STMCAR

Progetto a cura di

Gianluca Artioli e Sukhjinder Singh

Università degli studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Progettazione Elettronica Digitale

A.A. 2016/2017

Indice

1 Introduzione

Il corso Progettazione Elettronica Digitale ci ha permesso di studiare i componenti e le caratteristiche di un microcontrollore.

In particolare abbiamo utilizzato il microcontrollore STM32, primo microcontrollore basato sul nuovo core ARM Cortex-M3 che garantisce nuovi standard per quanto riguarda costi e performance, oltre ad essere in grado di operare a basso consumo.

Lo scopo di questa tesi sarà per l'appunto quella di avere un primo approccio con un microcontrollore, nello specifico l'STM32, capirne i vantaggi e la semplicità di utilizzo. In particolare useremo la board F4 Discovery di cui ne esplicheremo le funzionalità e potenzialità.

Per semplicità abbiamo diviso il tutto in 3 capitoli. Il primo riassume in poche righe l'architettura del progetto e le principali funzionalità della scheda utilizzata, il secondo esamina nel dettaglio il funzionamento globale delle componenti utilizzate e il terzo descrive lo sviluppo e la logica di programmazione.

2 STMCar – Architettura

Il nostro progetto come anticipato in precedenza è basato sul microcontrollore ST32. In particolare abbiamo utilizzato la scheda STM32F407VG, qui sotto in figura.

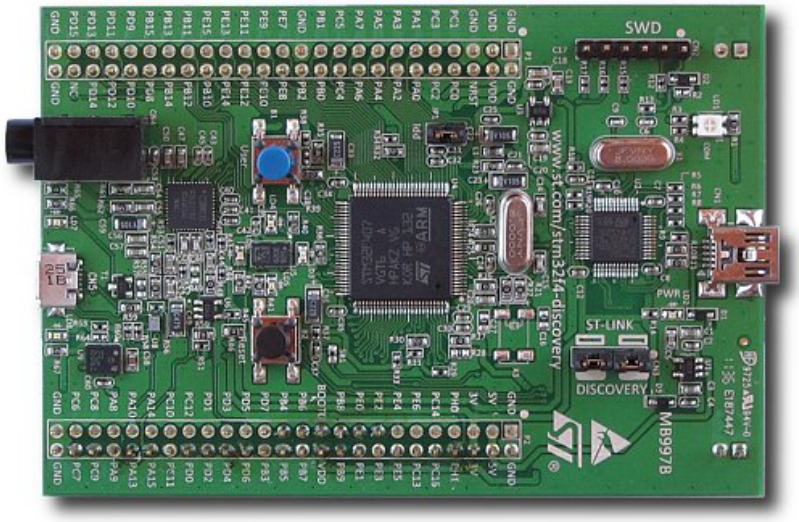


Figure 1 STM32F407

Riportiamo anche le principali caratteristiche della scheda:

Microcontrollore con conduttore Cortex-M4F da 168 MHz STM32F407VGT6 con memoria Flash da 1 MB, RAM da 192 KB
• Accelerometro a 3 assi MEMS LIS302DL o LIS3DSH a bordo scheda
• Microfono digitale omnidirezionale audio MEMS MP45DT02 a bordo scheda
• DAC audio stereo CS43L22 a bordo scheda
• Jack femmina da 3,5 mm per cuffie stereo
• Connettore Micro-AB USB OTG FS
• Connettore USB Mini-B ST-LINK/V2-A
• Connettore SWD
• ST-LINK/V2 a bordo scheda con interruttore di selezione modalità
• 2 x LED di stato: PC a ST-LINK attivo, +3,3 V all'accensione
• 2 x LED OTG USB: VBus presente, VBus sovracorrente
• 4 x LED utente
• Pulsanti di reset e utente
• Connettore di espansione per tutti i GPIO
• Alimentazione: USB VBus o esterna a +5 V c.c.
• Uscita per alimentatore esterno per applicazione: +3 V e +5 V.
• Dimensioni: 97 x 66 mm

Come si può dedurre dal titolo, nel nostro progetto abbiamo costruito una macchinina costituita da 2 motori e pilotata attraverso lo Smartphone che riesce inoltre a evitare eventuali ostacoli che potrebbe trovare sul suo percorso.
La scheda è stata collegata alla Smartphone attraverso il modulo Bluetooth HC-05 mostrato in figura. La comunicazione con la scheda STM avviene invece attraverso il protocollo seriale USART che spiegheremo in seguito.

