
CLASSE L8: INGEGNERIA
DELL'INFORMAZIONE
CORSO DI LAUREA IN INGEGNERIA INFORMATICA

MONOGRAFIA DI TIROCINIO

Studio di librerie grafiche e progetto di un oscilloscopio su
microcontrollore



Tutori Accademici:

Prof. Gianmario Pellegrino
Prof. Gianpiero Cabodi

Tutori Aziendali:

Ing. Salvatore Rinaudo
Ing. Michelangelo Grosso
Ing. Davide Lena

Candidato:

Giuseppe Carella



Sommario

1 Introduzione	3
2 Hardware e software utilizzati.....	5
2.1 I microcontrollori	5
2.2 La scheda STM32F429I-DISCO	5
2.3 Il microcontrollore STM32F429.....	6
2.4 Librerie del microcontrollore: <i>HAL (Hardware Abstraction Layer)</i>	7
2.4.1 Configurazione del Timer	8
2.4.2 Configurazione dell'ADC, del canale dell'ADC e del GPIOx.	9
2.4.3 Configurazione del DMA	10
2.5 Librerie del microcontrollore: <i>emWin</i>	11
2.5.1 I Widget	12
2.5.2 Graph Widget	13
2.6 Rotazione dello schermo	25
3 Progettazione di un oscilloscopio digitale	27
3.1 Che cos'è l'oscilloscopio	27
3.2 Architettura e soluzioni progettuali	29
3.2.1 Riscaldamento orizzontale	30
3.2.2 Riscaldamento verticale	31
4 Realizzazione del progetto	33
4.1 Configurazione della libreria grafica.....	33
4.2 Realizzazione del sistema di acquisizione	34
4.3 Configurazione dell'architettura del sistema di acquisizione.....	35
4.3.1 Configurazione del timer	35
4.3.2 Configurazione dell'ADC	35
4.3.3 Configurazione del DMA	37
4.4 Risultati	38
5 Conclusioni	45
Bibliografia.....	46

1 Introduzione

ST-Polito è una joint venture tra STMicroelectronics, al 75%, e il Politecnico di Torino, al 25%. Quest'azienda mira a far convergere la tecnologia dell'azienda con la ricerca dell'università per realizzare nuove applicazioni e prodotti pre-commerciali a basso costo. Le attività che si svolgono all'interno di ST-Polito comprendono lo sviluppo di servizi relativi alla progettazione, prototipazione e produzione di circuiti elettronici eterogenei, componenti e sistemi. Le aree di indagine dell'azienda sono l'efficienza energetica, la mobilità e la salute.

Il tirocinio riguarda la realizzazione di un semplice oscilloscopio che permetta di visualizzare la forma d'onda di una tensione analogica applicata in ingresso. Il sistema sviluppato si basa su un microcontrollore dotato delle periferiche necessarie all'acquisizione e alla conversione digitale di un segnale analogico su uno schermo *LCD* (*Liquid Crystal Display*) pilotato dal microcontrollore stesso (**Figura 1**).

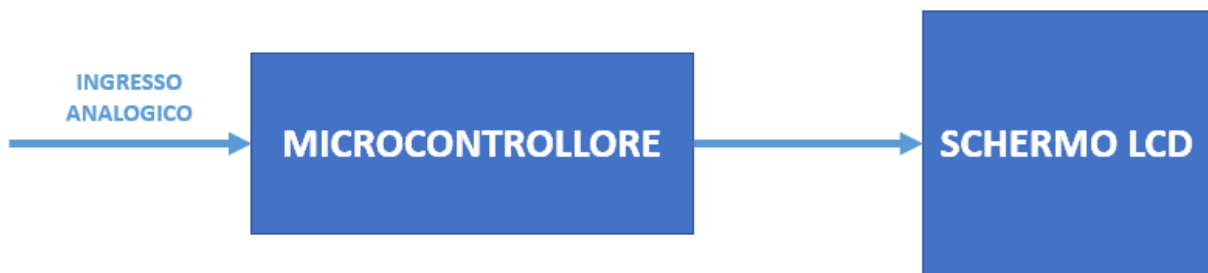


Figura 1: Schema a blocchi del sistema

In particolare si utilizza un *ADC* (*Analog to Digital Converter*) temporizzato mediante un timer e un canale *DMA* (*Direct Memory Access*) per il trasferimento dei dati in memoria. Per la visualizzazione si utilizza una libreria grafica commerciale (*emWin*).

Il lavoro si può suddividere in quattro fasi principali. La prima fase è consistita nello studio del funzionamento della libreria grafica *emWin*, anche mediante l'analisi del codice realizzato da altri sviluppatori. Dalla comprensione del codice è stato possibile capire come si possono inserire gli assi delle ascisse e delle ordinate, le scale numeriche, le griglie, e impostare colori e dimensioni di ciascun oggetto facente parte della visualizzazione di un grafico (**Figura 2**).

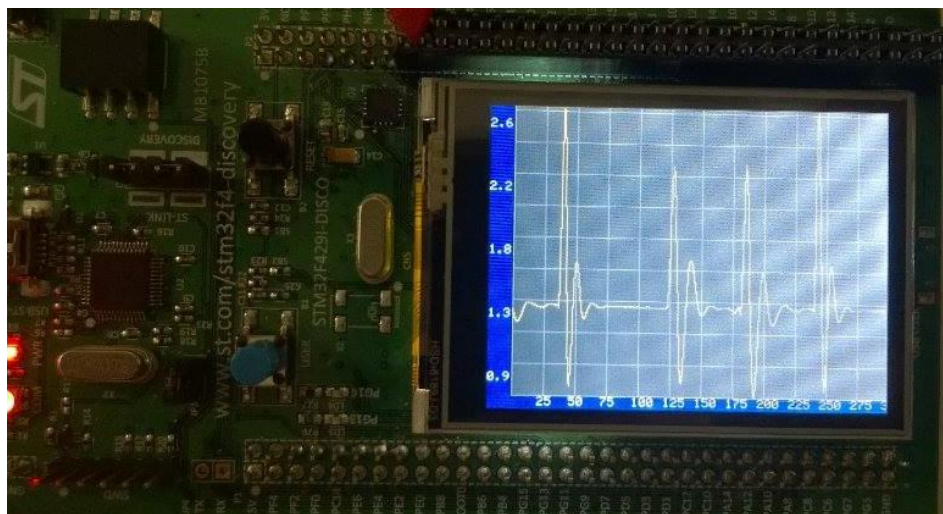


Figura 2: Rappresentazione di una funzione

Nella seconda fase, invece, è stata affrontata la progettazione di una serie di algoritmi utili per rappresentare la figura e adattarla alle dimensioni dello schermo. Durante la terza fase è stato configurato l'hardware della scheda per l'acquisizione e la conversione del segnale analogico in ingresso, attraverso l'uso di *HAL drivers*, una libreria che permette di configurare i vari componenti hardware di una scheda. Il prototipo è stato realizzato sulla scheda *STM32f429I-DISCOVERY* che ben si adatta agli scopi del tirocinio, in quanto provvista di un potente microcontrollore e di uno schermo *LCD*. Il sistema realizzato è stato infine validato utilizzando un generatore di funzioni arbitrarie.

La trattazione dell'elaborato è così suddivisa: il capitolo 2 presenta una descrizione tecnica della scheda, il suo schematico e un'analisi delle librerie *emWin*, usata per la gestione della grafica, e *HAL_drivers*, usata per la configurazione dell'hardware del microcontrollore. Il capitolo 3 descrive le specifiche generali di un oscilloscopio e le soluzioni progettuali proposte. Il capitolo 4 illustra le modalità di realizzazione dell'applicativo, approfondendo le parti più rilevanti del codice sviluppato per il microcontrollore, e riporta i risultati della validazione sperimentale. Il capitolo 5 contiene le conclusioni.

2 Hardware e software utilizzati

2.1 I microcontrollori



Figura 3: STM32F429ZIT6

In elettronica digitale il **microcontrollore** [1] o **microcontroller** o **MCU** (*MicroController Unit*) (**Figura 3**) è un dispositivo elettronico integrato su singolo chip, nato come evoluzione alternativa al *microprocessore* ed utilizzato generalmente in sistemi *embedded* ovvero per applicazioni specifiche (*special purpose*) di controllo digitale. È progettato per interagire direttamente con il mondo esterno tramite un programma residente nella propria memoria interna e mediante l'uso di pin specializzati o configurabili dal programmatore. Sono disponibili in 3 fasce di capacità elaborativa (ampiezza del bus dati): 8 bit, 16 bit e 32 bit. Generalmente sono dotati di *CPU* (*Central Processing Unit*) *CISC* (*Complex Instruction Set Computer*) con architettura *von Neumann*, anche se più di recente sono apparsi microcontrollori con *CPU* ad architettura *RISC* (*Reduced Instruction Set Computer*), come ad esempio il Texas Instruments MSP430, meglio predisposti per l'utilizzo dei moderni compilatori, piuttosto che dell'Assembly. Taluni microcontrollori complessi (come il Freescale 68302) hanno un processore *RISC* separato dal processore core. L'ampia gamma di funzioni di comando e controllo disponibili, sia analogiche che digitali, integrate sullo stesso chip, permette l'impiego delle *MCU* in sostituzione di schede elettroniche cablate tradizionali ben più complesse e costose. Per i microcontrollori sono pubblicati sistemi di sviluppo amatoriali e professionali anche in modalità *open source*.

2.2 La scheda STM32F429I-DISCO



Figura 4: STM32F429I-DISCO

La scheda *STM32f429I-DISCOVERY* (**Figura 4**) fa parte della serie *STM32 F4* progettata per sviluppare applicazioni prototipali. Offre tutto il necessario per poter realizzare e sviluppare applicazioni di vario tipo dotate o meno di un'interfaccia grafica. Basato sul processore *STM32F429ZIT6*, include un *ST-LINK/V2* come strumento di debug integrato, uno schermo LCD, una *SDRAM* (*Synchronous Dynamic Random Access Memory*) da 64 Mbit, un giroscopio, due connettori micro-USB, LED e pulsanti [2]. Di seguito si riportano le principali caratteristiche:

- Microcontrollore STM32F429ZIT6 caratterizzato da una memoria flash di 2MB, 256 KB di RAM in un package LQFP144.
- ST-LINK/V2 (interfaccia per la programmazione della flash del microcontrollore e per il debug).
- Alimentazione scheda: per mezzo di una porta USB o alimentazione esterna di 3V-5V.
- Schermo LCD da 2.4" (QVGA TFT).
- 64-Mbit di SDRAM.
- 6 LED:
 - LD1 (rosso/verde) per la comunicazione USB.
 - LD2 (rosso) per l'accensione a 3.3V.
 - 2 LED utilizzabili: LD3 (verde), LD4 (rosso).
 - 2 USB On-The-Go (OTG) LED.
- 2 pulsanti: (user e reset).
- USB OTG con connettore micro-AB. [2]

2.3 Il microcontrollore STM32F429

STM32F429ZIT6 è un microcontrollore appartenente alla serie *STM32* prodotto da *STMicroelectronics*. Di seguito sono riportate alcune delle sue caratteristiche principali:

- Core: ARM Cortex M4 con frequenza di 180MHz.
- Memorie:
 - 2 MB di memoria flash organizzata in due parti che permettono di eseguire operazioni di lettura e scrittura.
 - 256 + 4 KB di SRAM inclusi 4KB di RAM CCM (Core Coupled Memory).
- ADC da 2.4 MSPS (Mega Sample Per Second), 3x12 bit: 24 canali.
- Convertitori D/A da 2x12-bit.
- 17 timer: sono presenti 12 timer da 16-bit e 2 da 32-bit con frequenza di 180MHz.

In **Figura 5** è presente il diagramma a blocchi del microcontrollore in oggetto. Si noti la presenza di alcune periferiche. Il bus *APB2* (*Advanced Peripheral Bus*) è connesso ai timer, agli ADC, alle *USART* (*Universal Synchronous-Asynchronous Receiver/Transmitter*) e alle *SPI* (*Serial Peripheral Interface*). Il bus *APB1*, invece, è connesso a diversi timer (tra cui il timer *TIM2*), mentre l'*AHB1* (*Advanced High-performance Bus*) è direttamente collegato al *DMA* e ai registri della memoria flash.

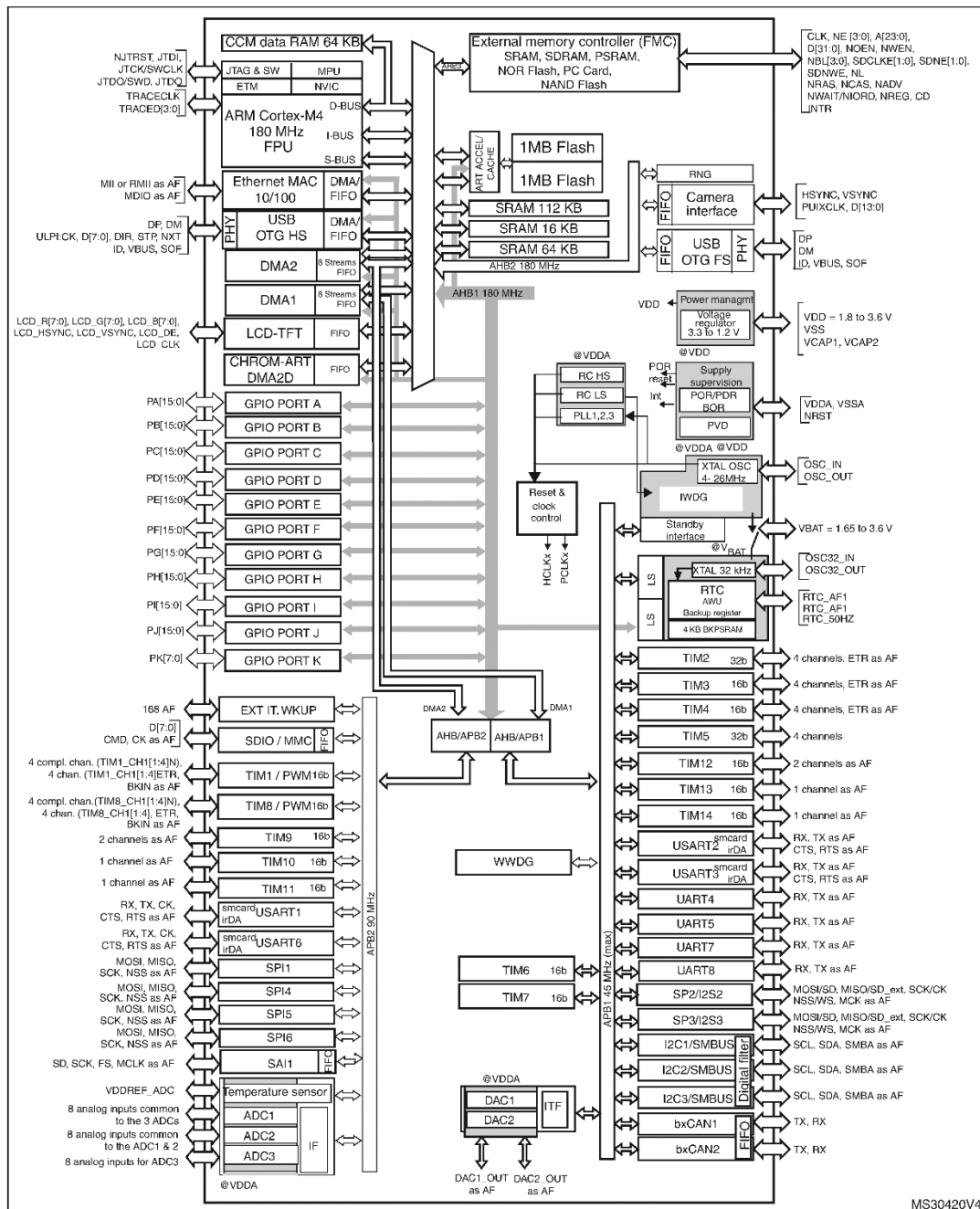


Figura 5: schematico di STM32F429ZIT6

2.4 Librerie del microcontrollore: HAL (Hardware Abstraction Layer)

Questa libreria, fornita insieme al microcontrollore, è composta da un insieme di API (Application Programming Interface) che favoriscono la gestione dell'hardware del microcontrollore da parte dello sviluppatore di un'applicazione. HAL è stata progettata tenendo conto di un'architettura generica; pertanto le sue funzioni possono essere utilizzate sui vari processori all'interno della stessa famiglia, favorendo la portabilità del codice. Le API più importanti sono quelle che permettono di inizializzare e configurare le periferiche e gestire il trasferimento di dati. Le API sono suddivise in due categorie:

- Generic API: API che adoperano funzioni generiche che vanno bene per qualunque microcontrollore della famiglia STM32.

