UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Biomedica

Incremento della risoluzione della FFT implementata su microcontrollori PIC, tramite memoria SRAM esterna e interfaccia SPI, per impieghi di fonocardiografia

Enhancing FFT resolution implemented on PIC microcontrollers, through external SRAM memory and SPI interface, for use of phonocardiography

Relatore: Chiar.mo
Prof. Piva Francesco

Tesi di Laurea di
Pettinari Lorenzo

A.A. 2021 / 2022

SOMMARIO

Capitolo 1: Introduzione

1.1 Microcontrollori PIC

1.2 Interfaccia I2C

1.3 Interfaccia SPI

1.4 Display TFT

1.5 Memoria SRAM

1.6 Fonocardiografia (PCG)

Capitolo 2: Scopo

2.1 Trasformate di Fourier e FFT

2.2 Analisi in frequenza del tracciato fonocardiografico

Capitolo 3: Materiali e metodi

3.1 Microcontrollore PIC18F47Q10

3.2 Software per la programmazione

3.3 Display TFT ST7796S

3.4 Memoria SRAM 23LC1024

Capitolo 4: Risultati

4.1 Configurazione microcontrollore PIC

4.2 Configurazione modulo ADCON

4.3 Configurazione moduli MSSP

4.4 Configurazione pin

4.5 Configurazione display TFT ST7796S

4.6 Configurazione memoria SRAM 23LC1024

4.7 Implementazione FFT con SRAM

4.8 Main

Capitolo 5: Discussione

Bibliografia

Introduzione

1.1 Microcontrollori PIC

Un microcontrollore (MCU) è un sistema programmabile che integra in un unico circuito diversi componenti tra cui un microprocessore (CPU), necessario per l'esecuzione di istruzioni, memorie e periferiche I/O.

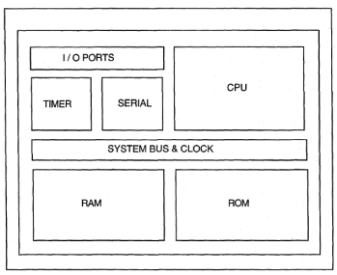
I microcontrollori possono essere utilizzati per attuare funzioni di controllo nei prodotti più diversi in ambito civile e industriale; ne esistono varie classi con caratteristiche e componenti differenti.

I microcontrollori PIC 18F elaborano istruzioni a 8 bit, altri microcontrollori a 16 bit, 24 bit o 32 bit; è necessario scegliere la tipologia più adeguata per ogni applicazione.

Le MCU a 32 bit dispongono di registri, unità logiche aritmetiche e bus a 32 bit e questo consente operazioni di calcolo con numeri più grandi; inoltre, esse sono in genere caratterizzate da velocità di elaborazione maggiore, da memorie di dimensioni più elevate e da funzionalità aggiuntive. Dall'altro lato, però, esse sono più costose e il consumo energetico è maggiore rispetto alle MCU a 8 bit.

Esistono microcontrollori PIC realizzati appositamente per il digital signal processing che prendono il nome di DS-PIC.

Si può considerare la struttura di un semplice microcontrollore tramite il seguente schema a blocchi:



La CPU preleva istruzioni dalla memoria programma attraverso il bus dati e indirizzi; le istruzioni sono prelevate da locazioni di memoria successive fino all'esecuzione di una istruzione di salto.

Una volta che un'istruzione è stata prelevata, si attende nel registro istruzioni o nella pipe fin quando la CPU non è pronta per decodificarla.

L'istruzione viene decodificata dalla logica di controllo in un segnale di controllo che viene inviato all'unità di esecuzione, la quale è costituita da un insieme di registri, da unità funzionali, come, ad esempio, l'unità logica aritmetica (ALU) e da bus interni [1].

Il sincronismo delle operazioni è garantito da un segnale di clock ovvero un'onda quadra con una determinata frequenza generata da un oscillatore, in genere un quarzo piezoelettrico. La gran parte dei microcontrollori sono dotati di oscillatore interno, ma è possibile utilizzare anche oscillatori esterni.

Sono presenti, poi in genere, 3 tipi di memorie di dimensione variabile:

- una memoria Flash non volatile, nella quale è contenuto il programma da eseguire ed è cancellabile elettronicamente, per cui può essere riscritta molte volte; il numero di scritture garantite è fornito dal costruttore ed è espresso nel parametro "Endurance" che può essere letto nel datasheet;
- una memoria RAM volatile, utilizzata per salvare le variabili di un programma durante la sua esecuzione, la quale è caratterizzata da tempi di lettura/scrittura molto rapidi;
- una memoria EEPROM non volatile, con tempi di lettura/scrittura molto minori rispetto alla RAM, la quale è utilizzata per salvare i parametri di un dispositivo o i dati raccolti, se è necessario che questi vengano mantenuti in assenza di alimentazione.

Per risolvere problemi del mondo reale un microcontrollore deve contenere dispositivi che permettano alla CPU di interagire con il mondo esterno.

Sono presenti pin che possono essere utilizzati come input e come output; è possibile impostare la funzione di ciascuno dei pin presenti grazie a opportuni registri.

Lo stato logico di ogni pin, impostato come output, può essere modificato dalla CPU, lo stato logico di ogni pin, impostato come input, può essere letto dalla CPU usando istruzioni apposite nel programma.

La comunicazione con periferiche esterne può avvenire attraverso diversi sistemi di trasmissione, ad esempio seriale (I2C e SPI) e parallela.

Alcune MCU hanno appositi moduli ADC integrati che permettono di convertire segnali analogici in digitale; questi moduli sono di importanza fondamentale per quanto riguarda l'acquisizione di segnali biomedicali che sono per natura analogici.

Microcontrollori più sofisticati hanno moduli "ADC with computations" che mettono a disposizione delle funzioni per elaborare il segnale una volta che questo è stato convertito in digitale, permettendo così di ridurre il carico di lavoro per la CPU.

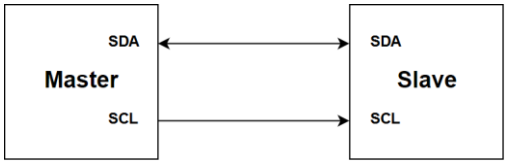
1.2 Interfaccia I2C

I2C (Inter-Integrated Circuit) è un protocollo di comunicazione seriale sincrono, il quale utilizza due fili e supporta una comunicazione half duplex; esso ha un'architettura di tipo master-slave.

È caratterizzato dall'avere velocità di trasferimento dati pari a 100 Kbps (standard mode), 400 Kbps (fast mode), fino a 3.4Mbps (high speed mode).

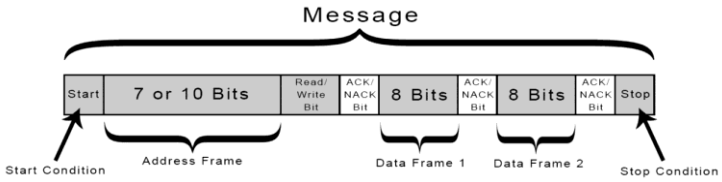
Ogni dispositivo I2C è caratterizzato dall'avere due pin: SCL (Serial Data Clock) e SDA (Serial Data).

Il clock è generato dal master ed in genere è compreso nel range dei KHz [2].



I dati sono trasferiti sotto forma di messaggi suddivisi in più frames; ogni messaggio contiene:

- Start Bit: SDA passa da valore logico alto a valore logico basso prima che SCL passi da valore logico alto a basso;
- Address Frame: una sequenza di 7 o 10 bit che identifica univocamente lo slave con il quale il master vuole dialogare;
- Read/Write Bit: un singolo bit che specifica se il master sta mandando dati (valore logico basso) o se il master sta richiedendo dati (valore logico alto);
- ACK bit: se un frame è stato correttamente ricevuto viene inviato un bit di acknowledge dal destinatario al mittente; questa è una conferma di avvenuta ricezione del frame;
- Data Frame: una sequenza di 8 bit di dati;
- Stop Condition: SDA passa da valore logico basso ad alto dopo che SCL è passato da un valore logico basso ad uno alto.



I principali vantaggi di questa interfaccia consistono nella sua semplicità, nell'utilizzo di soli due fili e nell'avere sempre la conferma di avvenuta ricezione di un frame; essa offre, però, velocità di comunicazione inferiore rispetto alla SPI.

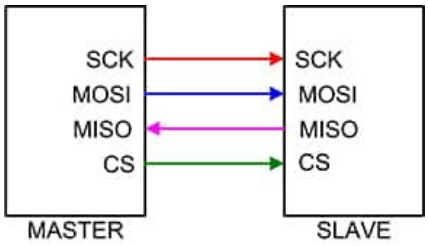
1.3 Interfaccia SPI

La SPI è una interfaccia di comunicazione seriale sincrona, la quale utilizza quattro fili e supporta una comunicazione full duplex.

Ha un'architettura di tipo master-slave e permette velocità di trasferimento dati fino a 400Mbps.

Ogni dispositivo SPI ha quattro pin: SCLK (Serial Clock), MOSI (Master Out Slave In), MISO (Master In Slave Out), CS (Chip Select).

Il master genera il clock su SCLK e seleziona lo slave con il quale vuole comunicare attraverso il CS [2].



SPI è uno standard "de facto" e non uno standard "de iure", perciò non è universalmente definito e ogni dispositivo adotta le proprie regole per la comunicazione [3].

È necessario considerare questo aspetto per permettere la corretta comunicazione tra le periferiche interconnesse, poiché master e slave devono essere compatibili.

La SPI è un'interfaccia non completamente sincrona perché i dati sono sincronizzati con il clock ma il CS può o meno essere sincrono.

Nel caso di un'interfaccia completamente sincrona è possibile dividere i fronti del clock in fronti di caricamento dati sul bus e fronti di campionamento; in corrispondenza di quest'ultimo il dato sul pin in ingresso è letto.

È possibile definire quattro modalità di trasmissione per l'interfaccia SPI. In generale il master, ovvero il microcontrollore in questo caso, supporta tutte e quattro le modalità, mentre lo slave ne supporta solo una o talvolta due.

Le quattro modalità dipendono dalla polarità (CPOL) e dalla fase (CPHA) del clock su SCKL ovvero se la trasmissione inizia quando il clock è alto o basso e se il campionamento avviene sul fronte di salita o di discesa del clock [3].

Mode	Polarity (CPOL)	Phase (CPHA)
0		
1		
2		
3		

Utilizzando opportuni registri del microcontrollore si può attivare e configurare l'interfaccia SPI in modo tale che si può comunicare correttamente con lo slave.

I principali vantaggi di questa interfaccia sono l'elevata velocità di trasmissione dei dati e la comunicazione di tipo full duplex; viene, però, utilizzato un maggior numero di connessioni rispetto a I2C e ciò implica una maggiore occupazione di pin del microcontrollore.

1.4 Display TFT

I display sono delle importanti periferiche che permettono al microcontrollore di interagire con l'uomo; i più utilizzati sono di tipo LCD TFT (thin-film transistor) che sono caratterizzati dall'avere una elevata risoluzione e un prezzo piuttosto contenuto.