Figure 2 Bluethoot HC-05

I motori sono stati gestiti attraverso un ponte-h, lo scopo di quest'ultimo è propio quello di permettere l'inversione del verso della corrente in un carico. In questo modo è possibile pilotare il verso di rotazione dei motori e quindi permettere alla macchina di muoversi nello spazio.
Come è possibile notare dalla figura, il modulo L298N presenta 2 pin in entrabi i lati della scheda per collegare i 2 motorini. Sono presenti inoltre 3 pin per l'alimentazione (12v,5v,GND) e 6 pin per gestire la rotazione e l'abilitazione dei motori. Come vedremo in seguito attraverso una particolare funzionalità della scheda STM32 saraà possibile anche settare la velocità di rotazione ai motori.

Figure 3 Ponte-h L298N

Il sensore utilizzato per calcolare la distanza da eventuali ostacoli è il celebre HCSR04, gia utilizzato in moltissimi progetti soprattutto su micocontrollori ATMEL (Arduino).
Il funzionamento del sensore è abbastanza sempice come spiegato anche dal datasheet. Si invia un segnale di trigger per 10us, il modulo invierà 8 cycle burst di ultrasuoni a 40 Khz e abilitarà il pin echo. Successivamente è possibile calcolare la distanza confrontando il tempo trascorso tra l'invio del segnale di trigger e l'abilitazione del pin di echo.