- **Extension API:** API che includono specifiche funzioni per una ben determinata famiglia di microcontrollori.

Per il progetto dell'oscilloscopio la libreria *HAL* è stata utile per configurare il *DMA*, l'*ADC* e il timer che temporizza l'acquisizione del segnale analogico da visualizzare.

2.4.1 Configurazione del Timer

La configurazione del timer richiede la dichiarazione della struttura *TIM_HandleTypeDef*, che permette di definire i parametri di funzionamento. Successivamente si attiva il clock del timer *TIM2* collegato al bus *APB1*, per mezzo della macro *__HAL_RCC_TIM2_CLK_ENABLE*. Tra i parametri da configurare vi sono il valore del prescaler, del periodo e la scelta del timer da utilizzare. Infine si inizializza la struttura dati. Le funzioni, strutture e macro adoperate sono le seguenti:

- **TIM_HandleTypeDef**
 - Descrizione: struttura contenente i parametri di configurazione del timer.
 - Parametri:
 - a) *TIM_TypeDef *Instance*: puntatore alla struct che gestisce i parametri di configurazione di uno dei registri del timer. A questo parametro bisogna assegnare l'indirizzo del timer che si vuole utilizzare.
 - b) *TIM_Base_InitTypeDef Init*: struttura contenente i parametri di configurazione relativi a frequenza e periodo.
- **TIM_Base_InitTypeDef Init**
 - Descrizione: struttura contenente i parametri di configurazione del timer relativi a frequenza e periodo.
 - Parametri:
 - a) *uint32_t Prescaler*: costante usata per dividere la frequenza massima del microcontrollore.
 - b) *uint32_t CounterMode*: specifica il "counter mode" (come specificato nel file *stm32f4xx_hal_tim.h*, alla voce *TIM_Counter_Mode*).
 - c) *uint32_t Period*: specifica il periodo.
 - d) *uint32_t ClockDivision*: specifica il "Clock Division" (come specificato nel file *stm32f4xx_hal_tim.h* alla voce *"TIM_ClockDivision"*).
- **__HAL_RCC_TIM2_CLK_ENABLE**: è una macro che permette di attivare il clock del timer collegato alla periferica *APB1*.
- **HAL_TIM_Base_Init**
 - Prototipo: *HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef *htim)*.
 - Descrizione: Questa funzione inizializza la struttura *htim*. Deve essere chiamata subito dopo aver configurato i parametri della struttura *htim*.
 - Parametri:
 - a) *htim*: puntatore alla struttura che gestisce la configurazione del timer.
 - Valore ritornato: HAL status.
- **HAL_TIM_Base_Start**
 - Prototipo: *HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef *htim)*.
 - Descrizione: funzione che fa partire il conteggio del timer.
 - Parametri:
 - a) *htim*: puntatore alla struttura che gestisce la configurazione del timer.
 - Valore ritornato: HAL status.
- **HAL_TIM_Base_Stop**
 - Prototipo: *HAL_StatusTypeDef HAL_TIM_Base_Stop (TIM_HandleTypeDef *htim)*.
 - Descrizione: funzione che interrompe il conteggio del timer.
 - Parametri:
 - a) *htim*: puntatore alla struttura contenente i parametri di configurazione del timer.
 - Valore di ritorno: HAL status.

2.4.2 Configurazione dell'ADC, del canale dell'ADC e del GPIOx.

Per poter utilizzare le funzionalità della periferica ADC è necessario definire tre strutture che servono, rispettivamente, per la configurazione dell'ADC, del GPIOx (nel caso di specie GPIOC) e del canale dell'ADC (in questo caso 13). Si abilitano i clock dei bus APB2 e AHBI (a cui sono connessi rispettivamente l'ADC e il DMA) per mezzo di due macro e una volta completato il tutto si possono chiamare le funzioni di inizializzazione. Le funzioni, macro e strutture utilizzate sono le seguenti:

- **ADC_HandleTypeDef**
 - Descrizione: è una struttura che definisce i parametri di configurazione dell'ADC (definita nel file *stm32f4xx_hal_adc.h*).
 - Parametri più importanti:
 - a) *ADC_TypeDef *Instance*: puntatore alla struttura che definisce i registri dell'ADC (questo puntatore deve essere assegnato a uno degli indirizzi dell'ADC).
 - b) *ADC_InitTypeDef Init*: struttura che gestisce alcuni parametri di configurazione dell'ADC (clock, risoluzione, allineamento dati, scelta del trigger, dell'output trigger, ecc.).
- **ADC_InitTypeDef Init**
 - Descrizione: struttura che configura alcuni parametri dell'ADC.
 - Parametri:
 - a) *uint_32t ClockPrescaler*: seleziona la frequenza del clock nell'ADC. Questo clock è comune per tutti gli ADC.
 - b) *uint_32t Resolution*: imposta la risoluzione dell'ADC. I possibili valori da assegnare sono in **Tabella 1**.
 - c) *uint_3t DataAlign*: specifica se l'allineamento dei dati dell'ADC è a destra o sinistra.
 - d) *uint_32_t NbrOfConversion*: specifica il numero di conversioni che deve effettuare l'ADC.
 - e) *uint_32_t ExternalTrigConvEdge*: abilita il trigger esterno. I parametri sono definiti nella **Tabella 2**.
 - f) *uint_32_t ExternalTrigConv*: abilita il trigger esterno relativo al timer TIM2.

Tabella 1: Valori ammessi per il parametro Resolution

ADC_RESOLUTION_12B	ADC con risoluzione da 12 bit.
ADC_RESOLUTION_10B	ADC con risoluzione da 10 bit.
ADC_RESOLUTION_8B	ADC con risoluzione da 8 bit.
ADC_RESOLUTION_6B	ADC con risoluzione da 6 bit

Tabella 2: Valori ammessi per il parametro ExternalTrigConvEdge

ADC_EXTERNALTRIGCONVEDGE_NONE	Nessun trigger esterno dell'ADC è attivato.
ADC_EXTERNALTRIGCONVEDGE_RISING	Trigger esterno dell'ADC abilitato sul fronte di salita.
ADC_EXTERNALTRIGCONVEDGE_FALLING	Trigger esterno dell'ADC abilitato sul fronte di discesa.
ADC_EXTERNALTRIGCONVEDGE_RISINGFALLING	Trigger esterno dell'ADC abilitato sia sul fronte di salita che di discesa.

- **__HAL_RCC_ADC2_CLK_ENABLE**: macro che abilita il clock di APB2.
- **__HAL_RCC_GPIOC_CLK_ENABLE**: macro che abilita il clock del GPIO collegato al bus AHBI.
- **HAL_GPIO_DeInit**
 - Prototipo: *void HAL_GPIO_DeInit (GPIO_TypeDef *GPIOx, uint32_t GPIO_Pin)*.
 - Descrizione: Inizializza le periferiche del GPIOx (x varia da A fino a K) al loro valore di default.

- Parametri:
 - a) *GPIOx*: struttura che gestisce la configurazione del *GPIO*.
 - b) *GPIO_Pin*: intero che definisce il bit sul quale andare a scrivere e/o leggere. Di norma questo intero è assegnato ad una variabile *GPIO_PIN_x* (x è il numero del pin che varia da 0 a 15).
 - Valore ritornato: nessuno.
- **HAL_GPIO_Init**
 - Prototipo: *void HAL_GPIO_Init (GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init).*
 - Descrizione: inizializza il *GPIOx* (x è compreso tra A e K). Questa funzione deve essere chiamata subito dopo aver compilato i parametri di configurazione delle strutture *GPIOx* e *GPIO_Init*.
 - Parametri:
 - a) *GPIOx*: puntatore alla struttura *GPIO_TypeDef*.
 - b) *GPIO_Init*: puntatore alla struttura *GPIO_Init* contenente i parametri di configurazione per uno specifico *GPIO*.
 - Valore di ritorno: nessuno.
 - **HAL_ADC_Init**
 - Prototipo: *HAL_StatusTypeDef HAL_ADC_Init (ADC_HandleTypeDef *hadc).*
 - Descrizione: inizializza l'*ADCx* (x è un intero compreso tra 1 e 3). I parametri di configurazione sono contenuti nella struttura *ADC_HandleTypeDef*.
 - Parametri:
 - a) *hadc*: puntatore alla struttura *ADC_HandleTypeDef*.
 - Valore di ritorno: HAL status.
 - **__HAL_ADC_ENABLE**: macro che abilita l'*ADC*.
 - **HAL_ADC_ConfigChannel**
 - Prototipo: *HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef *hadc, ADC_ChannelConfTypeDef *sConfig).*
 - Descrizione: funzione che configura un canale dell'*ADC*. I parametri di configurazione del canale sono contenuti nella struttura *ADC_ChannelConfTypeDef*.
 - Parametri:
 - a) *hadc*: puntatore alla struttura che contiene i parametri di configurazione dell'*ADC*.
 - b) *sConfig*: puntatore alla struttura che contiene i parametri di configurazione del canale dell'*ADC*.
 - Valore di ritorno: HAL status.
 - **HAL_ADC_Start**
 - Prototipo: *HAL_StatusTypeDef HAL_ADC_Start (ADC_HandleTypeDef *hadc).*
 - Descrizione: abilita l'*ADC* e inizia la conversione sul canale che è stato opportunamente configurato.
 - Parametri:
 - a) *hadc*: puntatore alla struttura contenente i parametri di configurazione dell'*ADC*.
 - Valore di ritorno: HAL status.
 - **HAL_ADC_Start_DMA**
 - Prototipo: *HAL_StatusTypeDef HAL_ADC_Start_DMA (ADC_HandleTypeDef *hadc, uint32_t *pData, uint32_t Length).*
 - Descrizione: abilita l'*ADC DMA request* dopo l'ultimo trasferimento (*Single-ADC mode*) ed abilita l'*ADC peripheral*.
 - Parametri:
 - a) *hadc*: puntatore alla struttura contenente i parametri di configurazione dell'*ADC*.
 - Valore di ritorno: HAL status.

2.4.3 Configurazione del DMA

Per la configurazione del *DMA* bisogna dichiarare la struttura che si occupa della sua configurazione, ovvero *DMA_HandleTypeDef*. Dopo aver abilitato il clock del bus *AHB1* a cui è collegato il *DMA2*, si inizializza (per

mezzo della funzione *HAL_DMA_DeInit*) e si configura il *DMA*. Completata questa parte si chiama una funzione che fa in modo che *ADC* e *DMA* possano comunicare tra loro tutte le volte che l'*ADC* ne fa richiesta (usando la funzione *HAL_ADC_Start_DMA* definita nel paragrafo 2.5.2). Le funzioni, macro e strutture utilizzate sono le seguenti:

- **DMA_HandleTypeDef**
 - Descrizione: struttura che definisce i parametri di configurazione del *DMA*.
 - Parametri più importanti:
 - a) *DMA_Stream_TypeDef *Instance*: puntatore alla struttura contenente i parametri di configurazione dei registri del *DMA* (*Instance* è uguagliato all'indirizzo di uno dei registri del *DMA*).
 - b) *DMA_InitTypeDef Init*: struttura che definisce alcuni parametri di configurazione del *DMA* (canale, tipo di trasferimento, ampiezza della periferica dati e memoria, priorità, ecc.).
- **DMA_InitTypeDef Init**
 - Descrizione: struttura che definisce alcuni parametri di configurazione del *DMA*.
 - Parametri:
 - a) *uint32_t Channel*: specifica il canale del *DMA*.
 - b) *uint32_t Direction*: specifica se l'informazione è trasferita dalla memoria alla periferica, da memoria a memoria oppure da periferica a memoria.
 - c) *uint32_t PeriphInc*: specifica se l'indirizzo del registro della periferica deve essere incrementato o meno.
 - d) *uint32_t MemInc*: specifica se l'indirizzo del registro della memoria deve essere incrementato o meno.
 - e) *uint32_t PeriphDataAlignment*: specifica la dimensioni dei dati nella periferica.
 - f) *uint32_t MemDataAlignment*: specifica le dimensioni dei dati in memoria.
 - g) *uint32_t Priority*: specifica la priorità per il *DMA* che può essere bassa, media o alta.
- **__HAL_RCC_DMA2_CLK_ENABLE**: macro che abilita il clock del bus *AHB1* a cui è collegato il *DMA2*.
- **HAL_DMA_DeInit**
 - Prototipo: *HAL_StatusTypeDef HAL_DMA_DeInit (DMA_HandleTypeDef *hdma)*.
 - Descrizione: funzione che inizializza i parametri di configurazione del *DMA* ai propri valori di default.
 - Parametri:
 - a) *hdma*: puntatore alla struttura *DMA_HandleTypeDef* che gestisce i parametri di configurazione del *DMA*.
 - Valore di ritorno: HAL status.
- **HAL_DMA_Init**
 - Prototipo: *HAL_StatusTypeDef HAL_DMA_Init (DMA_HandleTypeDef *hdma)*.
 - Descrizione: funzione che inizializza il *DMA* in relazione ai parametri di configurazione definiti nella struttura *DMA_HandleTypeDef*.
 - Parametri:
 - a) *hdma*: puntatore alla struttura contenente i parametri di configurazione del *DMA*.
 - Valore di ritorno: HAL status.

2.5 Librerie del microcontrollore: *emWin*

emWin è una libreria grafica sviluppata da *SEGGER* [6] in linguaggio di programmazione C che può essere utilizzata per lo sviluppo di interfacce grafiche su microcontrollore adattabile a vari sistemi operativi real-time e a schermi *LCD* di varia dimensione. In questa libreria sono disponibili delle *API* che permettono di gestire i colori, le dimensioni e il posizionamento sullo schermo per ciascuno degli oggetti creati. In questo tirocinio sono state adoperate delle *API* tali da generare la tipica grafica di un oscilloscopio, ovvero un piano cartesiano (assi delle ascisse e delle ordinate, griglie, grafici e colori dei punti visualizzati).

2.5.1 I Widget

Per utilizzare in maniera corretta *emWin* è necessario comprendere il concetto di *widget*. I Widget sono degli oggetti (una barra, un'icona, un bottone, ecc.) che rispondono a determinate proprietà. Quest'ultimi se accostati l'uno all'altro possono creare un'interfaccia grafica. Il vantaggio principale sta nel fatto che questi oggetti possono essere posizionati lungo uno schermo con grande semplicità usando delle opportune funzioni [4]. In **Figura 6** sono mostrati tutti i *widgets* disponibili nella versione V5.28 della libreria:










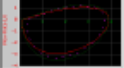

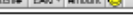
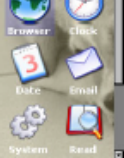




Name	Screenshot (classic)	Screenshot (skinned)	Description
BUTTON			Button which can be pressed. Text or bitmaps may be displayed on a button.
CHECKBOX			Check box which may be checked or unchecked.
DROPDOWN			Dropdown listbox, opens a listbox when pressed.
EDIT			Single-line edit field which prompts the user to type a number or text.
FRAMEWIN			Frame window. Creates the typical GUI look.
GRAPH			Graph widget, used to show curves or measured values.
HEADER			Header control, used to manage columns.
ICONVIEW			Icon view widget. Useful for icon based platforms as found in common hand held devices.
IMAGE			Image widget. Displays several image formats automatically.
KNOB			Knob widget which can be used to adjust uncountable values.
LISTBOX			Listbox which highlights items as they are selected by the user.
LISTVIEW			Listview widgets are used to creates tables.

