UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria Biomedica

Sviluppo di un dispositivo in grado di visualizzare, immagazzinare ed analizzare segnali trasdotti da microfono allo scopo di monitoraggio dell'attività cardiaca.

Development of a device capable of displaying, storing and analyze signals transduced by a microphone in order to monitor cardiac activity.

Relatore:

Prof. **Piva Francesco**

Tesi di Laurea di:

Matteo Boemio

A.A. 2020 / 2021

Indice

1. INTRODUZIONE ____ 3

1.1. FONOCARDIOGRAFIA ____ 3

1.2. CARATTERIZZAZIONE DEL FONOCARDIOGRAFO DIGITALE ____ 5

2. SCOPO ____ 6

3. MATERIALI E METODI ____ 6

3.1. COMPILATORE E PROGRAMMATORE ____ 6

3.2. LA SCELTA DEL PIC18F47K42 ____ 7

3.3. PERIFERICHE ____ 8

3.3.1. Encoder KY-040 ____ 8

3.3.2. Display grafico TFT ILI9481 ____ 10

3.3.3. Scheda di memoria MicroSD ____ 10

3.4. ALIMENTAZIONE ____ 11

3.5. STRUMENTI DA BANCO E PROGRAMMI ____ 12

4. RISULTATI ____ 13

4.1. CONNESSIONI ____ 13

4.1.1. Connessione in parallelo ____ 13

4.1.2. Connessione Seriale SPI ____ 14

4.2. MODULO ADC INTERNO ____ 15

4.3. TIMER0 ____ 17

4.4. INTERFACCIAMENTO UTENTE ____ 19

4.4.1. Struttura ed aspetto del programma ____ 19

4.4.2. Implementazione software encoder ____ 23

4.4.3. Implementazione Pulsanti ____ 25

4.5. INIZIALIZZAZIONE E CONTROLLI ____ 26

4.6. ACQUISIZIONE E VISUALIZZAZIONE DEL SEGNALE ____ 33

4.7. MEMORIZZAZIONE DEL SEGNALE ____ 36

4.7.1. Prestazioni ____ 36

4.7.2. Scrittura in memoria ____ 39

4.8. ANALISI DEL SEGNALE ____ 43

5. CONCLUSIONI E SVILUPPI FUTURI ____ 45

6. APPENDICE ____ 48

7. BIBLIOGRAFIA E SITOGRAFIA ____ 58

Introduzione

1.1. Fonocardiografia

La fonocardiografia è una tecnica di analisi medica che prevede l'auscultazione dei suoni prodotti dal muscolo cardiaco durante lo svolgersi del suo ciclo andando a rappresentare la risultante attraverso pennino su foglio o direttamente in formato digitale su schermo. Il tracciato ottenuto è il fonocardiogramma, segnale caratteristico che in condizioni di normale funzionamento cardiaco si presenta formato da due gruppi di oscillazioni. Il primo di 4-6 pulsazioni della durata di 0.10-0.14 s complessivi, denominato come primo tono, mentre, il secondo gruppo di oscillazioni presenta ampiezze minori ed una durata, anch'essa inferiore, di circa 0.06 s. Tra i due toni intercorre una pausa breve corrispondente alla sistole ventricolare, mentre, si individua l'instaurarsi di una pausa più lunga tra la fine del secondo tono e il primo del successivo ciclo cardiaco corrispondente, questa volta, alla diastole ventricolare. Per quanto riguarda la caratterizzazione fisiologica dei due toni essi sono prodotti principalmente dalla chiusura delle valvole. In particolare, le oscillazioni iniziali del primo tono, caratterizzate da bassa frequenza e piccola ampiezza, corrispondono alla messa in tensione del muscolo cardiaco, mentre le successive oscillazioni di maggiore ampiezza e frequenza, sono associabili alla chiusura delle valvole atrioventricolari. Le vibrazioni registrate in corrispondenza del secondo tono equivalgono meccanicamente alla chiusura delle valvole semilunari aortiche e polmonari. Talvolta sono presenti anche altri due toni: il terzo ed il quarto, rispettivamente seguente il secondo di circa 200ms e precedente il primo di 100ms. Il terzo tono è tipico nei bambini o in soggetti con gittata cardiaca particolarmente ponderosa, indice del termine del riempimento veloce ventricolare, è causato dalla vibrazione dell'apparato valvolare (mitrale e tricuspidale) a causa della decelerazione improvvisa del flusso sanguigno all'interno della cavità ventricolare. Il quarto tono è patologico e corrisponde a forti contrazioni atriali producenti improvvise tensioni a livello dei lembi valvolari e della muscolatura circostante generanti, per l'appunto, il suono auscultato.

Essendo un tracciato caratteristico il fonocardiogramma consente di individuare condizioni patologiche di diversa natura sulla base del confronto con tracciati fisiologici. Nella seguente *figura 1* sono evidenziati fonocardiogrammi patologici a confronto con il

tracciato A in condizioni di funzionamento normale del muscolo cardiaco. Stati patologici sono solitamente diagnosticati attraverso l'auscultazione di quello che in gergo medico viene chiamato soffio; la natura di questo suono sibilante deriva dall'instaurarsi di un moto turbolento all'interno del, normalmente imperturbato, flusso sanguigno dovuto principalmente a due cause: la stenosi e il rigurgito. Con il termine tecnico stenosi si vuole indicare il restringimento di un vaso o un orifizio, come una valvola, che causa un passaggio difficoltoso al bolo di sangue in eiezione. Il rigurgito, invece, si rifà ad una condizione di insufficienza di una valvola, tale per cui il sangue refluisca nella direzione opposta rispetto alla quale, fisiologicamente, dovrebbe dirigersi. Tramite analisi del fonocardiogramma è anche evidenziabile la presenza della patologia *Patent ductus arteriosus* (PDA): una condizione medica congenita che causa la comunicazione persistente tra aorta e arteria polmonare.

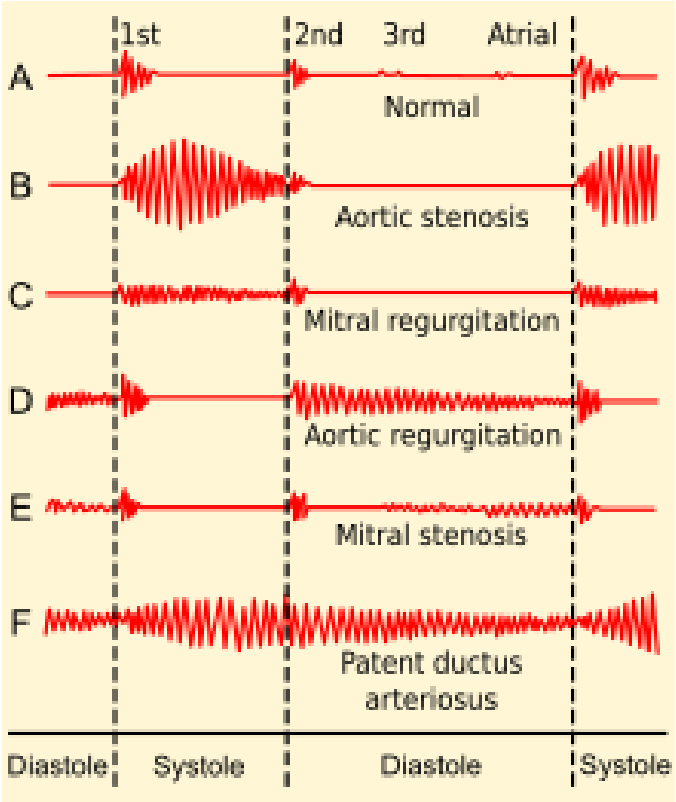

Figura 1: Fonocardiogramma in condizioni fisiologiche e patologiche

I soffi prima citati si presentano come rumori ad alta frequenza che spostano lo spettro da analizzare dai 100Hz fisiologici fino a sopra il 1KHz e spesso fino a 2KHz; quindi, la progettazione di uno strumento atto alla diagnostica consta sicuramente in una banda

passante ben più ampia se confrontato con un corrispettivo dispositivo utilizzato per il solo monitoraggio.

1.2. Caratterizzazione del fonocardiografo digitale

La moderna fonocardiografia digitale per espletare il suo compito si avvale di microfoni atti alla trasduzione del segnale sonoro in segnale elettrico, e di elettronica per il filtraggio, il campionamento e l'immagazzinamento di tali segnali. Non manca, poi, l'implementazione della visualizzazione dinamica o statica che sia su di un supporto visivo.

Il presente progetto si occupa della parte digitale di campionamento e registrazione del segnale, nonostante ciò, non si può prescindere dalla conoscenza dell'elettronica analogica a monte del processo di digitalizzazione. Essa è costituita da tre blocchi funzionali principali: il microfono, l'amplificatore per strumentazione e i filtri.

- Il microfono: è il trasduttore elementare, per questa applicazione ne è stato utilizzato uno con tecnologia elettrete.
- L'amplificatore: è un amplificatore per strumentazione INA128 utilizzato per innalzare il livello di segnale mantenendo comunque un basso livello di rumore.
- I filtri: hanno lo scopo di eliminare le componenti irrilevanti e/o di disturbo nel segnale acquisito. In particolare, è stato utilizzato un filtro di Butterworth del quarto ordine ottenuto come cascata di due medesimi filtri del secondo ordine e un filtro notch per l'eliminazione della componente a 50Hz.

Grazie all'elettronica appena elencata il segnale acquisito spazierà nella banda d'interesse, tra i 20Hz e 2KHz, con alto rapporto segnale/rumore.

La parte digitale, sviluppata attraverso un microcontrollore PIC e diverse periferiche, concettualmente si compone di tutti gli elementi necessari alla digitalizzazione, visualizzazione, memorizzazione ed analisi del segnale. Per quanto riguarda la digitalizzazione essa avverrà attraverso un processo di campionamento che per il teorema di Shannon, data la banda passante, non dovrà essere al di sotto dei 4KHz. La visualizzazione del segnale sarà possibile grazie ad un display grafico TFT, mentre la

memorizzazione avverrà in formato numerico su di una MicroSD, garantendo la portabilità dei dati.

2. Scopo

La bioingegneria si presenta come la scienza atta alla creazione e al miglioramento di strumenti diagnostici e terapeutici in grado di semplificare, migliorare e salvare la vita dell'uomo; essendo il presente progetto aderente a quest'ambito, si proporrà gli stessi obbiettivi limitatamente al campo della diagnosi, in particolare quella delle patologie cardiache, rientrando nella branca denominata fonocardiografia.

Si preannuncia l'obbiettivo di creare un dispositivo compatto, relativamente potente, nonché portatile in grado di monitorare l'attività cardiaca, o più generalmente di visualizzare, immagazzinare e, in parte, analizzare localmente i segnali analogici prodotti nello specifico da un microfono. Si tratta quindi dell'implementazione della parte digitale di un più complesso fonocardiografo il cui utilizzo potrebbe portare giovamento nella diagnosi di particolari patologie cardiache rendendo di più facile e precisa l'interpretazione i suoni prodotti dal cuore durante il suo ciclo di funzionamento. Fornisce inoltre la possibilità di effettuare una memorizzazione multimediale del segnale recepito potendo, quindi, aiutare il clinico nella ricostruzione dell'anamnesi di un paziente e consentendo, previa misurazione, l'analisi del segnale anche su computer dal quale poter effettuare qualsivoglia tipologia di elaborazione e trasformazione dei dati forniti.

3. Materiali e metodi

3.1. Compilatore e programmatore

Per effettuare la programmazione del microcontrollore PIC è necessario utilizzare un compilatore ed un programmatore. Questi due elementi si occupano, rispettivamente, di: tradurre il codice sorgente in codice oggetto e di effettuare la scrittura di tale codice nella memoria del microcontrollore; nello specifico caso dei microcontrollori PIC esso sarà di tipo esadecimale.

Per il presente progetto è stato utilizzato il compilatore MikroC Pro di MikroElektronika, mentre come programmatore USB MikroProg for PIC prodotto dalla medesima ditta,

coadiuvato dal relativo software. La licenza per l'utilizzo dei precedenti programmi ha un costo relativamente ingente se si considera l'esistenza di software equivalenti completamente gratuiti, eppure quest'onere è compensato dalla facilità di utilizzo e dalla vastità di librerie disponibili che vengono fornite all'utente.

Dal punto di vista tecnico: il linguaggio utilizzato dal compilatore è il C di ormai appurata diffusione, flessibilità e semplicità, mentre il programmatore, a livello software, si presenta come un'interfaccia nella quale è possibile selezionare varie opzioni. In particolare, tra queste risaltano le *"configuration word"* ovvero le impostazioni di configurazione del microcontrollore, la selezione avviene facilmente tramite menù a tendina dove, previa scelta del modello di PIC utilizzato, sono consultabili le descrizioni di ogni campo selezionabile sulla base delle indicazioni riportate nel relativo datasheet. Questa particolarità semplifica il processo di configurazione rispetto ad altri compilatori dove deve essere effettuato in codice attraverso comandi opportuni e definizioni numeriche dei registri sicuramente meno pratiche ed intuitive.

3.2. La scelta del PIC18F47K42

In generale i microcontrollori PIC (acronimo di "Programmable Interface Controller") si differenziano tra loro sulla base di molte caratteristiche inerenti a: dimensioni di memoria (RAM ed EEPROM), versioni (ad 8,16 o 32 bit), forma, dimensione, numero di pin, connessioni hardware disponibili in numero e tipo; nonché per numerosi altri parametri.

Allo scopo del suddetto progetto si indirizzò la scelta, in un primo momento, sul modello 18F4550 PDIP 40 pin prodotto dalla Microchip in quanto dotato di prestazioni accettabili ed una vasta diffusione, garantendo quindi un corposo supporto in rete per quanto riguarda l'implementazione lato software delle varie periferiche. In seguito ad un'analisi più approfondita sono emerse limitazioni insormontabili, in particolare in termini di velocità di processamento e numero di pin adibiti a connessioni hardware, del prima citato microcontrollore; tali problematiche hanno fatto sì che la preferenza migrasse su di un controllore di classe superiore: il PIC18F47K42 PDIP 40 pin prodotto dalla medesima casa del precedente.

A discapito della diffusione e della semplicità, questo modello, garantisce prestazioni ben maggiori rispetto al suo fratello minore sotto la maggior parte delle caratteristiche, in particolare le specifiche che hanno evidenziato il 18F47K42 come ottimo candidato riguardano: la velocità di esecuzione (64MHz massimi con clock interno), memoria flash di 128KB, la presenza di due porte di comunicazione SPI nonché il modulo ADC con profondità di campionamento di 12 bit.

3.3. Periferiche

Il microcontrollore PIC ha il compito di gestire le periferiche che svolgeranno le funzioni di interfacciamento e memorizzazione. Per promuovere un facile utilizzo dello strumento da parte dell'utente sono stati aggiunti i seguenti componenti: il display, in grado di fornire l'andamento del segnale e un'interfaccia visivamente intuitiva; pulsanti ed un encoder rotativo per permettere la selezione delle opzioni a discrezione dell'utilizzatore.