Figure 4 HCSR04

2.1 Struttura globale del progetto

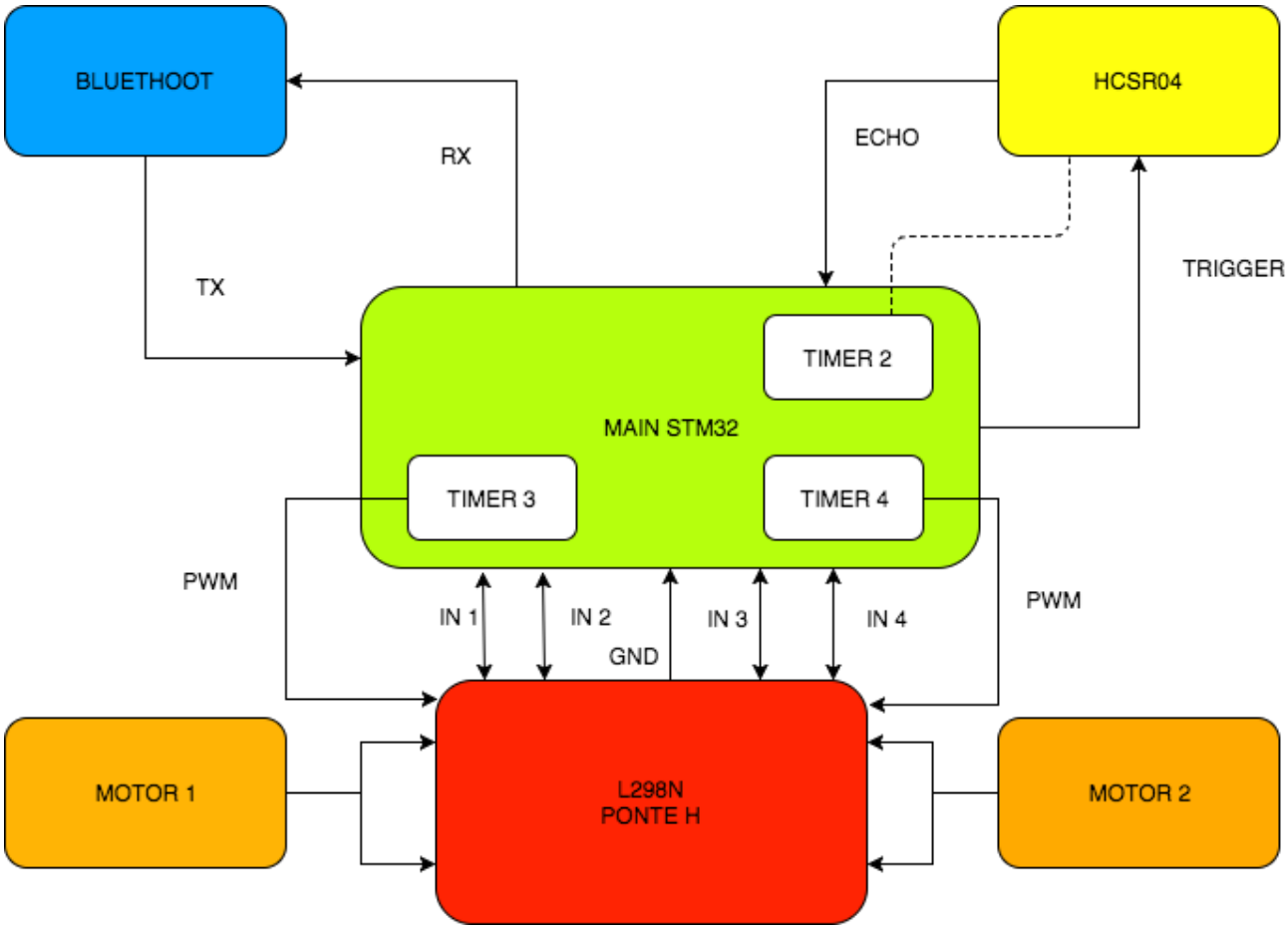


Figure 5 Struttura Generale Progetto

3 Funzionalità utilizzate

In questo capitolo riassumiamo brevemente le funzionalità utilizzate nel progetto e spieghiamo nel dettaglio il loro funzionamento.

Come è possibile vedere nella figura 5, la comunicazione attraverso il modulo bluethoot avviene per mezzo della comunicazione seriale.

La comunicazione seriale prende il proprio nome dalla modalità con cui vengono inviate le informazioni, ovvero il dispositivo che trasmette, invia le informazioni al dispositivo che le riceve un bit alla volta in modo sequenziale. Questa strategia di comunicazione nel corso degli anni si è dimostrata più economica e più robusta rispetto alla concorrente parallela, riuscendo pertanto a conquistare la quasi totalità delle comunicazioni digitali ad oggi esistenti. Di contro una tale strategia di comunicazione richiede, rispetto al caso parallelo, una architettura di gestione e controllo di gran lunga più complessa. Nel corso degli anni sono stati sviluppati circa a decina di standard, diversi per modalità di sincronizzazione, per dimensioni dei pacchetto e per l'interpretazione da dare a quest'ultimo.

Partiamo dalle modalità di sincronizzazione; in genere possiamo incontrare 2 tipi diversi di comunicazione: sincrona, asincrona.

In una comunicazione sincrona il trasmettitore invia i dati un bit per volta con una determinata frequenza costante, ovvero oltre al canale su cui viaggiano i dati ne esiste un altro tra i due dispositivi sul quale viaggia un segnale di clock. Grazie a quest'ultimo il ricevitore, all'interno del treno di impulsi che riceve dal trasmettitore, sa quando finisce un bit e quando ne inizia un altro

Nel caso asincrono invece non esiste nessun tipo di sincronia tra ricevitore e trasmettitore, quest'ultimo quando è pronto inizia a trasmettere serialmente i bit che vengono letti direttamente dal ricevitore. Il pacchetto inviato contiene sempre un bit di start, che segnala al ricevitore l'inizio della trasmissione, i dati da inviare che nel maggior parte dei casi sono otto e poi può contenere anche un bit di parità che ha lo scopo di verificare qualche eventuale errore nella trasmissione e dei bit di stop che segnalano al ricevitore la fine del pacchetto. Nel caso di invio di grosse quantità di dati, questo tipo di comunicazione risulta meno efficiente rispetto alla sincrona in quanto vengono più volte inviati bit che non contengono informazioni come i bit di start e stop.

3.1 USART

USART è l'acronimo di Universal Synchronous-Asynchronous Receiver-Trasmitter, ed è l'evoluzione del vecchio UART che non era capace di gestire le comunicazioni sincrone; nei computer si occupa della gestione della comunicazione tramite l'interfaccia RS-232. Il protocollo generalmente è composto da un bit di start, otto bit dati, un bit di parità e qualche volta può anche contare uno o due bit di stop. La comunicazione avviene tramite due canali dedicati, uno di trasmissione ed uno di ricezione, ovviamente intrecciati tra i due dispositivi: il canale di trasmissione di un dispositivo è connesso al canale di ricezione dell'altro e viceversa. Il canale di trasmissione normalmente si trova al livello logico alto, quando un dispositivo è pronto per trasmettere abbassa il livello sulla linea di trasmissione mandando così il primo bit, ovvero il bit di start. Successivamente il dispositivo invia i dati partendo dal meno significativo, per poi concludere con i bit di parità o di stop. Il bit di parità viene utilizzato per il controllo finale del pacchetto ricevuto e si base sul numero di 1 presenti nel pacchetto, i quali all'interno del pacchetto devono sempre essere in numero pari, così se ad esempio se tra i bit di dati ci sono cinque 1 il bit di parità sarà un altro 1.