Figura 6: Widgets disponibili

2.5.2 Graph Widget

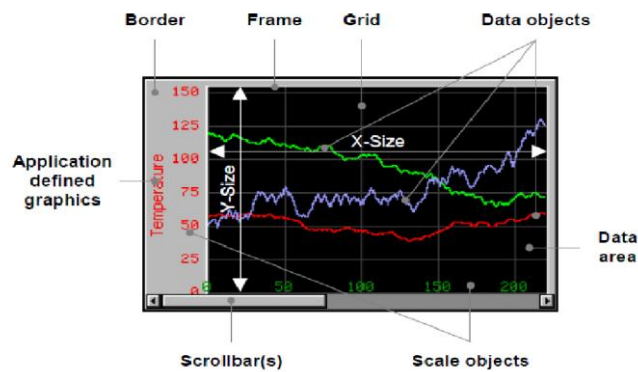


Figura 7: Graph Widget

Un graph widget è un diagramma cartesiano composto dai seguenti elementi:

- *Border*: bordo del graph widget.
- *Frame*: sottile cornice attorno al data area.
- *Data Area*: area nella quale sono mostrati le griglie e gli oggetti (*Data object*).
- *Grid*: griglie orizzontali e verticali contenute nel data area.
- *Data object*: vettori di punti che sono visualizzati sul data area.
- *Application defined graphics*: area nella quale è possibile inserire del testo o numeri.
- *Scrollbar*: se l'area di rappresentazione del data object è più grande del data area il graph widget visualizza automaticamente una barra di scorrimento orizzontale e/o verticale.
- *Scale objects*: scale orizzontali e verticali che possono essere collegate al graph widget.
- *X_Size*: intervallo rappresentato sull'ascissa del data area.
- *Y_Size*: intervallo rappresentato sull'ordinata del data area.

Lo schema di un graph widget è mostrato in **Figura 7**.

La procedura di creazione del graph widget è la seguente:

- Creo il graph widget e setto i valori desiderati.
- Creo il data object.
- Collego il data object al graph widget.
- Creo la scale object (opzionale).
- Collego la scale object (opzionale).

Vi sono due modalità di rappresentazione: modalità XY e modalità YT (**Figura 8**). La modalità XY è utile per rappresentare generici punti sul piano.

La modalità YT è utile per rappresentare un segnale nella base dei tempi. L'asse orizzontale rappresenta il tempo, mentre quello verticale la tensione.

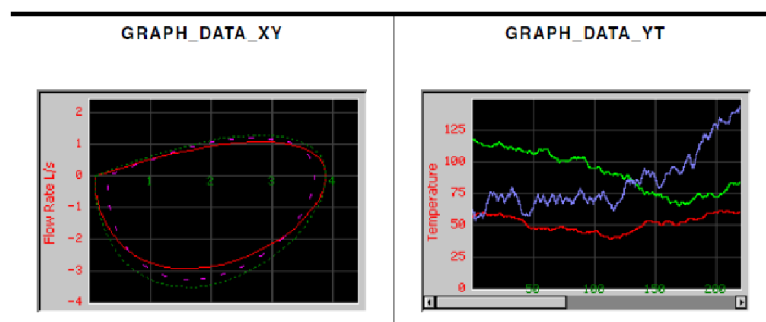


Figura 8: Tipi di rappresentazioni supportate

Un widget può essere disegnato in base alle sue proprietà usando opportune funzioni. Un widget deve essere anzitutto inizializzato per mezzo delle funzioni *GUI_init* che inizializza la libreria *emWin*, *WM_Exec* che disegna i

widget e *WM_ValidateWindow* che convalida la finestra contenente i widget. Altre funzioni, invece, servono per definire gli effetti (usando la funzione *WIDGET_SetDefaultEffect*), impostare il colore dei vari oggetti componenti il graph widget (usando la funzione *GRAPH_SetColor*), visualizzare griglie (usando la funzione *GRAPH_SetGridVis*), bordi (usando la funzione *GRAPH_SetBorder*) e scale numeriche (usando la funzione *GRAPH_SetScale*). Di seguito è mostrato un elenco delle funzioni disponibili:

- **WM_Exec:**
 - Descrizione: disegna i Widgets.
 - Prototipo: *int WM_Exec(void)*.
 - Valori ritornati: 0 se non ci sono finestre da realizzare, 1 se c'è una finestra da realizzare.
- **WM_ValidateWindow:**
 - Descrizione: convalida una finestra specifica, ovvero questa funzione fa sì che la finestra venga aggiornata. Normalmente è chiamata internamente e non è necessario che venga chiamata dall'utente.
 - Prototipo: *void WM_ValidateWindow(WM_HWIN hWin)*.
 - Parametri:
 - a) *hWin*: handle della finestra che si vuole convalidare.
- **GUI_Init:**
 - Descrizione: funzione che inizializza la libreria *emWin*. Questa funzione deve essere chiamata prima di generare un'interfaccia grafica.
 - Prototipo: *void GUI_init()*.
- **GUI_Clear:**
 - Descrizione: funzione che "pulisce" la finestra.
 - Prototipo: *void GUI_Clear(void)*.
 - Esempio: mostrare la scritta "Hello World" sullo schermo, attendere 1 secondo e pulire lo schermo.
GUI_DispStringAt("Hello world", 0, 0); // Testo da visualizzare sul display.
GUI_Delay(1000); // Attesa di 1 secondo.
GUI_Clear(); // Pulisce il display.
- **GUI_CURSOR_Show:**
 - Descrizione: mostra il cursore sul display.
 - Prototipo: *void GUI_CURSOR_Show(void);*
- **GUI_CURSOR_SetPosition**
 - Descrizione: setta la posizione del cursore.
 - Prototipo: *void GUI_CURSOR_SetPosition(int x, int y);*
 - Parametri:
 - a) *x*: posizione x del cursore.
 - b) *y*: posizione y del cursore.
- **GUI_COUNTOF(vettore)**
 - Descrizione: macro che restituisce la dimensione di un vettore.
 - Parametri:
 - a) Vettore di interi o float.
 - Valore di ritorno: dimensione del vettore.
- **WIDGET_SetDefaultEffect**
 - Descrizione: definisce l'effetto di visualizzazione di default per i widget.
 - Prototipo: *const WIDGET_EFFECT *WIDGET_SetDefaultEffect(const WIDGET_EFFECT *pEffect);*
 - Parametri:
 - a) *pEffect*: puntatore alla struttura che gestisce gli effetti del widget (*Tabella 3*).
 - Valore di ritorno: l'effetto usato precedentemente.

Tabella 3: Valori ammessi per il puntatore pEffect

&WIDGET_Effect_3D	Imposta l'effetto "3D"
&WIDGET_Effect_None	Imposta l'effetto "None"
&WIDGET_Effect_Simple	Imposta l'effetto "Simple"

- **GRAPH_CreateEx**

- Descrizione: funzione che crea un graph widget di una determinata dimensione.
- Prototipo: *GRAPH_Handle GRAPH_CreateEx(int x0, int y0, int xSize, int ySize, WM_HWIN hParent, int WinFlags, int ExFlags, int Id).*
- Parametri:
 - a) *X0*: dimensione orizzontale del graph widget.
 - b) *Y0*: dimensione verticale del graph widget.
 - c) *xSize*: dimensione orizzontale del data area.
 - d) *ySize*: dimensione verticale del data area.
 - e) *hParent*: struttura che gestisce la finestra principale ("parent window").
 - f) *WinFlags*: vi sono dei flag opportuni che possono essere assegnati (è consigliabile usare le costanti "WM_CF_SHOW | WM_CF_CONST_OUTLINE", che servono per rendere visibile il graph widget).

- **GRAPH_SetBorder:**

- Descrizione: definisce i bordi del graph widget (**Figura 9** e **Figura 10**).
- Prototipo: *void GRAPH_SetBorder(GRAPH_Handle hObj, unsigned BorderL, unsigned BorderT, unsigned BorderR, unsigned BorderB);*
- Parametri:
 - a) *hObj*: struttura che identifica l'oggetto graph widget al quale devono essere settati i bordi.
 - b) *BorderL*: dimensione del bordo sinistro.
 - c) *BorderT*: dimensione del bordo in alto.
 - d) *BorderR*: dimensione del bordo destro.
 - e) *BorderB*: dimensione del bordo in basso.

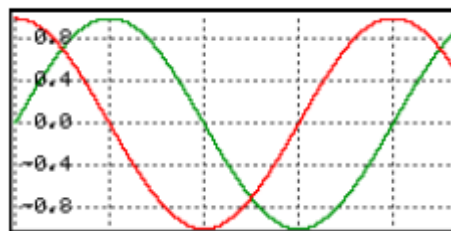


Figura 9: Prima di chiamare la funzione **GRAPH_SetBorder** la scala numerica (a sinistra della figura) è senza bordo.

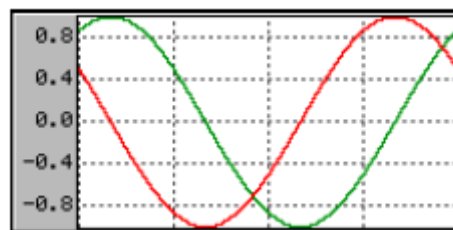


Figura 10: Dopo aver chiamato la funzione **GRAPH_SetBorder** la scala numerica presenta un bordo

- **GRAPH_SetColor**

- Descrizione: setta il colore desiderato per ciascun oggetto del graph widget (**Figura 11** e **Figura 12**).
- Prototipo: *GUI_COLOR GRAPH_SetColor(GRAPH_Handle hObj, GUI_COLOR Color, unsigned Index).*
- Parametri:
 - a) *hObj*: struttura che identifica l'oggetto graph widget.
 - b) *Color*: colore che si desidera impostare.
 - c) *Index*: intero che può assumere diversi valori (**Tabella 4**).
- Valore di ritorno: precedente colore.

Tabella 4: Valori permessi per la variabile Index

GRAPH_CI_BK	Setta il colore di sfondo.
GRAPH_CI_BORDER	Setta il colore della border area.
GRAPH_CI_FRAME	Setta il colore della sottile linea di frame.
GRAPH_CI_GRID	Setta il colore della griglia.

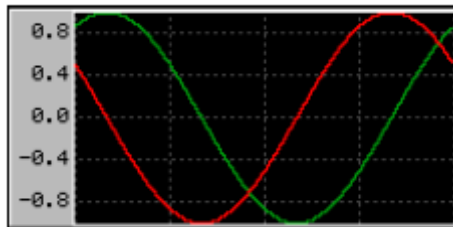


Figura 11: Prima di chiamare la funzione GRAPH_SetColor il data area è colorato di nero

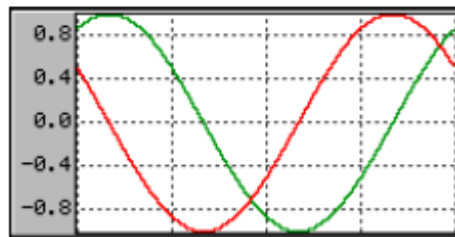


Figura 12: Dopo aver chiamato la funzione GRAPH_SetColor il colore del data area da nero diventa bianco.

- **GRAPH_SetGridVis**

- Descrizione: funzione che rappresenta le griglie sullo schermo (**Figura 13** e **Figura 14**).
- Prototipo: *unsigned GRAPH_SetGridVis(GRAPH_Handle hObj, unsigned OnOff).*
- Parametri:
 - a) *hObj*: struttura che identifica l'oggetto graph widget.
 - b) *OnOff*: può assumere due valori: 1 (se la griglia è visibile), 0 altrimenti (impostazione di default).
- Valore di ritorno: precedente valore relativo alla visibilità della griglia.

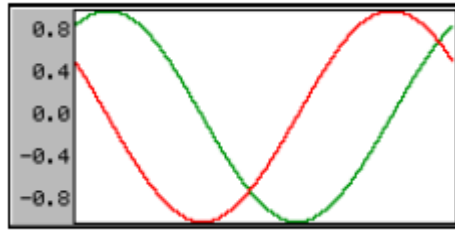


Figura 13: Prima di chiamare la funzione GRAPH_SetGridVis il data area è senza griglie.

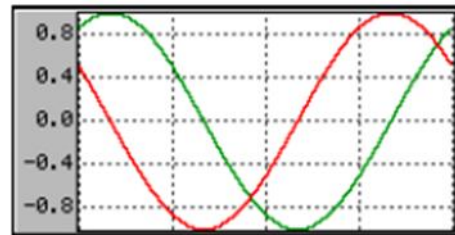


Figura 14: Dopo aver chiamato la funzione GRAPH_SetGridVis il data area presenta le griglie.

- **GRAPH_SetGridDistX**
GRAPH_SetGridDistY

- Descrizione: funzioni che impostano la distanza tra una griglia e l'altra lungo ascisse (*GRAPH_SetGridDistX*) e ordinate (*GRAPH_SetGridDistY*) (**Figura 15** e **Figura 16**).
- Prototipi:
 - a) *unsigned GRAPH_SetGridDistX(GRAPH_Handle hObj, unsigned Value).*
 - b) *unsigned GRAPH_SetGridDistY(GRAPH_Handle hObj, unsigned Value).*
- Parametri:
 - a) *hObj*: struttura che identifica l'oggetto "grafico".
 - b) *Value*: distanza in pixel tra una griglia e l'altra. Il valore di default è pari a 50.
- Valore di ritorno: precedente distanza tra due griglie consecutive.

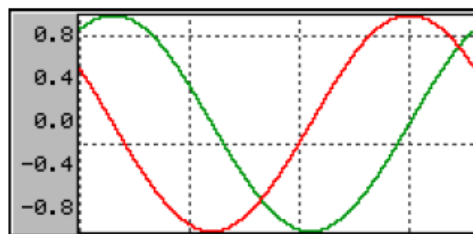


Figura 15: Prima di chiamare le funzioni GRAPH_SetGridDistX e GRAPH_SetGridDistY le righe della griglia sono ad una certa distanza tra loro.

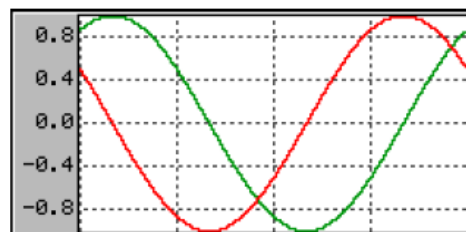


Figura 16: Dopo aver chiamato le funzioni GRAPH_SetGridDistX e GRAPH_SetGridDistY le distanze tra le righe delle griglie cambiano.

- **GRAPH_DATA_YT_Create**

- Descrizione: funzione che crea un grafico (data object). Questa richiede per ogni valore relativo all'asse x il corrispondente valore sull'asse y. Tipicamente è una funzione usata sui grafici dipendenti dal tempo.
- Prototipo: *GRAPH_DATA_Handle GRAPH_DATA_YT_Create(GUI_COLOR Color, unsigned MaxNumItems, I16 *pData, unsigned NumItems);*
- Parametri:
 - a) *Color*: colore della funzione.
 - b) *MaxNumItems*: massimo numero di valori numerici rappresentanti la funzione.
 - c) *pData*: puntatore al dato che deve essere aggiunto all'oggetto. Il puntatore deve essere un array di tipo *I16*, ovvero un signed short.
 - d) *NumItems*: numero di interi che devono essere aggiunti.
- Valore di ritorno: struttura del graph object se la creazione ha avuto successo, altrimenti 0.