3.3.1. Encoder KY-040

Una delle interfacce fisiche sul pannello è fornita dall'encoder rotativo ottico KY-040, esso assume varie funzioni a seconda della porzione di programma nella quale l'utente sta navigando. Nello specifico viene utilizzato ogni qual volta vi è la necessità di selezionare un'opzione da un menù con valori discreti oppure nella fase di analisi per evidenziare la porzione di segnale sulla quale si vuole concentrare lo studio.

Il modello KY-040 è un encoder relativo che quindi riesce a registrare solamente il verso di spostamento e quantificarne l'entità: è privo di riferimenti assoluti. Dispone inoltre della funzionalità di pressione che presenta le stesse caratteristiche di un pulsante. Per comprendere il principio di funzionamento è necessaria la conoscenza delle caratteristiche costruttive dell'encoder: esso è costituito di tre pin per la recezione della rotazione e di un pin adibito alla pressione. La posizione relativa dei pin per identificare movimenti rotativi è rappresentata in *figura 2*, all'interno sono presenti due interruttori congiungenti i pin A e C, e B e C. questi rispettivamente comporranno le due uscite digitali: CLK e DT.

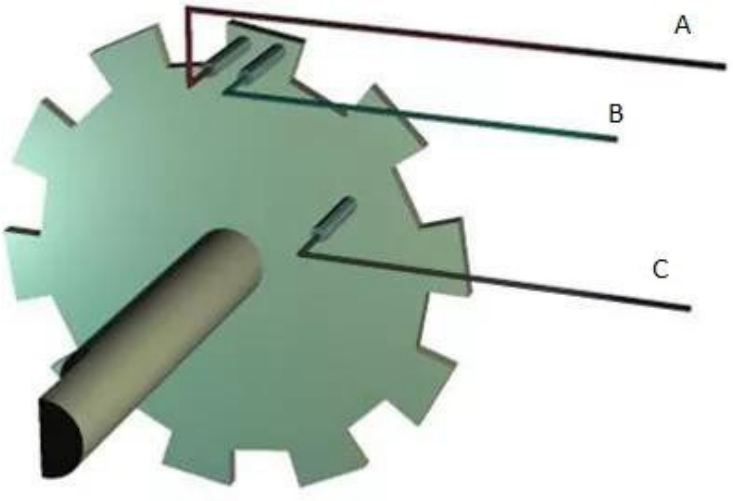

Figura 2: Configurazione dei pin interni all'encoder

Per ogni posizione i due interruttori si troveranno entrambi in condizione concorde: entrambi chiusi o entrambi aperti, durante la rotazione, invece, si noterà la commutazione di un interruttore in anticipo relativamente all'altro, ciò implica il verso di rotazione. In particolare, se l'interruttore tra A e C cambia stato in anticipo rispetto all'equivalente tra B e C allora si è assistiti ad una rotazione in verso orario, viceversa se il segnale in uscita evidenzia una commutazione contraria. Di seguito, in *figura 3*, è rappresentato l'andamento del segnale sui due pin di uscita CLK (in alto) e DT (in basso).

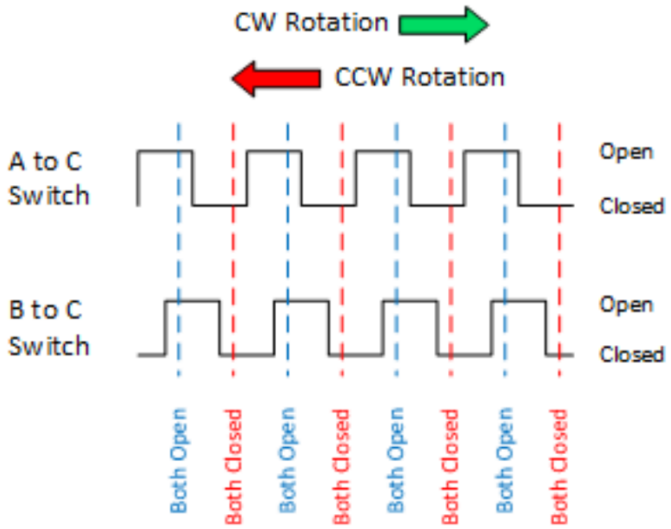

Figura 3: Segnali in uscita dall'encoder in funzione del verso di rotazione

In realtà l'encoder è stato utilizzato già montato su di una scheda che ne facilita l'utilizzo e la connessione. Essa è costituita da cinque pin dei quali due adibiti all'alimentazione:

V collegato al positivo e G a terra, con specifiche resistenze del valore di 10 KΩ atte a proteggere da sovratensioni; CLK e DT impiegati per l'uscita digitale del segnale rotativo nelle modalità descritte in precedenza; SW adottato per la rilevazione di una pressione implementando anche la funzionalità di pulsante.

3.3.2. Display grafico TFT ILI9481

Per lo sviluppo del presente progetto è imperativo l'utilizzo di un display grafico in grado di rendere visibile il segnale campionato in ingresso. Oltre a questa funzione primaria vi è stata la possibilità di implementare un menù di selezione rendendo la navigazione tra le opzioni di visualizzazione e memorizzazione molto più intuitiva. Per di più contribuisce a diminuire in numero gli elementi d'interfacciamento fisico con l'utente (pulsanti, encoder), poiché la loro funzione viene ridefinita a livello software e visualizzata sul display a seconda delle necessità e della schermata di selezione alla quale si sta facendo riferimento.

Il display utilizzato ha dimensione di 3.2" con risoluzione grafica 480x320 a colori ed utilizza la tecnologia TFT (thin film transistor). È connesso con il microcontrollore tramite una connessione in parallelo semplice e veloce, descritta in seguito, con la possibilità di alimentarlo a 5V o 3.3V. Il controllore del display supporta velocità di comunicazione fino ad un massimo di 20MHz risultando la più lenta tra le periferiche e ponendosi come stadio determinante della velocità di tutto il sistema, limitando necessariamente il clock del microcontrollore a 16MHz.

Lato software l'implementazione è stata facilitata dall'uso di una libreria disponibile in rete la quale, con opportune modifiche, fornisce tutte le funzioni necessarie a graficare e scrivere su schermo. In particolare, per la visualizzazione del segnale si ha la possibilità di utilizzare la funzione "display_drawPixel" oppure "writeLine" rispettivamente graficanti un singolo pixel e una linea congiungente due pixel in coordinate date.

3.3.3. Scheda di memoria MicroSD

La memorizzazione dei dati acquisiti deve soddisfare requisiti dettati dal tipo di applicazione e dalla semplicità di utilizzo: interoperabilità e trasportabilità. In un primo momento, concentrandosi più sulla velocità di comunicazione ed immagazzinamento si è utilizzata una memoria esterna di tipo EEPROM interfacciata con il microcontrollore

attraverso una comunicazione I2C. Durante i test è stata utilizzata la memoria 24C256 prodotta da ATMEL con buoni risultati in termini di prestazioni a fronte di scarsa capacità di memoria e pressocché nulla portabilità dei dati al di fuori dello strumento stesso, inoltre la connessione I2C per garantire velocità elevate deve essere effettuata a livello hardware con restrizioni sui pin utilizzabili.

Causa queste limitazioni si è spostata l'attenzione su di un lettore per schede MicroSD comunicante attraverso interfaccia SPI che, viceversa per quanto riscontrato nella EPPROM, garantisce una maggiore dimensione di memoria seppur accessibile a velocità minori, ed anche flessibilità nell'utilizzo di pin "*general purpose*". Come precedentemente accennato, la scelta è stata dettata anche da una questione di portabilità e leggibilità dei dati, infatti: le memorie MicroSD sono da tempo diffuse in commercio e leggibili dalla maggior parte dei computer. La memoria è estraibile attraverso un opportuno supporto che ne consente un facile utilizzo nonché una connessione agevole grazie ai pin integrati. Quest'ultimi sono sei in totale: due utilizzati per alimentare la scheda e quattro adibiti allo scambio di dati tramite comunicazione SPI che sarà descritta nel dettaglio successivamente.

La scheda di memorizzazione in sé è una SanDisk da 4GB di classe 4 (4MB/s minimi in scrittura), ma qualunque MicroSD di dimensioni comprese tra 4GB e 32GB è adatta in quanto unificate nel supportare il file system FAT32 (File Allocation Table) utilizzato nel presente progetto, al contrario MicroSD con capacità minori o maggiori del range citato, non saranno compatibili in quanto rispettivamente formattate in FAT12 o FAT16 e exFAT.

Dal punto di vista della tecnologia è una memoria flash: un dispositivo a stato solido con la caratteristica di essere non volatile e quindi in grado di mantenere i dati immagazzinati anche ad alimentazione assente, qualità indispensabile visto lo scopo del progetto.

3.4. Alimentazione

Il microcontrollore e le sue periferiche sono alimentati tramite batteria attraverso il modulo mb v2 prodotto dalla Elegoo. La suddetta scheda è stata pensata per l'alimentazione in fase di sviluppo e quindi dotata di due serie di pin sottostanti al modulo stesso per devolvere tensione alla due coppie di linee, positivo e terra,

normalmente presenti nelle classiche breadboard. La tensione di alimentazione delle due linee può essere modificata individualmente tramite il posizionamento opportuno di due jumper posti nella parte superiore, in particolare il collegamento dei pin off e 3.3V garantisce una tensione di 3.3V lungo la linea, mentre connettendo il pin 5V ad off si ottiene un'alimentazione di 5V. Vi è anche la possibilità di rendere inattiva una o entrambe le linee cortocircuitando i due pin off tramite il jumper. Di seguito in *figura 4*, si è riportato lo schematico dei pin precedentemente descritti.

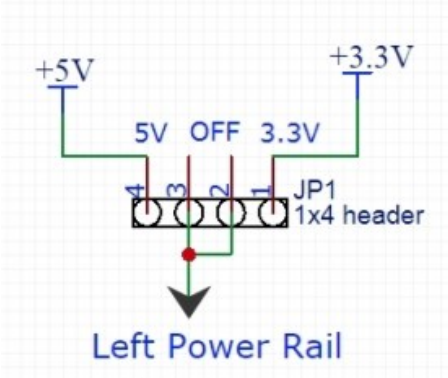

Figura 4: Configurazione dei pin di settaggio del modulo di alimentazione

Oltre all'alimentazione delle linee, necessaria per le periferiche, il modulo presenta sulla parte superiore quattro pin, due a 3.3V e due a 5V, con relativi contatti per la terra: due di questi saranno usati per alimentare il microcontrollore stesso.

L'alimentazione a monte del modulo è fornita da una classica batteria da 9V (codice IEC 6LR61) il cui voltaggio viene opportunamente abbassato ad opera dei due regolatori di tensione, tipo AMS1117 prodotti da Advanced Systems, integrati a bordo del modulo.

3.5. Strumenti da banco e programmi

Oltre a microcontrollore e periferiche sono stati utilizzati anche degli strumenti da banco per effettuare i test, questi sono:

- oscilloscopio *RIGOL MSO5102* utilizzato principalmente per la valutazione delle prestazioni di alcune porzioni di codice.
- generatore di onde *SIGLENT SDG2042X* impiegato per generare segnali noti e valutarne l'effettiva recezione e memorizzazione da parte del dispositivo.

Per disegnare gli schematici dei circuiti è stato utilizzato il software *CircuitMaker* dovendo anche creare, utilizzando uno strumento del programma stesso, quelle periferiche che non erano già presenti nel database.

La schematizzazione del programma sottoforma di diagramma di flusso è stata effettuata tramite il software *DIA* permettendo una maggiore consapevolezza in fase di programmazione.

4. Risultati

4.1. Connessioni

4.1.1. Connessione in parallelo

La connessione tramite interfaccia parallela del display garantisce velocità e semplicità di utilizzo a livello software a fronte di un numero di pin occupati elevato relativamente ad altre modalità di connessione. In particolare, vengo utilizzati tredici pin dei quali:

- D0, D1, D2, D3, D4, D5, D6, D7: sono la porta dati.
- RST: pin di reset.
- WR: write operation.
- RS: register select.
- RD: read operation.
- CS: chip select.

Di seguito lo schema circuitale secondo il quale si è proceduto a collegare il microcontrollore con il display senza particolari attenzioni sulla tipologia di pin scelti, fermo restando che la porta di comunicazione dati equivale direttamente alla porta D del PIC18F47K42 in modo da facilitare la programmazione e ridurre il numero d'istruzioni.

Sia l'intera porta dati D che i pin relativi alle altre funzioni sono stati configurati opportunamente come uscite attraverso i registri TRIS relativi. Inoltre, tramite i registri ANSEL ogni porta implicata è settata in digitale.

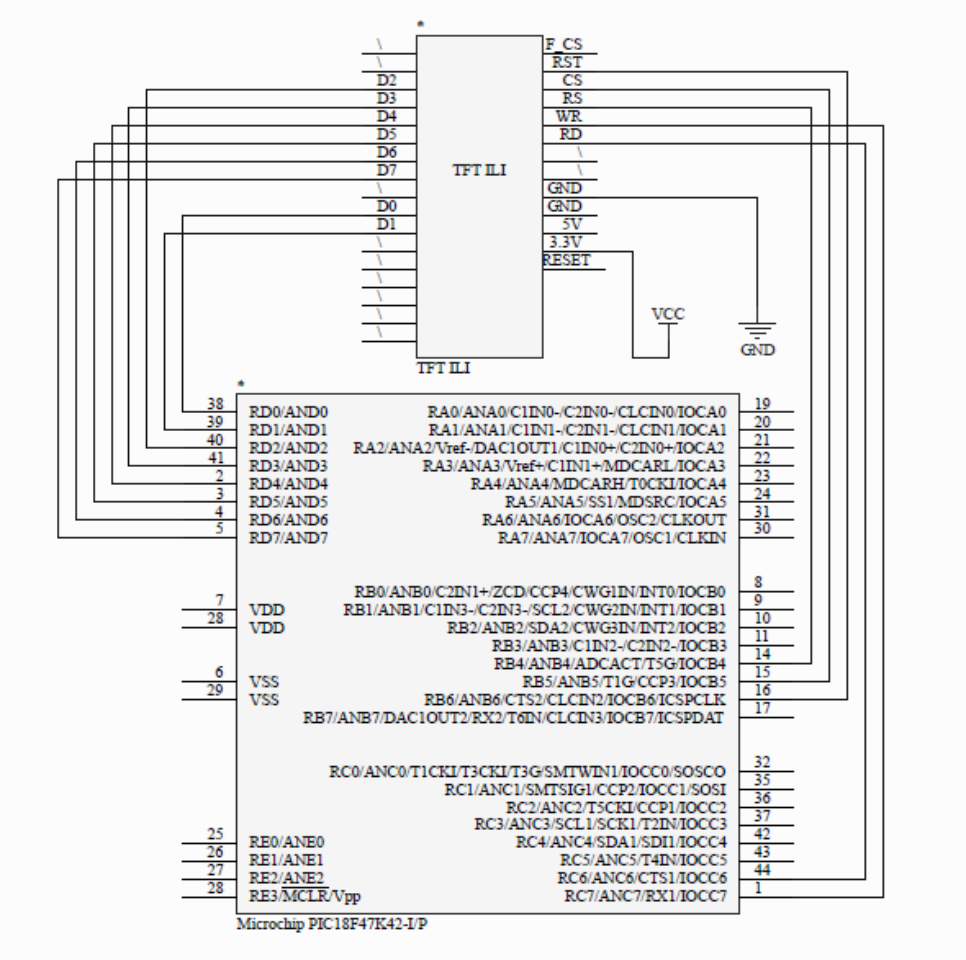

Figura 5: Connessione in parallelo del display

4.1.2. Connessione Seriale SPI

La connessione seriale è stata implementata per la comunicazione del microcontrollore con la scheda montante la MicroSD. In linea del tutto generale l'interfaccia SPI gestisce una comunicazione Master-Slave sincrona e full duplex ottenuta tramite registri a scorrimento contenenti i dati da scambiare. Consta di quattro linee di collegamento:

- MISO: Master Input Slave Output
- MOSI: Master Output Slave Input
- CS: Chip Select (equivalentemente anche Slave Select)
- SCK: Serial Clock

L'inizio della comunicazione è gestito dal dispositivo Master così come il segnale di clock, che viene fornito dal medesimo, attraverso una linea dedicata. Prima dell'effettivo avvio della comunicazione il dispositivo Master attiva la linea CS relativa allo Slave con il quale

intende dialogare, fornendo in un successivo momento il clock con frequenza caratterizzante la velocità della comunicazione stessa. Una volta identificato ed attivato lo Slave, i bit appartenenti al registro a scorrimento del Master vengono traslati verso l'esterno attraverso la linea MOSI partendo dal bit più significativo (MSB). Il bit traslato si trasferisce nel registro dello Slave, il quale a sua volta inizia a svuotare il proprio registro inviando il bit più significativo attraverso la linea MISO. La comunicazione termina in corrispondenza dell'ottavo bit trasmesso.

