = 0; //Im
  W[1][0] = cos(-2 * PI / Nc);  //Re
  W[1][1] = sin(-2 * PI / Nc);  //Im
  for(i = 2; i < (Nc / 4 + 1); i++)
  {
    pot_c(W[1][0], W[1][1], i);
    W[i][0] = h;
    W[i][1] = b;
  }
  for(j = 0; j < Log_N; j++)
  {
   for(i = 0; i < Nc; i++)
    {
      if(!(i & n) {
        t_re = f[i][0];  //parte Re di a0
        t_im = f[i][1];  //parte Im di a0
        z = (i * a) % (n * a);
        if (z > (Nc / 4))
            {
```

```
                Temp_Re = -1 * W[Nc/2-z][0];  //Re
                Temp_Im = W[Nc/2-z][1];  //Im
            }
        else
            {
                Temp_Re = W[z][0];  //Re
                Temp_Im = W[z][1];  //Im
            }

        prodotto_c(Temp_Re, Temp_Im, f[i + n][0], f[i + n][1]);
        f[i][0] = t_re + h;
        f[i][1] = t_im + b;
        f[i + n][0] = t_re - h;
        f[i + n][1] = t_im - b;
      } //if
    }  // for
    n = n * 2;
    a = a / 2;
  }  //for

for(t = 0; t < Nc; t)
  {
    t_re = f[t][0];
    t_im = f[t][1];
    f[t][0] = (int) 10*log10((sqrt(pow(t_re, 2) + pow(t_im, 2))/Nc
+0.5));
  }
}

void FFT(double f[][2], int x)
{
  int m;
  transform(f)

  for(m = 0; m < Nc; m++)
  {
    f[m][0] *= x;
  }
}
```

7. BIBLIOGRAFIA E SITOGRAFIA

1. Multimodal Cardiovascular Imaging: Principles and Clinical Applications.
Olle Pahlm, Galen S. Wagner. McGraw-Hill Education / Medical.

2. Libreria grafica GFX. https://github.com/adafruit/Adafruit-GFX-Library

3. Digital phonocardiography: a PDA-based approach. M. Brusco and H. Nazeran.
Proceedings of the 26th Annual International Conference of the IEEE EMBS
San Francisco, CA, USA. 2004

4. Characteristics of phonocardiography waveforms that influence automatic feature recognition. S. Stainton, C. Tsimenidis and A. Murray.
Computing in Cardiology Conference (CinC) 2017

5. Fourier Transforms and the Fast Fourier Transform (FFT) Algorithm. Paul Heckbert. Notes Comput. Graph. 1995, 3, 15–463.

6. Codice in C++ della FFT basata sull'algoritmo di Cooley-Tukey
https://it.wikipedia.org/wiki/Trasformata_di_Fourier_veloce

7. The Scientist and Engineer's Guide to Digital Signal Processing. Steven W. Smith.
California Technical Publishing