- **GRAPH_SCALE_Create**

- Descrizione: crea la scala numerica relativa al graph object.
- Prototipo: *GRAPH_SCALE_Handle GRAPH_SCALE_Create(int Pos, int TextAlign, unsigned Flags, unsigned TickDist);*
- Parametri:
 - a) *Pos*: posizione relativa della scala (**Tabella 5**).
 - b) *TextAlign*: allineamento del testo usato per rappresentare i numeri della scala. Sono dei valori preimpostati (**Tabella 6** e **Tabella 7**).
 - c) *Flags*: i valori permessi sono mostrati nella **Tabella 8**
 - d) *TickDist*: distanza da una tacca della scala alla successiva.
- Valore di ritorno: struttura dell'oggetto scala se la sua creazione ha avuto successo, altrimenti 0.

Tabella 5: Valori permessi per la variabile Pos

BORDER_BOTTOM	Bordo inferiore.
BORDER_LEFT	Bordo sinistro.
BORDER_RIGHT	Bordo destro.
BORDER_TOP	Bordo superiore.

Tabella 6: Allineamento orizzontale

GUI_TA_LEFT	Allineamento orizzontale a sinistra.
GUI_TA_HCENTER	Allineamento orizzontale al centro.
GUI_TA_RIGHT	Allineamento orizzontale a destra.

Tabella 7: Allineamento verticale

GUI_TA_TOP	Allineamento verticale in alto.
GUI_TA_VCENTER	Allineamento verticale al centro.
GUI_TA_BOTTOM	Allineamento verticale in basso.

Tabella 8: Valori permessi per la variabile Flags

GRAPH_SCALE_CF_HORIZONTAL	Flag che crea la scala orizzontale.
GRAPH_SCALE_CF_VERTICAL	Flag che crea la scala verticale.

- **GRAPH_AttachScale**

- Descrizione: funzione che collega l'oggetto scala all'oggetto grafico (**Figura 17** e **Figura 18**).
- Prototipo: `void GRAPH_AttachScale(GRAPH_Handle hObj, GRAPH_SCALE_Handle hScale).`
- Parametri:
 - a) *hObj*: struttura che gestisce la configurazione dell'oggetto grafico.
 - b) *hScale*: struttura che gestisce l'oggetto scala che deve essere aggiunto all'oggetto grafico.

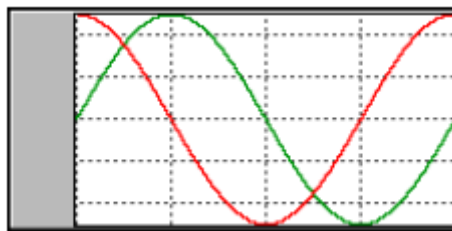


Figura 17: Prima di chiamare la funzione GRAPH_AttachScale il bordo sinistro è privo della scala numerica.

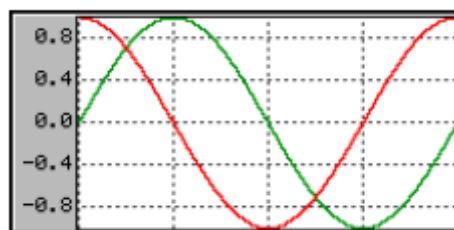


Figura 18: Dopo aver chiamato la funzione GRAPH_AttachScale il bordo sinistro presenta la scala numerica.

- **GRAPH_SCALE_SetPos**

- Descrizione: funzione che setta la posizione della scala (**Figura 19** e **Figura 20**).
- Prototipo: `int GRAPH_SCALE_SetPos(GRAPH_SCALE_Handle hScaleObj, int Pos).`
- Parametri:
 - a) *hScaleObj*: struttura dell'oggetto "scala".
 - b) *Pos*: posizione della scala in relazione al graph widget.
- Valore di ritorno: precedente posizione dell'oggetto "scala".
- Informazioni aggiuntive: Il parametro *Pos* specifica nel caso della scala orizzontale la distanza verticale in pixel dal bordo superiore del graph widget al testo della scala. Nel caso della scala verticale il parametro specifica la distanza orizzontale dal bordo sinistro del graph widget alla posizione orizzontale del testo.

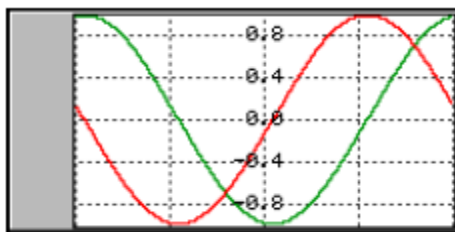


Figura 19: Prima di chiamare la funzione `GRAPH_SCALE_SetPos` la scala numerica di questo graph widget è al centro del data area.

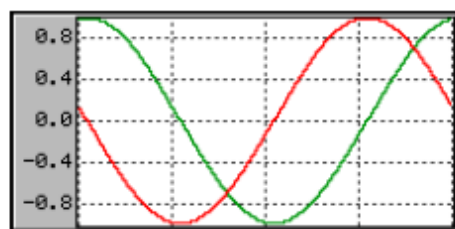


Figura 20: Dopo aver chiamato la funzione `GRAPH_SCALE_SetPos` la scala numerica è posta a sinistra in corrispondenza del bordo.

- **`GRAPH_SCALE_SetOff`**

- Descrizione: funzione che imposta un offset che ha lo scopo di scorrere l'oggetto scala in direzione negativa o positiva lungo l'asse delle ordinate (**Figura 21** e **Figura 22**).
- Prototipo: `int GRAPH_SCALE_SetOff(GRAPH_SCALE_Handle hScaleObj, int Off)`
- Parametri:
 - a) *hScaleObj*: struttura che gestisce la configurazione dell'oggetto scala.
 - b) *Off*: offset usato per rappresentare la scala.
- Valore di ritorno: offset usato in precedenza.

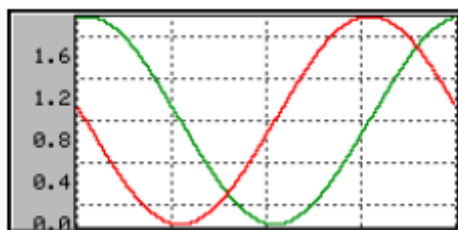


Figura 21: Prima di chiamare la funzione `GRAPH_SCALE_SetOff` l'origine degli assi è circa 0.

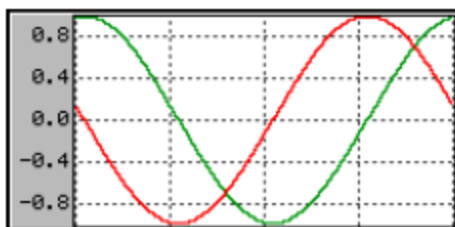


Figura 22: Dopo aver chiamato la funzione `GRAPH_SCALE_SetOff` l'origine degli assi è circa -1.

- **GRAPH_SetGridOffY**

- Descrizione: funzione che aggiunge un offset alle griglie orizzontali (**Figura 23** e **Figura 24**).
- Prototipo: *unsigned GRAPH_SetGridOffY(GRAPH_Handle hObj, unsigned Value).*
- Parametri:
 - a) *hObj*: struttura che gestisce la configurazione del graph widget.
 - b) *Value*: offset usato.
- Valore di ritorno: precedente offset usato per rappresentare le griglie orizzontali.

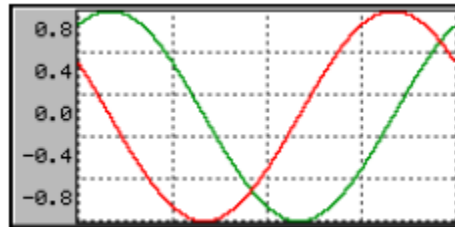


Figura 23: Prima di chiamare la funzione GRAPH_SetGridOffY

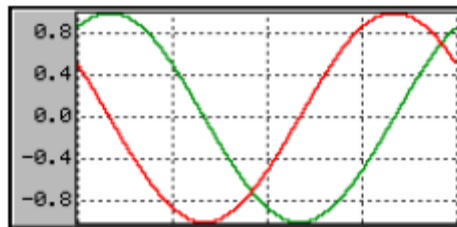


Figura 24: Dopo aver chiamato la funzione GRAPH_SetGridOffY l'offset cambia valore

- **GRAPH_DATA_YT_SetAlign**

- Descrizione: funzione che imposta l'allineamento del data object sul data area (**Figura 25** e **Figura 26**).
- Prototipo: *void GRAPH_DATA_YT_SetAlign(GRAPH_DATA_Handle hDataObj, int Align);*
- Parametri:
 - a) *hDataObj*: struttura che gestisce i dati relativi al data object da rappresentare.
 - b) *Align*: intero che può assumere dei valori predefiniti (**Tabella 9**).

Tabella 9: Valori permessi per la variabile Align

GRAPH_ALIGN_RIGHT	Allineamento del data object al lato destro (impostazione di default).
GRAPH_ALIGN_LEFT	Allineamento del data object a sinistra.

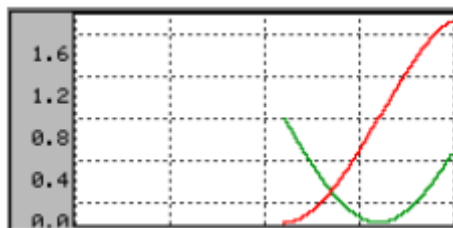


Figura 25: Prima di chiamare la funzione GRAPH_DATA_YT_SetAlign il data object è allineato a destra.

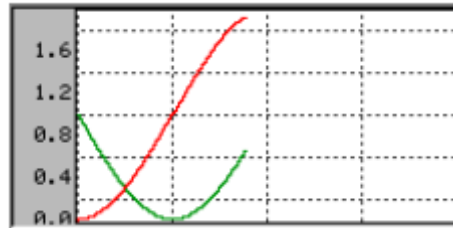


Figura 26: Dopo aver chiamato la funzione `GRAPH_DATA_YT_SetAlign` il data object è allineato a sinistra.

- **GRAPH_AttachData**

- Descrizione: funzione che collega un data object al graph widget già esistente (**Figura 27** e **Figura 28**).
- Prototipo: `void GRAPH_AddGraph(GRAPH_Handle hObj, GRAPH_DATA_Handle hData).`
- Parametri:
 - a) `hObj`: struttura che gestisce i parametri di configurazione del graph widget.
 - b) `hData`: struttura che si occupa della gestione dei dati relativi al grafico che si vuole realizzare.

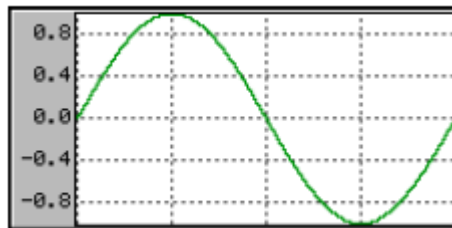


Figura 27: Prima di chiamare la funzione `GRAPH_AttachData` il data object non è ancora visualizzato sullo schermo.

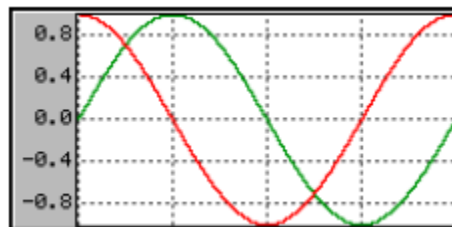


Figura 28: Dopo aver chiamato la funzione `GRAPH_AttachData` il data object (in rosso) è visualizzato sullo schermo.

- **GRAPH_DATA_YT_SetOffY**

- Descrizione: imposta l'offset per rappresentare i punti sullo schermo LCD (**Figura 29** e **Figura 30**).
- Prototipo: `void GRAPH_DATA_YT_SetOffY(GRAPH_DATA_Handle hDataObj, int Off).`
- Parametri:
 - a) `hDataObj`: struttura che gestisce i punti relativi al data object che si vuole rappresentare sullo schermo LCD.
 - b) `Off`: offset che deve essere usato per disegnare il grafico.
- Informazioni aggiuntive: l'intervallo di valori verticali che sono mostrati nelle ordinate sullo schermo LCD è compreso tra $0 - (Ysize-1)$, dove `Ysize` è la dimensione del data area.

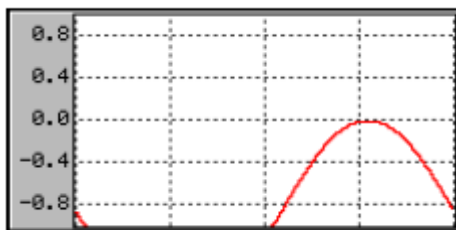


Figura 29: Prima di chiamare la funzione `GRAPH_DATA_YT_SetOffY` il data area ha un offset che non permette di vedere completamente il data object.

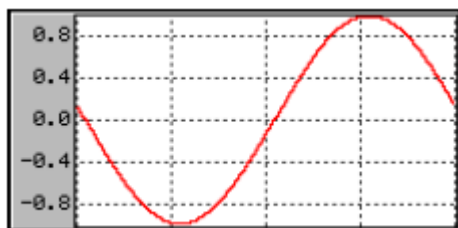


Figura 30: Dopo aver chiamato la funzione `GRAPH_DATA_YT_SetOffY` la funzione è adattata alle dimensioni del data area e può essere vista totalmente.

- **GRAPH_SCALE_SetFactor**

- Descrizione: funzione che imposta un fattore tale da rappresentare in maniera logica i numeri sulla scala. E' possibile, quindi, realizzare una relazione tra il grafico rappresentato sul data area e la scala in questione (**Figura 31** e **Figura 32**).
- Prototipo: `float GRAPH_SCALE_SetFactor(GRAPH_SCALE_Handle hScaleObj, float Factor)`.
- Parametri:
 - a) `hScaleObj`: struttura che gestisce l'oggetto scala.
 - b) `Factor`: fattore per calcolare i numeri della scala.
- Informazioni aggiuntive: senza l'uso di questa funzione l'unità dell'oggetto scala è il pixel. Con questa funzione, invece, è possibile convertire il valore di un pixel nell'unità di misura desiderata.

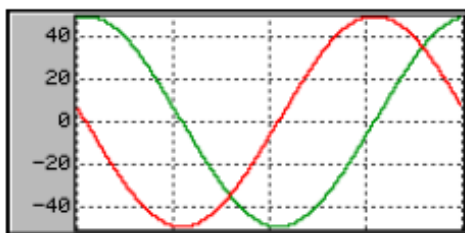


Figura 31: Prima di chiamare la funzione `GRAPH_SCALE_SetFactor` i numeri sulla scala numerica sono compresi tra -35 e 50.

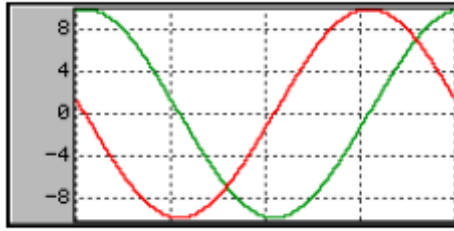


Figura 32: Dopo aver chiamato la funzione `GRAPH_SCALE_SetFactor` i numeri sulla scala numerica sono compresi tra -6 e 10.

- **GRAPH_SCALE_SetTickDist**

- Descrizione: funzione che imposta la distanza tra un numero e il suo successivo della scala (**Figura 33** e **Figura 34**).
- Prototipo: `unsigned GRAPH_SCALE_SetTickDist(GRAPH_SCALE_Handle hScaleObj,int Dist)`.
- Parametri:
 - a) `hScaleObj`: struttura che gestisce l'oggetto scala.
 - b) `Dist`: distanza in pixel tra i numeri.

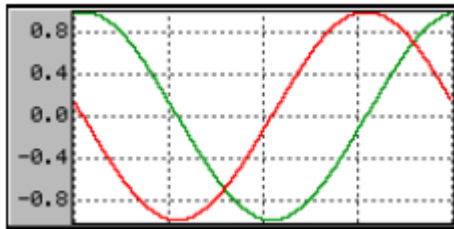


Figura 33: Prima di chiamare la funzione `GRAPH_SCALE_SetTickDist` la distanza tra un numero e il suo successivo è di 0.4.