Esistono quattro modalità di connessione SPI che sono combinazioni di due parametri binari identificativi: la polarità del clock e la fase dello stesso. La polarità indica l'effettuazione delle operazioni sul fronte di salita o di discesa del clock, mentre la fase configura su quale di questi due fronti deve avvenire la lettura dei dati. Nel particolare si è utilizzata la modalità zero con clock non invertito (operazioni sul fronte di salita) e lettura sempre sul medesimo fronte.

Nello specifico le connessioni fisiche sono state implementate come in *figura 6*.

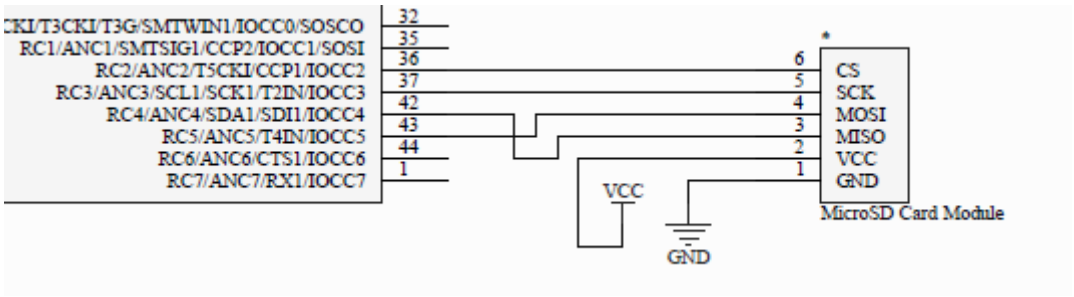

Figura 6: Connessione seriale del modulo MicroSD

Il dispositivo slave è la scheda montante la MicroSD, mentre il ruolo del Master è ricoperto dal microcontrollore. Per quanto descritto prima, in fase di programmazione, i pin relativi a CS, SCK e MOSI saranno configurati come output, mentre il restante MISO in modalità input, naturalmente tutti settati in digitale.

4.2. Modulo ADC interno

Il modulo ADC del PIC18F47K42 è un convertitore analogico-digitale che sfrutta la tecnologia sample and hold per produrre, dato un segnale analogico in ingresso, un risultato a 12bit, memorizzato temporaneamente in due registri ADRESH e ADRESL, rispettivamente il byte più e meno significativo.

Ovviamente, la precisione del segnale acquisito, a valle dell'elettronica relativa al microfono, dipende fortemente dalla frequenza e dalla profondità di campionamento. La prima caratteristica è direttamente proporzionale alla frequenza del clock scelto in fase di settaggio dei registri relativi, a meno di eventuali divisori e moltiplicatori, anch'essi configurabili. La profondità di campionamento, invece, rappresenta il parametro principe per comprendere l'entità dell'approssimazione fatta nella trasformazione del segnale da continuo a discreto in ogni istante adibito al campionamento.

Scendendo nelle specifiche del PIC scelto, il modulo ADC lavorerà ad una profondità di 12bit: il segnale in ingresso verrà recepito e campionato quantizzando la dinamica in 2^{12}=4096 livelli. A valle del microfono un amplificatore con guadagno regolabile consente di avere in uscita un segnale opportunamente variabile in ampiezza. Ciò è richiesto in quanto il modulo ADC viene controllato da una serie di registri, tra i quali, quello relativo ai voltaggi di riferimento: ADREF permette la selezione dei bit che specificano i voltaggi negativo (NREF) e positivo (PREF). In particolare, essi sono stati settati in modo da riferirsi al voltaggio con il quale lo stesso MCU viene alimentato ovvero 3.3V, senza dover utilizzare potenziali di riferimento esterni giovando all'economia, in termini di pin occupati, di tutto il sistema.

Date le precedenti considerazioni si può facilmente calcolare l'errore di quantizzazione introdotto dall'operazione di campionamento, infatti, avendo una dinamica del segnale in ingresso pari al massimo a 3.3V e 4096 livelli di quantizzazione il rapporto tra i due restituisce circa 0.8mV ovvero l'ampiezza di ciascun livello. L'errore massimo che si può commettere sarà quindi pari alla metà dell'ampiezza dell'intervallo di quantizzazione, ovvero ±0.4mV contro i ± 1.6mV che invece avrebbe introdotto lo stesso processo a bordo del PIC18F4550 equipaggiato con un modulo ADC a soli 10bit.

Tra i registri di settaggio del modulo ADC alcuni sono relativi alla fonte di clock; quest'ultima viene specificata tramite un singolo bit del registro ADCON0 indicando la volontà di utilizzare l'oscillatore dedicato FRC oppure lo stesso segnale di clock che dirige il MCU(HFINTOSC). Si è scelto di utilizzare la seconda opzione in quanto superiore in termini di prestazioni e stabilità. A tal proposito, entrambi gli oscillatori, essendo interni,

sono implementati tramite un circuito oscillante di tipo RC incline ad instabilità dovute a variazioni di temperatura. È comunque garantita dal produttore una variabilità massima del ±1% rispetto alla periodicità dichiarata per quanto riguarda l'utilizzo dell'HFINTOSC.

Di pari passo con la selezione del clock interno a frequenza FOSC va definito anche il registro ADCLK indicante il valore del divisore di tale frequenza. Non è possibile, infatti, fornire all'ADC il clock alla frequenza originale del microcontrollore. Inoltre, dal datasheet si apprendono i range raccomandati per la scelta del parametro ed anche i relativi T_{AD} (tempo di conversione per singolo bit) per ogni combinazione FOSC e divisore. Avendo un clock con FOSC pari a 16MHz si è scelto un divisore pari a 16 selezionando l'opzione più veloce (T_{AD}=1.0 µs) rimanendo comunque conformi alle specifiche fornite dal produttore.

TABLE 36-1: ADC CLOCK PERIOD (TAD) VS. DEVICE OPERATING FREQUENCIES(1,4)

ADC Clock Period (TAD)		Device Frequency (Fosc)						
ADC Clock Source	CS<5:0>	64 MHz	32 MHz	20 MHz	16 MHz	8 MHz	4 MHz	1 MHz
Fosc/2	000000	31.25 ns(2)	62.5 ns(2)	100 ns(2)	125 ns(2)	250 ns(2)	500 ns(2)	2.0 µs
Fosc/4	000001	62.5 ns(2)	125 ns(2)	200 ns(2)	250 ns(2)	500 ns(2)	1.0 µs	4.0 µs
Fosc/6	000010	125 ns(2)	187.5 ns(2)	300 ns(2)	375 ns(2)	750 ns(2)	1.5 µs	6.0 µs
Fosc/8	000011	187.5 ns(2)	250 ns(2)	400 ns(2)	500 ns(2)	1.0 µs	2.0 µs	8.0 µs
...	...	...	...	...	...	...	...	...
Fosc/16	000111	250 ns(2)	500 ns(2)	800 ns(2)	1.0 µs	2.0 µs	4.0 µs	16.0 µs(3)
...	...	...	...	...	...	...	...	...
Fosc/128	111111	2.0 µs	4.0 µs	6.4 µs	8.0 µs	16.0 µs(3)	32.0 µs(2)	128.0 µs(2)
FRC	CS(ADCON0<4>) = 1	1.0-6.0 µs	1.0-6.0 µs	1.0-6.0 µs	1.0-6.0 µs	1.0-6.0 µs	1.0-6.0 µs	1.0-6.0 µs

Legend: Shaded cells are outside of recommended range.
Note 1: See TAD parameter for FRC source typical TAD value.
2: These values violate the required TAD time.
3: Outside the recommended TAD time.
4: The ADC clock period (TAD) and total ADC conversion time can be minimized when the ADC clock is derived from the system clock FOSC. However, the FRC oscillator source must be used when conversions are to be performed with the device in Sleep mode.

Figura 7: Tabella dei tempi di conversione da datasheet del PIC18F47K42

Dalla precedente tabella si evince, nonostante non sia esplicitamente specificato, come l'oscillatore dedicato FRC pecchi di un'instabilità pronunciata nei propri cicli di clock, in quanto il valore T_{AD} è mutevole in un range tra 1.0 µs e 6.0 µs.

4.3. TIMER0

I microcontrollori PIC dispongono di un vasto assortimento di timer interni in grado di generare segnali periodici in funzione di diverse fonti di clock interne oppure esterne.

Solitamente sono utilizzati come trigger per una o più routine di interrupt, mentre nel presente progetto, uno di essi, assume la funzione di conteggio dei secondi dal momento di attivazione del suddetto timer.

Il funzionamento si basa sulla variazione di un bit del registro PIR3, in particolare il bit denominato TMR0IF. Il valore di questo bit è tipicamente pari a zero, ma in caso di un overflow esso assume il valore uno. L'overflow è dettato da una saturazione dei registri TMR0H e TMR0L, rispettivamente byte più e meno significativo, se si adotta la configurazione a 16bit, altrimenti dal solo TMR0H in 8bit. Questi registri vengono incrementati di 1 ad ogni impulso di clock, quindi, dimensionando opportunamente i prescaler e i postscaler, nonché scegliendo la fonte di clock più adatta, si possono ottenere attivazioni del flag TMR0IF con periodicità diverse. È inoltre possibile precaricare i registri TMR0H e TMR0L ottenendo una ancor più variabile temporizzazione.

Nel presente progetto è emersa la necessità di implementare un timer con temporizzazione pari ad un secondo, per far ciò è necessario configurare opportunamente i registri di controllo del timer sulla base dei seguenti calcoli.

Il massimo delay ottenibile è pari a:

$$T_{istruzione} \cdot Prescaler \cdot 2^{bit} = T_{\max Delay}$$

Dove con *bit* si è indicato il numero di bit disponibile al contatore: 16 o 8 a seconda della selezione scelta attraverso il bit MD16 del registro T0CON0. Il tempo di istruzione dipende dal clock scelto attraverso il registro T0CON1 tramite i tre bit denominati CS. In particolare, si è scelto un timer a 16bit e clock pari a FOSC/4 che implica un tempo di istruzione pari a 0.25µs in quanto la frequenza dell'oscillatore è di 16MHz. Ponendo il tempo massimo di delay pari proprio ad 1 secondo si può risolvere la precedente equazione in funzione di *Prescaler* trovando il valore, tra i disponibili, più consono all'applicazione:

$$Prescaler = \frac{1s}{0.25\mu s \cdot 2^{16}} \cong 61.04$$

Tra i valori assegnabili al prescaler, tramite i tre bit CKPS del registro T0CON1, il più congruo è pari a 64. Effettuata la scelta si ricalcola il tempo di massimo delay inserendo questa volta il valore del prescaler, ottenendo un intervallo pari all'incirca 1.05 s tra un overflow e l'altro. Per compensare questo sfasamento dal tempo desiderato è necessario precaricare i registri TMR0H e TMR0L con un opportuno valore, in particolare il tempo di overflow dei registri può esser caratterizzato dalla seguente equazione:

$$T_{overflow} = T_{istruzione} \cdot Prescaler \cdot (2^{bit} - Preload)$$

Come prima, ponendo il tempo di overflow pari ad 1 secondo si ottiene che il *Preload* necessario allo scopo è 3036, ovvero 0x0BDC in esadecimale, facilmente distribuibile sui due registri adibiti al conteggio come: TMR0H=0x0B e TMR0L=0xDC.

4.4. Interfacciamento utente

Grazie al display grafico ed alle periferiche di interazione fisica l'utente è in grado di navigare nel programma e di ricevere riscontri da esso. Nel presente paragrafo verranno discusse, in un'ottica concettuale, le varie porzioni del programma, i rispettivi collegamenti pilotati da specifiche interazioni utente ed il funzionamento a livello software di quest'ultime.

4.4.1. Struttura ed aspetto del programma

Il programma è strutturato in modo da fornire diverse schermate di visualizzazione a seconda della scelta utente, quest'ultima è espletabile tramite periferiche fisiche la cui funzione varia in base alla sezione nella quale si sta navigando. La possibilità di ridefinire via software le operazioni associate a queste periferiche consente di diminuirne il numero, in particolare: due pulsanti e un solo encoder rotativo.

All'accensione il dispositivo visualizza una schermata di caricamento caratterizzata da una grafica ed una barra di progressione fornente informazioni sull'avanzare dell'inizalizzazione. Durante questo processo sono stati implementati vari controlli, se uno di essi non risulta soddisfatto l'avanzamento della barra si arresta e il relativo errore viene stampato a schermo richiedendo, laddove necessario, l'intervento dell'utente ed attendendo un determinato lasso di tempo per poi effetturare nuovamente il controllo.