Sono display a matrice attiva in cui ogni pixel è costituito da tre sub-pixel che vanno a creare la colorazione RGB; ognuno di questi è pilotato da un singolo transistor.

Ogni display possiede un controller che permette alla CPU di interfacciarsi con il driver; di questi ne esistono diverse tipologie tra cui ILI9488, ILI9486 e ST7796S. Ogni tipologia ha

le proprie caratteristiche ed è necessario comprenderne il funzionamento per consentire un corretto uso del display.

Il driver mette a disposizione funzioni per l'inizializzazione e per impostare i parametri di funzionamento.

ILI9486, ILI9488 e ST7796S sono utilizzati per TFT 380x420 e permettono vari tipi di comunicazione tra cui parallela a 8bit e SPI.

Nella trasmissione parallela sono usati 13 pin: 8 pin per inviare i dati e 5 pin di controllo.

I pin di controllo sono un pin di reset (RST), un chip select (CS) per selezionare la periferica, un pin per selezionare se si inviano comandi o dati (RS), un pin di scrittura (WR) e un pin di lettura (RD).

La trasmissione parallela è caratterizzata da un'elevata velocità di comunicazione però questo avviene a discapito di un'elevata occupazione di pin del microcontrollore.

Nella trasmissione seriale SPI sono usati solamente 6 pin: un chip select (CS), un pin di reset (RST), un pin per selezionare se si inviano comandi o dati (DC), un pin per segnali in ingresso (SDI), un pin per segnali in uscita (SDO) e un pin per il clock di trasmissione (SCK).

ST7796 permette una trasmissione SPI in modalità 0 e clock su SCK con frequenza fino a 12MHz.

È necessario, quindi, impostare i registri del microcontrollore (master) affinché questo possa comunicare correttamente con il display (slave).

Il clock su SCK è derivato dal clock del microcontrollore ed è un suo sottomultiplo; è possibile selezionare mediante opportuni registri del microcontrollore il rapporto tra clock su SCK e clock del microcontrollore.

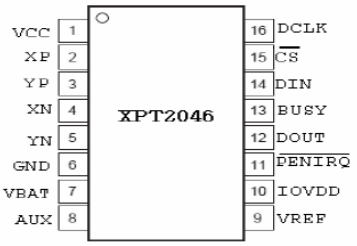
Per poter utilizzare un display è necessario che questo venga prima inizializzato: devono essere impostati opportuni registri in modo tale che si riesca a mostrare correttamente a schermo ciò che si desidera ottenere.

L'inizializzazione dipende dal particolare driver utilizzato, ad esempio per ILI9486 in trasmissione parallela:

```
void Init_ILI9486(){
    //È impostata la direzione dei PIN di controllo come output
    LCD_WR_Direction = 0;
    LCD_RS_Direction = 0;
    LCD_RD_Direction = 0;
    LCD_CS_Direction = 0;
    LCD_RST_Direction = 0;

    LCD_CS = 1;
    LCD_RST = 1;
    LCD_RS = 1;
    LCD_RD = 1;
    LCD_WR = 1;

    LCD_RST = 0; //Reset
    __delay_ms(10);
    LCD_RST = 1;

    LCD_CS = 0;
    Send_Command_8bit(0x01); //Software Reset
    LCD_CS = 1;
    __delay_ms(10);

    LCD_CS = 0;
    Send_Command_8bit(0x11); //Sleep OUT
    LCD_CS = 1;
    __delay_ms(10);

    LCD_CS = 0;
    Send_Command_8bit(0x3A); //Interface pixel format
    Send_Data_8bit(0x66); //18bit RGB, 18bit CPU interface format
    LCD_CS = 1;
    __delay_ms(10);

    LCD_CS = 0;
    Send_Command_8bit(0x36); //Memory access control
    Send_Data_8bit(0xE8);
    LCD_CS = 1;
    __delay_ms(10);

    LCD_CS = 0;
    Send_Command_8bit(0xC2); //Power control 3 for normal mode
    //Frequenza circuito elevatore di tensione. Maggiore è la frequenza, maggiore è la qualità del display, ma è
aumentato il consumo di corrente
    Send_Data_8bit(0x44);
```

```
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0xC5); //VCOM control
Send_Data_8bit(0x00);
Send_Data_8bit(0x00);
Send_Data_8bit(0x00);
Send_Data_8bit(0x00);
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x13); //Normal mode ON
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x29); //Display ON
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x20); //Display inversion OFF
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x51); //Controllo luminosità = 0xFF max, 0x00 min
Send_Data_8bit(0xFF);
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x2A); //Column address set
Send_Data_8bit(0x00);
Send_Data_8bit(0x00);
Send_Data_8bit(0x01);
Send_Data_8bit(0xDF);
Send_Command_8bit(0x00);
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x2B); //Row address set
Send_Data_8bit(0x00);
Send_Data_8bit(0x00);
Send_Data_8bit(0x01);
Send_Data_8bit(0x3F);
Send_Command_8bit(0x00);
LCD_CS = 1;
__delay_ms(10);

LCD_CS = 0;
Send_Command_8bit(0x38); //IDLE mode OFF
LCD_CS = 1;
__delay_ms(10);
```

```
LCD_CS = 0;
Send_Command_8bit(0xB1); //Frame rate control
Send_Data_8bit(0xD0); //Framerate 96Hz.
Send_Data_8bit(0x11);
LCD_CS = 1;
__delay_ms(10);    }
```



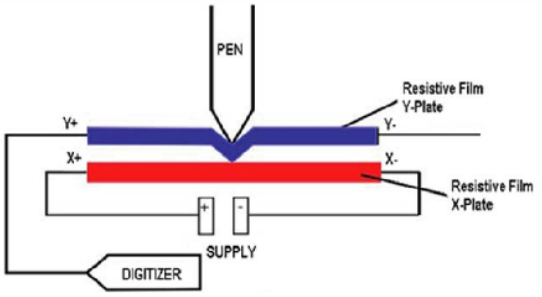
Alcuni display possono essere dotati di un controller per il touch screen integrato, ad esempio XPT2046 utilizza una comunicazione di tipo SPI attraverso dei pin dedicati. I touch screen utilizzati nei display a prezzo contenuto sono generalmente di tipo resistivo.

Sono presenti due strati di ossido di indio-stagno trasparenti e conduttivi separati da uno spazio vuoto; quando si effettua una pressione sullo schermo i due strati entrano in contatto elettrico.

Ognuno di questi strati presenta due elettrodi; in uno strato questi sono perpendicolari alla direzione x, mentre nell'altro lo sono rispetto alla direzione y.

Tra gli elettrodi di uno dei due strati è applicata una tensione in continua ed è misurata la tensione tra gli elettrodi dell'altro strato; i due prendono rispettivamente il nome di strato attivo e strato passivo.

Quando si tocca lo schermo si realizza un partitore di tensione, così che è possibile risalire alla posizione nella direzione corrispondente allo strato attivo; successivamente, la posizione nella seconda direzione è ottenuta seguendo un procedimento analogo, ma scambiando i collegamenti elettrici e, in questo modo, quello che precedentemente era lo strato attivo diventa strato passivo e viceversa [4].



I pin usati da XPT2046 sono: T_CLK per il clock della comunicazione SPI fino a 2,5 MHz, T_CS chip select, T_DO per i segnali in uscita, T_DI per i segnali in ingresso, T_IRQ interrupt output.

Per avere un segnale in uscita che può essere fornito al microcontrollore, è necessario inviare prima un byte di controllo con una particolare struttura; questo byte indica al controller i parametri di funzionamento e quale segnale si desidera ottenere in uscita.

Il byte di controllo per XPT2046 è costituito da: uno start bit (S) che è necessario sia settato a 1, tre bit (A_2, A_1, A_0) che permettono di scegliere il valore da avere in uscita, un bit (MODE) che permette di selezionare il numero di livelli usati dal convertitore A/D, un bit (SER/DFR') con il quale si può selezionare la tensione di riferimento, due bit (PD_1 e PD_0) per accendere il modulo A/D e impostare lo spegnimento tra una conversione e la successiva.

1.5 Memoria SRAM

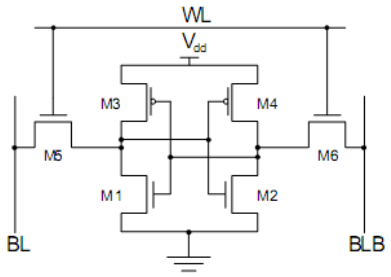
La memoria SRAM, static random access memory, è un tipo di memoria RAM volatile che non necessita di cicli di aggiornamento per mantenere i dati memorizzati, al contrario della DRAM.

Queste memorie per tale motivo sono in genere costose e caratterizzate da tempi di lettura e scrittura molto rapidi e, al contempo, da bassi consumi energetici in condizioni statiche.

In virtù di tali caratteristiche le SRAM sono utilizzate per la cache, per i registri interni delle CPU e in tutte le applicazioni in cui è richiesta elevata velocità di scrittura/lettura.

Per memorizzare un bit si utilizzano in genere celle 6T, costituite da 6 transistor MOSFET.

Il dato è mantenuto all'interno della cella da un bistabile costituito da due invertitori CMOS; sono presenti, inoltre, due transistor che fungono da dispositivi di accesso, grazie ai quali è possibile leggere o modificare il contenuto della cella [8].



Le celle di memoria di ogni SRAM sono raggruppate in parole di dimensione variabile in base alla tipologia di SRAM scelta; in generale queste sono costituite da 8 bit.

Ad ogni parola corrisponde un indirizzo che la identifica in modo univoco.

1.6 Fonocardiografia (PCG)

Un fonocardiogramma è la registrazione dei toni e dei rumori prodotti dal cuore durante l'azione di pompaggio del sangue nei vasi.

Esaminando un tracciato fonocardiografico normale, in un ciclo cardiaco possono essere rilevati quattro toni differenti; in ordine compaiono: primo (S1), secondo (S2), terzo (S3) e quarto tono cardiaco (S4). In generale, S3 e S4 sono molto deboli e per tale motivo vengono ascoltati principalmente S1 e S2 [5].

Il tono cardiaco S1 è prodotto nella prima fase della contrazione ventricolare ed è dovuto alle vibrazioni prodotte dalla chiusura delle valvole tricuspide e mitrale.

La componente più intensa di S1 è prodotta dalla chiusura della valvola mitrale e per tale motivo i cambiamenti di intensità di tale tono sono attribuibili principalmente alle forze che agiscono su di essa.

Questo può essere dovuto a varie cause, tra le quali variazioni della forza contrattile del ventricolo sinistro, variazioni della struttura mitralica o dell'intervallo PR.

In condizioni di riposo i suoni prodotti dalle valvole tricuspide e mitrale sono molto ravvicinati, tanto da non essere distinguibili; le principali cause di divisione del tono S1 sono legate ai ritardi nella contrazione del ventricolo di destra come, ad esempio, un blocco di branca destra [6].

Il tono cardiaco S2 è prodotto dalla chiusura delle valvole aortica (A2) e polmonare (P2) in diastole; la componente A2 è di intensità maggiore rispetto P2.