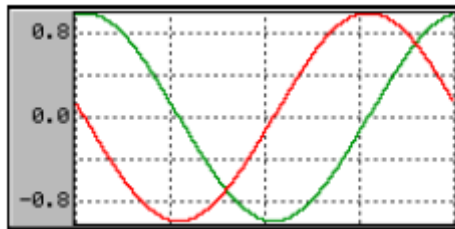


Figura 34: Dopo aver chiamato la funzione `GRAPH_SCALE_SetTickDist` la distanza tra un numero e il suo successivo è 0.8.

- **GRAPH_SCALE_SetNumDecs**

- Descrizione: imposta il numero di cifre decimali dopo la virgola dell'oggetto scala (**Figura 35** e **Figura 36**).
- Prototipo: `int GRAPH_SCALE_SetNumDecs(GRAPH_SCALE_Handle hScaleObj, int NumDecs);`
- Parametri:
 - a) `hScaleObj`: struttura che gestisce l'oggetto scala.
 - b) `NumDecs`: numero di cifre decimali dopo la virgola.
- Valore di ritorno: numero di cifre decimali dopo la virgola poste in precedenza.

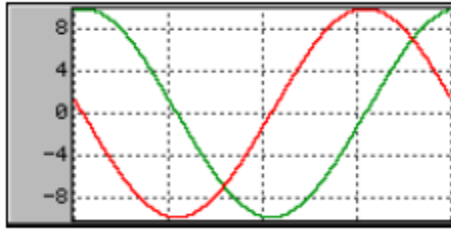


Figura 35: Prima chiamare la funzione `GRAPH_SCAL_SetNumDecs` non sono impostate cifre decimali dopo la virgola dell'oggetto scala.

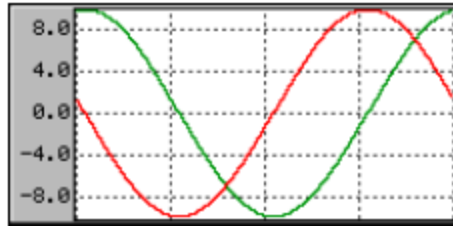


Figura 36: Dopo aver chiamato la funzione `GRAPH_SCALE_SetNumDecs` è impostata una cifra decimale dopo la virgola dell'oggetto scala.

2.6 Rotazione dello schermo

Per quanto riguarda questo aspetto è necessario impostare dei driver opportuni in fase di configurazione iniziale del sistema. E' possibile definire tre tipi di driver

- a) `GUIDRV_LIN_OSX_32`: schermo orizzontale da sinistra verso destra.
- b) `GUIDRV_LIN_OSY_32`: schermo orizzontale da destra verso sinistra.
- c) `GUIDRV_LIN_OS_32`: schermo verticale.

Per modificare i driver è necessario modificare la variabile `DISPLAY_DRIVER_0` definita nel file `LCDConf_stm32f429i_disco_MB1075.c` e attribuirle uno dei tre valori definiti in precedenza. In **Figura 37** e **Figura 38** è possibile vedere due modalità di rappresentazione:

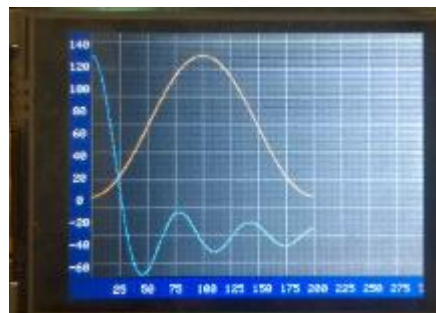


Figura 37: Visualizzazione orizzontale con il driver `GUIDRV_LIN_OSX_32`

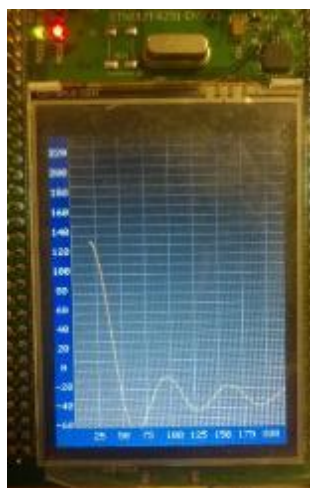


Figura 38: Visualizzazione verticale con il driver GUIDRV_LIN_OS_32

3 Progettazione di un oscilloscopio digitale

3.1 Che cos'è l'oscilloscopio



Figura 39: Oscilloscopio digitale

L'oscilloscopio (**Figura 39**) è uno strumento di misura elettronico che consente di visualizzare, su un grafico bidimensionale, l'andamento temporale dei segnali elettrici e di misurare tensioni, correnti, potenze ed energie elettriche. L'asse orizzontale del grafico solitamente rappresenta il tempo, rendendo l'oscilloscopio adatto ad analizzare grandezze periodiche. L'asse verticale rappresenta la tensione.

Un oscilloscopio classico è normalmente costituito da una scatola rettangolare su cui sono presenti uno schermo e numerose manopole e pulsanti di comando. Allo schermo è sovrapposto un reticolo allo scopo di favorire la lettura dei dati. Ogni intervallo del reticolo è chiamato *divisione*: sull'asse orizzontale le divisioni sono solitamente 10, sull'asse verticale variano da 6 in su, dipendentemente dalla geometria dello schermo. Questa modalità costruttiva è rimasta invariata per decenni fino all'avvento degli oscilloscopi digitali dotati di microprocessore, e di conseguenza in grado di poter installare a bordo veri e propri sistemi operativi i quali, uniti alla tecnologia degli schermi sensibili al tocco, hanno cambiato radicalmente l'aspetto e l'operatività di questo strumento. Il segnale da misurare viene introdotto attraverso un apposito connettore, solitamente di tipo coassiale *BNC* (*Bayonet Neill Concelman*) adatto per frequenze relativamente basse, o tipo N, utilizzato quando le frequenze in gioco sono molto elevate. Al connettore d'ingresso possono essere collegate le sonde, accessori particolari usati per prelevare segnali dai circuiti studiati. In modalità semplice, un punto luminoso percorre lo schermo da sinistra a destra a velocità costante, ridisegnando ripetutamente una linea orizzontale. La velocità di scansione è selezionabile per mezzo di una manopola presente sul pannello, la quale comanda il circuito chiamato *base dei tempi*: questo circuito genera precisi intervalli di tempo, che possono spaziare da pochi secondi a qualche nanosecondo; i valori, espressi in unità di tempo per divisione, sono riportati sulla manopola. In assenza di segnale, la traccia è solitamente al centro dello schermo, e l'applicazione di un segnale all'ingresso, provoca la deflessione verso l'alto o verso il basso, in funzione della polarità del segnale. La scala verticale è espressa in volt per divisione e può essere regolata da decine a millesimi di volt (sono comunque disponibili modelli in grado di effettuare misure dell'ordine dei microvolt). L'altezza iniziale del grafico (*offset*) può comunque essere decisa dall'utente, così come è possibile escludere la componente in corrente continua presente nel segnale in esame, nonché scegliere l'impedenza di ingresso. In questo modo si ottiene la visualizzazione di un grafico di tensione in funzione del tempo. Se il segnale è periodico, è possibile ottenere una traccia stabile regolando la base dei tempi in modo da coincidere con la frequenza del segnale o un suo sottomultiplo. Per esempio, se si ha in ingresso un segnale sinusoidale a 50 Hz, si può regolare la base dei tempi in modo che una scansione orizzontale avvenga in 20 millisecondi. Questo sistema è chiamato *sweep continuo* (in italiano *spazzata continua*, nel senso di continuità di una funzione). L'oscillatore della base dei tempi, non essendo sincronizzato con il segnale in analisi, impedisce di avere la traccia del segnale stabile e ferma, perciò la si vedrà fluttuare sullo schermo più o meno velocemente da destra a sinistra o viceversa. Per ottenere una traccia stabile gli oscilloscopi moderni dispongono di una funzione chiamata *trigger* (innesco): questo circuito, inventato nel 1946 dai due fondatori della società Tektronix, fa partire la scansione solo in corrispondenza del verificarsi di un evento sul segnale in ingresso, per esempio il superamento di una soglia di tensione positiva o negativa. Dopo avere completato la scansione da sinistra a destra, l'oscilloscopio rimane in attesa di un nuovo evento. In questo modo la visualizzazione rimane sincronizzata al segnale e la traccia è perfettamente stabile. La soglia di sensibilità del trigger, così come altri parametri, è regolabile dall'utente. Il circuito del trigger può essere configurato per mostrare una sola scansione di un segnale non periodico, come un singolo impulso o sequenze di impulsi non

ripetitivi. È possibile introdurre un ritardo tra l'evento e l'inizio della visualizzazione, in modo da analizzare parti del segnale che altrimenti sarebbero fuori dal campo di visualizzazione.

Le modalità di trigger normalmente presenti sono:

- trigger esterno: con un segnale applicato ad un apposito ingresso ed indipendente da quello analizzato.
- trigger a soglia: basato sul superamento di un livello prefissato, in salita oppure in discesa.
- trigger video: è uno speciale circuito che estrae i riferimenti di sincronismo di riga o di quadro dal segnale televisivo. Utile per lavorare con segnali video.
- trigger sulla rete elettrica: disponibile negli oscilloscopi alimentati dalla rete in corrente alternata. È utile in elettrotecnica.

Può però capitare che il segnale non soddisfi mai le condizioni di trigger e lo schermo rimanga buio, quindi senza informazione su ciò che sta accadendo. In tal caso è possibile usare l'*auto-trigger*, regolabile generalmente con il comando *GATE*, il quale può essere impostato a:

- normal: in assenza di trigger, lo schermo resta nero.
- auto: trascorso un certo periodo di tempo dall'ultima visualizzazione, la traccia parte senza sincronizzazione.
- single (detto anche *one-shot* o *single sweep*): ogni singolo *sweep* dev'essere abilitato mediante un comando esterno (tasto o segnale).

Un altro controllo, per quanto riguarda i trigger, è l'*hold-off*, ovvero il “trattenimento” del trigger. Esso permette di fissare un tempo, successivo allo spazzolamento, in cui il verificarsi della condizione di trigger è ignorata. Risulta utile in particolari situazioni, ad esempio quando si vuole visualizzare una traccia in cui il trigger viene soddisfatto due o più volte, in un unico sweep. Molti oscilloscopi permettono di escludere la base dei tempi e fornire all'asse orizzontale un segnale esterno: è la modalità X-Y, utile per visualizzare le relazioni di fase tra due segnali, per esempio in ambito radiotelevisivo. Applicando due segnali sinusoidali in rapporto armonico di frequenza agli ingressi, sullo schermo vengono visualizzate particolari figure, chiamate figure di Lissajous, con le quali è possibile conoscere i rapporti di fase e di frequenza tra essi. Alcuni oscilloscopi hanno sullo schermo dei cursori che possono essere spostati ed utilizzati per misurare con precisione intervalli di tempo o differenze di potenziale. Molti oscilloscopi hanno due o più ingressi verticali, consentendo di visualizzare diversi segnali contemporaneamente. Le regolazioni verticali sono separate mentre la base dei tempi ed il trigger è comune. Per visualizzare diversi segnali contemporaneamente, da diversi canali in ingresso, si possono adottare principalmente due soluzioni:

1. si costruisce un oscilloscopio con la possibilità di disegnare contemporaneamente due o più tracce.
2. si utilizza lo stesso punto luminoso per visualizzare le due tracce (o più di due).

Il secondo caso può essere realizzato in modalità *alternate* o in modalità *chopped*.

Supponendo che vi siano due tracce A e B, con la modalità *alternate* viene visualizzata alternativamente una scansione la traccia A e l'altra scansione la traccia B. Tale modo è utile per visualizzare segnali a frequenza elevata, poiché più è alta la frequenza e meno si nota l'alternanza delle tracce. Con una sufficientemente alta ci si potrebbe illudere che le due tracce siano visualizzate esattamente allo stesso istante temporale.

Con la modalità *chopped* sono invece visualizzate alternativamente piccolissime porzioni dei due segnali A e B. Essa è utile per visualizzare segnali a bassa frequenza, sebbene i due segnali non vengano visualizzati sullo schermo in modo continuo. Questo perché essendo bassa la frequenza, la modalità *alternate* le visualizzerebbe troppo lentamente per far sì che sembrino sullo stesso schermo contemporaneamente. Esistono modelli con doppio trigger, che permettono di visualizzare un segnale con una base dei tempi diversa. In questo modo è possibile avere la modalità *zoom*, in cui una porzione del segnale mostrato su una traccia può essere mostrato espanso sull'altra traccia. Per analizzare eventi non ripetitivi alcuni oscilloscopi sono dotati di *memoria di traccia*, un sistema che mantiene visualizzata l'ultima traccia apparsa. In alcuni modelli digitali la scansione può durare ore, e la traccia visualizzata scorre sullo schermo da destra a sinistra come avverrebbe in un registratore su striscia di carta. Normalmente ogni oscilloscopio è dotato di un circuito chiamato calibratore, il quale genera costantemente un segnale, costituito da un'onda quadra di ampiezza e frequenza nota. Collegando l'ingresso di misura all'uscita del calibratore è possibile

controllare il funzionamento e fare pratica con lo strumento, ma soprattutto eseguire saltuariamente l'operazione di *compensazione* delle sonde (che sono dotate di un trimmer capacitivo di compensazione) [5].

3.2 Architettura e soluzioni progettuali

La soluzione sviluppata permette l'acquisizione di un valore di tensione nel tempo a partire da un certo istante (reset del sistema), e la successiva visualizzazione sullo schermo. In questa fase preliminare non è dunque prevista la gestione di un segnale periodico o del *trigger*. Tuttavia, il progetto realizzato è alla base dello sviluppo di sistemi più complessi. Inoltre, l'intervallo di tensioni utilizzabili è fissato dalla tensione di alimentazione del microcontrollore, non avendo previsto un circuito per il condizionamento del segnale in ingresso.

Il sistema è basato sull'uso di un microcontrollore e le relative periferiche disponibili: un timer (*TIM*) ha il compito di sincronizzare l'acquisizione dei dati da parte dell'*ADC*, che converte la tensione in ingresso in un valore digitale. Per ogni acquisizione l'*ADC* comunica con il *DMA* il quale trasferisce i dati in memoria.

I valori dei punti acquisiti, memorizzati in un vettore, sono quindi elaborati via software. Prima di tutto vengono calcolati il massimo e il minimo dei campioni, che saranno utili per definire la scala verticale e la posizione dell'origine dell'asse delle ordinate. Successivamente, viene effettuato un riscalamento del vettore in modo da adattare il numero dei punti acquisiti alla dimensione orizzontale disponibile (in pixel).

Infine, i valori acquisiti sono visualizzati sullo schermo. Lo schema in **Figura 40** sintetizza l'architettura complessiva del sistema.