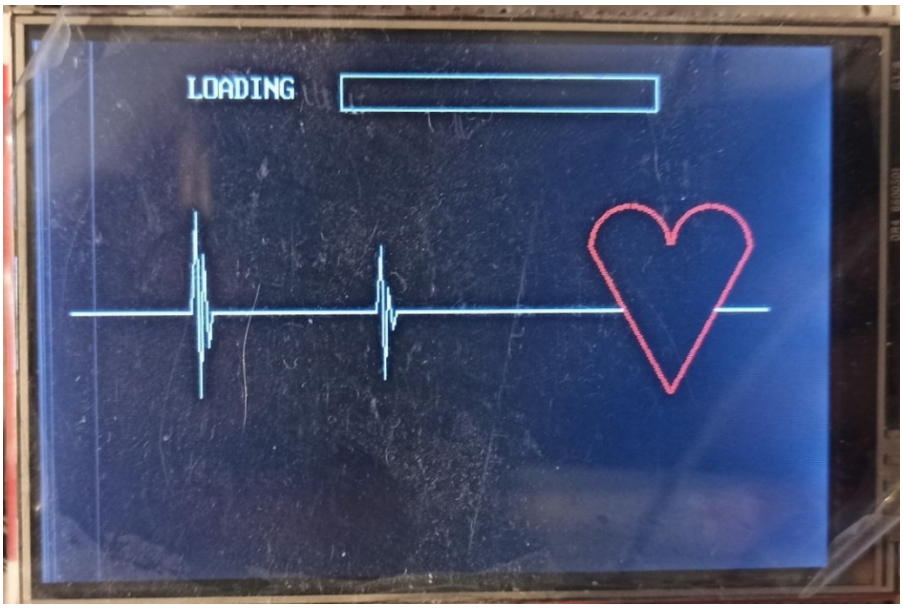

Figura 8: Schermata di caricamento

Se il processo di caricamento è andato a buon fine si può procedere con la visualizzazione della prima schermata, la cui vista è rappresentaviva del reale aspetto grafico del programma, questa fornisce subito l'andamento del segnale nel tempo, prelevato dall'ADC e opportunamente dimensionato in rapporto allo schermo. Oltre al segnale sono indicate a lato le funzioni relative ai due pulsanti, mentre l'encoder, se ruotato, va a modificare il tempo di acquisizione del segnale: parametro d'ingresso nella fase di registrazione che può assumere i valori discreti di 5s, 10s, 15s oppure 30s.

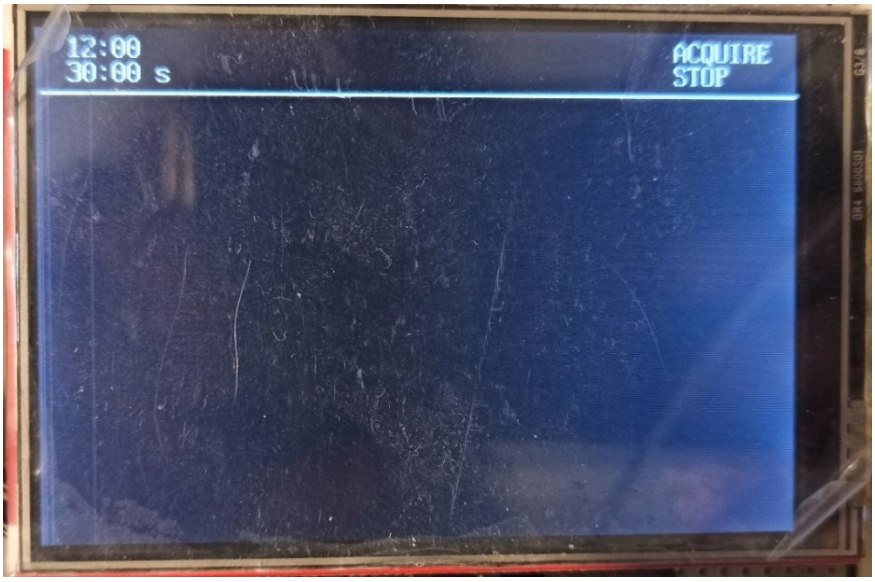

Figura 9: Schermata di visualizzazione dinamica

Alla pressione del tasto A è associata la funzione *"acquire"* che dà inizio alla memorizzazione del segnale fonocardiografico. Il tasto B, sia nella prima schermata di visualizzazione che durante l'acquisizione, funge da "*stop*" ovvero interrompe il campionamento lasciando in vista l'andamento del segnale al momento dell'interazione.

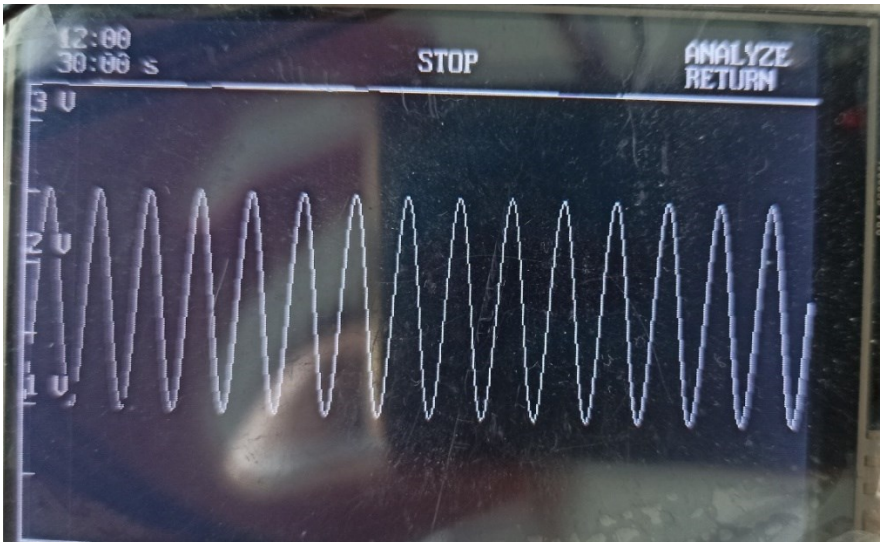

Figura 10: Schermata di visualizzazione statica

Successivamente compariranno in cascata due barrette grafiche verticali che, tramite rotazione dell'encoder, potranno esser posizionate sullo schermo andando ad evidenziare il tempo intercorso tra le due fornendo un prima analisi temporale.

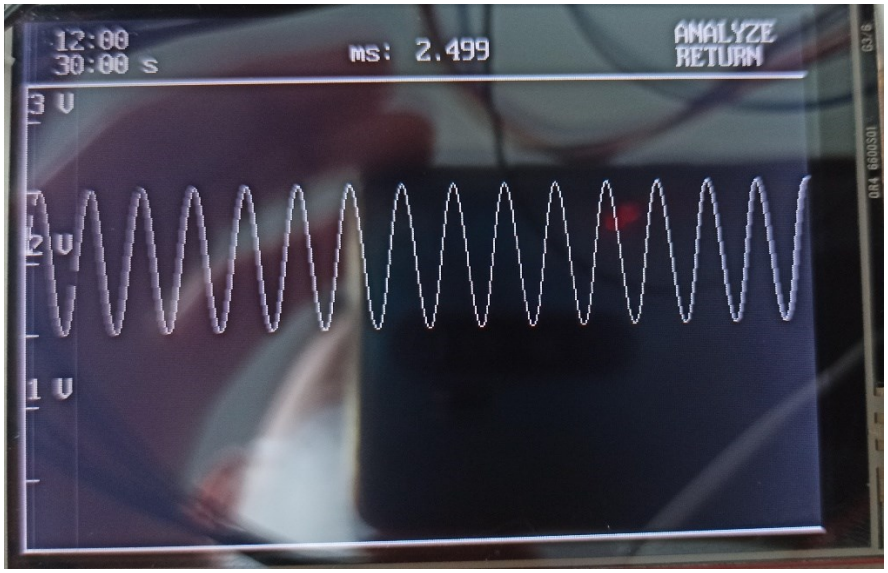

Figura 11: Schermata di analisi con segnale in ingresso sinusoidale 1V a 400Hz

Di seguito, in *figura 12*, è rappresentato il diagramma di flusso descrivente qualitativamente il programma e le procedure in funzione delle interazioni effettuate.

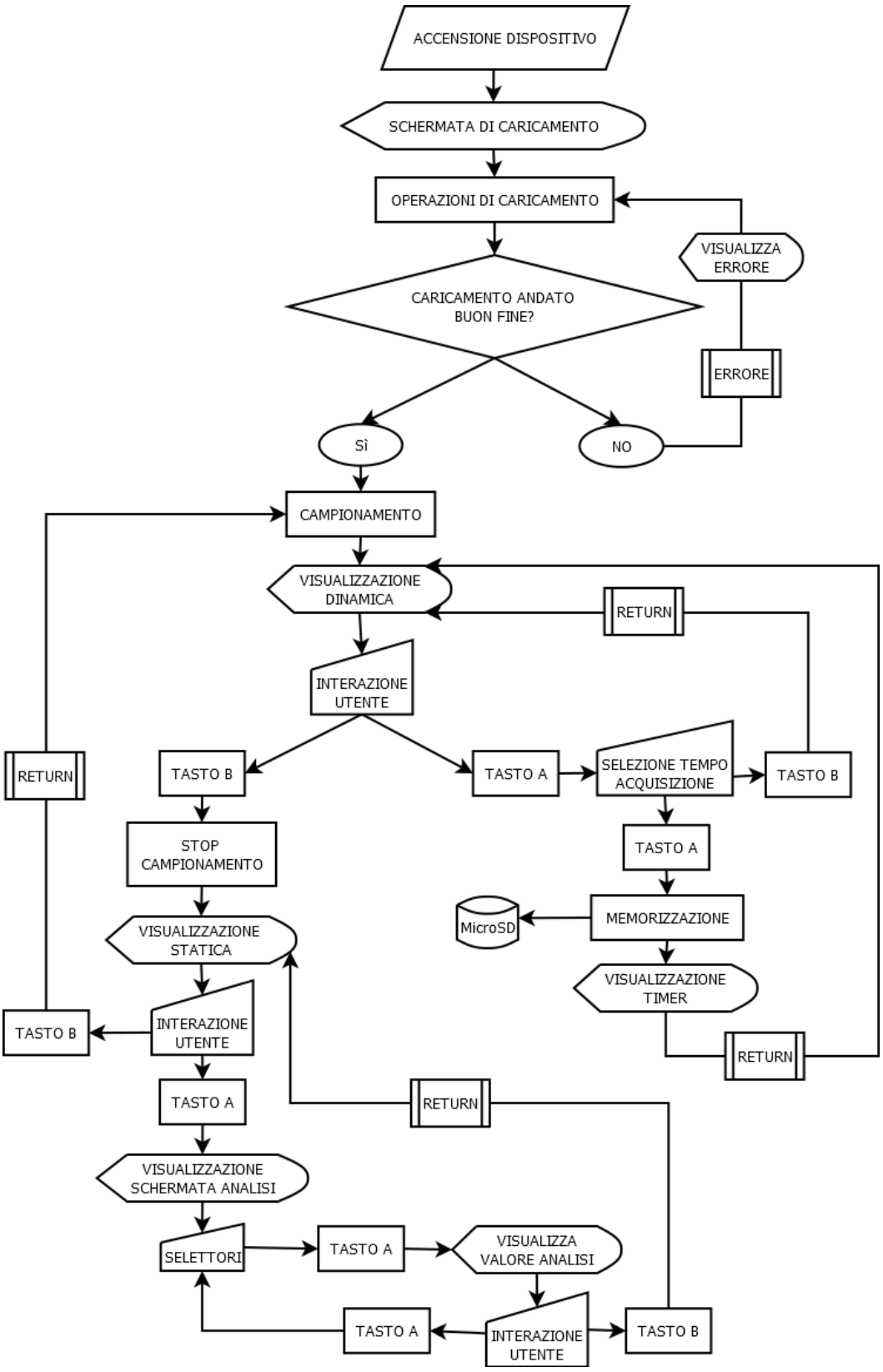

Figura 12: Diagramma di flusso del programma

4.4.2. Implementazione software encoder

Di seguito viene riportata una frazione di codice, del tutto generale, che evidenzia il funzionamento software dell'encoder rotativo. Questa stessa implementazione, con opportune modifiche, sarà usata nel programma completo ogni qual volta l'utente interagirà con l'encoder, per questo motivo si è preferito descriverla decontestualizzata, evitando ridondanze.

```
/*Per facilità di utilizzo i 3 pin dell'encoder vengono
definiti con i seguenti nomi*/
#define ENCODER_SW      RB0_bit
#define ENCODER_DT      RB1_bit
#define ENCODER_CLK     RB2_bit

/*Pin relativi all'encoder selezionati come ingressi e tutta
la porta B configurata in digitale.*/
TRISB = 0b00000111;
ANSELB= 0b00000000;

/*Definizione delle variabili: rotposition è una variabile
che funge da contatore venendo incrementata o decrementata
a seconda del verso di rotazione. Value e rot sono due
variabili di appoggio per effettuare il confronto tra
posizione precedente ed attuale dell'encoder.*/
short rotposition=0;
int  value, rot;

/*Si assegna alla variabile rot il valore iniziale del pin
CLK in modo da rendere invariante il codice dalla posizione
iniziale dell'encoder stesso la quale è ignota*/
rot=ENCODER_CLK;
```

*/*Loop infinito che controlla ripetutamente lo stato dell'encoder, se esso è equivalente allo stato iniziale, o in generale, allo stato precedente, allora non si è avuta nessuna rotazione e non viene effettuata nessuna operazione in quanto la condizione del primo if non è soddisfatta. Altrimenti il codice prosegue controllando l'uguaglianza o meno del pin* DT *con* CLK *se essi risultano diversi c'è stata una chiusura dell'interruttore A in anticipo (rotazione in senso orario), altrimenti se uguali la rotazione è avvenuta in senso antiorario. Infine il valore di* rot *viene ridefinito con ciò che è stato appena letto sul pin* CLK *in modo da rendere possibile il controllo successivo relativamente alla posizione attuale.*/

```
while(1)
     {
      value=ENCODER_CLK;
          if(value!= rot)
          {
               if(ENCODER_DT!=value)
                    {
                    rotposition++;
                         }
                     else{
                         rotposition--;
                         }
          }
rot=value;
     }
```

Il precedente listato presenta problemi di quel che in gergo è chiamato *bouncing:* ovvero una o una serie di letture errate causa rimbalzi meccanici creatisi vista la non idealità costruttiva della periferica. Solitamente questa incertezza viene risolta ponendo un ritardo tra una lettura e l'altra, consentendo l'assestarsi meccanico del sistema. In questa applicazione particolare, essendo la periferica utilizzata per effettuare operazioni

dispendiose anche dal punto di vista temporale, non sono stati sempre adottati ritardi in quanto la mole di operazioni svolta successivamente alla lettura è risultata sufficientemente importante da consentire un implicito *debouncing.*

4.4.3. Implementazione Pulsanti

Così come per l'encoder anche l'utilizzo dei pulsanti risulta frequente durante lo svolgersi del programma, meritandosi una descrizione più generale che può essere caratterrizata al caso specifico direttamente rifacendosi al codice completo in appendice.

Prima di descrivere l'implementazione software è necessario evidenziare che per ogni pulsante la connessione necessita di una resistenza di pull up del valore di 10KΩ in modo da garantire il livello logico richiesto.

Il codice, riportato nelle seguenti righe, risulta semplificato dalla libreria *button* offerta dal compilatore. Essa consta di una sola funzione, per l'appunto Button, che richiede in ingresso quattro valori: la porta dove è collegato il pulsante, il relativo pin, il tempo di pressione necessario a far scattare l'interruttore (in millisecondi) e lo stato attivo, ovvero se si considera la pressione o il rilascio come evento trigger. Restituisce il valore 255 nel caso che il pin sia rimasto nello stato definito come attivo per il tempo indicato, altrimenti zero.

```
/*il programma si rifà ad un generico tasto posto sul pin RC1, questo deve essere impostato come ingresso tramite l'opportuno registro TRIS e come digitale tramite ANSEL*/
TRISC = 0b00000010;
ANSELC= 0b00000000;

/*le variabili dichiarate hanno la funzione di flag, definite come un singolo bit*/
bit state,action;

/*se il pulsante vine premuto per un intervallo di 1ms il valore di state si aggiorna ad uno. Se nel frattempo il pulsante è stato rilasciato, ovvero ha cambiato il suo stato
```

```
attivo, allora si resetta il valore del flag state e si
genera qualsiasi tipo di azione, in questo caso
l'aggiornamento di una variabile*/

if(Button(&PORTC,1,1,1)){
              state=1;
           }
      if(state && Button(&PORTC,1,1,0)){
               state=0;
               action=1;
      }
```

Grazie alle funzione Button il *debuoncing* dei pulsanti non è strettamente necessario in quanto viene già controllato il tempo di attivazione del bottone eliminando falsi contatti dovuti a rimbalzi meccanici le cui durate saranno auspicabilmente diverse da quelle selezionate in codice per la pressione o il rilascio.