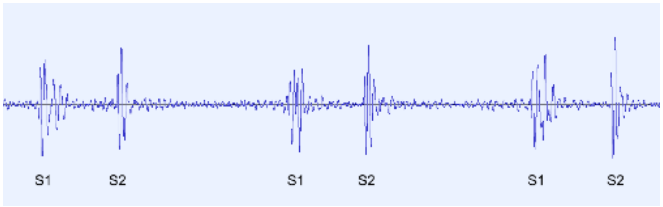
Dal momento che la pressione nell'aorta è significativamente maggiore rispetto alla pressione in arteria polmonare, la valvola aortica si chiude prima della valvola polmonare; per tale motivo A2 precede P2.

A differenza di S1, in condizioni normali, è possibile distinguere i suoni prodotti dalle due valvole durante la fase di inspirazione; in questa fase si ha, infatti, un aumento del ritorno venoso nella parte destra del cuore, viene prolungato perciò il flusso di sangue attraverso l'arteria polmonare, questo fa sì che la valvola polmonare impieghi più tempo per chiudersi [6] [7].

Il tono cardiaco S3 è prodotto dal riempimento passivo ventricolare nella prima fase della diastole; in genere questo tono è molto debole ed è riscontrato fisiologicamente in bambini, giovani adulti, atleti, oppure nel caso di ipercinesia o nel terzo semestre di gravidanza. Il tono S3 tende ad affievolirsi con l'età fino a scomparire verso i 40 anni [7].

Il tono cardiaco S4 è prodotto dal riempimento attivo dei ventricoli dovuto alla contrazione atriale.

S4 è in genere più intenso di S3, anche se ugualmente debole, e può essere prodotto da cuori sani e forti [7].



Scopo

2.1 Trasformate di Fourier e FFT

La trasformata continua di Fourier (CFT) è uno strumento matematico che consente di decomporre un segnale continuo e aperiodico in una somma di infinite sinusoidi con opportuna fase, ampiezza e frequenza.

Il passaggio, dalla descrizione di un segnale dal punto di vista del suo andamento temporale alla sua descrizione dal punto di vista della decomposizione in somma di armoniche, viene chiamato passaggio dal dominio del tempo al dominio della frequenza; la rappresentazione che si ottiene prende il nome di spettro del segnale.

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-\mathrm{i}2\pi f t} dt$$

La trasformata di Fourier trova molte applicazioni in ingegneria e in fisica ed è di fondamentale importanza nello studio dei segnali.

Nella pratica si ha però a che fare con segnali campionati, quindi discreti e aventi lunghezza finita; essi sono gli unici che possono essere acquisiti ed elaborati da un calcolatore.

Per lavorare con questi segnali viene introdotta la trasformata discreta di Fourier (DFT) che può essere considerata come un'approssimazione numerica della CFT.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$$

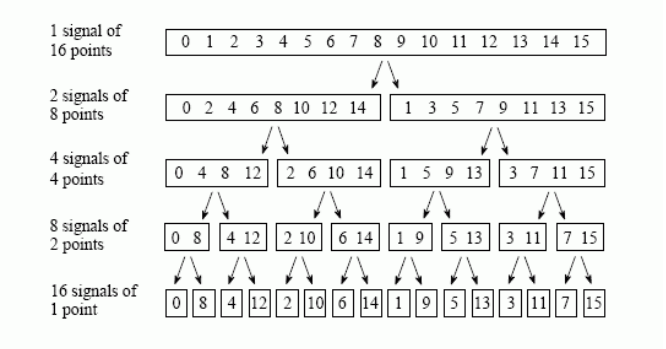
Per calcolare la DFT in modo efficiente di una serie finita di N punti è stato sviluppato l'algoritmo della Fast Fourier Transform (FFT).

Il numero di operazioni effettuate dalla FFT è proporzionale a $N\log_2 N$; si ha una complessità minore rispetto alle operazioni necessarie per implementare la DFT nel modo tradizionale, il cui numero è proporzionale a N^2 [11].

Ipotesi necessaria per l'applicazione di tale algoritmo è che il segnale sia costituito da certo un numero di punti N nel dominio del tempo e che questo numero sia una potenza di 2.

La FFT opera scomponendo tale segnale in N segnali nel dominio del tempo composti da punti singoli; successivamente, sono calcolati gli N spettri relativi a questi N segnali e, infine, sono sintetizzati per ottenere un unico spettro in frequenza.

Questo viene effettuato per step successivi, in ognuno dei quali il vettore dei campioni è spezzato in due e i campioni di posto pari e dispari sono posti in vettori diversi; è necessario effettuare $\log_2 N$ step per compiere questa decomposizione [12].



La decomposizione può essere anche vista come un algoritmo di ordinamento "bit reversal" che consiste nel riordinare l'ordine dei campioni andando a invertire l'ordine dei bit nel numero di ogni campione; questo può facilmente essere compreso osservando la figura seguente [12].

Sample numbers in normal order		Sample numbers after bit reversal	
Decimal	*Binary*	*Decimal*	*Binary*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0100
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

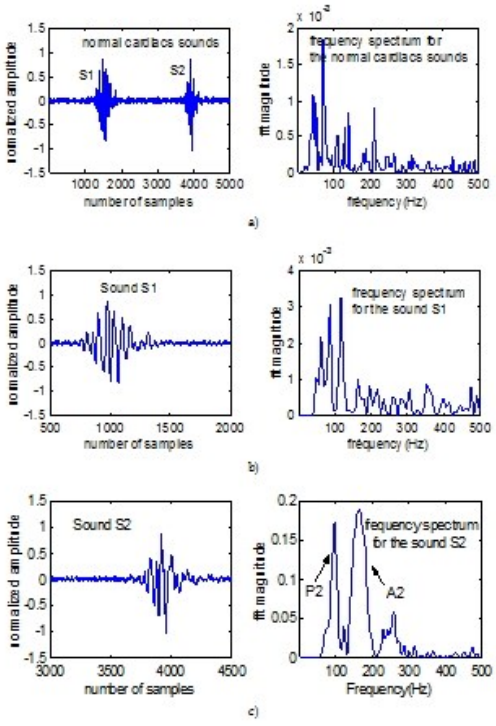
Successivamente viene calcolato lo spettro relativo ad ognuno degli N segnali costituiti da un solo punto.

Infine, è necessario che gli N spettri in frequenza così ottenuti siano ricombinati nell'esatto ordine inverso con cui ha avuto luogo la decomposizione nel dominio del tempo.

2.2 Analisi in frequenza del tracciato fonocardiografico

Generalmente il segnale fonocardiografico è studiato per auscultazione; questa è una procedura medica ben consolidata, la quale permette di identificare varie patologie cardiache.

Degenerazioni dei suoni cardiaci possono indicare, ad esempio, variazioni delle strutture anatomiche attraverso le quali scorre il sangue che portano ad un cambiamento della sua dinamica e delle vibrazioni causate dal suo flusso.



Alcune patologie sono, però, difficilmente identificabili con questo metodo: si necessita, perciò, di medici esperti con una preparazione accurata [9].

Grazie all'utilizzo della FFT è possibile andare ad analizzare il contenuto in frequenza di un segnale purché sia valida l'ipotesi di segnale stazionario.

Il segnale PCG è classificato come non-stazionario, tuttavia la FFT può essere considerata uno strumento per poter avere una comprensione di base del suo contenuto in frequenza [10].

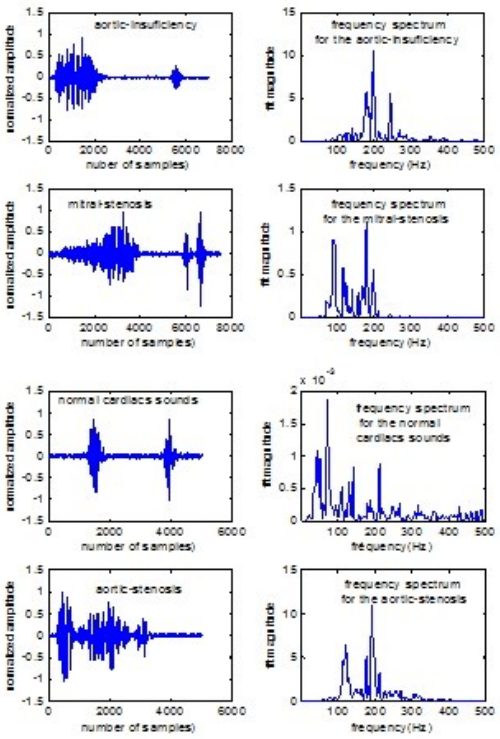
Si può osservare che un tracciato non patologico ha componenti in frequenza comprese tra i 40 e i 200 Hz [10].

La FFT può inoltre essere applicata separatamente ai suoni cardiaci S1 e S2: il primo ha uno spettro compreso tra 10 e 180 Hz, il secondo ha uno spettro compreso tra 50 e 250 Hz [10].

In molti casi patologici il contenuto in frequenza del segnale fonocardiografico può variare rispetto a un segnale normale.

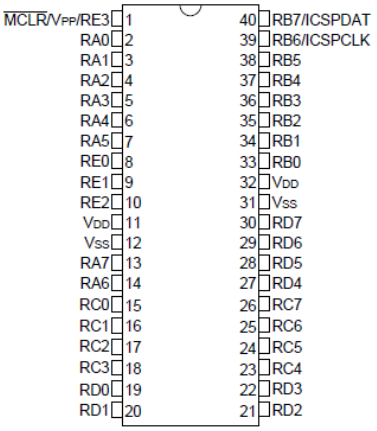
In alcune patologie si può constatare una perdita considerevole del contenuto in frequenza, ad esempio, nel caso di stenosi aortica o stenosi mitrale.

In altre patologie si possono osservare componenti in frequenza che non appaiono in soggetti sani, ad esempio, nel caso di insufficienza aortica si osservano frequenze maggiori di 200 Hz [10].



Materiali e metodi

3.1 Microcontrollore PIC18F47Q10



Il microcontrollore utilizzato è un PIC18F47Q10; questa MCU a 8 bit è stata scelta poiché riesce a soddisfare le necessità del progetto ed è un buon compromesso tra costo di acquisto, consumo energetico e prestazioni.

La tensione di alimentazione è compresa tra 1,8V e 5,5V; in questo caso il PIC è stato alimentato a 3,3V poiché questa è la tensione richiesta dal display TFT.