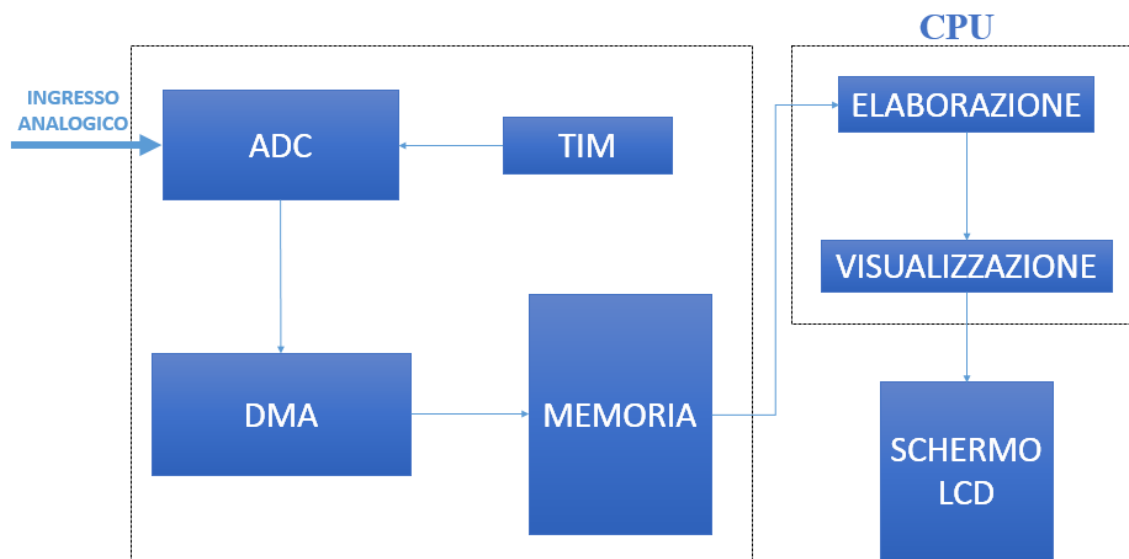


Figura 40: Diagramma a blocchi dell'architettura di progetto

La **Figura 41** mostra lo schema a blocchi generale del firmware: all'uscita dal reset, si configurano le periferiche necessarie e poi si lancia l'acquisizione dei dati in ingresso, a cui seguono elaborazione e visualizzazione.

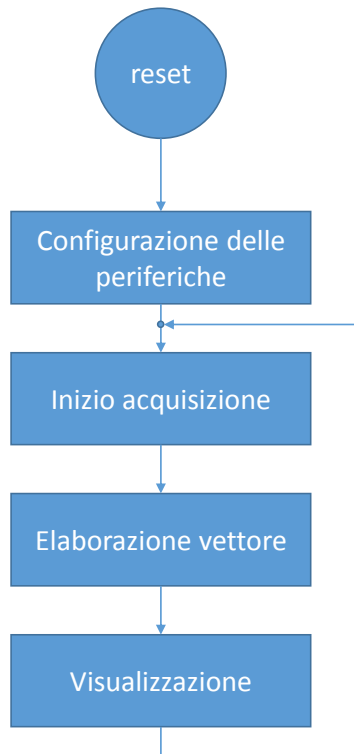
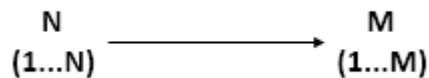


Figura 41: Diagramma a blocchi del firmware

3.2.1 Riscalamento orizzontale

In questo paragrafo è descritto l'algoritmo per adattare il data object alle dimensioni di uno schermo LCD. L'algoritmo gestisce due casi:

- **Primo caso:** situazione in cui il numero di dati è maggiore di DIM_GRAPH_X (massimo numero di punti rappresentabili sullo schermo LCD). Questo algoritmo modifica le dimensioni di un vettore di N punti attraverso un'elaborazione matematica che permette di ottenere un nuovo vettore di M punti, con $M < N$. In pratica, il valore del j -esimo punto del vettore risultante è pari alla media di $\frac{N}{M}$ valori del vettore iniziale.



Di seguito è riportato il codice:

```

for (i=0, j=0; i<M; i++)
{
    s=0;
    j0 = j;
    for ( ; j*M<(i+1)*N; j++)
        s+=(vett_orig[j]);
    vett_ris[i] = s/(j - j0);
}
  
```

- **Secondo caso:** in questo caso la dimensione del vettore è minore o uguale a *DIM_GRAPH_X*; pertanto l'algoritmo si limita a copiare i punti contenuti in *vett_orig* nel vettore *vett_ris* senza effettuare alcun tipo di elaborazione matematica.

3.2.2 Riscaldamento verticale

Terminato il riscaldamento orizzontale è necessario manipolare nuovamente i dati in modo tale da adattare il vettore di punti alle dimensioni dell'asse verticale dello schermo LCD. Questo è possibile attraverso il calcolo di un semplice rapporto:

$$k = \frac{f_{max} - f_{min}}{RMAX - RMIN}$$

I parametri utilizzati per il calcolo del rapporto di scala *k* sono i seguenti:

- *float fmax*: massimo del vettore di *punti* (che è calcolato con la funzione *CalcoloMassimo*).
- *float fmin*: minimo del vettore di *punti* (che è calcolato con la funzione *CalcoloMinimo*).
- *RMAX*: valore del punto massimo in pixel sulle ordinate dello schermo (in questo caso 298).
- *RMIN*: valore del punto minimo in pixel sulle ordinate dello schermo (in questo caso 0).

I valori del vettore di punti sono divisi per la costante *k* in modo da adattarsi all'ordinata dello schermo LCD in **Figura 42**:



Figura 42: Esempio delle dimensioni in pixel di uno schermo LCD

Si consideri il seguente esempio: si supponga di voler rappresentare una generica funzione i cui punti siano compresi tra 3 e 10. È necessario, quindi, modificare il vettore di punti in modo che sia compreso tra 0 e 227. Il problema si risolve nel seguente modo:

$$k = \frac{f_{max} - f_{min}}{RMAX - RMIN} = \frac{10 - 3}{227 - 0}$$

Con un semplice ciclo *for* si divide il punto *i*-esimo del vettore per la costante *k* (**Figura 43**). Inoltre è necessario traslare l'intervallo dei valori da rappresentare di una quantità pari a $\frac{-f_{min}}{k} + RMIN$. Quest'ultimo passaggio è effettuato utilizzando opportune funzioni della libreria, come riportato nel paragrafo 4.2.

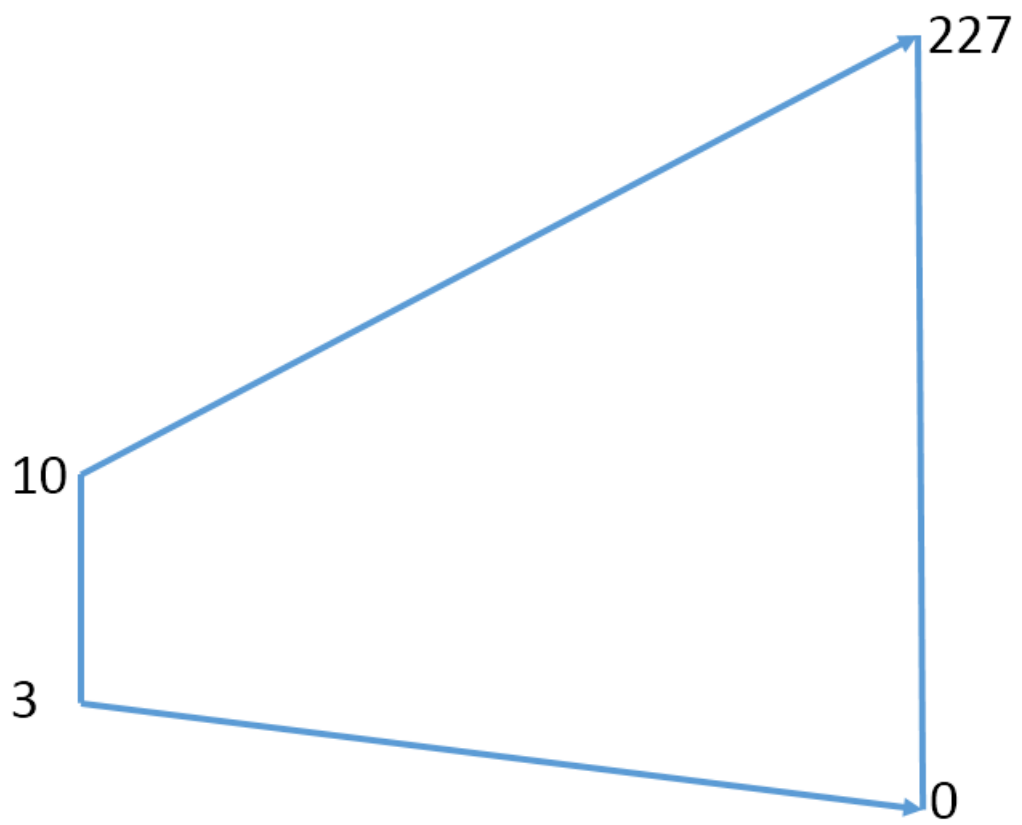


Figura 43:
Traslazione dei punti

4 Realizzazione del progetto

In questo capitolo si discute delle modalità di realizzazione del progetto. Il paragrafo 4.1 descrive come configurare il graph widget adoperando le API della libreria *emWin* trattate nel capitolo 2. Il paragrafo 4.2 analizza l'algoritmo adoperato per adattare il data object alle dimensioni dello schermo LCD (zoom). Infine il paragrafo 4.3 parla della configurazione dell'hardware del sistema di acquisizione. Il paragrafo 4.4 riporta i risultati ottenuti dalla validazione sperimentale.

4.1 Configurazione della libreria grafica

In questo paragrafo sono descritte le API della libreria *emWin* adoperate per creare l'interfaccia grafica di un graph widget. Questa fase del lavoro è realizzata nella funzione *_GraphDemo*, sviluppata per gestire la visualizzazione di un vettore di dati. Le sue caratteristiche sono le seguenti:

- Descrizione: funzione che gestisce la grafica del graph widget (per mezzo delle API della libreria *emWin*) e il data object che deve essere acquisito e visualizzato sullo schermo LCD (utilizzando la funzione *_ShowGraph* descritta nel paragrafo 4.2).
- Prototipo: *static void _GraphDemo(void)*.

All'interno di *_GraphDemo* è chiamata la funzione *WIDGET_SetDefaultEffect* che permette di definire l'effetto grafico desiderato. Per la creazione del graph widget, invece, è assegnata alla struttura *hGraph* la funzione *GRAPH_CreateEx*. Successivamente sono settati i bordi del data area: per fare questo è chiamata la funzione *GRAPH_SetBorder* che permette di definire le dimensioni del bordo sinistro, destro, alto e basso (le dimensioni dei bordi alto e destro sono poste uguali a 0). La libreria permette anche di gestire i colori dei diversi oggetti che formano il graph widget; per questo motivo è dichiarata la funzione *GRAPH_SetColor*. Gli oggetti colorati sono lo sfondo, il border area, la griglia e la sottile linea di frame. Un altro oggetto fondamentale della grafica è la griglia. Quest'ultima è aggiunta al data area per mezzo della funzione *GRAPH_SetGridVis*. Sono definite anche le distanze orizzontali e verticali tra una linea e l'altra della griglia per mezzo delle funzioni *GRAPH_SetGridDistX* e *GRAPH_SetGridDistY*. La funzione *GRAPH_SetGridOffY*, invece, permette di impostare un offset per le griglie. Per la creazione dell'ascissa e ordinata sono adoperate le strutture dati *hScaleH* (per l'ascissa) e *hScaleV* (per l'ordinata) alle quali è assegnata la funzione *GRAPH_SCALE_Create*. Le loro dimensioni sono impostate per mezzo delle funzioni *GRAPH_SCALE_SetPos* e aggiunte allo schermo tramite la *GRAPH_AttachScale*. Infine è chiamata la funzione *GRAPH_DATA_YT_Create* che è assegnata alla struttura *hdata[0]* e che permette di gestire la fase di acquisizione esposta nel paragrafo 4.2. Di seguito è mostrato il codice realizzato:

```
WIDGET_EFFECT *pEffectOld; /*puntatore alla struttura che gestisce gli effetti dell'interfaccia grafica*/

GRAPH_Handle hGraph; /*struttura che gestisce la configurazione del "graph object"*/

GRAPH_DATA_Handle hData[MAX_NUM_DATA_OBJ]; /*vettore di struct che gestisce i valori numerici della
funzione rappresentata sul graph widget. In questo caso MAX_NUM_DATA_OBJ è pari a 0, perché si è deciso di
rappresentare una sola funzione sullo schermo*/

int xSize, ySize; /*dimensioni dello schermo LCD*/

GRAPH_SCALE_Handle _hScaleH; /*struttura che gestisce la scala orizzontale del data area (ascissa). */
GRAPH_SCALE_Handle _hScaleV; /*struttura che gestisce la scala verticale del data area (ordinata). */

pEffectOld = WIDGET_SetDefaultEffect(&WIDGET_Effect_Simple); /

WM_SetCallback(WM_HBKWIN, _cbBk);

hGraph = GRAPH_CreateEx(0,0,xSize,ySize, WM_HBKWIN, WM_CF_SHOW | WM_CF_CONST_OUTLINE, 0,
0); /* Come si può notare i primi due valori sono uguali a 0, perché l'obiettivo è quello di rappresentare per intero
il data area sullo schermo LCD senza eventuali cornici esterne.*/

GRAPH_SetBorder(hGraph, BORDER_LEFT, BORDER_TOP, BORDER_RIGHT, BORDER_BOTTOM);
```

```

GRAPH_SetColor(hGraph, COLOR_BK, GRAPH_CI_BK);
GRAPH_SetColor(hGraph, COLOR_BORDER, GRAPH_CI_BORDER);
GRAPH_SetColor(hGraph, COLOR_FRAME, GRAPH_CI_FRAME);
GRAPH_SetColor(hGraph, COLOR_GRID, GRAPH_CI_GRID);
GRAPH_SetGridVis(hGraph, 1);
GRAPH_SetGridDistX(hGraph, GRID_DIST_X);
GRAPH_SetGridDistY(hGraph, GRID_DIST_Y);
_hScaleH = GRAPH_SCALE_Create(BORDER_BOTTOM >> 1, GUI_TA_VCENTER,
GRAPH_SCALE_CF_HORIZONTAL, TICK_DIST_H);
_hScaleV = GRAPH_SCALE_Create(BORDER_LEFT >> 1, GUI_TA_HCENTER,
GRAPH_SCALE_CF_VERTICAL, TICK_DIST_V);
GRAPH_SCALE_SetPos(_hScaleH, Data_ySize + (BORDER_BOTTOM/2));
GRAPH_SCALE_SetOff(_hScaleH, -5);
GRAPH_AttachScale(hGraph, _hScaleH);
GRAPH_AttachScale(hGraph, _hScaleV);
hData[0] = GRAPH_DATA_YT_Create(_aColorData[0], 1000, 0, 0);

```

4.2 Realizzazione del sistema di acquisizione

Terminata la fase di configurazione inizia una seconda fase che consiste nella realizzazione del sistema di acquisizione. Anche in questo caso è fondamentale adoperare *emWin*, perché contiene delle API opportune per la rappresentazione delle funzioni sullo schermo LCD. Tutte le operazioni necessarie per la configurazione del sistema di acquisizione sono contenute nella funzione *_ShowGraph*:

- Descrizione: funzione che si occupa della gestione del sistema di acquisizione.
- Prototipo: *static void _ShowGraph(GRAPH_Handle hGraph, GRAPH_DATA_Handle hData[], int DataCount, void (* pfAddData)(GRAPH_DATA_Handle hData, int DataID))*.
- Parametri:
 - a) *hGraph*: struttura che gestisce la configurazione del graph widget.
 - b) *hData*: vettore di struct che si occupa della gestione dei dati numerici della funzione che si vuole rappresentare sul data area.
 - c) *DataCount*: numero di forme d'onda.
 - d) *void (* pfAddData)*: puntatore alla funzione che si occupa dell'acquisizione dei valori numerici relativi al data object che si vuole rappresentare sul data area.

La prima funzione dichiarata è *GRAPH_DATA_YT_SetAlign* che imposta l'allineamento della curva sul data area e permette di sfruttare al meglio le dimensioni dello schermo. Successivamente è chiamata la funzione *GRAPH_AttachData* che lega il data object al data area. Il vettore di punti, ovviamente, non è adattato alle dimensioni dello schermo; per questo motivo si è realizzato un riscalamento orizzontale (paragrafo 3.2.1) e verticale (paragrafo 3.2.2).

Terminata la fase di manipolazione dei dati numerici, si è stabilita una relazione numerica tra la funzione e l'asse delle ordinate del data area. Per risolvere questo problema sono utilizzate alcune funzioni della libreria *emWin*:

- *GRAPH_DATA_YT_SetOffY*: l'offset che permette di visualizzare l'intero insieme di dati è $\frac{-f_{min}}{k} + RMIN$.
- *GRAPH_SCALE_SetFactor*: il fattore usato per calcolare i numeri da disegnare sullo schermo è pari a *k*.