Nell'esempio è stata proposta, come azione a seguito della pressione del pulsante, non a caso la variazione del valore di un flag, infatti: questa tipologia di implementazione è stata sfruttata proprio per entrare ed uscire dalle *subroutine* del programma in base al valore dei flag in questione.

4.5. Inizializzazione e controlli

All'accensione del dispositivo è necessario inizializzare tutti i vari moduli utilizzati, nonché effettuare controlli sullo stato del dispositivo, in particolare per quanto riguarda la memoria. Di seguito è riportata la porzione di codice, con ampi commenti, relativa agli aspetti appena elencati. Il monitoraggio del caricamento è consultabile dall'utente grazie ad una schermata la cui programmazione grafica è omessa in quanto risultante di un susseguirsi quasi periodico delle identiche funzioni. Sono invece riportate le righe di codice relative alle stringhe stampate a schermo durante i controlli a scopo di feedback, le definizioni dei registri e le funzioni di inizializzazione di periferiche e connessioni.

```
/*inclusione delle librerie: standard input output del
compilatore, mentre Display_LIB e __Lib_FAT32 esterne,
relative, rispettivamente al display ed alla memoria
MicroSD.*/

#include "Display_LIB.c"
```

```
#include <stdio.h>
#include "__Lib_FAT32.h"
```

*/*definizione di due nuovi tipi di variabile per facilitare la scrittura in fase di programmazione*/*

```
typedef unsigned short          uint8;
typedef signed short            int8;
```

*/*Definizione delle dipendenze esterne per l'utilizzo della libreria MMC (Multi Media Card) necessarie in un secondo momento per poter inizializzare la libreria* __Lib_FAT32. *La prima istruzione seleziona il pin adibito a Chip Select, mentre la seconda ne individua la direzione*/*

```
sfr sbit Mmc_Chip_Select at LATC2_bit;
sfr sbit Mmc_Chip_Select_Direction at TRISC2_bit;
```

*/*Definizione di due funzioni per l'inizializzazione della comunicazione SPI con velocità diverse. In entrambe è stata utilizzata la funzione* SPI1_Remappable_Init_Advanced *di MikroC, essa richiede in ingresso quattro parametri: la modalità di funzionamento, l'istante di campionamento del dato, la polarità del clock e il fronte d'onda scelto per la trasmissione. Procedendo in ordine: la modalità di funzionamento fa riferimento alla selezione del master (in questo caso il PIC) e definisce il divisore del clock. L'istante di campionamento può essere selezionato alla fine o nel centro dell'intervallo, la polarità del clock e la scelta del fronte d'onda (in salita o in discesa) caratterizzano la tipologia di comunicazione SPI che si vuole effettuare tra le 4 esistenti. In questo caso si tratta della modalità zero con polarità del clock impostata a valore logico basso e fronte d'onda da basso ad alto. Entrambe le funzioni adottano gli stessi parametri a differenza del divisore del clock, rispettivamente 64 e 4, consentendo una*

maggiore velocità di comunicazione nell'utilizzo della seconda funzione./

```
void initSPI(void)
{
SPI1_Remappable_Init_Advanced(_SPI_REMAPPABLE_MASTER_OSC_DI
V64,_SPI_REMAPPABLE_DATA_SAMPLE_MIDDLE,_SPI_REMAPPABLE_CLK_
IDLE_LOW, _SPI_REMAPPABLE_LOW_2_HIGH);
}

void initFastSPI(void)
{
SPI1_Remappable_Init_Advanced(_SPI_REMAPPABLE_MASTER_OSC_DI
V4,_SPI_REMAPPABLE_DATA_SAMPLE_MIDDLE,_SPI_REMAPPABLE_CLK_I
DLE_LOW, _SPI_REMAPPABLE_LOW_2_HIGH);
}

/*definisco la variabile err, utilizzata per rilevare errori
durante il caricamento*/
int8 err;

/*Configurazione I/O dei pin tramite i registri TRIS in
funzione delle necessità delle varie periferiche: 0
corrisponde ad un'uscita, 1 ad un ingresso. Il valore logico
dei pin è settato come basso attraverso i registri PORT. La
configurazione in digitale dei pin è effettuata mediante i
registri ANSEL*/
void main() {
     TRISC = 0b00000011;
     TRISB = 0b00000111;
     TRISD = 0b00000000;
     PORTB = 0b00000000;
     PORTC = 0b00000000;
     PORTD = 0b00000000;
```

```
    ANSELD= 0b00000000;
    ANSELB= 0b00000000;
    ANSELC= 0b00000000;
```

*/*Selezione della frequenza dell'oscillatore interno ad alta frequenza, tramite il registro* OSCFRQ *è stata impostata a 16MHz.*/*

```
    OSCFRQ=0b00000101;
```

*/*Istruzioni necessarie all'avviamento del display, la funzione* display_Starts *inizializza la comunicazione, mentre le seguenti si occupano di aspetti grafici relativi al testo, rispettivamente: il colore, la dimensione, il tipo di font. Infine la funzione* display_setRotation *pone lo schermo in orizzontale.*/*

```
    display_Starts();
    display_setTextColor(255, 255, 255, 0, 0, 0);
    display_setTextSize(1);
    display_setTextFont(8,16,2);
    display_setRotation(1);
```

*/*Il PIC in questione possiede la possibilità di selezionare le caratteristiche dei pin, in particolare, quando si effettuano interfacce come la presente SPI va specificato a quali pin ci si riferisce e la qualità di ognuno di essi tramite apposite funzioni. Anche questa volta è stata utilizzata la rispettiva libreria fornita da MikroC per gestire questa funzionalità chiamata PPS (Periperal Pin Select). Inizialmente si sblocca la possibilità di ridefinire i pin, successivamente tramite* PPS_Mapping_NoLock *si definisce: il pin, la sua configurazione I/O e a quale canale della connessione SPI corrisponde. Ripetuta questa operazione per ogni pin coinvolto si blocca nuovamente la*

definizione PPS. Infine, si può inizializzare la comunicazione SPI a bassa velocità/*

```
Unlock_IOLOCK();
PPS_Mapping_NoLock(_RC4,_INPUT,_SDI1);
PPS_Mapping_NoLock(_RC3,_OUTPUT,_SCK1);
PPS_Mapping_NoLock(_RC5,_OUTPUT,_SDO1);
Lock_IOLOCK();
initSPI();
```

*/*Il seguente blocco di codice si riferisce al primo controllo effettuato dal programma in fase di avvio: la verifica della presenza della MicroSD. È necessario inizializzare la libreria esterna per la gestione della FAT, ciò viene fatto attraverso la funzione FAT32_Init che, oltre a caricare la libreria, restituisce il valore -1 nel caso non sia individuabile un supporto sui pin scelti per la comunicazione oppure esso non sia correttamente formattato. Un semplice* if*, coadiuvato da un ciclo* while*, consente di effettuare controlli periodici, ogni secondo, in caso il supporto di memorizzazione non sia pervenuto alla prima inizializzazione. L'utente viene informato dell'eventuale errore tramite la stampa a schermo e, allo stesso modo, notificato del successo dell'operazione in caso essa vada a buon fine. Oltre al testo un ulteriore feedback è dato dall'accensione di un led rosso per l'errore o verde in funzione del successo dell'operazione*/*

```
err = FAT32_Init();
Delay_ms(1);
if(err<0)
{
        while(err<0)
        {
                err=FAT32_Init();
                Delay_ms(1000);
```

```
                display_setTextColor(255, 0, 0, 0, 0, 0);
                display_setCursor(90,60);
                display_puts("ERROR SD NOT FOUND!");
                PORTE.RE2=1;
            }
            fillRect(89,59,300,25,0,0,0);
            display_setTextColor(102, 255, 30, 0, 0, 0);
            display_setCursor(90,60);
            display_puts("OK SD FOUND!");
            PORTE.RE2=0;
            PORTE.RE1=1;
            Delay_ms(300);
            PORTE.RE1=0;
            fillRect(89,59,300,25,0,0,0);
    }
```

*/*Superate le criticità relative all'individuazione del supporto la comunicazione è instaurata e stabile, può essere velocizzata richiamando la funzione* initFastSPI.**/*

```
    initFastSPI();
```

*/*Il secondo blocco di controllo verifica la presenza del file denominato* SYSTEM.TXT *all'interno della memoria. Questo file serve per la memorizzazione del numero relativo all'ultimo record scritto in modo da poter adottare una numerazione progressiva dei set di dati immagazzinati. Tramite la funzione* FAT32_Exists *si verifica l'esistenza o meno del file prima citato, se essa restituisce -1 significa che vi è stato un errore a livello di comunicazione; perciò, si effettua nuovamente il controllo ogni secondo. Se la funzione restituisce il valore 1 indica la presenza del file e quindi si procederà alla lettura dello stesso immagazzinando il valore nella variabile* indice. *In questa fase le funzioni utilizzate appartengono sempre alla*

medesima libreria e sono: FAT32_Open *per aprire il file specificandone nome e modalità,* FAT32_Read *per leggere effettivamente il file e memorizzarlo in una variabile. Esiste anche la possibilità che sia il primo avviamento del dispositivo oppure che la memoria sia stata formattata, in quel caso la funzione* FAT32_Exists *restituisce 0 in quanto il file cercato è inesistente, questa casistica implica la creazione del file tramite* FAT32_Open *con relativa scrittura dell'indice pari a 001 attraverso la funzione* FAT32_Write*. Anche durante questo blocco di controllo i vari feedback vengono forniti attraverso stampa a schermo di stringhe di testo e accensione di led.*/*

```
err=FAT32_Exists("SYSTEM.TXT");
if(err<0)
{
        while(err<0)
        {
            err=FAT32_Exists("SYSTEM.TXT");
            Delay_ms(1000);
            display_setTextColor(255, 0, 0, 0, 0,
0);
            display_setCursor(90,60);
            display_puts("ERROR");
            PORTE.RE2=1;
        }
        fillRect(89,59,300,25,0,0,0);
        display_setTextColor(102, 255, 30, 0, 0, 0);
        display_setCursor(90,60);
        display_puts("OK ERROR SOLVED!");
        PORTE.RE2=0;
        PORTE.RE1=1;
        Delay_ms(300);
        PORTE.RE1=0;
        fillRect(89,59,300,25,0,0,0);
```

```
}
else
{
     if(err==1)
     {
      fhandle = FAT32_Open("SYSTEM.TXT", FILE_READ);
      Delay_ms(20);
      FAT32_Read(fhandle, indice, sizeof(indice));
     }
     if(err==0)
     {
      display_setCursor(90,60);
      display_puts("CREATING FILE SYSTEM");
      fhandle = FAT32_Open("SYSTEM.TXT", FILE_APPEND);
      FAT32_Write(fhandle,"001",3);
     }
}
err=FAT32_Close(fhandle);
```

4.6. Acquisizione e visualizzazione del segnale

Una volta terminata l'inizializzazione si genera una schermata che fornisce, oltre ad informazioni necessarie per l'interazione, anche l'andamento del segnale nel tempo in "*real time*". Questa visualizzazione è possibile grazie al modulo ADC del PIC, descritto in precedenza nel suo funzionamento e configurazione. Per quanto riguarda l'aspetto software è riportato di seguito il blocco di codice commentato relativo a questa particolare subroutine del programma completo. In realtà questa porzione ospita anche i codici relativi alle interazioni di encoder e pulsanti, non riportate poiché precedentemente discusse.

*/*Previa inizializzazione del modulo ADC tramite l'apposita funzione* ADC_Init *si entra in un loop infinito dove il campionamento è ad opera della funzione* ADC_Get_Sample *di MikroC la quale necessita della specifica del canale, facilmente associabile ad un pin tramite il datasheet del*

PIC, e restituisce in uscita il valore campionato sottoforma di unsigned *ovvero un intero della dimensione di due byte. Il dato letto dal convertitore è memorizzato all'interno di un array* lettura_scalata *di 512 posizioni. La dimensione dell'array non è casuale, bensì scelta sulla base della successiva operazione di memorizzazione, infatti, quest'ultima scrive direttamente in settori della memoria la cui grandezza è proprio di 512 byte; il funzionamento dettagliato verrà descritto in seguito.*/*

```
unsigned lettura_scalata[512];
ADC_Init();
while(1){
       for (X0=1; X0<=512; X0++)
          {
          lettura_scalata[X0] = ADC_Get_Sample(0);
          }
```

*/*Una volta immagazzinati i valori devono essere graficati, ma prima rapportati alle dimensioni dello schermo: il secondo ciclo for si occupa proprio di questa operazione andando a moltiplicare opportunamente il valore di ogni campione per un coefficiente dato dal rapporto tra il numero effettivo di pixel dedicati alla parte di visualizzazione (280 pixel) e la dinamica coperta dall'ADC ovvero da 0 a 4095. In realtà il coefficiente in questione risulta essere negativo a causa della modalità di scrittura adottata dal display che utilizza come origine di coordinate lo spigolo in alto a sinistra con direzione positiva dell'asse verso il basso. Sempre a causa di questa configurazione al valore deve essere sommata l'altezza dello schermo fruibile, con una piccola tolleranza evitando di arrivare a saturare tutta la profondità del display. I valori modificati e resi interi dall'operatore* (int) *antistante la riduzione sono poi immagazzinati nello*

stesso vettore lettura_scalata *con notevole risparmio della memoria RAM del PIC.*/

```
       for (X0=1; X0<=478; X0++)
 {
     lettura_scalata[X0]=(int)(-
     0.068376*lettura_scalata[X0]+279.5);
 }
```

*/*Di seguito la visualizzazione del segnale: un punto se si tratta del primo campione e una linea per congiungere i successivi fornendo un aspetto grafico migliore rispetto ad una rappresentazione utilizzante i soli punti non interpolati tra loro.*/*

```
for (X0=1; X0<=478; X0++)
{
  if (X0 == 1)
  {
   display_drawPixel(X0,lettura_scalata[X0],255,255,255);
   delay_ms(1);
  }
  else
  {
writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],255,255,255);
      }
     }
```

/*Si effettua la stessa operazione con il colore dello sfondo (nero) in modo da cancellare la traccia dopo che essa è stata visualizzata.*/

```
     for (X0=1; X0<=478; X0++)
     {
      if (X0 == 1)
```

```
        {
         display_drawPixel(X0,lettura_scalata[X0],0,0,0);
         delay_ms(1);
        }
        else
        {
            writeLine(X0-1,lettura_scalata[X0-
            1],X0,lettura_scalata[X0],0,0,0);
        }
       }
}
```

4.7. Memorizzazione del segnale

La registrazione del segnale su supporto è parte integrante del progetto e proprio a causa della sua rilevanza è la porzione più complessa del programma. Nel seguente paragrafo sarà spiegato il funzionamento e le operazioni effettuate per memorizzare i dati, introducendo inizialmente le problematiche riscontrate, i test e le soluzioni che sono state adottate in fase di progettazione per far fronte, in particolare, a specifiche relative alla velocità di scrittura.

