UNIVERSITA’ DEGLI STUDI DI SALERNO

Facoltà di ingegneria

Corso di laurea in ingegneria elettronica

PROGETTO IN

Misure per L’Automazione

Manuale Programmatore

INSEGNANTE

Prof. Vincenzo Paciello

STUDENTE

Flavio Della Calce

A.A 2021/2022

1. **Introduzione**
   1.1 Descrizione del software realizzato.

2. **Caratteristiche del software**
   2.1 Descrizione dei registri utilizzati.
   2.2 Descrizione delle funzioni sviluppate.
   2.3 Diagramma a blocchi del software realizzato.

1. INTRODUZIONE

Descrizione del software realizzato.

Il software realizzato ha come obiettivo quello di:

- Inizializzare una comunicazione seriale con il microcontrollore;
- Consentire l'attivazione di segnali PWM;
- Consentire la modifica dei parametri dei segnali PWM attivi;
- Consentire la variazione del baud rate;
- Consentire la lettura dell'ID del microcontrollore;

Per poter implementare queste funzionalità sono stati utilizzate diverse periferiche e sono stati definite diverse funzioni che verranno illustrate nel capitolo successivo. Il programma utilizzato per la realizzazione del software è Keilμvision, un software progettato per tutti i microcontrollori con architettura ARM cortex. In particolare, il microcontrollore su cui è stato implementato il software realizzato è l'STM32 NUCLEO F401RE. Per ulteriori informazioni riguardanti le funzionalità offerte e la scheda di sviluppo utilizzata consultare il manuale utente.

2. CARATTERISTICHE DEL SOFTWARE

2.1 Descrizione dei registri utilizzati.

In questa sezione vengono descritti i registri utilizzati e le loro funzionalità.

RCC

```
#define RCC_APB1ENR *(long*)  0x40023840        //RCC APB1 enable register
#define RCC_AHB1ENR *(long*)  0x40023830        //RCC AHB1 enable register
```

L'RCC (Reset and Clock Control) è un registro essenziale la cui funzione è quella abilitare il clock alle differenti periferiche utilizzate. I due registri dichiarati consentono di abilitarlo sulle periferiche da noi utilizzate, ovvero APB1 e AHB1. Si noti che il clock non è abilitato di default a ogni periferica per limitare il consumo della scheda.

GPIOA

Per GPIOA si intende "General Purpose Input/Output" relativo alla porta A, ovvero i pin della scheda di sviluppo che costituiscono tale porta.

Nello sviluppo del software sono stati utilizzati tre registri che appartengono alla categoria GPIOA, ovvero:

1) GPIOA_MODER
2) GPIOA_ODR
3) GPIOA_AFRL

1) GPIOA_MODER

```
#define GPIOA_MODER *(long*)  0x40020000        //GPIOA mode register
```

Il registro GPIOA_MODER definisce la modalità (ingresso/uscita/alternata) relativa al singolo pin, in particolare nella progettazione della scheda di sviluppo sono stati utilizzati sei pin di cui quattro in modalità output per fornire l'onda quadra relativa ai segnali PWM e gli altri due per implementare la comunicazione seriale in modalità alternata (Rx e Tx).

2) GPIOA_ODR

```
#define GPIOA_ODR   *(long*)  0x40020014        //GPIOA Output data register
```

Il registro GPIOA_ODR contiene tutti gli stati di output dei pin della porta GPIOA e serve quindi a gestire il bit in uscita e viene utilizzato nella generazione dell'onda quadra relativa ai segnali PWM.

3) GPIOA_AFRL

```
#define GPIOA_AFRL  *(long*)  0x40020020        //GPIOA AF register low
```

Il registro GPIOA_AFRL serve a selezionare il particolare tipo di funzione alternata sul singolo pin; in questo caso è stato utilizzato il register LOW in quanto stiamo utilizzando in modalità alternata i pin PA2 e PA3 che ricadono nella categoria pin LOW ovvero da PA0 fino a PA7.

USART2

L'USART è una periferica del microcontrollore adibita alla gestione della comunicazione seriale.

Nello sviluppo del software sono stati utilizzati quattro registri che appartengono alla categoria USART2, ovvero:

1) USART2_CR1
2) USART2_BRR
3) USART2_SR
4) USART2_DR

1) ***USART2_CR1***

```
#define USART2_CR1   *(long*)   0x4000440C        //USART2 control register 1
```

L'USART2_CR1 è un registro che presenta diverse funzionalità. Viene utilizzato nel progetto per: abilitare l'USART2 (bit 13 UE), abilitare la trasmissione e la ricezione (bit 2 RE e 3 TE) e per generare un interrupt USART ogni volta che ORE=1 o RXNE=1 nel registro USART_SR (bit 5).