Per quanto riguarda invece i bit di stop, generalmente si possono avere tre opzioni, ovvero 1, 1,5 o 2 bit di stop, durante l'invio del bit di stop il trasmettitore riporta la linea nello stato di riposo. Il concetto di 1,5 bit, ovviamente non ha senso se non si definisce il concetto di bit in una comunicazione di questo tipo. In una comunicazione seriale un bit viene definito da un tempo, ovvero stabilita la velocità della comunicazione ad esempio 115200 baud (bit al secondo o bps) un singolo bit durerà 1/115200 secondi, circa 8.68 us pertanto dal momento in cui inizia la comunicazione (nel caso asincrono) ogni 8,68 us si avrà un bit diverso nel pacchetto, in questo caso campionando ogni 8,68 us il ricevitore sarà in grado di distinguere un bit da un altro. In questo contesto 1.5 bit non è altro che un tempo di 8,68*1.5 us durante il quale il trasmettitore mantiene costante il livello sul canale. Le caratteristiche peculiari della comunicazione spesso vengono riassunte tramite una sigla, che indica la velocità, la presenza di bit di parità e di stop, così ad esempio una sigla come 19200n1, indica una comunicazione con una velocità di 19200 baud, nessun bit di parità ed un bit di stop.

Come detto una comunicazione su RS-232 ha due canali, uno per la trasmissione ed uno per la ricezione, nei moderni computer si usano segnali di tipo TTL, ovvero 5V equivalgono ad un 1 logico e 0V ad uno zero. Nel caso di comunicazioni tra dispositivi relativamente vicini questa configurazione lavora bene, tuttavia se le distanze tra i dispositivi iniziano a diventare grandi, questa configurazione potrebbe diventare difficile da gestire, in quanto la resistenze interna dei fili su cui viaggiano i segnali potrebbero dissipare quest'ultimi portando ad esempio al ricevitore una tensione di 2.0V che a quel punto potrebbe essere interpretata come uno zero ma che magari in partenza era 1. Per ovviare a questo problema e creare un tipo di comunicazione robusta alle distanze senza variare la configurazione a due canali, nel corso degli ultimi anni la RS-232 è stata sostituita con la RS-485. Questo nuovo standard risolve il problema delle distanze ragionando in modo differenziale, ovvero i due canali di trasmissione vengono virtualmente sostituiti con due generici canali A e B, nel caso in cui su A c'è una tensione di circa 200 mV maggiore rispetto a B, il ricevitore percepirà un 1, e nel caso contrario uno 0. In questo modo se ad esempio abbiamo 5V sul canale A e 4.8V sul canale B a monte, ipotizzando che i due canali siano costruttivamente uguali e che pertanto dissipano la stessa potenza, anche se il ricevitore ad esempio legge 2 Volt sul canale A e 1,8 sul canale B sarà comunque in grado di interpretare il bit in modo esatto. Oggi nei moderni computer la porta seriale con l'interfaccia RS-232 è sparita e per utilizzarla bisogna ricorrere a convertitori digitali capaci di riadattare lo standard USB.

3.2 TIMER

Sempre osservando la figura 3 si nota che per gestire i motori sono stati utilizzati i timer interni alla scheda STM, inoltra per abilitare e pilotare il senso di rotazione sono state utilizzate le porte GPIO presenti sulla Discovery.

Poiché un microcontrollore normalmente non dispone dell'hardware adatto ad eseguire operazioni parallele si utilizzano le periferiche di timing.

Questo periferiche vengono utilizzate dal microcontrollore per la generazione di interrupt ad istanti prefissati di tempo, quando un interrupt viene generato il microcontrollore congela le operazioni che stava facendo e corre ad eseguire quelle relative all'interrupt.