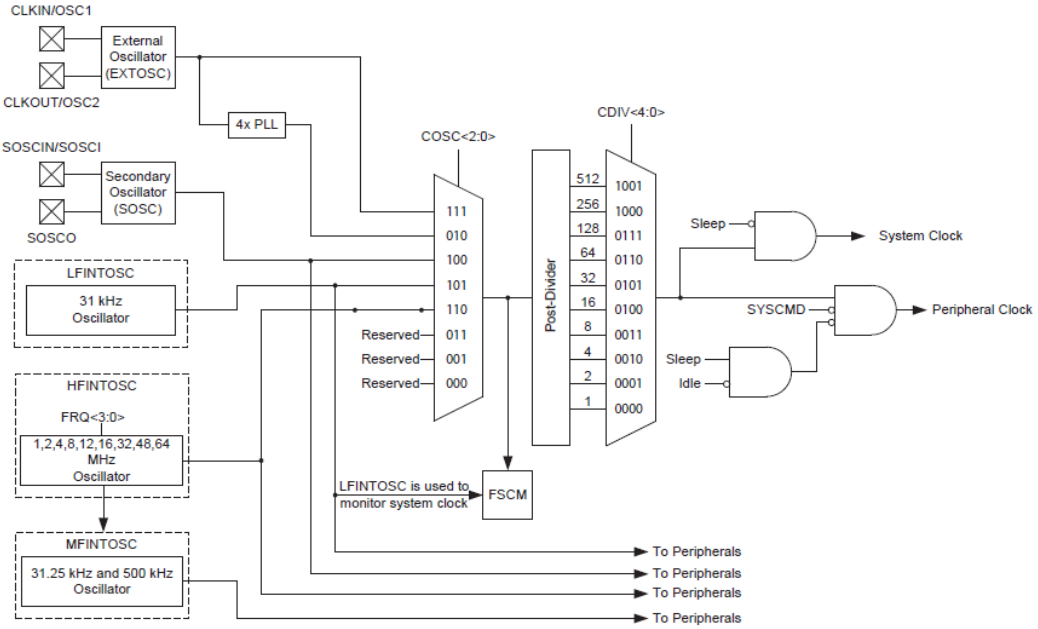
Il microcontrollore è dotato di oscillatore interno con frequenza di clock selezionabile fino ad un massimo di 64 MHz; per le esigenze di questo progetto è stata selezionata la frequenza massima attraverso il registro OSCFRQ.

Per il calcolo della FFT sono eseguite dal microprocessore molte istruzioni e l'avere una frequenza di clock elevata permette di portare a termine il calcolo in tempi ridotti.

Possono essere utilizzati, in alternativa, oscillatori esterni, come ad esempio un quarzo piezoelettrico connesso a opportuni pin.

È presente anche un oscillatore interno a bassa frequenza (31 KHz) e un oscillatore interno a media frequenza (da 31,25 KHz a 500 KHz).

La frequenza del segnale generato da una di queste sorgenti può essere divisa per una costante, in modo da ottenere un suo sottomultiplo, grazie alla presenza di un postscaler.



Questa MCU è dotata di memoria programma Flash di 128 KB, una memoria dati SRAM di 3615 Bytes e una memoria EEPROM di 1024 Bytes.

La memoria programma ha dimensioni opportune per contenere le istruzioni da eseguire e le librerie utilizzate per la comunicazione con le periferiche; la memoria dati, invece, ha una capacità piuttosto limitata e per il calcolo della FFT è utilizzata una SRAM esterna in modo tale da contenere il numero elevato di variabili create.

In totale sono presenti 40 pin: 36 pin I/O, due di alimentazione e due di massa; 35 di questi possono essere utilizzati come ingressi analogici.

Due moduli MSSP permettono al microcontrollore di interfacciarsi con le periferiche esterne che utilizzano una comunicazione seriale I2C o SPI; in particolare in questo progetto sono stati utilizzati entrambi i moduli per il display TFT e per la SRAM esterna.

I moduli devono essere correttamente configurati per poter comunicare con le periferiche; in particolare, per SPI è necessario scegliere la modalità per la comunicazione e il microcontrollore supporta tutte e quattro le possibili modalità; deve poi essere stabilito il clock di comunicazione da inviare su SCK.

Ogni modulo utilizza dei pin dedicati; in particolare, il modulo MSSP1 usa RC3 come SCK, RC4 come SDI e RA5 come CS, SDO, invece, può essere scelto tra i pin disponibili.

Il modulo MSSP2 usa RB1 come SCK, RB2 come SDI, RB0 come CS e SDO anche in questo caso può essere scelto.

I byte inviati e ricevuti durante la comunicazione sono inseriti in particolari registri di buffer SSPxBUF; quando l'invio o la ricezione di un byte è terminato, viene settato ad 1 il bit SSPxIF nel registro PIR3, tale bit ha la funzione di flag per indicare la fine del trasferimento del byte.

È, inoltre, presente un modulo ADC^2 (analog-digital converter with computation) a 10bit che permette di campionare in 1024 livelli segnali analogici in ingresso su particolari pin selezionati.

Sebbene il numero di livelli di campionamento non sia molto elevato (esistono, infatti, convertitori AD esterni a 12, 16 o 24 bit) è possibile ottenere frequenze di campionamento piuttosto elevate.

È possibile selezionare la frequenza di campionamento attraverso opportuni registri; in particolare, il registro ADCLK permette di selezionare il clock del modulo AD, il registro ADACQ permette di scegliere il tempo di acquisizione del segnale prima del campionamento.

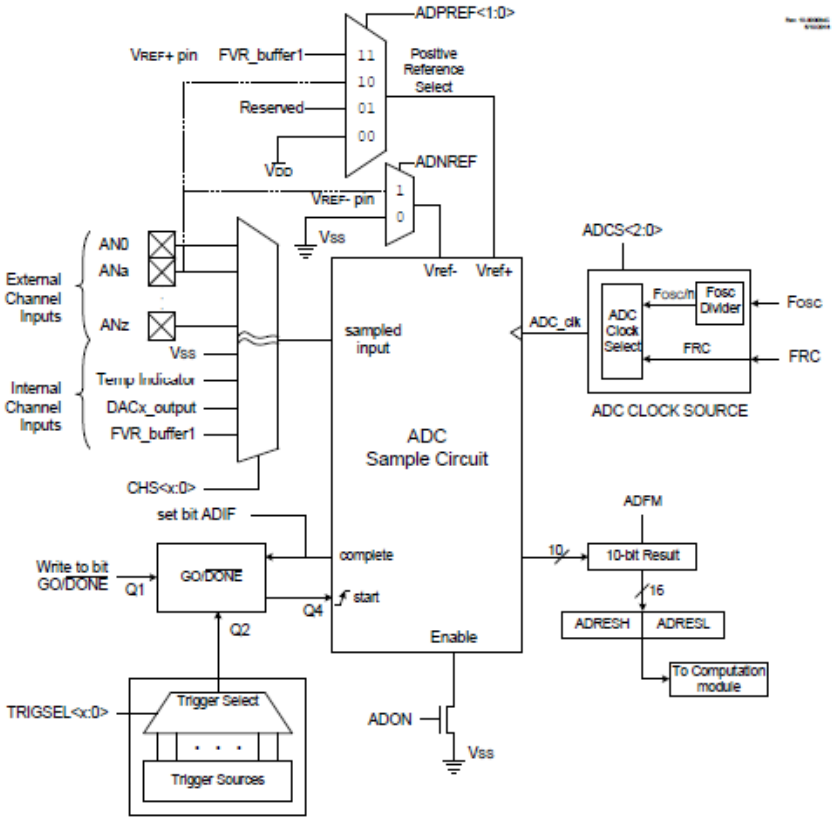
Per poter acquisire il segnale analogico in ingresso è necessario selezionare l'intervallo di tensione nel quale andare ad individuare i 1024 livelli; si devono, quindi, scegliere le tensioni di riferimento positiva e negativa attraverso il registro ADREF.

Una volta configurato correttamente il modulo, per poter iniziare una conversione è necessario accendere il modulo AD settando ad 1 il bit ADON nel registro ADCON0 e settare ad 1 il bit ADGO nello stesso registro.

Quando la conversione è terminata, il bit ADGO è resettato e il dato è inserito nei registri ADRESH e ADRESL; i due bit più significativi possono essere inseriti o in ADRESH o in ADRESL in base al tipo di giustificazione scelta attraverso il bit ADFM nel registro ADCON0.

Nel caso di giustificazione a destra i due bit più significativi sono inseriti nel registro ADRESH, i restanti otto nel registro ADRESL.

Questo modulo permette inoltre di effettuare elaborazioni sul segnale acquisito, ad esempio, può essere effettuato un filtraggio passa-basso con frequenza di cutoff selezionabile ed è possibile calcolare la media del segnale in ingresso e il numero di attraversamenti degli zeri. Questa caratteristica del modulo ADC^2 è di interesse in molte applicazioni poiché permette di effettuare elaborazioni senza impiegare il microprocessore, per cui quest'ultimo può, così, compiere contemporaneamente altre operazioni.



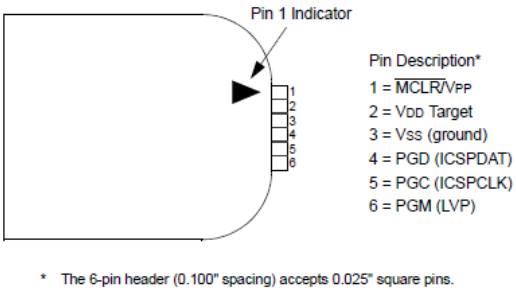
3.2 Software per la programmazione

Per la programmazione del microcontrollore PIC è stato utilizzato MPLAB X, un ambiente di sviluppo integrato che fornisce diverse funzionalità tra le quali: creare un codice sorgente utilizzando l'editor integrato, effettuare debug, permettere la compilazione del codice attraverso un compilatore.

È stato utilizzato il compilatore XC8 specifico per MCU PIC a 8bit. Talvolta per facilitare la creazione del codice C è stato utilizzato il tool MPLAB Code Configurator (MCC), il quale permette di configurare i registri del microcontrollore utilizzando un'interfaccia grafica e di generare, successivamente, un codice C.

L'affidabilità, però, di questo tool è limitata ed è stato spesso necessario modificare il codice generato in modo da renderlo funzionante.

Per permettere la programmazione del PIC è stato utilizzato un PICKIT 3 collegato al computer via USB e al microcontrollore attraverso opportuni pin.



3.3 Display TFT ST7796S

È stato utilizzato un display TFT da 4 pollici con risoluzione 480x320 e driver ST7796S.

La comunicazione utilizzata è seriale di tipo SPI in modalità 0, con frequenza massima testata su SCK di 12 MHz.

Il display possiede, inoltre, un controller XPT2046 per il touch screen, che utilizza una comunicazione di tipo SPI attraverso pin dedicati in modalità 0; il clock massimo su SCK è pari a 2,5 MHz.

Il display utilizza 9 pin: 4 sono usati la comunicazione SPI (CS, SDI, SDO e SCK), 2 sono di controllo, DC permette di selezionare se si inviano comandi o dati, RST permette di effettuare il reset. Si hanno, inoltre: un pin di alimentazione a 3,3V, un pin di massa e un pin di alimentazione per la retroilluminazione.

Per permettere un corretto utilizzo del display è necessario effettuare un'inizializzazione andando a settare particolari registri, inviando comandi e dati.

Innanzitutto, è necessario effettuare un soft reset con il comando 0x01; viene poi indicato di uscire dalla modalità di riposo con il comando 0x11.