2) ***USART2_BRR***

```
#define USART2_BRR   *(long*)   0x40004408        //USART2 baud rate register
```

L'USART2_BRR è un registro che viene utilizzato per settare il baud rate della comunicazione seriale della scheda di sviluppo. Il contenuto di questo registro è l'USART_DIV che prevede una parte frazionaria e una mantissa. L'USART_DIV è un divisore della frequenza di clock che decide la velocità della comunicazione secondo la seguente formula:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

In cui OVER8 è in genere settato uguale a 0, Tx/Rx baud è il valore di baud rate che si vuole settare e fCK è la frequenza di conteggio del microcontrollore e da questa equazione è possibile ricavare tramite una semplice operazione di formula inversa l'USART_DIV.

3) ***USART2_SR***

```
#define USART2_SR    *(long*)   0x40004400        //USART2 status register
```

L'USART2_SR è un registro che tiene conto dello stato relativo alla periferica e viene utilizzato per verificare se è stato inviato o ricevuto un carattere.

4) ***USART2_DR***

```
#define USART2_DR    *(long*)   0x40004404        //USART2 data register
```

L'USART2_DR è un registro dati della periferica e serve a contenere ciò che è stato immesso sulla seriale.

NVIC

```
#define NVIC_ISER0   *(long*)   0xE000E100        //NVIC ISER0
#define NVIC_ISER1   *(long*)   0xE000E104        //NVIC ISER1
```

L'NVIC (Nested Vectored Interrupt Controller) è un registro interno al Cortex (ovvero al processore del Microcontrollore) in cui vengono mappate, gestite ed abilitate o meno

le sorgenti interne ed esterne di Interrupt. In questo caso sono stati abilitati il Timer2, Timer3 e Timer4 come sorgenti di interrupt (il Timer2 è quello che produce eventi di interrupt a maggiore frequenza) agendo sul registro NVIC_ISER0 ed sono stati abilitati il Timer5 e l'USART2 sempre come sorgenti di interrupt agendo sul registro NVIC_ISER1.

MCUID

```
#define MCUID1      *(long*)  0x1FFF7A10        //ID register 1
#define MCUID2      *(long*)  0x1FFF7A14        //ID register 2
#define MCUID3      *(long*)  0x1FFF7A18        //ID register 3
```

L'MCUID1, MCUID2 e MCUID3 sono tre registri che sono stati dichiarati con la finalità di visualizzare l'ID del microcontrollore; in particolare, l'ID della scheda di sviluppo è una sequenza binaria a 96 bit contenuta in 3 registri ognuno da 32 bit (UID1, UID2, UID3).

Timers Control Registers

```
//Timers Control Registers
#define TIM2_CR1 *(long*) 0x40000000
#define TIM3_CR1 *(long*) 0x40000400
#define TIM4_CR1 *(long*) 0x40000800
#define TIM5_CR1 *(long*) 0x40000C00
```

TIM2_CR1, TIM3_CR1, TIM4_CR1 e TIM5_CR1 sono dei registri utilizzati per controllare i vari timer utilizzati e per impostare le differenti modalità che la periferica offre all'utente; in particolare, nella progettazione è stato abilitato il conteggio settando in maniera opportuna i bit del registro.

Timers Status Registers

```
//Timers Status Registers
#define TIM2_SR *(long*)  0x40000010
#define TIM3_SR *(long*)  0x40000410
#define TIM4_SR *(long*)  0x40000810
#define TIM5_SR *(long*)  0x40000C10
```

TIM2_SR, TIM3_SR, TIM4_SR e TIM5_SR sono dei registri di stato dei timer e tramite questi possiamo verificare se sono accaduti diversi eventi, come ad esempio la fine del conteggio.

Timers Prescalers

```
//Timers Prescalers
#define TIM2_PSC *(long*) 0x40000028
#define TIM3_PSC *(long*) 0x40000428
#define TIM4_PSC *(long*) 0x40000828
#define TIM5_PSC *(long*) 0x40000C28
```

TIM2_PSC, TIM3_PSC, TIM4_PSC e TIM5_PSC sono dei registri che consentono di settare il valore del divisore della frequenza di clock a cui conteranno i vari Timer e di gestire la frequenza di conteggio (all'interno di questi registri verrà impostato il valore per il quale andrà divisa la frequenza di clock, nel nostro caso il valore è 16MHz).

Timers Auto Reload Register

```
//Timers Auto Reload Registers
#define TIM2_ARR *(long*) 0x4000002C
#define TIM3_ARR *(long*) 0x4000042C
#define TIM4_ARR *(long*) 0x4000082C
#define TIM5_ARR *(long*) 0x40000C2C
```

TIM2_ARR, TIM3_ARR, TIM4_ARR e TIM5_ARR sono dei registri in cui va inserito un valore intero a 32 bit che costituisce il numero di fronti di clock che verranno contati dal timer, terminato il conteggio il registro si auto-ricarica dello stesso valore.

Timers DMA – Interrupt Enable Registers

```
//Timers DMA/Interrupt Enable Registers
#define TIM2_DIER *(long*)  0x4000000C
#define TIM3_DIER *(long*)  0x4000040C
#define TIM4_DIER *(long*)  0x4000080C
#define TIM5_DIER *(long*)  0x40000C0C
```

TIM2_DIER, TIM3_DIER, TIM 4_DIER e TIM5_DIER sono dei registri che hanno il compito di abilitare le differenti sorgenti di interrupt che la periferica dispone.