- *GRAPH_SCALE_SetOff*: il fattore usato per disegnare la scala e posizionarla verticalmente in maniera opportuna è $\frac{-f_{min}}{k} + RMIN$ (in pratica grazie a questa funzione l'origine degli assi coincide con il minimo della funzione).
- *GRAPH_SetGridOffY*: l'offset usato è $\frac{-f_{min}}{k} + RMIN$.
- *GRAPH_SCALE_SetTickDist*: la distanza tra un numero e l'altro sulla scala delle ordinate è 50.
- *GRAPH_SCALE_SetNumDecs*: in questa funzione è stata inserita 1 cifra decimale dopo la virgola. Questa viene chiamata soltanto se la differenza in valore assoluto tra f_{max} ed f_{min} risulta minore di 10. Questa condizione è stata introdotta per rappresentare in maniera adeguata i numeri sulla scala numerica, mantenendo invariata la dimensione del bordo che contiene la scala.

Per la rappresentazione del vettore di punti sullo schermo *LCD* si è creata un'opportuna funzione:

```
static void AggiungiDati(GRAPH_DATA_Handle hData, int DataID)
```

Quest'ultima carica un dato alla volta e per questo motivo è inserita all'interno di un ciclo *while* che itera fino a quando il numero di cicli è inferiore a *DIM_GRAPH_X* (numero massimo di punti rappresentabili sul data area).

All'interno di *AggiungiDati* è presente un'ulteriore funzione che si occupa di aggiungere il dato sullo schermo:

```
void GRAPH_DATA_YT_AddValue(GRAPH_DATA_Handle hDataObj, I16 Value);
```

4.3 Configurazione dell'architettura del sistema di acquisizione

Questo paragrafo descrive la metodologia utilizzata per la configurazione del timer, del *DMA* e dell'*ADC* che consentono l'acquisizione dei valori di tensione di un ingresso analogico e la conversione in un vettore di valori numerici.

Prima di tutto occorre configurare il *timer* che è in grado di sincronizzare le operazioni dell'*ADC*. Ogni volta che l'*ADC* campiona e converte un valore di tensione, richiede l'intervento di un canale *DMA* che, a sua volta, trasferisce in memoria l'informazione. Si è supposto di acquisire una sequenza di 1000 punti ad una frequenza di 200 Hz. Nei seguenti paragrafi sono presenti le configurazioni del *timer*, *ADC* e *DMA* in linguaggio di programmazione C.

4.3.1 Configurazione del timer

I principali parametri per definire il funzionamento del timer sono quelli del *prescaler* e del *periodo*, che permettono di definire la temporizzazione del sistema. In questo caso, il clock di sistema ha una frequenza di 18MHz, che è divisa per 900 (*prescaler*) e poi per 100 (*counter*) per arrivare alla frequenza desiderata di 200Hz.

```
void Init_TIM2( TIM_HandleTypeDef *myTime){
    __HAL_RCC_TIM2_CLK_ENABLE();

    myTime->Instance = TIM2;

    myTime->Init.Prescaler = 900 - 1; // timer running at 200 Hz
    myTime->Init.Period = 100 - 1;
    myTime->Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    myTime->Init.CounterMode = TIM_COUNTERMODE_UP;
    HAL_TIM_Base_Init(myTime);
}
```

4.3.2 Configurazione dell'ADC

I principali parametri di configurazione dell'*ADC* sono quelli del *GPIOC* che permette di abilitare il pin 3 per l'acquisizione del vettore di punti, del trigger esterno dell'*ADC* e del trigger esterno relativo al timer *TIM2*.

```

ADC_HandleTypeDef Init_Adc(){
    ADC_HandleTypeDef myAdc;
    ADC_ChannelConfTypeDef sConfig;
    GPIO_InitTypeDef GPIO_Init;
    __HAL_RCC_ADC2_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_AHB1_RELEASE_RESET();
    HAL_GPIO_DeInit(GPIOC, GPIO_PIN_3);
    GPIO_Init.Pin = GPIO_PIN_3;
    GPIO_Init.Mode = GPIO_MODE_ANALOG;
    GPIO_Init.Speed = GPIO_SPEED_FAST;
    GPIO_Init.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOC, &GPIO_Init);
    myAdc.Instance = ADC2;
    HAL_ADC_DeInit(&myAdc);
    myAdc.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
    myAdc.Init.Resolution = ADC_RESOLUTION_12B;
    myAdc.Init.ScanConvMode = DISABLE;
    myAdc.Init.ContinuousConvMode = DISABLE;
    myAdc.Init.DiscontinuousConvMode = DISABLE;
    myAdc.Init.NbrOfDiscConversion = 0;
    myAdc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_RISING;
    myAdc.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T2_TRGO;
    myAdc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    myAdc.Init.NbrOfConversion = 1;
    myAdc.Init.DMAContinuousRequests = DISABLE;
    myAdc.Init.EOCSelection = DISABLE;
    HAL_ADC_Init(&myAdc);
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    sConfig.Offset = 0;
    sConfig.Channel = ADC_CHANNEL_13;
    __HAL_ADC_ENABLE(&myAdc);
    if(HAL_ADC_ConfigChannel(&myAdc, &sConfig)!=HAL_OK)

```

```

        Errore();

    return myAdc;
}

```

4.3.3 Configurazione del DMA

I principali parametri per la configurazione del DMA sono quelli relativi alla scelta del canale del DMA (*DMA_CHANNEL_1*), alla direzione del trasferimento che in questo caso va da periferica a memoria (*DMA_PERIPH_TO_MEMORY*) e alla dimensione dei dati in periferica e memoria.

```

DMA_HandleTypeDef Init_Dma_Adc(ADC_HandleTypeDef *myAdc){
    static DMA_HandleTypeDef hdma_adc;

    __HAL_RCC_DMA2_CLK_ENABLE();

    HAL_DMA_DeInit(&hdma_adc);

    hdma_adc.Instance = DMA2_Stream2;

    hdma_adc.Init.Channel = DMA_CHANNEL_1;
    hdma_adc.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD;
    hdma_adc.Init.Mode = DMA_CIRCULAR;
    hdma_adc.Init.Priority = DMA_PRIORITY_HIGH;
    hdma_adc.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
    hdma_adc.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_HALFFULL;
    hdma_adc.Init.MemBurst = DMA_MBURST_SINGLE;
    hdma_adc.Init.PeriphBurst = DMA_PBURST_SINGLE;

    HAL_DMA_Init(&hdma_adc);

    __HAL_LINKDMA(myAdc, DMA_Handle, hdma_adc);

    return hdma_adc;
}

```

4.4 Risultati

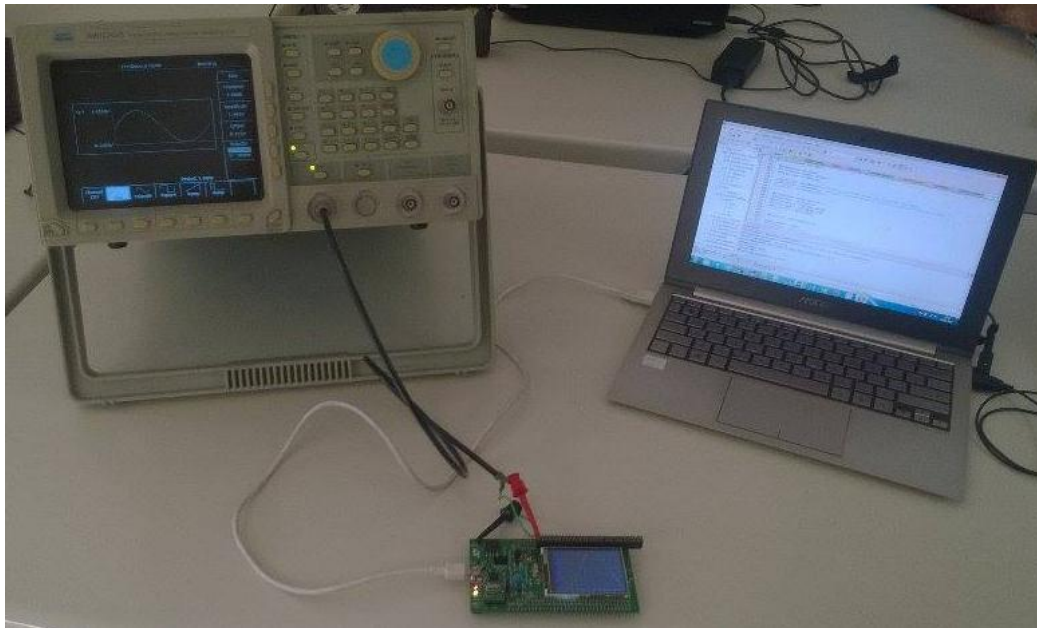


Figura 44: Collegamenti

Per verificare i risultati ottenuti un generatore di funzioni arbitrarie è stato collegato al pin *PC3* della scheda, che a sua volta è collegata al pc attraverso un cavo *USB* (**Figura 44**). Il PC, oltre ad alimentare la scheda, si occupa anche di caricare il firmware in memoria. L'ADC permette di acquisire tensioni comprese tra 0 e la tensione di alimentazione, pari a 2.9 V. Di seguito sono esposti una serie di esempi che permettono di validare il sistema realizzato. I primi esempi (casi 1,2 e 3) permettono di vedere la rappresentazione della funzione seno a frequenze variabili. Dai test risulta che il valore massimo di frequenza tale da poter vedere sullo schermo LCD un data object accettabile è circa 20 Hz (caso 4); infatti impostando una frequenza molto più grande si ha che la funzione seno non è più distinguibile. Gli ultimi due esempi (caso 5 e 6) mostrano l'affidabilità dello scalamento verticale: nel caso 5 è riportato l'esempio della funzione triangolo i cui valori in ampiezza sono compresi tra 1.6V e 2.4V, mentre nel caso 6 è rappresentata l'onda quadra i cui valori in ampiezza sono compresi tra 1.1 e 2.9 V.

- **Caso 1:** il generatore di funzioni è stato impostato in modo da generare una funzione seno con 1 Hz di frequenza, ampiezza 1.450V ed offset 0.725V (**Figura 45**). Questi parametri di configurazione ipotizzano un'impedenza di ingresso del ricevitore pari a 50Ω e generano, con un'impedenza tendente ad infinito, la forma d'onda seno di ampiezza 2.9V, correttamente rappresentata sull'oscilloscopio realizzato (**Figura 46**). Si noti anche la correttezza della scala.

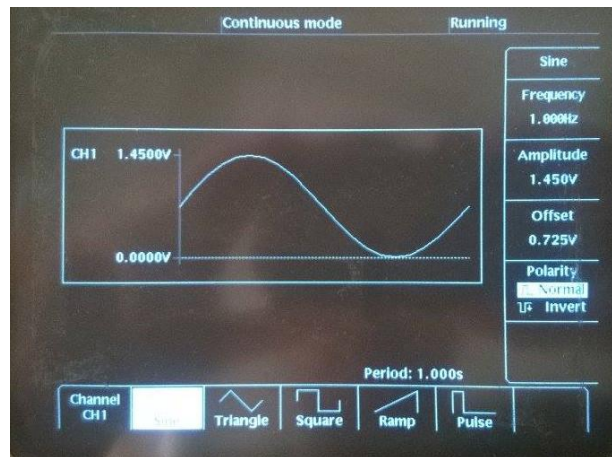


Figura 45: Interfaccia del generatore di funzioni

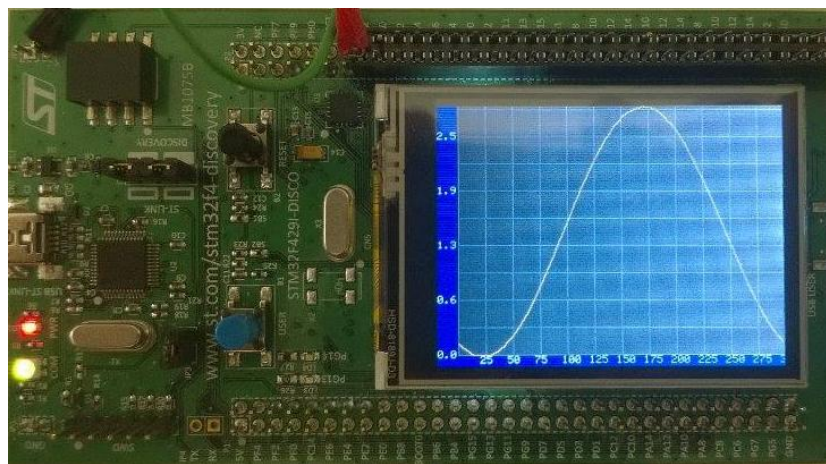


Figura 46: Seno di ampiezza 2.9V con frequenza di 1Hz

- **Caso 2:** il generatore di funzioni è stato impostato in modo da generare una funzione seno con 5 Hz di frequenza, ampiezza 1.450V ed offset 0.725V supponendo una linea a 50Ω (Figura 47 e Figura 48). Si noti anche la correttezza della scala e la presenza del puntatore opportunamente configurato per future applicazioni.

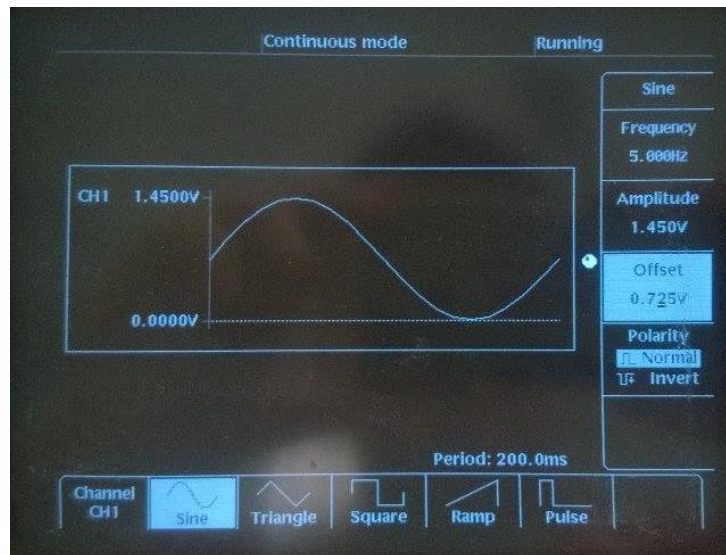


Figura 47: Generatore di funzioni

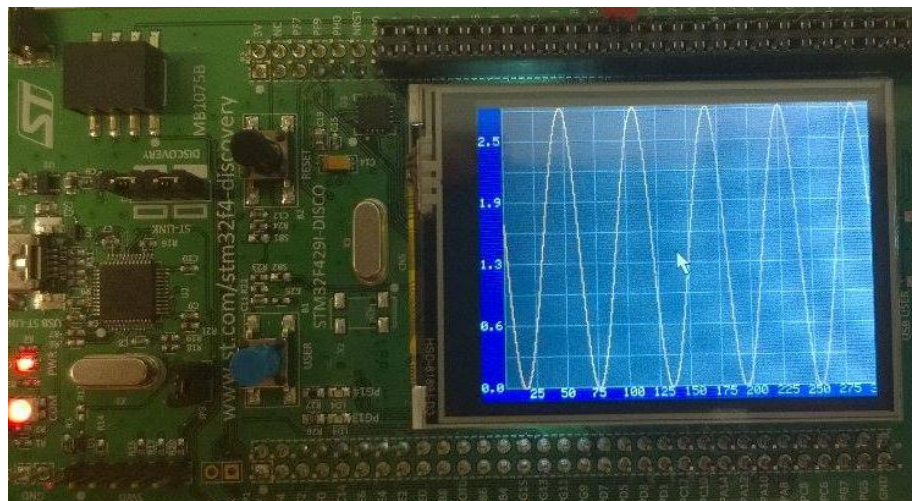


Figura 48: Seno di ampiezza 2.9V con frequenza di 5Hz

- **Caso 3:** Il generatore di funzioni è stato impostato in modo da generare un funzione seno con 20Hz di frequenza, ampiezza 1.450V ed offset 0.725V supponendo una linea a 50Ω (*Figura 49* e *Figura 50*).