Con il comando 0x3A è possibile selezionare il numero di bit utilizzati per i colori RGB; se si vuole inviare un byte per ogni colore è possibile selezionare il formato a 18bit e i due bit meno significativi di ogni byte inviato sono scartati. Altri driver come ILI9488 supportano anche il formato a 24 bit.

Il comando 0x36, memory access data control, permette di scegliere la rotazione.

È possibile, inoltre, scegliere la modalità di funzionamento tra normale o risparmio energetico; per operare in modalità normale è inviato il comando 0x13.

Si può controllare il consumo energetico utilizzando particolari comandi, ad esempio Power Control 3 in Normal Mode (0xC2); questo comando permette di aumentare la qualità dell'immagine a scapito però del consumo di corrente.

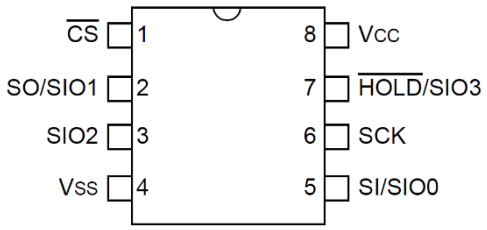
È necessario definire a quale parte della memoria frame il microcontrollore può avere accesso e questo è possibile grazie ai comandi Column Address Set (0x2A) e Row Address Set (0x2B).

Una volta inizializzato correttamente il display, per scrivere nella memoria frame viene usato il comando 0x2C e possono essere inviati 2 o 3 byte per scegliere il colore RGB di ogni pixel.

Sono state create funzioni all'interno di librerie che permettono di impostare lo sfondo del display e di selezionare in modo agevole il colore di singoli pixel; esistono, inoltre, librerie grafiche che permettono di scrivere sullo schermo numeri e stringhe di caratteri.

3.4 Memoria SRAM 23LC1024

La memoria SRAM esterna utilizzata è la 23LC1024; essa ha capacità di 1 Mbit ed utilizza una comunicazione SPI in modalità 0 e supporta una frequenza di clock su SCK fino a 20 MHz. In questo progetto è stata usata la frequenza massima permessa dal modulo MSSP del MCU pari a 16 MHz.



Le caratteristiche di questa memoria la rendono adatta al progetto sviluppato, essa ha, infatti, una elevata capacità in modo tale da contenere l'elevato numero di variabili create dal calcolo della FFT e comunica molto velocemente con il microcontrollore.

Sono presenti 8 pin: 4 per la comunicazione SPI (CS, SDI, SDO e SCK), 2 pin di alimentazione e 2 pin non utilizzati; i due pin non utilizzati vengono sfruttati se si adotta una comunicazione SDI o SQI.

È possibile scegliere tra tre modalità di funzionamento: Byte Operation, Page Operation e Sequence Operation.

Nella prima modalità, le operazioni di lettura e scrittura sono limitate a un singolo byte, ad ogni singolo byte corrisponde un indirizzo unico a 24 bit.

Nella seconda modalità si individuano 4096 pagine da 32 bytes ognuna; all'interno di ogni pagina, per le operazioni di scrittura e lettura, l'indirizzo è incrementato automaticamente.

Nella terza modalità, invece, i byte vengono scritti e letti automaticamente in locazioni di memoria successive incrementando l'indirizzo a partire da un indirizzo iniziale.

È possibile selezionare la modalità di funzionamento modificando i bit 7 e 6 di un registro dedicato (MODE register); questo può essere effettuato inviando l'istruzione 0x01 grazie alla quale viene indicato di scrivere nel registro, è inviato poi il dato da inserire.

Per scrivere in una singola locazione di memoria in Byte Operation occorre inviare l'istruzione 0x02, è necessario inviare l'indirizzo a 24bit della locazione di memoria e infine il dato da inserire.

In modo analogo per leggere il dato all'interno di una particolare locazione di memoria, occorre inviare l'istruzione 0x03 e l'indirizzo a 24bit, il dato viene restituito quindi nel buffer SPI della MCU.

Risultati

Durante il percorso di tirocinio in laboratorio è stato configurato il microcontrollore PIC con i suoi moduli ADCON e MSSP, successivamente interfacciato con il display TFT e con la memoria SRAM esterna.

Infine è stata implementata la FFT utilizzando quest'ultima.

4.1 Configurazione microcontrollore PIC

Inizialmente, per permettere al microcontrollore di operare correttamente, sono stati definiti i Configuration Bits ovvero una raccolta di bits che possono essere modificati durante la programmazione e vengono letti in fase di reset del microcontrollore.

Questi permettono di andare a disabilitare e abilitare alcune funzioni: è possibile, ad esempio, selezionare la sorgente del clock.

Ogni MCU ha i suoi Configurations Bits specifici che ne permettono il corretto funzionamento.

In questo caso sono stati generati utilizzando MCC attraverso impostazioni selezionate tramite interfaccia grafica.

```
// Configuration bits: selected in the GUI

// CONFIG1L
#pragma config FEXTOSC = OFF    // External Oscillator mode Selection bits->Oscillator not enabled
#pragma config RSTOSC = HFINTOSC_64MHZ    // Power-up default value for COSC bits->HFINTOSC with HFFRQ = 64 MHz and CDIV = 1:1

// CONFIG1H
#pragma config CLKOUTEN = OFF    // Clock Out Enable bit->CLKOUT function is disabled
#pragma config CSWEN = ON    // Clock Switch Enable bit->Writing to NOSC and NDIV is allowed
#pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable bit->Fail-Safe Clock Monitor enabled

// CONFIG2L
```

```
#pragma config MCLRE = EXTMCLR    // Master Clear Enable bit->MCLR pin (RE3) is MCLR
#pragma config PWRTE = OFF    // Power-up Timer Enable bit->Power up timer disabled
#pragma config LPBOREN = OFF    // Low-power BOR enable bit->Low power BOR is disabled
#pragma config BOREN = SBORDIS    // Brown-out Reset Enable bits->Brown-out Reset enabled , SBOREN bit is ignored

// CONFIG2H
#pragma config BORV = VBOR_190    // Brown Out Reset Voltage selection bits->Brown-out Reset Voltage (VBOR) set to 1.90V
#pragma config ZCD = OFF    // ZCD Disable bit->ZCD disabled. ZCD can be enabled by setting the ZCDSEN bit of ZCDCON
#pragma config PPS1WAY = ON    // PPSLOCK bit One-Way Set Enable bit->PPSLOCK bit can be cleared and set only once; PPS registers remain locked after one clear/set cycle
#pragma config STVREN = ON    // Stack Full/Underflow Reset Enable bit->Stack full/underflow will cause Reset
#pragma config XINST = OFF    // Extended Instruction Set Enable bit->Extended Instruction Set and Indexed Addressing Mode disabled

// CONFIG3L
#pragma config WDTCPS = WDTCPS_31    // WDT Period Select bits->Divider ratio 1:65536; software control of WDTPS
#pragma config WDTE = OFF    // WDT operating mode->WDT Disabled

// CONFIG3H
#pragma config WDTCWS = WDTCWS_7    // WDT Window Select bits->window always open (100%); software control; keyed access not required
#pragma config WDTCCS = SC    // WDT input clock selector->Software Control

// CONFIG4L
#pragma config WRT0 = OFF    // Write Protection Block 0->Block 0 (000800-001FFFh) not write-protected
#pragma config WRT1 = OFF    // Write Protection Block 1->Block 1 (002000-003FFFh) not write-protected
#pragma config WRT2 = OFF    // Write Protection Block 1->Block 2 (004000-005FFFh) not write-protected
#pragma config WRT3 = OFF    // Write Protection Block 1->Block 3 (006000-007FFFh) not write-protected

// CONFIG4H
#pragma config WRTC = OFF    // Configuration Register Write Protection bit->Configuration registers (300000-30000Bh) not write-protected
#pragma config WRTB = OFF    // Boot Block Write Protection bit->Boot Block (000000-0007FFh) not write-protected
#pragma config WRTD = OFF    // Data EEPROM Write Protection bit->Data EEPROM not write-protected
#pragma config SCANE = ON    // Scanner Enable bit->Scanner module is available for use, SCANMD bit can control the module
#pragma config LVP = ON    // Low Voltage Programming Enable bit->Low voltage programming enabled. MCLR/VPP pin function is MCLR. MCLRE configuration bit is ignored

// CONFIG5L
#pragma config CP = OFF    // UserNVM Program Memory Code Protection bit->UserNVM code protection disabled
#pragma config CPD = OFF    // DataNVM Memory Code Protection bit->DataNVM code protection disabled

// CONFIG6L
#pragma config EBTR0 = OFF    // Table Read Protection Block 0->Block 0 (000800-001FFFh) not protected from table reads executed in other blocks
#pragma config EBTR1 = OFF    // Table Read Protection Block 1->Block 1 (002000-003FFFh) not protected from table reads executed in other blocks
#pragma config EBTR2 = OFF    // Table Read Protection Block 1->Block 2 (004000-005FFFh) not protected from table reads executed in other blocks
```

```
#pragma config EBTR3 = OFF    // Table Read Protection Block 1->Block 3 (006000-007FFFh) not protected from table reads executed in other blocks

// CONFIG6H
#pragma config EBTRB = OFF    // Boot Block Table Read Protection bit->Boot Block (000000-0007FFh) not protected from table reads executed in other blocks
```

Dopo aver selezionato attraverso i Configuration Bits l'oscillatore interno come sorgente di clock sono stati settati i registri per configurare quest'ultimo.

In particolare il registro OSCFRQ per selezionare una frequenza di 64MHz, ovvero la massima possibile, il registro OSCON1 per selezionare la modalità di funzionamento dell'oscillatore interno e per impostare come unitaria la divisione della frequenza di clock da parte del postscaler.

Anche in questo caso è stato utilizzato MCC per facilitare il settaggio di questi registri.

```
// NOSC HFINTOSC; NDIV 1;
   OSCCON1 = 0x60;
// HFFRQ 64_MHz;
   OSCFRQ = 0x08;
```

4.2 Configurazione modulo ADCON

Per permettere di acquisire un segnale analogico esterno è stato necessario configurare il modulo convertitore analogico-digitale interno a 10bit.

È stato selezionato come ingresso analogico il pin RA1 settando il registro ADPCH.

Attraverso il registro ADCON0 è stata selezionata la giustificazione a destra del risultato della conversione settando a 1 il bit ADFM, è stato selezionato il clock derivato dall'oscillatore interno come clock della conversione settando a 0 il bit ADCS, infine è stato attivato il modulo AD settando a 1 il bit ADON.

Il registro ADCON2 permette di selezionare la modalità di operazione del modulo: settando ADMD[0:2] a 000 si è scelto infatti di operare in basic-mode.

Con ADCLK è stato selezionato il sottomultiplo della frequenza di clock proveniente dall'oscillatore interno da fornire al convertitore AD; in questo caso si è scelto Fosc/8 ovvero 8MHz.

ADREF permette di selezionare le tensioni positiva e negativa di riferimento per individuare i 1024 livelli di campionamento del segnale analogico in ingresso; come tensione negativa di riferimento è stata scelta la massa e come tensione positiva l'alimentazione pari a 3,3V.