4.7.1. Prestazioni

La memorizzazione del segnale avviene direttamente durante il campionamento implicando la necessità di ottenere prestazioni in scrittura elevate, evitando così, la perdita di numerosi campioni durante l'intervallo di tempo sfruttato per la registrazione. Per garantire una velocità sufficiente questa porzione di programma è stata ridotta il più possibile dovendo anche rinunciare alla contemporanea visualizzazione del segnale, la quale avrebbe notevolmente rallentato tutto il processo causa l'esecuzione di molteplici istruzioni grafiche.

Auspicando ad individuare il metodo più rapido di scrittura durante il campionamento si è scelto un approccio sperimentale evitando di calcolare tempi di completamento di tutte le varie istruzioni presenti in un ciclo di scrittura in modo così dispendioso ed incerto. I test sono stati effettuati ponendo un'uscita logica alta su di un generico pin del microcontrollore all'inizio del ciclo di operazioni del quale si vuole misurare la durata, e

un'uscita logica bassa alla fine del medesimo ciclo. In questo modo, tramite oscilloscopio, è possibile individuare i fronti d'onda dell'uscita sul pin selezionato e quantificare la velocità, comprensiva di campionamento e scrittura, di varie soluzioni andando ad individuare gli stadi determinanti del processo e quali funzioni risultino migliori per l'applicazione.

Valutando la velocità di scrittura della funzione FAT32_Write si è riscontrato che il tempo impiegato per immagazzinare 512 byte in memoria durante il campionamento è pari a 7.6 s, ovvero una velocità di campionamento e scrittura pari a circa 67.4 byte/secondo.

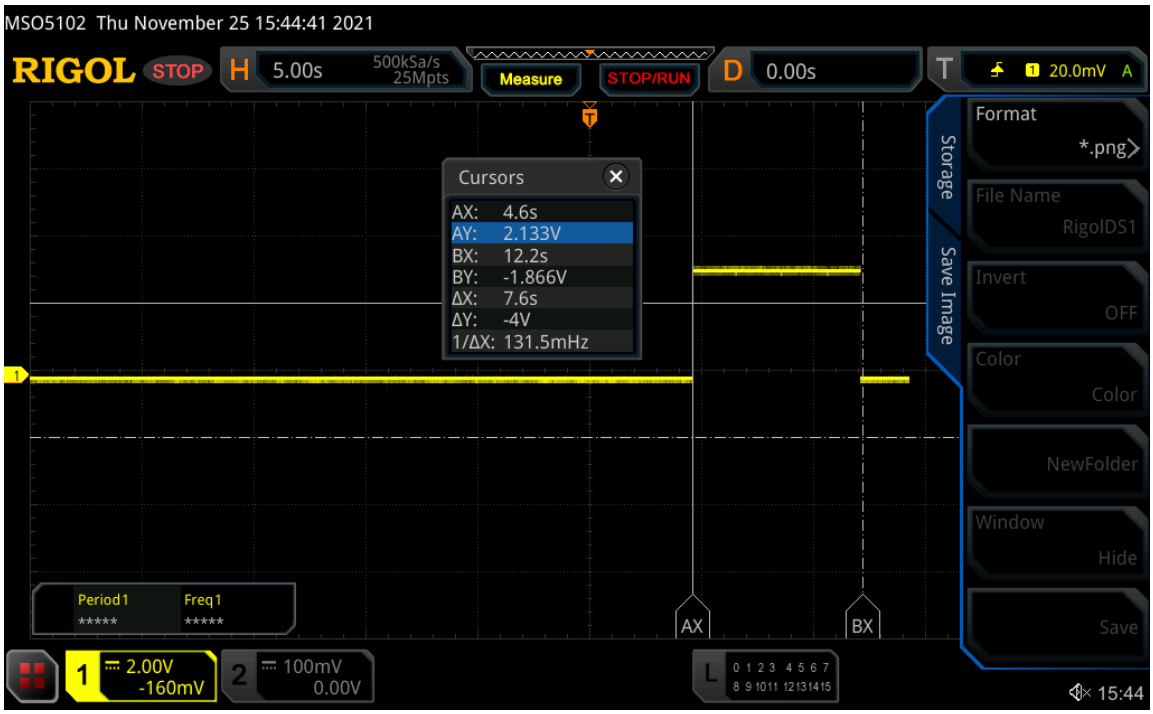

Figura 13: Test scrittura di 512 byte con funzione FAT32_Write

Dai test effettuati, nelle medesime condizioni, ma utilizzando la funzione FAT32_Dev_Write_Sector si nota come questa sia di gran lunga più performante, arrivando ad impiegare soli 14.08ms per la scrittura di 512 byte, come si può notare in *figura 14* dall'indicatore posto in basso a sinistra. Raggiungendo quindi la velocità di circa 36.3 Megabyte/secondo. Ogni campione occupa 2 byte di memoria, essendo quantizzato a 12 bit; quindi, la frequenza complessiva di campionamento e scrittura è di circa 18 KHz, nettamente superiore a quella richiesta dalle specifiche sulla banda passante (4KHz).

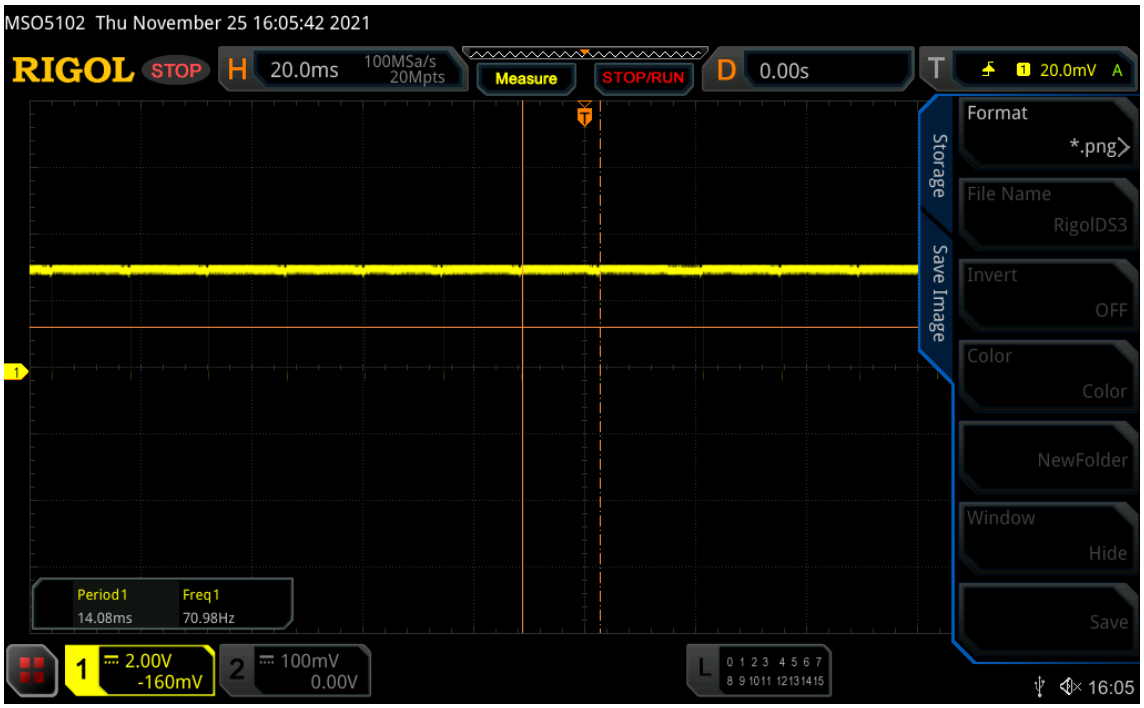

Figura 14: Test scrittura di 512 byte con funzione FAT32_Dev_Write_Sector

Dalle precedenti valutazioni è ovvio che per raggiungere prestazioni massime in scrittura è necessario evitare l'utilizzo della funzione *FAT32_Write* causa i controlli che essa effettua implicitamente prima di scrivere effettivamente i dati. A fronte di questo inconveniente la funzione scelta è stata *FAT32_Dev_Write_Sector*, sempre appartenente alla medesima libreria della precedente, che fornisce la possibilità di scrivere direttamente nei settori di memoria, senza effettuare controlli sulla condizione di tali porzioni né di mantenere in RAM l'indirizzo del puntatore all'ultimo dato scritto. Questa modalità di scrittura "grezza" favorisce, da un lato, lo scopo arrivando a velocità di molto superiori alla soluzione precedente, dall'altro si esime dall'effettuare controlli creando possibili sovrascritture di dati. Questo problema è stato risolto tramite l'utilizzo del file di testo SISTEM.TXT sempre presente in memoria che garantisce la progressività dei settori utilizzati in scrittura assicurando che essi non si sovrappongano. Questa soluzione presenta comunque inconvenienti relativi alla modificazione dei file da terzi: l'eliminazione o la creazione di documenti di testo da parte di un computer, per esempio, non è rilevabile dal dispositivo che potrebbe andare a sovrascrivere settori già occupati oppure evitare di scriverne alcuni che invece sono stati cancellati con conseguente sottrazione di spazio d'archiviazione.

4.7.2. Scrittura in memoria

L'effettiva scrittura in memoria è influenzata da due fattori fondamentali e tra loro correlati: la durata temporale e la grandezza in byte della registrazione. Seppur sussista una relazione matematica tra le due, si è preferito sovradimensionare lo spazio a disposizione per la memorizzazione utilizzando come controllo il solo scorrere temporale di un timer, evitando così, saturazioni di memoria impreviste causa l'instabilità, piccola ma presente ad ogni ciclo, della frequenza di campionamento. Come già accennato in precedenza le opzioni selezionabili per la durata sono quattro: 5 s, 10 s, 15 s e 30 s; la scelta di una di queste implica che, nella successiva fase di memorizzazione, sia riservata una porzione più o meno ampia in memoria. Il calcolo dei settori per l'allocazione è stato effettuato sperimentalmente valutando se la scrittura nel lasso di tempo scelto aveva saturato o meno i settori a lei riservati. Effettuando una serie di prove ripetute si ottiene un valore di convergenza approssimabile al numero di byte che ogni registrazione occupa a seconda della sua durata, a questo valore va sommato un contributo per garantire la prima citata sovrastima della mole di dati in ingresso. Nel seguente codice vengono riportate le istruzioni dedite a svolgere il processo di creazione del file in base alle precedenti considerazioni ed anche i comandi necessari a adottare una denominazione progressiva del file.

```
/*Dichiarazione delle variabili: sc1 è un particolare valore
denominato __SECTOR che la funzione di scrittura utilizzata
necessita in ingresso per puntare al primo settore del file
creato; *res è un puntatore a carattere che servirà per
immagazzinare una stringa risultato della concatenazione di
due; str[] è una stringa fissa che contiene l'estensione del
file di testo.*/

__SECTOR sc1;
char indice[10], *res, str[]=".txt";
int file;

/*Per effettuare una numerazione successiva all'avvio viene
letto il file SYSTEM.TXT all'interno del quale è salvato il
numero dell'ultimo record prodotto. La lettura bufferizza
```

questo valore all'interno della stringa indice *e, previa traduzione, sottoforma di intero nella variabile* file. *Successivamente la stringa* indice *è concatenata con ".txt" tramite la funzione* strcat *in modo da poter dare il nome, memorizzato in* res, *al file che verrà creato.*/

```
file=atoi(indice);
indice[7]=0;
res=strcat(indice,str);
```

/*In questa porzione di codice vi sono in realtà una serie di if che caratterizzano il file a seconda della temporizzazione scelta. Per semplicità è riportato un solo caso, del tutto equivalente alle restanti tre opzioni selezionabili. Per creare una partizione di memoria velocemente e di dimensioni note, viene utilizzata la funzione FAT32_MakeSwap sempre dotazione della libreria __Lib_FAT32, questa richiede in ingresso rispettivamente: il nome comprendente l'estensione del file, il numero di settori da riservare e il puntatore al settore iniziale.*/

```
    if(secondi==30){
      FAT32_MakeSwap(res, 100, &sc1);
      }
```

Una volta creato il file con il nome adeguato si deve effettuare la scrittura dei dati campionati all'interno della memoria MicroSD. Questa operazione deve procedere indisturbata per il tempo previsto in fase di selezione, ciò impone l'implementazione di un timer in grado di contare i secondi trascorsi. Per far ciò è stato sfruttato il TIMER0 interno al microcontrollore il cui funzionamento ed i relativi calcoli necessari per la calibrazione sono già stati discussi in precedenza; di seguito sono invece riportate le porzioni di codice relative all'implementazione.

/*Si controlla il flag TMR0IF valutando se esso è variato, nel caso affermativo significa che è intercorso 1 secondo dall'accensione del timer, allora si scala il contatore

relativo ai secondi e si imposta nuovamente il timer nella condizione iniziale, ovvero: flag con valore zero e il preload dei registri pari a 3036 suddiviso sui due registri TMR0H e TMR0L. Ponendo così il timer nella condizione di iniziare immediatamente il nuovo conteggio./*

```
while(secondi>0){
     if(PIR3.TMR0IF == 1){
               secondi--;
               TMR0H=0x0B;
               TMR0L=0xDC;
               PIR3.TMR0IF=0;
          }
```

Sempre all'interno del ciclo while soprastante, l'operazione di scrittura è affidata alla funzione FAT32_Dev_Write_Sector. Questa procedura opera scrivendo in memoria interi settori, sia che il dato passato ecceda o sia in difetto rispetto alla dimensione fissa, 512 byte, di essi. Si effettua il campionamento sulla base di questa specifica allocando i dati all'interno di un array di opportune dimensioni come per quanto accadeva in fase di visualizzazione.

*/*Il dato campionato da parte del modulo ADC è un intero di particolare formato in quanto a 12 bit, per poterlo scrivere in memoria esso deve essere trasformato in un carattere tramite la funzione di MikroC:* IntToStr. *In particolare, avendo 512 valori interi occupanti due byte che una volta trasformati saranno caratteri da un solo byte si è adottata la definizione di una matrice* txt *per bufferizzare i dati convertiti. Una volta riempito l'array con i dati e, conseguentemente tradotti, essi vengono passati in blocco alla funzione di scrittura, la quale, registra a partire dal settore* sc1 *in poi, incrementando ad ogni ciclo il suo contatore senza limitazioni in quanto, allo scadere del tempo, l'uscita è forzata dalla condizione di while sulla variabile* secondi.*/*

```
char txt[512][7];
     for(X0=0; X0<512; X0++){
                    lettura_scalata[X0]                    =
ADC_Get_Sample(0);
                    IntToStr(lettura_scalata[X0],txt[X0]);
     }
FAT32_Dev_Write_Sector(sc1+k,txt);
k++;
}
```

Durante il processo di registrazione sono presenti a video anche degli indicatori grafici: una stringa riportante la dicitura “REC”, il conteggio decrescente dei secondi e un led rosso fisso, atti ad allarmare l’utente verso spegnimenti errati durante la fase di scrittura in memoria, la quale risulta particolarmente soggetta a danneggiamenti dovuti ad azioni del genere.