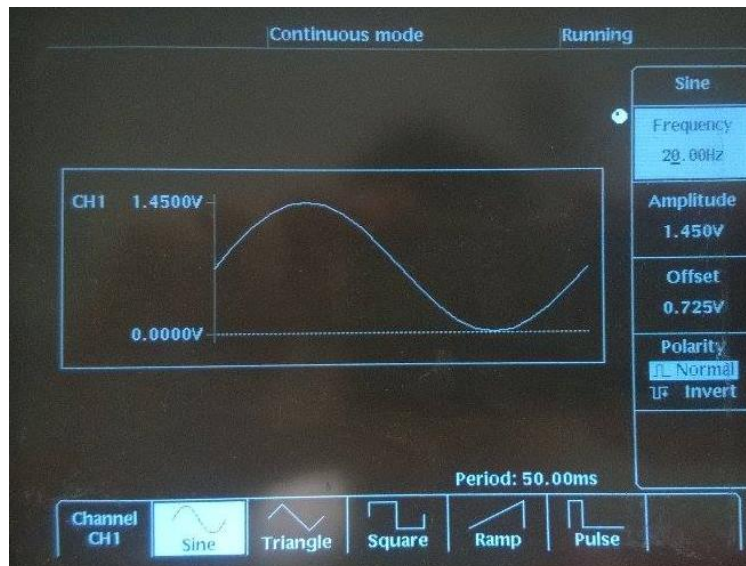


Figura 49: Generatore di funzioni

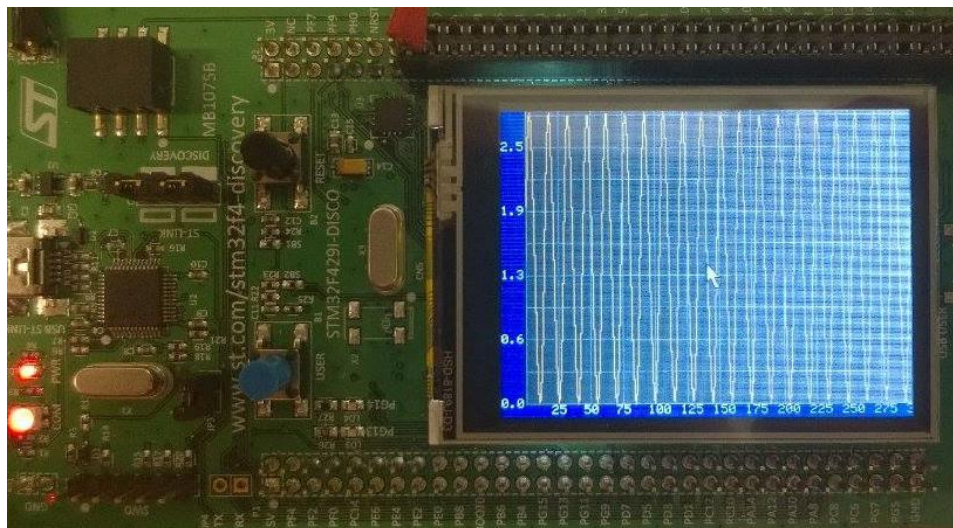


Figura 50: Funzione seno di ampiezza 2.9V con frequenza di 20Hz

- **Caso 4:** il generatore di funzioni è stato impostato in modo da generare una funzione seno con 70Hz di frequenza, ampiezza 1.450V ed offset 0.725V supponendo una linea da 50Ω (*Figura 51* e *Figura 52*). In questo caso la figura non è molto chiara, perché la frequenza è molto alta. La frequenza massima accettabile si aggira attorno ai 20Hz.

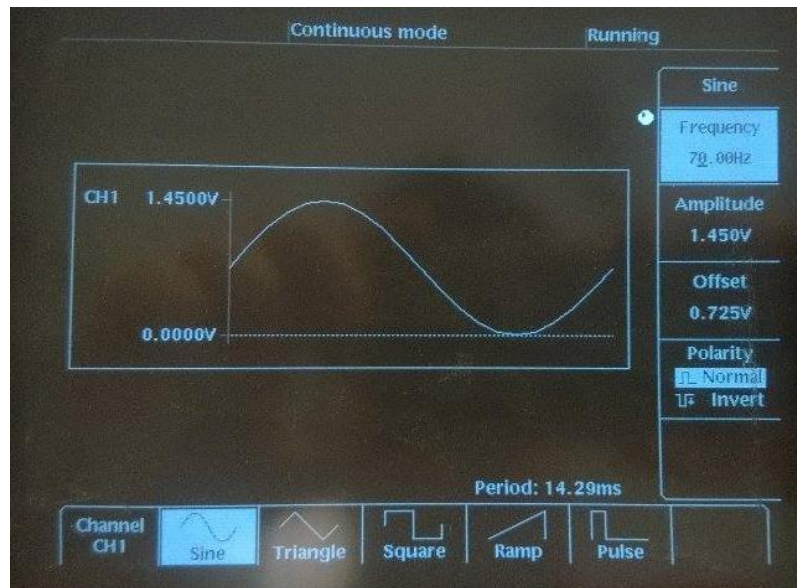


Figura 51: Generatore di funzioni

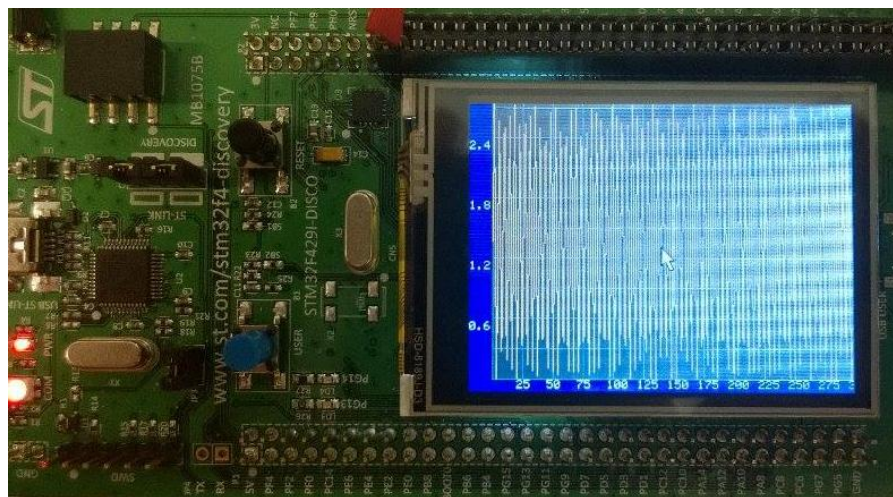


Figura 52: Funzione seno di ampiezza 2.9V con frequenza di 70Hz

- **Caso 5:** il generatore di funzioni è stato impostato in modo da generare una funzione triangolo con 3Hz di frequenza, ampiezza 0.400V ed offset 1V supponendo una linea a 50Ω (*Figura 53 e Figura 54*).

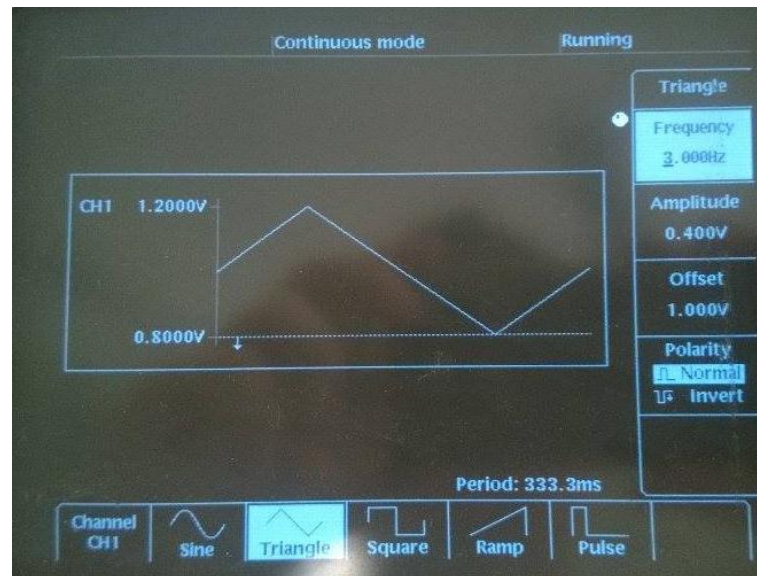


Figura 53: Generatore di funzioni

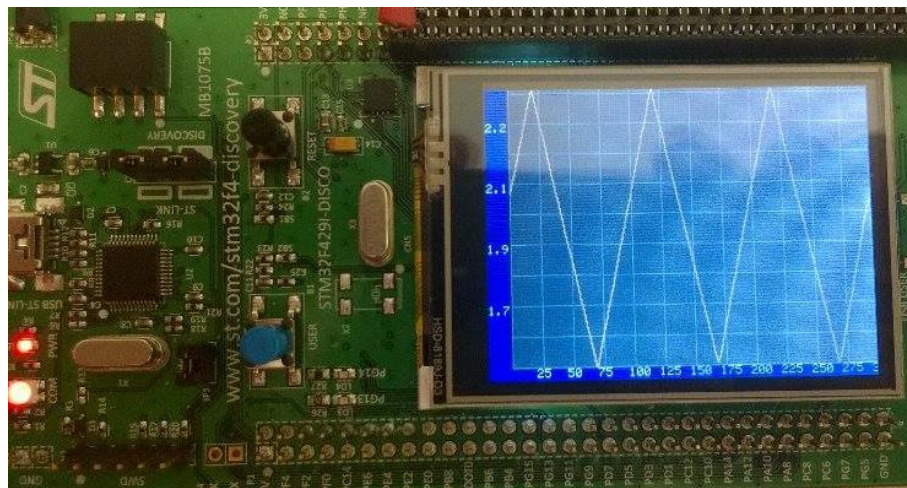


Figura 54: Onda triangolare di ampiezza 0.8V con frequenza di 3Hz

- **Caso 6:** il generatore di funzioni è stato impostato in modo da generare una funzione onda quadra con 4Hz di frequenza, ampiezza 0.900V ed offset 1V supponendo una linea a 50Ω (*Figura 55* e *Figura 56*).

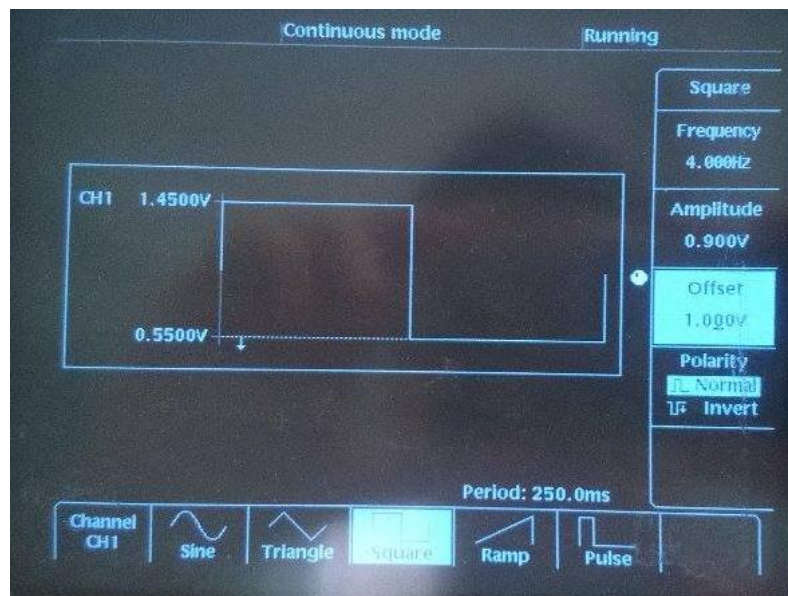


Figura 55: Generatore di funzioni

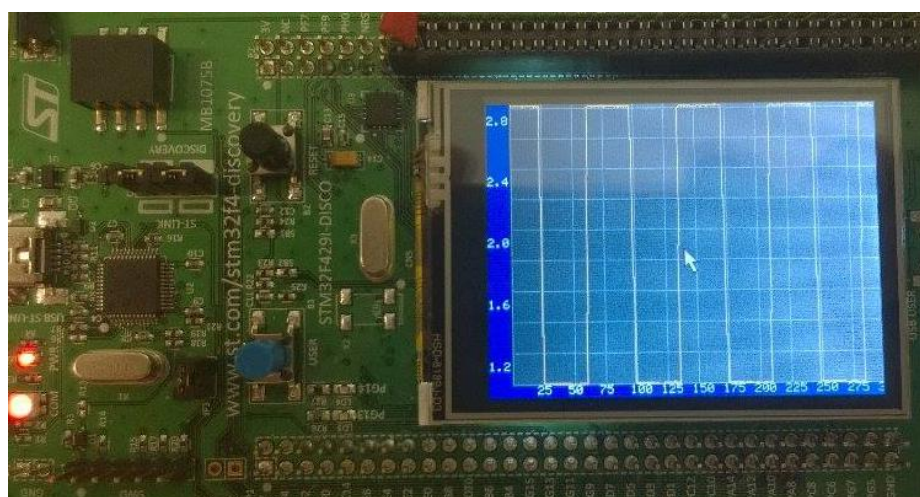


Figura 56: Onda quadra di ampiezza 1.8V con frequenza di 4Hz

5 Conclusioni

È stato realizzato un prototipo di un semplice oscilloscopio, utilizzando una scheda a microcontrollore che acquisisce una forma d'onda per poi rappresentarla su uno schermo *LCD*. Quest'ultimo è montato sulla scheda ed è sicuramente molto utile per vedere in modo immediato e a basso costo l'andamento di una tensione nel tempo.

L'attuale sistema è la versione preliminare del progetto di un oscilloscopio più completo, poiché gli aspetti da migliorare sono ancora tanti: innanzitutto i lavori futuri di maggiore priorità sono quelli che riguardano il trigger, il condizionamento del segnale in ingresso e la gestione dello zoom.

Attualmente è infatti possibile acquisire e visualizzare una forma d'onda non periodica e compresa tra i limiti della tensione di alimentazione del microcontrollore utilizzato, elaborandone i valori di tensione letti in modo tale che possa adattarsi alle dimensioni dello schermo e occuparlo interamente.

La realizzazione del meccanismo di trigger richiede un processo di acquisizione continuo (in un buffer circolare), la definizione di una soglia della tensione in ingresso e la computazione della frequenza con cui il segnale oltrepassa tale soglia, e di conseguenza un'elaborazione più complessa dei campioni acquisiti.

Per la gestione di segnali al di fuori della tensione di alimentazione del microcontrollore, si richiede lo sviluppo di un circuito di condizionamento esterno alla scheda.

Infine, un altro aspetto da considerare è la realizzazione della funzionalità di modifica della visualizzazione (zoom e scorrimento), gestita mediante opportuni tasti funzione virtuali sul touch screen, che permettano all'utente di gestire V/div e s/div .

Il lavoro fin qui svolto, concentrandosi sull'analisi delle periferiche del processore e delle librerie grafiche, fornisce gli strumenti essenziali per future estensioni delle funzionalità del sistema.

Bibliografia

- [1] <http://it.wikipedia.org/wiki/Microcontrollore>
- [2] <http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF259090>
- [3] http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user_manual/DM00105879.pdf
- [4] https://www.segger.com/cms/admin/uploads/productDocs/UM03001_emWin5.pdf
- [5] <http://it.wikipedia.org/wiki/Oscilloscopio>
- [6] <https://www.segger.com/index.html>