ADACQ permette di andare a scegliere il tempo di acquisizione del segnale in ingresso ovvero il numero di periodi di clock precedenti la conversione del segnale; se la sorgente del segnale in ingresso ha una bassa impedenza di uscita, allora questo tempo può essere breve e può essere, quindi, aumentata la velocità di campionamento. In questo caso è stato selezionato un tempo di acquisizione del segnale pari a 10 periodi di clock in modo tale da avere un buon compromesso tra velocità di campionamento e corretta acquisizione del segnale.

Anche in questo caso per facilitare la configurazione del modulo ADCON è stato utilizzato MCC.

```
// ADLTHL 0;
    ADLTHL = 0x00;
    // ADLTHH 0;
    ADLTHH = 0x00;
    // ADUTHL 0;
    ADUTHL = 0x00;
    // ADUTHH 0;
    ADUTHH = 0x00;
    // ADSTPTL 0;
    ADSTPTL = 0x00;
    // ADSTPTH 0;
    ADSTPTH = 0x00;
    // ADRPT 0;
    ADRPT = 0x00;
    // ADPCH ANA1;
    ADPCH = 0x01;
    // ADCAP Additional uC disabled;
    ADCAP = 0x00;
    // ADPRE 0;
    ADPRE = 0x00;
```

```
// ADDSEN disabled; ADGPOL digital_low; ADIPEN disabled; ADPPOL VSS;
ADCON1 = 0x00;
// ADCRS 1; ADMD Basic_mode; ADACLR disabled; ADPSIS ADRES;
ADCON2 = 0x10;
// ADCALC First derivative of Single measurement; ADTMD disabled; ADSOI ADGO not cleared;
ADCON3 = 0x00;
// ADAOV ACC or ADERR not Overflowed;
ADSTAT = 0x00;
// ADNREF VSS; ADPREF VDD;
ADREF = 0x00;
// ADACT disabled;
ADACT = 0x00;
// ADCS FOSC/8;
ADCLK = 0x03;
// ADGO stop; ADFM right; ADON enabled; ADCONT disabled; ADCS FOSC/ADCLK;
ADCON0 = 0x84;
// ADACQ 10;
ADACQ = 0x0A;
```

4.3 Configurazione moduli MSSP

Per permettere la comunicazione SPI con il display e la memoria SRAM è stato necessario configurare entrambi i moduli MSSP; ognuno in modo specifico in base alla periferica con il quale si deve interfacciare.

Il modulo MSSP1 è stato utilizzato per il display TFT; quest'ultimo utilizza una comunicazione SPI in modalità 0, la frequenza di clock massima supportata su SCK è pari a 12 MHz; è stata perciò selezionata una frequenza pari a 8 MHz poiché è il sottomultiplo più elevato selezionabile della frequenza di clock generata dall'oscillatore interno entro 12 MHz. MSSP1 utilizza RA5 come CS, RC3 come SCK e RC4 come SDI; è stato scelto poi RC5 come SDO.

Per inizializzare MSSP1 è stato utilizzato il seguente codice generato tramite MCC e successivamente modificato:

```
//Setup PPS Pins
SSP1CLKPPS = 19;
SSP1DATPPS = 20;
RC3PPS    = 15;
RC5PPS    = 16;
//SPI setup
```

```
    SSP1STAT = 0x40;
    SSP1CON1 = 0x0A;
    SSP1ADD = 0x01;
    TRISCbits.TRISC3 = 0;
    SSP1CON1bits.SSPEN = 1;
```

Per trasferire dati dal microcontrollore al display, e viceversa, sono state utilizzate due funzioni differenti da quelle generate da MCC.

Per inviare un byte al display questo viene inserito all'interno del buffer SSP1BUF; è possibile stabilire quando la trasmissione del byte è completa rilevando quando il bit flag SSP1IF nel registro PIR3 è settato a 1.

Per ricevere un byte dal display viene pulito il buffer SSP1BUF, si deve attendere poi che SSP1IF venga settato a 1 e questo indica che il dato inviato dal display ora è nel buffer.

```
void SPI1_WriteByte(uint8_t byte)
{
    unsigned char data_flush;
    SSP1BUF=byte;

    while(!PIR3bits.SSP1IF);
    PIR3bits.SSP1IF=0;
    data_flush=SSP1BUF;
}

uint8_t SPI1_ReadByte(void)
{
    SSP1BUF=0xff;
    while(!PIR3bits.SSP1IF);
    PIR3bits.SSP1IF=0;
    return(SSP1BUF);
}
```

Il modulo MSSP2 è stato utilizzato per la comunicazione con la memoria SRAM; questa usa una comunicazione SPI in modalità 0 con una frequenza massima di clock su SCK pari a 20 MHz.

È stata selezionata la frequenza massima disponibile ovvero 16 MHz, pari alla frequenza di clock generata dall'oscillatore interno diviso 4.

MSSP2 utilizza RB0 come CS, RB1 come SCK e RB2 come SDI; è stato scelto RB3 come SDO.

Analogamente al modulo MSSP1, è stato utilizzato un codice per l'inizializzazione generato tramite MCC:

```
//Setup PPS Pins
   SSP2CLKPPS = 9;
   SSP2DATPPS = 10;
   RB1PPS    = 17;
   RB3PPS    = 18;
   //SPI setup
   SSP2STAT = 0xC0;
   SSP2CON1 = 0x00;
   SSP2ADD = 0x01;
   TRISBbits.TRISB1 = 0;
   SSP2CON1bits.SSPEN = 1;
```

Analogamente a MSSP1 sono state utilizzate due funzioni per inviare e ricevere byte:

```
void SPI2_WriteByte(uint8_t byte)
{
   unsigned char data_flush;
   SSP2BUF=byte;

   while(!PIR3bits.SSP2IF);
   PIR3bits.SSP2IF=0;
   data_flush=SSP2BUF;
}

uint8_t SPI2_ReadByte(void)
{
   SSP2BUF=0xff;
   while(!PIR3bits.SSP2IF);
   PIR3bits.SSP2IF=0;
   return(SSP2BUF);
}
```

4.4 Configurazione pin

Per permettere la corretta comunicazione con le periferiche è stato necessario selezionare la direzione I/O dei pin e indicare quali di questi vengono utilizzati per segnali analogici e quali per segnali digitali.

Oltre ai pin utilizzati dai moduli MSSP e ADCON, sono stati utilizzati RD1 e RD2 come pin di controllo per il display, rispettivamente DC e RST.

Anche in questo caso la selezione dei pin è stata effettuata tramite MCC ed è stato generato il seguente codice:

```
/* LATx register */
    LATE = 0x00;
    LATD = 0x00;
    LATA = 0x00;
    LATB = 0x00;
    LATC = 0x00;
/* TRISx registers */
    TRISE = 0x07;
    TRISA = 0xDF;
    TRISB = 0xF4;
    TRISC = 0xD7;
    TRISD = 0xF9;
 /* ANSELx registers */
    ANSELD = 0xF9;
    ANSELC = 0xC7;
    ANSELB = 0xF0;
    ANSELE = 0x07;
    ANSELA = 0xDF;

    SSP2DATPPS = 0x0A;   //RB2->MSSP2:SDI2;
    SSP1CLKPPS = 0x13;   //RC3->MSSP1:SCK1;
    RB1PPS = 0x11;   //RB1->MSSP2:SCK2;
    RC3PPS = 0x0F;   //RC3->MSSP1:SCK1;
    RB3PPS = 0x12;   //RB3->MSSP2:SDO2;
    RC5PPS = 0x10;   //RC5->MSSP1:SDO1;
    SSP1DATPPS = 0x14;   //RC4->MSSP1:SDI1;
    SSP2CLKPPS = 0x09;   //RB1->MSSP2:SCK2;
```

4.5 Configurazione display TFT ST7769S

Per poter permettere al microcontrollore di comunicare con il display in modo tale da mostrare a schermo ciò che si desidera, è stato necessario creare una libreria opportuna.

Questa libreria è costituita da più header files.

ST7769.h contiene le funzioni necessarie per l'inizializzazione, per settare il background e scegliere l'orientamento del display.

ADD_ST7769.h contiene le funzioni per poter scegliere il colore desiderato dei singoli pixel dello schermo.

Sono stati, successivamente, assegnati degli alias ai pin utilizzati per la comunicazione in un file header Def.h.

```
#define TFT_CS         PORTAbits.RA5     //Posizione fissa per questo PIC
#define TFT_RST        PORTDbits.RD2
#define TFT_DC         PORTDbits.RD1

#define TFT_SDO        PORTCbits.RC4
#define TFT_SCK        PORTCbits.RC3     //Posizione fissa per questo PIC
#define TFT_SDI        PORTCbits.RC5     //Posizione fissa per questo PIC

#define TFT_CS_DIR      TRISAbits.TRISA5
#define TFT_RST_DIR     TRISDbits.TRISD2
#define TFT_DC_DIR      TRISDbits.TRISD1

#define TFT_SDO_DIR     TRISCbits.TRISC4
#define TFT_SCK_DIR     TRISCbits.TRISC3
#define TFT_SDI_DIR     TRISCbits.TRISC5

#define TFTWIDTH 320
#define TFTHEIGHT 480
```

Sono state definite le funzioni che permettono di inviare dati o comandi al display:

```
void sendCommand(uint8_t cmd) {
  TFT_DC = 0; //Quando DC è a 0 è inviato un comando, quando è a 1 è inviato un dato
  SPI1_WriteByte(cmd);
  TFT_DC = 1;
}

void sendData(uint8_t cmd) {
  TFT_DC = 1;
  SPI1_WriteByte(cmd);
}
```

Per inizializzare il display è stata utilizzata la funzione INITST7796:

```
void InitST7796(void){
    TFT_RST_DIR = 0;
    TFT_CS_DIR = 0;
```

```
    TFT_DC_DIR = 0;
    TFT_SDI_DIR = 0;
    TFT_SDO_DIR = 1;
    TFT_SCK_DIR = 0;

    __delay_ms(100);
    TFT_RST = 0;
    __delay_ms(100);
    TFT_RST = 1;
    __delay_ms(200);

     TFT_CS = 0;
     sendCommand(0x01); //Software Reset
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x11); //Sleep OUT
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x28); //Display OFF
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x3A); //Interface pixel format
     sendData(0x66); //76 = 18bit RGB, 18bit MCU interface format
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x36); //Memory access control
     sendData(0xE8);
     TFT_CS = 1;
     __delay_ms(10);

     //Power control 3 for normal mode
     //Frequenza circuito elevatore di tensione. Maggiore è la frequenza maggiore è la qualità del display ma a
scapito del consumo di corrente
     TFT_CS = 0;
     sendCommand(0xC2); //Power Control
     sendData(0x0A); //Medium
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x13); //Normal mode ON
     TFT_CS = 1;
     __delay_ms(10);

     TFT_CS = 0;
     sendCommand(0x53); //Write CTRL display value
     sendData(0x2C);
     TFT_CS = 1;
     __delay_ms(10);
```

```
    TFT_CS = 0;
    sendCommand(0x51); //Brightness
    sendData(0xFF);
    TFT_CS = 1;
    __delay_ms(10);

    TFT_CS = 0;
    sendCommand(0x2A); //Column Address Set
    sendData(0x00);
    sendData(0x00);
    sendData(0x01);
    sendData(0xDF);
    sendCommand(0x00);
    TFT_CS = 1;
    __delay_ms(10);

    TFT_CS = 0;
    sendCommand(0x2B); //Page Address Set
    sendData(0x00);
    sendData(0x00);
    sendData(0x01);
    sendData(0x3F);
    sendCommand(0x00);
    TFT_CS = 1;
    __delay_ms(10);

    //IDLE mode OFF
    TFT_CS = 0;
    sendCommand(0x38);
    TFT_CS = 1;
    __delay_ms(10);

    TFT_CS = 0;
    sendCommand(0x29); //Display ON
    TFT_CS = 1;
    __delay_ms(10);
}
```