In questo modo si garantisce l'esecuzione delle istruzioni relative all'interrupt in istanti prefissati di tempo.
Normalmente le periferiche di timing all'interno di un microcontrollore sono di due tipi: Counter e Timer.
I timers hanno un funzionamento simile ad degli orologi, infatti, ad ogni colpo di clock il timer incrementa un registro contatore, quando il contatore raggiunge un determinato valore, o semplicemente va in overflow, il timer può generare un interrupt. E' quindi possibile generare interrupt ad istanti precisi di tempo conoscendo la frequenza di clock del microcontrollore e fissando il valore della soglia che il contatore deve superare per generare l'interrupt.
I parametri caratteristici dei timer sono il range e la risoluzione. Per range di un timer si intende la massima quantità di tempo che un timer può contare, ovvero il massimo tempo tra due interrupt successivi. Viceversa la risoluzione indica il minor tempo possibile tra due interrupt successivi.
I contatori hanno un funzionamento molto simile ai timers, anche per i contatori, infatti, viene incrementato un registro contatore in occasione di un determinato impulso o evento che può essere interno al microcontrollore come lo è un colpo di clock per i timers o esterno a quest'ultimo come ad esempio un colpo di clock da un dispositivo esterno.
Nel caso del counter è possibile usare un interrupt generato da un timer per incrementare il proprio registro contatore il quale verrà poi usato per generare interrupt di range maggior rispetto a quelli generabili dal singolo timer. Per il settaggio dei timers spesso si utilizzano diversi registri quelli storici o più comuni sono: il prescaler e il postscaler. Il registro di prescaler è presente in quasi tutti i timers del microcontrollore e si usa per dividere la frequenza di clock di quest'ultimo in modo da incrementare il registro contatore del timer con frequenze più basse, ovvero più lentamente. In questo modo si aumenta sia il range che la risoluzione del timer. Il postscaler invece si trova a valle del timers e fissa il moltiplicatore per la generazione dell'interrupt, ovvero, quanti overflow del timer sono necessari a creare un interrupt
I microcontrollori a bordo delle schede Discovery montano svariate periferiche timers e nello specifico il microcontrollore che stiamo usando monta ben 12 timer.

Timer	Channel 1			Channel 2			Channel 3			Channel 4		
	PP1	PP2	PP3	PP1	PP2	PP3	PP1	PP2	PP3	PP1	PP2	PP3
TIM 1	PA8	PE9		PA9	PE10		PA10	PE13		PA11	PE14	
TIM 2	PA0	PA5	PA15	PA1	PB3		PA2	PB10		PA3	PB11	
TIM 3	PA6	PB4	PC6	PA7	PB5	PC7	PB0	PC8		PB1	PC9	
TIM 4	PB6	PD12		PB7	PD13		PB8	PD14		PB9	PD15	
TIM 5	PA0	PH10		PA1	PH11		PA2	PH12		PA3	PI0	
TIM 8	PC6	PI5		PC7	PI6		PC8	PI7		PC9	PI2	
TIM 9	PA2	PE5		PA3	PE6							
TIM 10	PB8	PF6										
TIM 11	PB9	PF7										
TIM 12	PB14	PH6		PB15	PH9							
TIM 13	PA6	PF8										
TIM 14	PA7	PF9										

Figure 6 Timer on STM32

In STMCar abbiamo utilizzato il Timer 2 per settare un contatore e calcolare la differenza tra il momento in cui il segnale di Echo è al livello logico alto e quando invece viene risettato al livello logico basso (Falling). In questo modo è possibile calcolare la distanza e configurare quindi il sensore di prossimità HCSR04.

3.3 PWM

l termine PWM è l'acronimo di Pulse-Width Modulation (Modulazione di larghezza di impulso), trascurando i dettagli di tale tecnica per i quali rimandiamo a testi più specialistici, nell'ottica del semplice microcontrollore si tratta di un treno di impulsi di durata variabile ad

una data frequenza. Ogni impulso non è altro che un onda quadra con un periodo fisso, dipendente dalla frequenza del treno, composta da una parte alta (3.3 Volt) che dura per un certo "Ton" e una parte bassa (0 Volt) che dura per un certo "Toff" come mostrato in nella figura 1.

Se la frequenza del treno è sufficientemente alta rispetto alle frequenze dei device a valle è possibile far arrivare a quest'ultimi una tensione variabile tra 0 e 3.3 Volt semplicemente cambiando i tempi di on ed off dell'onda quadra. Il tempo del treno, ovvero il tempo tra due impulsi successivi, ovviamente sarà l'inverso della frequenza di quest'ultimo o più semplicemente la somma dei tempi di on ed off, T = Ton + Toff.

Il rapporto tra il Ton e il tempo globale T, prende il nome di "*duty cycle*", pertanto se vogliamo creare una tensione variabile ci basterà agire su quest'ultimo, nel caso per esempio in cui vogliamo una tensione nulla, imporremo un duty cycle del 0%, ovvero solo Toff, nel caso in cui vogliamo una tensione media di 3.3 Volt, useremo il duty cycle massimo, ovvero solo Ton e nel caso in cui ad esempio vogliamo una tensione di 0.33 Volt imporremo un duty cycle del 10% in quanto 0.33 volt è esattamente il 10% della tensione massima.