In seguito all’avvenuta registrazione, oltre ai feedback grafici, deve essere implementato l’aggiornamento del file SISTEM.TXT in modo da poter effettuare nuove registrazioni senza il pericolo di sovrascrivere passati record.

*/*Si incrementa il numero* file *letto in fase di caricamento del programma, garantendo una numerazione progressiva delle registrazioni. In seguito, esso è tradotto in una stringa tramite la funzione* ShortToStr, *il risultato è memorizzato sempre nella variabile* indice *giovando all’economia del programma e instaurando un ciclo chiuso con le precedenti istruzioni che nominavano i file proprio sulla base del valore di* indice *concatenato con la stringa ".txt". Successivamente si apre il file* SYSTEM.TXT *in scrittura tramite la funzione* FAT32_Open *e vi si scrive il nuovo valore di* indice *attraverso la procedura* FAT32_Write, *specificando il file nel quale si intende scrivere, il dato e la dimensione di esso. Aprendo il file in scrittura il puntatore si posizionerà nella prima locazione sovrascrivendo il*

vecchio indice con il nuovo. Alla fine del processo il file viene chiuso da FAT32_Close.

```
file++;
ShortToStr(file,indice);
fhandle = FAT32_Open("SYSTEM.TXT", FILE_WRITE);
FAT32_Write(fhandle,indice,sizeof(indice));
err=FAT32_Close(fhandle);
```

Al termine di questa serie di operazioni la scrittura è completata e il programma torna in automatico alla schermata di visualizzazione primaria dove graficherà nuovamente il segnale e attenderà un'ipotetica, ulteriore, interazione utente.

4.8. Analisi del segnale

Durante la visualizzazione del segnale è possibile utilizzare un pulsante per effettuare un fermo immagine. Questa azione oltre ad interrompere il campionamento genera la comparsa in cascata di due assi ortogonali, uno verticale ed uno orizzontale, indicanti rispettivamente il tempo e l'ampiezza del segnale. La calibrazione dell'asse corrispondente alle ampiezze è stata effettuata in modo piuttosto semplice considerando i seguenti calcoli basati sui riferimenti relativi al modulo ADC del PIC:

$$Pixel\ al\ livello = \frac{280\ Pixel}{4095\ Livelli} \cong 0.068376\ Pixel/Livello$$

$$mV\ al\ livello = \frac{3.3\ V}{4095\ Livelli} \cong 0.805860\ mV/Livello$$

$$mV\ al\ Pixel = \frac{0.805860\frac{mV}{Livello}}{0.068376\frac{Pixel}{Livello}} \cong 11.785726\frac{mV}{Pixel} \cong 11.8\frac{mV}{Pixel}$$

Conoscendo il valore in mV per ogni pixel relativo all'area di visualizzazione si può costruire sull'asse verticale una scala graduata riportante le ampiezze, scelta con sensibilità di 0.5 V.

Per quanto riguarda l'asse dei tempi è stato calibrato al completamento del progetto tramite approccio sperimentale: analizzando segnali periodici forniti da un generatore di forme d'onda e, tramite le barrette adibite alla selezione, andando ad individuare i pixel intercorsi tra due periodi conoscendo così il rapporto tempo/Pixel necessario sia per l'implementazione dell'asse stesso che per il calcolo della distanza temporale tra i due selettori. Essendo i test basati su misure effettuate tramite il posizionamento dei selettori esse peccano di incertezza, si è quindi preferito quantificare il coefficiente tramite una serie di misure ripetute su segnali sinusoidali a frequenze diverse stimando una media di 0.083ms/pixel. Questa procedura di taratura ha prodotto un algoritmo di analisi con una buona precisione, come si può notare dalla *figura 15* dove i selettori sono stati precedentemente impostati andando ad individuare un periodo di una sinusoide a 200Hz, il quale corrisponde a 5ms contro i vicini 4.999ms individuati dal dispositivo.

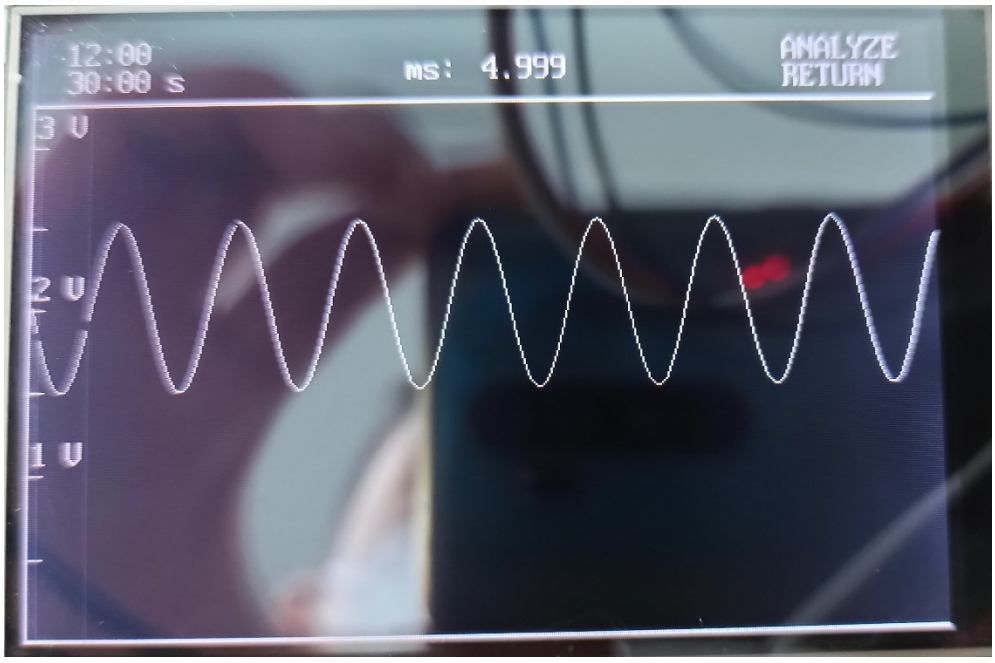

Figura 15: Schermata di analisi con segnale in ingresso sinusoidale 1V a 200Hz

Successivamente alla generazione degli assi vi è, come nella maggior parte delle sezioni di codice, una ridefinizione delle azioni associate agli interruttori: il pulsante A assume la funzione di "Analyze" ovvero se premuto genera il primo dei due selettori. Esso si presenta come una linea verticale la cui posizione può essere facilmente variata, verso destra o verso sinistra, ruotando l'encoder. All'interazione, il medesimo tasto, assume la dicitura "Confirm" associata alla conferma della posizione del primo selettore qualora

esso sia valutato in posizione corretta a discrezione dell'utente. La pressione, oltre ad immagazzinare la posizione selezionata, genera il secondo selettore in modo del tutto equivalente, mentre un ulteriore conferma restituirà a video il tempo intercorso tra le due posizioni, intanto che i selettori verranno cancellati lasciando spazio ad ulteriori analisi qualora esse fossero necessarie.

Per questo paragrafo non è stata riportata nessuna porzione di codice in quanto costruito sulla base del funzionamento di encoder e pulsanti, le cui descrizioni sono già state affrontate in precedenza. Inoltre, il codice sarebbe risultato poco rilevante dal punto di vista informatico in quanto consistente di istruzioni grafiche atte alla creazione dei selettori ed al loro movimento generato tramite cancellazione e riscrittura; nonché istruzioni di simile natura, per evitare che la traccia del segnale venga cancellata durante queste operazioni di spostamento dei cursori.

5. Conclusioni e sviluppi futuri

In conclusione: il progetto sviluppato si dimostra completo di tutte le qualità necessarie per un dispositivo atto al monitoraggio con delle caratteristiche tipiche di strumenti diagnostici. Inoltre, si è posta particolare attenzione nel rendere di facile comprensione e interfacciamento il dispositivo approcciando la progettazione con un'ottica volta all'utilizzo "sul campo" prescindendo, in parte, dagli scopi puramente accademici. Naturalmente il prodotto è ancora ben distante da un'ipotetica commercializzazione sia per fattezze che per prestazioni, ma ne getta le basi fornendo numerosi spunti volti al miglioramento e all'ampliamento delle funzionalità. A tal proposito si è ragionato su possibili sviluppi considerando classi di microcontrollori superiori come i PIC24 oppure la famiglia dsPIC disponendo, in questa evenienza, di migliori caratteristiche tecniche, in particolare: velocità, dimensione della memoria RAM e profondità di campionamento. Tramite l'utilizzo di un microcontrollore più evoluto, come i sopracitati, si potrebbe implementare uno strumento dalle prestazioni maggiorate, consentendo di aggiungere funzionalità coadiuvanti la diagnostica come: l'implementazione della FFT (Fast Fourier Transform), la visualizzazione del suo tracciato e memorizzazione contemporanea al segnale temporale. Simili potenzialità, inserite in un contesto ospedaliero, non forniscono in realtà una svolta nell'ambito della fonocardiografia interpretativa se non

calibrate a dovere tramite una ponderosa mole di dati patologici e non dai quali il programma può "apprendere"; perciò, un ulteriore passo potrebbe essere quello di costruire un database tramite registrazioni effettuate dallo strumento e tracciati contemporanei provenienti da ormai appurate apparecchiature diagnostiche, come un ECG, per valutare un complessivo andamento dei parametri vitali in funzione dei suoni auscultabili. Nel caso si avesse a disposizione una cospicua mole di dati i segnali potrebbero essere processati da un algoritmo di *machine learning* andando ad individuare le patologie in relazione alla forma caratteristica dei segnali sia in tempo che in frequenza.

Approcciando la questione da un punto di vista locale si hanno comunque buoni margini di miglioramento sia considerando il lato hardware che quello software. Per quanto riguarda il primo aspetto sicuramente l'utilizzo di un display di più grandi dimensioni e risoluzione assisterebbe una visualizzazione più semplice e precisa. Inoltre, l'aggiunta di periferiche come un orologio ed un convertitore analogico/digitale esterno dotato di RAM propria aiuterebbero rispettivamente: la memorizzazione dei file inserendo la data di creazione oltre che graficare l'orario su display e, dall'altro lato, la capacità di campionare e bufferizzare nell'immediato evitando la perdita di valori e auspicabilmente migliorandone la qualità disponendo di maggiore profondità di campionamento. Relativamente al lato software, invece, si necessita sicuramente di un miglioramento del codice dal punto di vista delle prestazioni e delle interazioni. In particolare, l'operazione di *tuning* dovrebbe migliorare la velocità del programma semplificando le istruzioni più dispendiose ristrutturandole dal punto di vista concettuale inserendo delle routine di interrupt, le quali gioverebbero alla fluidità del programma, nello specifico durante le interazioni, ma anche nella fase di campionamento evitando la procedura di *polling*, ovvero il continuo controllo, del timer adibito a contatore.

Per quanto riguarda sviluppi futuri già facilmente implementabili senza l'obbligo di modificare il microcontrollore o stravolgere concettualmente il programma, si potrebbe codificare una funzionalità di individuazione del battito cardiaco tramite l'analisi del già campionabile ed immagazzinabile fonocardiogramma, potendo graficare il valore su schermo durante la fase di visualizzazione dinamica fornendo un'informazione

aggiuntiva. Inoltre, disponendo della funzionalità di memorizzazione su supporto multimediale, i file sono facilmente trasferibili su computer dove si potrebbe implementare un programma, per esempio in ambiente Matlab, attraverso il quale analizzare i dati in modo più flessibile.