Per settare la rotazione del display è stata utilizzata la funzione setRotation:

```
uint16_t _width;    //< Larghezza display
uint16_t _height;   //< Altezza display
uint8_t rotation;  //< Rotazione display (da 0 a 3)

void setRotation(unsigned m) {
rotation = m % 4;
   switch (rotation) {
      case 0:
         m = (0x40 | 0x08);
         _width  = TFTWIDTH;
         _height = TFTHEIGHT;
         break;
      case 1:
```

```
            m = (0x20 | 0x08);
            _width  = TFTHEIGHT;
            _height = TFTWIDTH;
            break;
        case 2:
            m = (0x80 | 0x08);
            _width  = TFTWIDTH;
            _height = TFTHEIGHT;
            break;
        case 3:
            m = (0x40 | 0x80 | 0x20 | 0x08);
            _width  = TFTHEIGHT;
            _height = TFTWIDTH;
            break;
    }

    TFT_CS = 0;
    sendCommand(0x36);
    sendData(m);
    TFT_CS = 1;
}
```

È stata definita la funzione per impostare lo sfondo consentendo di scegliere il colore:

```
void Set_Background(unsigned x,unsigned y,unsigned z)
{
    unsigned i, j;

     TFT_CS = 0;
     sendCommand(0x2C);
     for (i=0; i<480; i++)
     {
      for (j=0; j<320; j++)
      {
       sendData(x);
       sendData(y);
       sendData(z);
      }
     }
     sendCommand(0x00);
     TFT_CS = 1;

    setRotation(1);
}
```

Sono state definite, infine, le funzioni set_address e write_pixel le quali permettono, rispettivamente, di andare a selezionare un pixel specifico del display e di colorarlo del colore desiderato.

In particolare la funzione write_pixel ha come ingressi le coordinate del pixel da colorare e il colore RGB desiderato.

```
void set_address(unsigned x, unsigned y) {
 unsigned xt, xb, yt, yb;
 xt = (x >> 8);
 xb = x;
 yt = (y >> 8);
 yb = y;
    TFT_CS = 0;
    sendCommand(0x2A);
    sendData(xt);
    sendData(xb);
    sendData(xt);
    sendData(xb);
    sendCommand(0x00);
    TFT_CS = 1;
    __delay_us(1);

    TFT_CS = 0;
    sendCommand(0x2B);
    sendData(yt);
    sendData(yb);
    sendData(yt);
    sendData(yb);
    sendCommand(0x00);
    TFT_CS = 1;
    __delay_us(1);
}

void write_pixel(unsigned x, unsigned y, unsigned r, unsigned g, unsigned b)  {
  set_address(x, y);
  TFT_CS = 0;
  sendCommand(0x2C);
  sendData(r);
  sendData(g);
  sendData(b);
  TFT_CS = 1;
}
```

4.6 Configurazione memoria SRAM 23LC1024

È stata creata una libreria per permettere alla MCU di comunicare con la SRAM.

In questa libreria sono state implementate funzioni per inizializzare la memoria SRAM, per scrivere e leggere byte nelle locazioni di memoria.

Inoltre, sono state implementate le funzioni per scrivere e leggere variabili di tipo double; ogni double in XC8 ha dimensione di 24 bit, per cui è stato necessario decomporlo in 3 bytes, i quali sono stati scritti in locazioni di memoria successive.

Si è scelto di utilizzare la modalità di funzionamento Byte Operation; in fase di scrittura e di lettura è perciò necessario specificare l'indirizzo di ogni locazione di memoria da scrivere e da leggere.

Sono stati assegnati degli alias ai pin utilizzati nella comunicazione e alle funzioni per il trasferimento dei byte tramite SPI per facilitare la scrittura del codice:

```
#define R_CS    PORTBbits.RB0
#define R_SDI   PORTBbits.RB3
#define R_SDO   PORTBbits.RB2
#define R_SCK   PORTBbits.RB1

#define R_CS_DIR    TRISBbits.TRISB0
#define R_SDI_DIR   TRISBbits.TRISB3
#define R_SDO_DIR   TRISBbits.TRISB2
#define R_SCK_DIR   TRISBbits.TRISB1

#define WriteByte   SPI2_WriteByte
#define ReadByte    SPI2_ReadByte
```

È stata creata una funzione per inizializzare la SRAM andando a impostare la direzione I/O dei pin utilizzati per la comunicazione e definendo la modalità di funzionamento, ovvero Byte Operation.

```
void RAM_init(){
   //È impostata la direzione I/O pin
   R_CS_DIR = 0;
```

```
    R_SDI_DIR = 0;
    R_SDO_DIR = 1;
    R_SCK_DIR = 0;

    R_CS = 0;
    WriteByte(0x01); //Write command
    WriteByte(0x00); //Byte mode
    R_CS = 1;

}
```

Sono state, quindi, scritte le funzioni per scrivere e leggere byte in determinate locazioni di memoria della SRAM.

La funzione writeByte_RAM prende in ingresso l'indirizzo a 24 bit della locazione di memoria e il byte da scrivere al suo interno; per inviare l'indirizzo a 24 bit, utilizzando un buffer della comunicazione SPI a 8 bit, è stato necessario decomporlo in 3 byte.

Analogamente la funzione readByte_RAM prende in ingresso l'indirizzo a 24 bit della locazione di memoria da leggere e restituisce il byte al suo interno.

```
void writeByte_RAM(uint24_t address, uint8_t data){
    uint8_t a, b, c;

    a = address >> 16;
    b = address >> 8;
    c = address;
    R_CS = 0;
    WriteByte(0x02);
    WriteByte(a);
    WriteByte(b);
    WriteByte(c);
    WriteByte(data);
    R_CS = 1;
}

uint8_t readByte_RAM(uint24_t address){
    uint8_t a, b, c, data;

    a = address >> 16;
    b = address >> 8;
    c = address;

    R_CS = 0;
    WriteByte(0x03);
    WriteByte(a);
```

```
    WriteByte(b);
    WriteByte(c);
    data = ReadByte();
    R_CS = 1;

    return data;
}
```

Infine sono state scritte le funzioni per leggere e scrivere double nella SRAM; variabili di tipo double sono, infatti, le variabili utilizzate nel calcolo della FFT.

```
void writedouble_RAM(uint24_t address, double data){
    uint8_t bytes[3];

    *(double*)(bytes) = data;

    writeByte_RAM(address, bytes[0]);
    writeByte_RAM(address + 1, bytes[1]);
    writeByte_RAM(address + 2, bytes[2]);
}

double readdouble_RAM(uint24_t address){
    uint8_t bytes[3];
    double result;

    bytes[0] = readByte_RAM(address);
    bytes[1] = readByte_RAM(address + 1);
    bytes[2] = readByte_RAM(address + 2);

    result = *(double*)(bytes);

    return result;
}
```

4.7 Implementazione FFT con SRAM

Per l'implementazione della FFT utilizzando la memoria SRAM esterna è stata creata una libreria dedicata.

La FFT è calcolata grazie ad una funzione che prende in ingresso un vettore di 512 campioni (variabili di tipo int) e un fattore di scala per permettere di visualizzare meglio le componenti in frequenza sul display.

Il vettore dei campioni è innanzitutto ordinato utilizzando la funzione ordina, successivamente il vettore ordinato è copiato nella SRAM. Ogni campione occupa 3 locazioni di memoria dal momento che viene convertito in double grazie ad un operatore di cast.

Il campione nella cella zero del vettore è scritto nelle locazioni di memoria 0, 1 e 2.

Nelle locazioni di memoria 3, 4, 5 è scritta la parte immaginaria del campione ovvero 0.

Il campione i-esimo quindi è scritto nelle locazioni di memoria i, i+1, i+2 e la sua parte immaginaria, che per tutti i campioni è pari a 0, è scritta nelle locazioni di memoria i+3, i+5, i+6; con "i" che va da 0 a 511.

I coefficienti calcolati per la FFT sono scritti a partire dalla locazione di memoria jj = (512*6) + 100, ovvero 100 locazioni dopo quella occupata dalla parte immaginaria dell'ultimo campione.

Il coefficiente i-esimo è scritto nelle locazioni di memoria jj+i, jj+i+1, jj+i+2 e la sua parte immaginaria è scritta nelle locazioni jj+i+3, jj+i+4, jj+i+5; con "i" che va da 0 a (512/4).

Una volta calcolata la FFT le componenti in frequenza sono scritte nello stesso vettore dei campioni fornito in ingresso.