Nelle STM32 Discovery boards un segnale di tipo PWM viene generato mediante l'utilizzo dei timer interni al microcontrollore (già incontrati nella puntata precedente) ricordiamoci che il microcontrollore che stiamo utilizzando monta a bordo ben 12 timers (fig 6).

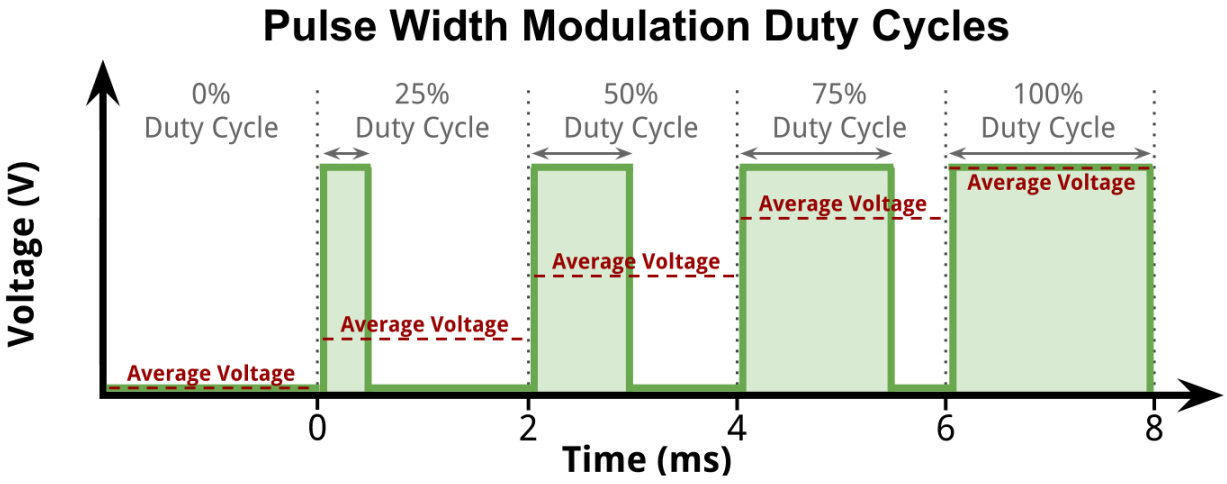


Figure 7 Duty Cycle

Nel progetto abbiamo utilizzato in particolare i Timer 3 e 4 per generare un un impulso PWM a 20Khz con Prescaler 10. Questo come sara descritto nel capitolo successivo è stato collegato al pin enable del ponte H per entrambi i motori utilizzando rispettivamente il pin B6 per il timer 3 e il pin A6 per il timer 4.
In questo modo settando un opportuno “pulse” nella configurazione della struttura PWM è stato possibile gestire la velocità di rotazione dei motori.

3.4 GPIO

Il General Purpose Input/Output (anche noto come GPIO) è un'interfaccia disponibile su alcuni dispositivi elettronici.
Un dispositivo microprocessore o interfaccia può avere una o più connessioni GPIO su un'interfaccia con dispositivi e periferiche esterne. Queste possono agire come input, per leggere i segnali digitali dalle altre parti del circuito, o output, per controllare o segnalare agli altri dispositivi. GPIO sono spesso collocati in gruppi, tipicamente di 8 pin - una porta GPIO - che usualmente hanno GPIO individuali configurabili o come input o come output. In alcuni casi, GPIO possono essere configurati per produrre degli CPU interrupt ed essere in grado di utilizzare il Direct Memory Access per spostare efficientemente grandi quantità di dati per e dal dispositivo.

Nel progetto abbiamo utilizzato come GPIO configurati come output i pin (D) 12,13,14,15 per gestire il verso di rotazione dei motori. Inoltre per configurare il sensore di prossimità HCSR04 è stato configurato come GPIO di output il pin C6 (Trigger) e come GPIO di input il pin C7(Echo).

4 Progettazione e implementazione

In questo capitolo descriviamo la progettazione e implementazione del software per il funzionamento del progetto.

La configurazione delle componenti principali è stata fatta attraverso STMCubeMx un software particolare e molto utile per configurare anche graficamente la Discovery.

Qui sotto è riportata la configurazione su Cube del nostro progetto, che come si può facilmente notare è in linea con quanto descritto nei capitoli precedenti e mostrato anche in figura 3. Per quanto riguarda l'implementazione è stato usato l'ambiente di sviluppo Keil UVision v5. Questo è scaricabile gratuitamente dal sito STM. Abbiamo preferito quest'ultimo rispetto ad altri IDE come per esempio CoIDE per la sua semplicità di utilizzo.