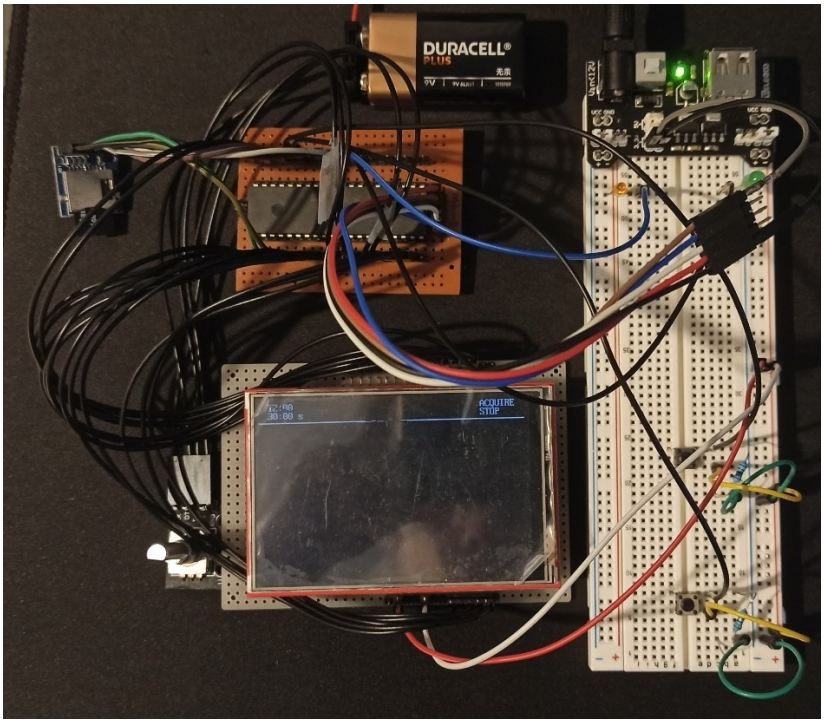

Figura 16: Aspetto del dispositivo completo durante la fase di progettazione

6. Appendice

```
#include "Display_LIB.c"
#include <stdio.h>
#include "__Lib_FAT32.h"
#define ENCODER_SW      RB0_bit
#define ENCODER_DT      RB1_bit
#define ENCODER_CLK     RB2_bit
typedef unsigned short          uint8;
typedef signed short            int8;
sfr sbit Mmc_Chip_Select at LATC2_bit;
sfr sbit Mmc_Chip_Select_Direction at TRISC2_bit;
void initSPI(void)
{

SPI1_Remappable_Init_Advanced(_SPI_REMAPPABLE_MASTER_OSC_DIV6
4,                          _SPI_REMAPPABLE_DATA_SAMPLE_MIDDLE,
_SPI_REMAPPABLE_CLK_IDLE_LOW, _SPI_REMAPPABLE_LOW_2_HIGH);
}
void initFastSPI(void)
{

SPI1_Remappable_Init_Advanced(_SPI_REMAPPABLE_MASTER_OSC_DIV4
,                           _SPI_REMAPPABLE_DATA_SAMPLE_MIDDLE,
_SPI_REMAPPABLE_CLK_IDLE_LOW, _SPI_REMAPPABLE_LOW_2_HIGH);
}
int8 err;
void main() {
     unsigned long i,X0,k;
     short fhandle;
     signed short rotposition=0,secondi=0,select;
     int  value, rot, file=0, an1, an2, diff;
     char indice[10], txtc, *res, txtdiff[7];
     bit state, acquire, acquire1, state1, stop, state2, sel,
analyze1, analyze2 ;
     __SECTOR sc1;
     unsigned lettura_scalata[512];
     char txt[512][7];
     char str[]=".txt";
     TRISE = 0b00000000;
     TRISC = 0b00000011;
     PORTC = 0b00000000;
     TRISB = 0b00000111;
     PORTB = 0b00000000;
     TRISD = 0b00000000;
     PORTD = 0b00000000;
     ANSELD= 0b00000000;
     ANSELB= 0b00000000;
     ANSELC= 0b00000000;
     ANSELE= 0b00000000;
     T0CON0= 0b00010000;
     T0CON1= 0b01000110;
     TMR0H=0x0B;
```

```
TMR0L=0xDC;
IPR3.TMR0IP=0;
PIE3.TMR0IE=1;
INTCON0.GIEL=1;
T0CON0= 0b10010000;
OSCFRQ=0b00000101;
INTCON.GIEL=1;
display_Starts();
display_setTextColor(255, 255, 255, 0, 0, 0);
display_setTextSize(1);
display_setTextFont(8,16,2);
display_setRotation(1);
Schermata_Caricamento();
delay_ms(5000);
Unlock_IOLOCK();
PPS_Mapping_NoLock(_RC4,_INPUT,_SDI1);
PPS_Mapping_NoLock(_RC3,_OUTPUT,_SCK1);
PPS_Mapping_NoLock(_RC5,_OUTPUT,_SDO1);
Lock_IOLOCK();
display_fillRoundRect(181,20,38,20,4,102,255,30);
display_drawRect(181,20,38,20,40,115,51);
display_drawRect(182,21,36,18,40,115,51);
initSPI();
display_fillRoundRect(219,20,38,20,4,102,255,30);
display_drawRect(219,20,38,20,40,115,51);
display_drawRect(220,21,36,18,40,115,51);
Delay_ms(1);
err = FAT32_Init();
Delay_ms(1);
if(err<0)
{
        while(err<0)
        {
              err=FAT32_Init();
              Delay_ms(1000);
              display_setTextColor(255, 0, 0, 0, 0, 0);
              display_setCursor(90,60);
              display_puts("ERROR SD NOT FOUND!");
              PORTE.RE2=1;
        }
        fillRect(89,59,300,25,0,0,0);
        display_setTextColor(102, 255, 30, 0, 0, 0);
        display_setCursor(90,60);
        display_puts("OK SD FOUND!");
        PORTE.RE2=0;
        PORTE.RE1=1;
        Delay_ms(300);
        PORTE.RE1=0;
        fillRect(89,59,300,25,0,0,0);
}
display_setTextColor(255, 255, 255, 0, 0, 0);
display_fillRoundRect(257,20,38,20,4,102,255,30);
display_drawRect(257,20,38,20,40,115,51);
```

```
display_drawRect(258,21,36,18,40,115,51);
initFastSPI();
display_fillRoundRect(295,20,38,20,4,102,255,30);
display_drawRect(295,20,38,20,40,115,51);
display_drawRect(296,21,36,18,40,115,51);
err=FAT32_Exists("SYSTEM.TXT");
if(err<0)
{
         while(err<0)
         {
                err=FAT32_Exists("SYSTEM.TXT");
                Delay_ms(1000);
                display_setTextColor(255, 0, 0, 0, 0, 0);
                display_setCursor(90,60);
                display_puts("ERROR");
                PORTE.RE2=1;
         }
         fillRect(89,59,300,25,0,0,0);
         display_setTextColor(102, 255, 30, 0, 0, 0);
         display_setCursor(90,60);
         display_puts("OK ERROR SOLVED!");
         PORTE.RE2=0;
         PORTE.RE1=1;
         Delay_ms(300);
         PORTE.RE1=0;
         fillRect(89,59,300,25,0,0,0);
}
else
{
     if(err==1)
     {
      fhandle = FAT32_Open("SYSTEM.TXT", FILE_READ);
      Delay_ms(20);
      FAT32_Read(fhandle,indice,sizeof(indice));
     }
     if(err==0)
     {
      display_setCursor(90,60);
      display_puts("CREATING FILE SYSTEM");
      fhandle = FAT32_Open("SYSTEM.TXT", FILE_APPEND);
      FAT32_Write(fhandle,"001",3); //scrivo
     }
}
err=FAT32_Close(fhandle);
file=atoi(indice);
display_fillRoundRect(333,20,38,20,4,102,255,30);
display_drawRect(333,20,38,20,40,115,51);
display_drawRect(334,21,36,18,40,115,51);
fillRect(80,15,300,30,0,0,0);
display_setTextColor(255, 255, 255, 0, 0, 0);
display_setTextSize(1.8);
display_setCursor(90,20);
display_puts("READY");
```

```
      PORTE.RE1=1;
      Delay_ms(300);
      PORTE.RE1=0;
      fillScreen(0,0,0);
      Schermata_vis_din();
      ADC_Init();
      rot=ENCODER_CLK;
      state=0;
      acquire=0;
      acquire1=0;
      state1=0;
      stop=0;
      analyze1=0;
      analyze2=0;
      ShortToStr(file,indice);
      indice[7]=0;
      res=strcat(indice,str);
      while(1)
      {
        for (X0=1; X0<=512; X0++)
           {
           lettura_scalata[X0] = ADC_Get_Sample(0);
           }
        for (X0=1; X0<=478; X0++)
       {
       lettura_scalata[X0]=(int)(-
0.068376*lettura_scalata[X0]+279.5);
       }
      for (X0=1; X0<=478; X0++)
      {
       if (X0 == 1)
       {
        display_drawPixel(X0,lettura_scalata[X0],255,255,255);
        delay_ms(1);
       }
       else
       {
        writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],255,255,255);
       }
      }
      for (X0=1; X0<=478; X0++)
      {
       if (X0 == 1)
       {
        display_drawPixel(X0,lettura_scalata[X0],0,0,0);
        delay_ms(1);
       }
       else
       {
        writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],0,0,0);
       }
```

```
      }
      if(Button(&PORTC,1,1,1)){
               state=1;
            }
       if(state && Button(&PORTC,1,1,0)){
                state=0;
                acquire=1;
       }
       if(Button(&PORTC,0,1,1)){
               state1=1;
            }
       if(state1 && Button(&PORTC,0,1,0)){
                state1=0;
                stop=1;
       }
       if(stop==1){
        PORTE.RE2=1;
        display_setCursor(400,5);
        display_puts("ANALYZE ");
        display_setCursor(400,20);
        display_puts("RETURN");
        display_setCursor(240,15);
        display_puts("STOP");
        for (X0=1; X0<=478; X0++)
        {
         if (X0 == 1)
            {

display_drawPixel(X0,lettura_scalata[X0],255,255,255);
             delay_ms(1);
             }
             else
             {
             writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],255,255,255);
             }
        }
        writeLine(1,40,1,320,255,255,255);
        writeLine(2,40,2,320,255,255,255);
        writeLine(2,319,479,319,255,255,255);
        writeLine(2,318,479,318,255,255,255);
        writeLine(2,277,10,277,255,255,255);
        writeLine(2,234,10,234,255,255,255);
        writeLine(2,191,10,191,255,255,255);
        writeLine(2,148,10,148,255,255,255);
        writeLine(2,105,10,105,255,255,255);
        writeLine(2,62,10,62,255,255,255);
        display_setCursor(5,44);
        display_puts("3 V");
        display_setCursor(5,130);
        display_puts("2 V");
        display_setCursor(5,216);
        display_puts("1 V");
```

```
      }
      while(stop==1){
       if(Button(&PORTC,0,1,1)){
              state1=1;
           }
      if(state1 && Button(&PORTC,0,1,0)){
               state1=0;
               stop=0;
      }
      if(Button(&PORTC,1,1,1)){
              state=1;
           }
      if(state && Button(&PORTC,1,1,0)){
               state=0;
               analyze1=1;
               rotposition=15;
               display_setCursor(200,15);
               display_puts("SET CURSOR 1");
               display_setCursor(400,5);
               display_puts("CONFIRM");
               display_setCursor(400,20);
               display_puts("       ");
      }
      while(analyze1==1){
             value=ENCODER_CLK;
             if(value!= rot){
                        if(ENCODER_DT!=value){
                                              rotposition++;
                                              }
                        else{
                             rotposition--;
                             }

writeLine((rotposition+1)*3,41,(rotposition+1)*3,317,0,0,0);
                       writeLine((rotposition-
1)*3,41,(rotposition-1)*3,317,0,0,0);

writeLine((rotposition)*3,41,(rotposition)*3,317,255,255,255)
;
                       for (X0=1; X0<=478; X0++)
                       {
                        if (X0 == 1)
                        {

display_drawPixel(X0,lettura_scalata[X0],255,255,255);
                         }
                          else
                          {
                           writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],255,255,255);
                          }
                         }
            }
```

```
            if(Button(&PORTC,1,1,1)){
            state=1;
             }
             if(state && Button(&PORTC,1,1,0)){
              state=0;
              an1=(rotposition)*3;
              analyze1=0;
              display_setCursor(200,15);
              display_puts("SET CURSOR 2");
              rotposition++;
              analyze2=1;
              }
            rot=value;
      }
      while(analyze2==1){
             value=ENCODER_CLK;
             if(value!= rot){
                        if(ENCODER_DT!=value){
                                              rotposition++;
                                              }
                        else{
                             rotposition--;
                             }

writeLine((rotposition+1)*3,41,(rotposition+1)*3,317,0,0,0);
                        writeLine((rotposition-
1)*3,41,(rotposition-1)*3,317,0,0,0);

writeLine((rotposition)*3,41,(rotposition)*3,317,255,255,255)
;

writeLine(an1,41,an1,317,255,255,255);
                        for (X0=1; X0<=478; X0++)
                        {
                         if (X0 == 1)
                         {

display_drawPixel(X0,lettura_scalata[X0],255,255,255);
                         }
                          else
                          {
                           writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],255,255,255);
                          }
                        }
            }
            if(Button(&PORTC,1,1,1)){
            state=1;
             }
             if(state && Button(&PORTC,1,1,0)){
              state=0;
              an2=(rotposition)*3;
              analyze2=0;
```

```
                display_setCursor(400,5);
                display_puts("ANALYZE");
                display_setCursor(200,15);
                display_puts("                  ");
                display_setCursor(200,15);
                display_puts("Diff: ");
                diff=an1-an2;
                diff=abs(diff);
                IntToStr(diff,txtdiff);
                display_setCursor(240,15);
                display_puts(txtdiff);
                }
              rot=value;
       }
       display_setCursor(400,20);
       display_puts("RETURN");
       writeLine(an1,41,an1,317,0,0,0);
       writeLine(an2,41,an2,317,0,0,0);
       }
       display_setCursor(400,5);
       display_puts("ACQUIRE");
       display_setCursor(400,20);
       display_puts("STOP    ");
       for (X0=1; X0<=478; X0++)
        {
         if (X0 == 1)
            {
             display_drawPixel(X0,lettura_scalata[X0],0,0,0);
             delay_ms(1);
             }
             else
             {
             writeLine(X0-1,lettura_scalata[X0-
1],X0,lettura_scalata[X0],0,0,0);
             }
        }
        writeLine(1,20,1,320,0,0,0);
        writeLine(2,20,2,320,0,0,0);
        writeLine(2,319,479,319,0,0,0);
        writeLine(2,318,479,318,0,0,0);
        fillRect(2,41,35,320,0,0,0);
       PORTE.RE2=0;
       display_setCursor(200,15);
       display_puts("               ");
       while(acquire==1){
          display_setCursor(220,15);
          display_puts("SELECT ACQ. TIME");
       while(acquire1==0){
              value=ENCODER_CLK;
              if(value!= rot){
                        if(ENCODER_DT!=value){
                                              select++;
                                              }
```

```
                    else{
                         select--;
                         }
        if((select >3) || (select<-3))
        {
         select=0;
        }
        if(select==0)
        {
         display_setCursor(20,20);
         display_puts("30:00 s");
         secondi=30;
        }
        if((select==1) || (select==-1))
        {
         display_setCursor(20,20);
         display_puts("15:00 s");
         secondi=15;
        }
        if((select==2) || (select==-2))
        {
         display_setCursor(20,20);
         display_puts("10:00 s");
         secondi=10;
        }
        if((select==3) || (select==-3))
        {
         display_setCursor(20,20);
         display_puts("5:00  s");
         secondi=5;
        }
        }
if(Button(&PORTC,1,1,1)){
        state=1;
     }
  if(state && Button(&PORTC,1,1,0)){
         state=0;
         acquire1=1;
  }
       rot=value;
}
display_setCursor(220,15);
display_puts("                ");
PORTE.RE2=1;
display_setCursor(200,15);
display_puts("REC");
Delay_ms(30);
if(secondi==30){
FAT32_MakeSwap(res,100, &sc1);
}
if(secondi==15){
FAT32_MakeSwap(res,50, &sc1);
}
```

```
       if(secondi==10){
       FAT32_MakeSwap(res,35, &sc1);
       }
       if(secondi==5){
       FAT32_MakeSwap(res,16, &sc1);
       }
       i=0;
       X0=1;
       k=0;
       Delay_ms(30);
       while(secondi>0)
       {
         if(PIR3.TMR0IF == 1){
             secondi--;
             TMR0H=0xFF;
             TMR0L=0xFF;
             PIR3.TMR0IF=0;
         }
              for(X0=0; X0<512; X0++){
                     lettura_scalata[X0] = ADC_Get_Sample(0);
      IntToStr(lettura_scalata[X0],txt[X0]);
      }
FAT32_Dev_Write_Sector(sc1+k,txt);
k++;
}
     }
         Delay_ms(500);
         PORTE.RE2=0;
         display_setCursor(200,15);
         display_puts("    ");
         display_setCursor(240,15);
         display_puts("    ");
         file++;
         ShortToStr(file,indice);
         fhandle = FAT32_Open("SYSTEM.TXT", FILE_WRITE);
         FAT32_Write(fhandle,indice,sizeof(indice));
         err=FAT32_Close(fhandle);
         Delay_ms(30);
         indice[7]=0;
         res=strcat(indice,str);
         Delay_ms(1000);
         acquire=0;
         acquire1=0;
       }
    }
}
```

7. Bibliografia e sitografia

1. A. Bellini e A. Guidi, *“Linguaggio C. Una guida alla programmazione con elementi di Objective-C”,* Milano, McGraw-Hill Education, 2018.

2. Dhaker, Piyu. "Introduction to SPI interface." *Analog Dialogue* 52.3 (2018): 49-53.

3. M. Brusco e H. Nazeran, "Digital phonocardiography: a PDA-based approach", *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* , 2004, pp. 2299-2302, doi: 10.1109/IEMBS.2004.1403668.

4. Satheesh, M. B., et al. "Microcontroller and SD Card Based Standalone Data Logging System using SPI and I2C Protocols for Industrial Application." *Int. J. Adv. Res. Electr. Electron. Instrum. Eng* 5.4 (2016): 2208-2214.

5. Ibrahim, Dogan. *PIC Microcontroller Projects in C: Basic to Advanced*. Newnes, 2014.

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

7. Specifiche FAT32. https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system