Sono state poi implementate delle funzioni necessarie al calcolo della FFT tra cui prodotto_c per il calcolo del prodotto tra numeri complessi, pot_c per il calcolo della potenza di un numero complesso.

```
#define Nc 512

const double PI = 3.141592655358;
double h, b;
int lettura[Nc];

unsigned Log2()
{
  unsigned k = Nc;
```

```
 unsigned i = 0;
 while(k) {
  k >>= 1;
  i++;
 }
 return i - 1;
}


unsigned reverse(unsigned Log_N, unsigned 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;
}


void ordina_campioni()
{
 unsigned i, nuovo_indice;
 unsigned Log_N = Log2();
 double temp, ap;

for(i = 0; i < Nc; i++)
 {
  nuovo_indice = reverse(Log_N, i);
  if ((i != nuovo_indice) && (i < nuovo_indice))
   {
    temp = readdouble_RAM(i*6);
    ap = readdouble_RAM(nuovo_indice*6);
    writedouble_RAM(i*6, ap);
    writedouble_RAM(nuovo_indice*6, temp);
   }
 }
}


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;
   }
 }
```

```
}

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
}


//(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);
   }
  }
}


void transform(int *lettura)     //Funzione per il calcolo della FFT del vettore dei campioni
{

   double t_re, t_im, Temp_Re, Temp_Im;
   double fi_re, fi_im;
   double w1_re, w1_im;
   unsigned t, i, j, z, n = 1, a = Nc / 2, Log_N = Log2();

   ordina(lettura);

   //Il vettore dei campioni è copiato nella RAM
   for(i = 0; i<Nc; i++){

      writedouble_RAM(i*6, ((double) lettura[i]) );
      writedouble_RAM((i*6)+3, 0 );

   }

   unsigned jj=(Nc*6)+100;

   writedouble_RAM(jj, 1);
   writedouble_RAM(jj+3, 0);
   writedouble_RAM(jj+6, (double) cos(-2 * PI / Nc));
   writedouble_RAM(jj+9, (double) sin(-2 * PI / Nc));

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

```
    {
      w1_re=readdouble_RAM(jj+6);
      w1_im=readdouble_RAM(jj+9);
      pot_c(w1_re, w1_im, i);
      writedouble_RAM(jj+(i*6), h);
      writedouble_RAM((jj+3)+(i*6), b);

    }

    for(j = 0; j < Log_N; j++)
    {
    for(i = 0; i < Nc; i++)
       {
       if(!(i & n))
       {

          t_re = readdouble_RAM(i*6);
          t_im = readdouble_RAM((i*6)+3);

          z = (i * a) % (n * a);

          if (z > (Nc / 4))
             {
                Temp_Re = -1 * readdouble_RAM(jj+((Nc/2-z)*6));
                Temp_Im = readdouble_RAM((jj+3)+((Nc/2-z)*6));
             }
          else
             {
                Temp_Re = readdouble_RAM(jj+(z*6));
                Temp_Im = readdouble_RAM((jj+3)+(z*6));
             }

          fi_re = readdouble_RAM((i+n)*6);
          fi_im = readdouble_RAM((i+n)*6 + 3);
          prodotto_c(Temp_Re, Temp_Im, fi_re, fi_im);
          writedouble_RAM(i*6, t_re + h);
          writedouble_RAM(i*6 + 3, t_im + b);
          writedouble_RAM((i+n)*6, t_re - h);
          writedouble_RAM((i+n)*6 + 3, t_im - b);

       }
       }
       n = n * 2;
       a = a / 2;
    }

    for(t = 0; t < Nc; t++)    //Il modulo in frequenza è scritto nel vettore dei campioni
    {
       t_re = readdouble_RAM(t*6);
       t_im = readdouble_RAM(t*6 + 3);

       lettura[t] =  (int) 20 + 10*(log10(sqrt(pow(t_re, 2) + pow(t_im, 2))/Nc));
    }
}


void FFT(int *lettura, int x)
```

```
{
 int m;
 transform(lettura);

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

Infine è stata implementata una funzione all'interno della libreria per tracciare a schermo lo spettro in frequenza.

```
void drawFFT(int campioni[Nc]){
   for(int j=0; j<((Nc/2)); j++){
      for(int i=0; i<=campioni[j]; i++){
            write_pixel(j*2,320-i,255,255,255);

         }
   }
}
```

4.8 Main

Nel main del programma da eseguire sul microcontrollore sono state, innanzitutto, incluse le librerie generate da MCC, le librerie del display, della SRAM e della FFT.

Prima della funzione main è stato definito il vettore per contenere 512 campioni acquisiti; questo consente di andare a memorizzare tale vettore nella memoria dati che ha una capacità di 3615 bytes.

Se il vettore dei campioni fosse definito all'interno della funzione main, allora questo sarebbe stato salvato nella memoria EEPROM, la quale ha una capacità di soli 1024 bytes e non potrebbe perciò contenere un vettore di 512 variabili int.

Poiché il modulo ADCON offre una risoluzione di campionamento pari a 10 bit, il segnale analogico in ingresso è campionato su 1024 livelli; è sufficiente, perciò, una variabile di tipo int a 16 bit per contenere il valore di ogni campione.

Nella funzione main è stata chiamata la funzione SYSTEM_Initialize generata tramite MCC che contiene al suo interno le inizializzazioni dei moduli MSSP1, MSSP2, ADCON e contiene, inoltre, le configurazioni dei pin e dell'oscillatore interno.

Sono state, successivamente, abilitate le interfacce SPI1 e SPI2 mediante opportune istruzioni per permettere la comunicazione con display e SRAM, dopodiché questi ultimi sono stati inizializzati.

Sono stati acquisiti i 512 campioni, per fare ciò è stato selezionato il canale dove è presente l'ingresso analogico ovvero RA1 ed è stato acceso il modulo AD.

Una volta acquisiti i campioni è stata applicata la FFT e quest'ultima è stata graficata a schermo.

```
//File generati da MCC
#include "mcc_generated_files/mcc.h"
//Librerie display
#include "Def.h"
#include "ST7796.h"
#include "Add_ST7796.h"
//Libreria SRAM
#include "23LC1024.h"
//Libreria FFT
#include "FFT_SPI.h"

//È definito un vettore di 512 campioni
int campioni[512];

void main(void)
{
   //Inizializzazione moduli MSSP1, MSSP2, ADCON, configuro pin e oscillatore
   SYSTEM_Initialize();

   SSP1CON1bits.SSPEN = 1; //Abilito SPI1
   SSP2CON1bits.SSPEN = 1; //Abilito SPI2

   //Inizializzazione il display
   InitST7796();
   //Background nero
   Set_Background(0,0,0);

   //Inizializzazione la SRAM
   RAM_init();

   //Configurazione modulo AD
```

```
    ADPCH = 0x01;           //Ingresso analogico AN1
    ADCON0bits.ADON = 1;    //È acceso il modulo AD

    for(int i = 0; i<512; i++){
        ADCON0bits.ADGO = 1;
        while(ADCON0bits.ADGO == 1){};
        campioni[i] = (ADRESH << 8) + ADRESL;

    }

    FFT(campioni,1);

    while (1)
    {
        drawFFT(campioni);
    }
}
```

Discussione

Durante il percorso di tesi è stata implementata la FFT su microcontrollore PIC18F47Q10 utilizzando memoria SRAM esterna 23LC1024.

Inizialmente, è stato configurato il microcontrollore per poter operare correttamente e comunicare con la memoria SRAM e il display TFT; questo è stato in parte possibile grazie al tool di MPLAB MCC. Sovente è stato però necessario andare a leggere il datasheet per comprendere come configurare correttamente i registri.

La parte sicuramente più complessa del lavoro è consistita nella creazione delle librerie per il display TFT e per la memoria SRAM; questo è stato fatto interamente partendo da pochissime informazioni reperibili in internet ed è stato necessario leggere il datasheet di queste periferiche e fare numerosi test per garantire il loro corretto funzionamento.

In particolare, per il display ho incontrato molte difficoltà nella scrittura della funzione di inizializzazione e questo ha comportato uno studio approfondito del datasheet e dei comandi disponibili; è stato possibile ottenere una soluzione confrontando la funzione scritta con funzioni analoghe per driver differenti.

Una ulteriore difficoltà riguarda la comunicazione SPI con le due periferiche: nei datasheet, infatti, non viene indicata chiaramente la modalità di comunicazione utilizzata, per cui questa è stata dedotta utilizzando i diagrammi temporali forniti dai costruttori.

Per il display TFT non è specificato nel datasheet neanche la frequenza di clock di comunicazione massima SPI; poiché sono reperibili pochissime informazioni in merito sul web, è stato necessario effettuare delle prove e testare varie frequenze di clock.

Per quanto riguarda la memoria SRAM, ulteriore problematica risiede nella scrittura di variabili di dimensione maggiore di 8 bit: queste sono state decomposte in più variabili da 8

bit, poi scritte in locazioni di memoria successive e sono state create funzioni per scrivere e leggere variabili int a 16 bit, 24 bit, 32 bit e variabili di tipo double a 24 bit.

Infine, è stata implementata la FFT che ha comportato il dover risolvere problemi legati alla memorizzazione e al corretto recupero nella memoria SRAM delle variabili create per il calcolo della trasformata.

I campioni e i coefficienti calcolati sono definiti come numeri complessi a cui corrispondono due variabili di tipo double: una per la parte reale e una per la parte immaginaria e queste hanno una dimensione di 24 bit e occupano perciò 3 locazioni di memoria consecutive.

Il progetto creato ha permesso di acquisire 512 campioni attraverso l'utilizzo del modulo ADCON, applicare poi al vettore contenente i campioni la FFT e, infine, mostrare il risultato sullo schermo.

Un successivo sviluppo di questo progetto potrebbe essere l'utilizzo di un modulo convertitore AD esterno; questo permetterebbe di avere una risoluzione maggiore dei 10 bit forniti dal modulo interno. È però necessario che il modulo disponga di una velocità di campionamento adeguata all'utilizzo.

Inoltre, introducendo ed utilizzando una seconda SRAM in parallelo, si consentirebbe di utilizzare la prima per la memorizzazione di un numero elevato di campioni e la seconda per il calcolo della FFT.

Bibliografia

[1] R. Bannatyne and G. Viot, "Introduction to microcontrollers," *WESCON/97 Conference Proceedings*, 1997, pp. 564-574, doi: 10.1109/WESCON.1997.632384.

[2] D. Trivedi, A. Khade, K. Jain and R. Jadhav, "SPI to I2C Protocol Conversion Using Verilog," *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1-4, doi: 10.1109/ICCUBEA.2018.8697415.

[3] Usach, Miguel. AN-1248 Application Note: *SPI Interface*. Analog Devices, Inc., September 2015

[4] J. Calpe-Maravilla, I. Medina, M. J. Martínez and A. Carbajo, "Dual touch and gesture recognition in 4-wire resistive touchscreens," SENSORS, 2014 IEEE, 2014, pp. 787-790, doi: 10.1109/ICSENS.2014.6985117.

[5] G. V. H. Prasad and P. R. Kumar, "Performance analysis of feature selection methods for feature extracted PCG signals," 2015 13th International Conference on Electromagnetic Interference and Compatibility (INCEMIC), 2015, pp. 225-231, doi: 10.1109/INCEMIC.2015.8055885.

[6] Dornbush S, Turnquest AE. Physiology, Heart Sounds. 2021 Jul 22. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2022 Jan–. PMID: 31082054.

[7] Chizner MA. Cardiac auscultation: rediscovering the lost art. Curr Probl Cardiol. 2008 Jul;33(7):326-408. doi: 10.1016/j.cpcardiol.2008.03.003. PMID: 18513577.

[8] Trinadh Kumar, A. S. S., and B. V. V. Satyanarayana. "Low voltage high speed 8T SRAM cell for ultra-low power applications." Int J Eng Technol 7.3.29 (2018): 70-74.

[9] Debbal, S. M., and Fethi Bereksi-Reguig. "Time-frequency analysis of the first and the second heartbeat sounds." Applied Mathematics and Computation 184.2 (2007): 1041-1052.

[10] Debbal S.M, Hamza Ch. “Pathologies cardiac discrimination using the Fast Fourir Transform (FFT) The short time Fourier transforms (STFT) and the Wigner distribution”. J Cardiology Interventions, 1 (1); DOI: http;//doi.org/03.2021/1.1004.

[11] Cooley, James W. and John W. Tukey. “An algorithm for the machine calculation of complex Fourier series.” *Mathematics of Computation* 19 (1965): 297-301.

[12] Steven W. Smith. 1997. The scientist and engineer's guide to digital signal processing. California Technical Publishing, USA.