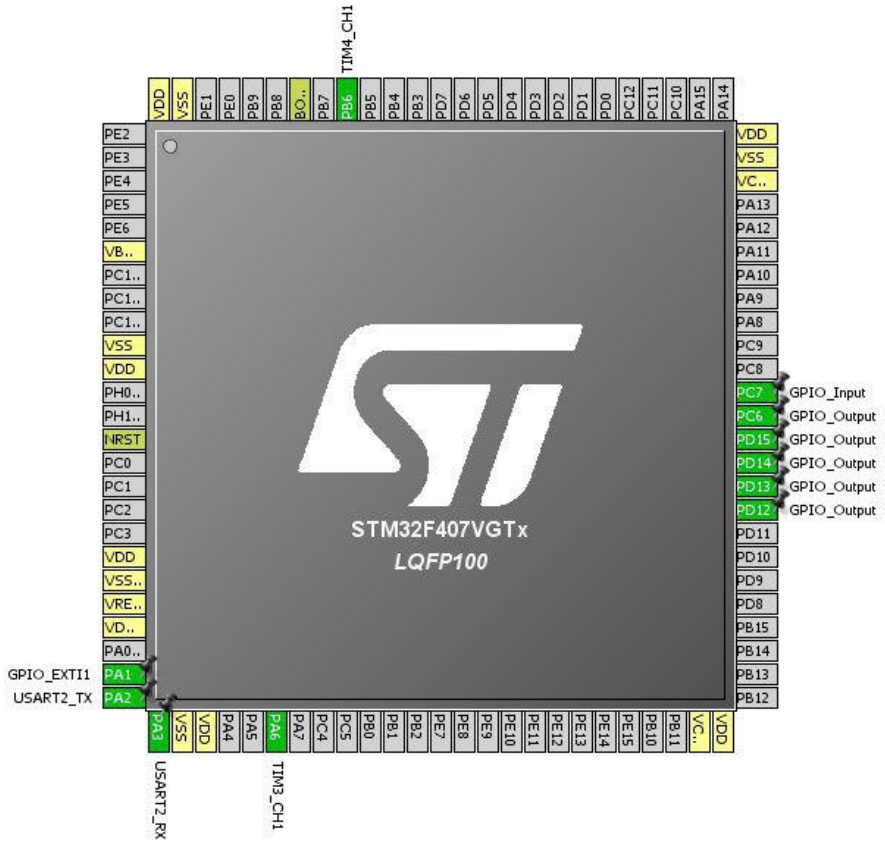
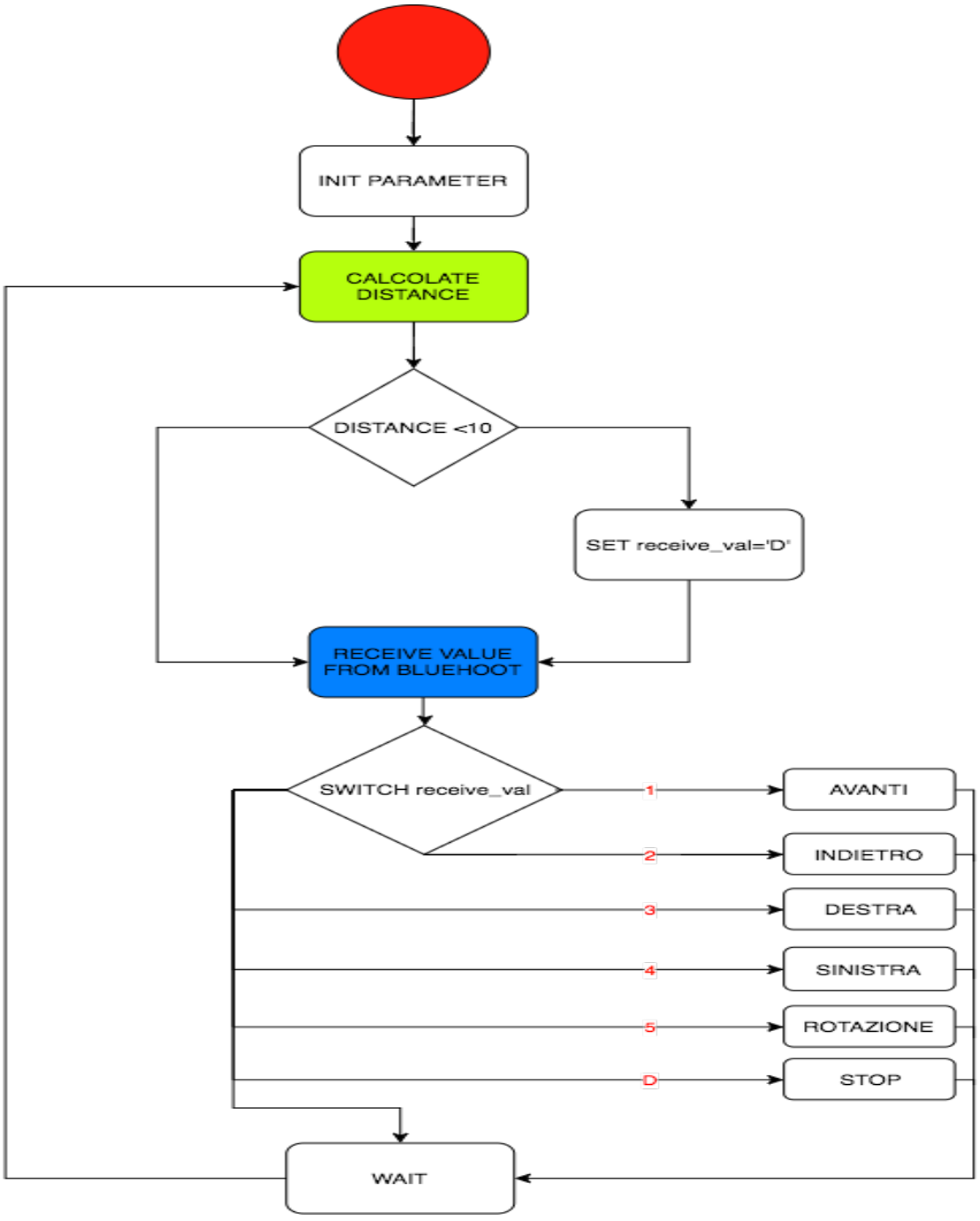


Figure 8 STMCUBE Configuration

Il progetto si basa su pochi componenti che come è possibile notare dal diagramma riportato, devono essere sincronizzati e inizializzati prima di poter avere dei risultati in output. Dopo la prima fase di configurazione, all'interno di un ciclo infinto è stato calcolata la distanza e settata la variabile utilizzata anche come variabile di ritorno per le comunicazioni seriali con il modulo Bluetooth. Successivamente il programma aspetta di ricevere qualcosa sulla seriale tramite il Bluetooth, in seguito in base al comando inviato saranno attivate opportune funzioni per compiere determinate azioni e muovere o fermare del tutto la macchina.



4.1 Implementazione

In questo paragrafo abbiamo riportato il software scritto per il funzionamento del progetto. Abbiamo organizzato il progetto in due file header .h dove sono stati definite le strutture dei componenti utilizzati.

Nel file main.c sono stati inclusi i file .h e attraverso delle funzioni sono stati inizializzati tutti i componenti definiti.

```
//INIT HCSR04
void HCRSR04_init(GPIO_TypeDef* ECHOPORT,uint16_t ECHOPIN,GPIO_TypeDef*
TRIGGERPORT,uint16_t TRIGGERPIN){
        hc.ECHO_GPIOx=ECHOPORT;
        hc.ECHO_GPIO_Pin=ECHOPIN;
        hc.TRIGGER_GPIOx=TRIGGERPORT;
        hc.TRIGGER_GPIO_Pin=TRIGGERPIN;
}
//INIT MOTOR INPUT
void Motor_init(GPIO_TypeDef* F1PORT,uint16_t F1PIN,GPIO_TypeDef* B1PORT,uint16_t
B1PIN,GPIO_TypeDef* F2PORT,uint16_t F2PIN,GPIO_TypeDef* B2PORT,uint16_t B2PIN){
        mt.FORWARD1_GPIOx=F1PORT;
        mt.FORWARD1_GPIO_Pin=F1PIN;
        mt.BACK1_GPIOx=B1PORT;
        mt.BACK1_GPIO_Pin=B1PIN;

        mt.FORWARD2_GPIOx=F2PORT;
        mt.FORWARD2_GPIO_Pin=F2PIN;
        mt.BACK2_GPIOx=B2PORT;
        mt.BACK2_GPIO_Pin=B2PIN;
}
```

5 Conclusioni e Sviluppi Futuri