Timers Event Generation Registers

```
//Timers Event Generation Register
#define TIM2_EGR *(long*) 0x40000014
#define TIM3_EGR *(long*) 0x40000414
#define TIM4_EGR *(long*) 0x40000814
#define TIM5_EGR *(long*) 0x40000C14
```

TIM2_EGR, TIM3_EGR, TIM4_EGR e TIM5_EGR sono dei registri che resettano il contatore e generano un aggiornamento dei registri.

2.2 Descrizione delle funzioni sviluppate.

In questa sezione vengono descritte le funzioni utilizzate e il loro scopo.

Printstring

```
void printstring(char* str){

  while(*str !=   '\0') {
      USART2_SR  &=   (~(1<<6));                          //USART 2 Transmission complete flag set to 0
      USART2_DR = *str;
      while(! (( USART2_SR & (1<<6) ) >>6)  );            //Wait for transmission complete
      str++;
  }
}
```

La funzione printstring ha lo scopo di mettere a video tutto ciò che effettivamente si visualizza su Termite quando la scheda di sviluppo è collegata tramite cavo USB al PC. In queste 5 righe di codice viene eseguito un ciclo while in cui si entra se il valore della stringa (posto uguale al valore di USART2_DR) a cui punta il puntatore è diverso dal carattere di fine stringa '\0'. Se la condizione è verificata si entra nel ciclo while in cui la flag della trasmissione completata di USART2 viene settata a 0 con un'operazione di 'and' ponendo a 1 tutti i e 32 i bit tranne il bit 6 e tramite l'utilizzo di un secondo ciclo while interno al primo si attende il termine della comunicazione del carattere tramite USART2, infatti il bit 6 del registro USART2_SR diventa 1 quando la trasmissione è completa e la riga di codice aspetta proprio che tale bit diventi 1.

Menù

```
void menu() {
  printstring("Selezionare l'opzione desiderata:\n"
              "- Digitare 1 per : definire il numero di segnali PWM attivi (0 di default)\n"
              "- Digitare 2 per : modificare i parametri di un segnale PWM attivo\n"
              "- Digitare 3 per : definire il baud rate della comunicazione seriale (9600 di default)\n"
              "- Digitare 4 per : leggere l'ID del microcontrollore\n\n");
  submenu=0;
```

La funzione menù si serve della funzione printstring per mettere a video su Termite il menù principale da cui poi si possono scegliere le varie funzionalità che offre il software realizzato. Viene inoltre settata la static int submenù a 0, dove questa variabile statica è molto utile per gestire i vari "sottomenù" appunto che il programma presenta.

SetDuty

```
void SetDuty(int Duty) {              //Set New Duty Value

  int Ton;
//  GPIOA_ODR ^= (1<<5);
  switch (SelectedSignal) {            //Scelta numero del segnale PWM di cui modificare il Duty

    case 1 :  {                        //Imposta nuovo Duty per il segnale 1
      Ton = TIM2_ARRvalueON;
      TIM2_ARRvalueON = (int)( ((int)TIM2_ARRvalueON + (int)TIM2_ARRvalueOFF) * Duty / 100);
      TIM2_ARRvalueOFF = TIM2_ARRvalueOFF + Ton - TIM2_ARRvalueON;
    }
    break;

    case 2 :  {                        //Imposta nuovo Duty per il segnale 2
    //  GPIOA_ODR ^= (1<<5);
      Ton = TIM3_ARRvalueON;
      TIM3_ARRvalueON = (int)((TIM3_ARRvalueON + TIM3_ARRvalueOFF)*Duty/100);
      TIM3_ARRvalueOFF = TIM3_ARRvalueOFF+Ton-TIM3_ARRvalueON;
    }
    break;

    case 3 :  {                        //Imposta nuovo Duty per il segnale 3
      Ton = TIM4_ARRvalueON;
      TIM4_ARRvalueON = (int)((TIM4_ARRvalueON + TIM4_ARRvalueOFF)*Duty/100);
      TIM4_ARRvalueOFF = TIM4_ARRvalueOFF+Ton-TIM4_ARRvalueON;
    }
    break;
```

```
    case 4 : {                          //Imposta nuovo Duty per il segnale 4
      Ton = TIM5_ARRvalueON;
      TIM5_ARRvalueON = (int)((TIM5_ARRvalueON + TIM5_ARRvalueOFF)*Duty/100);
      TIM5_ARRvalueOFF = TIM5_ARRvalueOFF+Ton-TIM5_ARRvalueON;
    }
    break;
  }

    SelectedSignal = 0;
    menu();
}
```

La funzione SetDuty ha lo scopo di permettere all'utente di settare il nuovo valore del Duty Cycle; quindi, la funzione ci restituisce il valore Duty inteso proprio come nuovo valore del Duty cycle scelto. Viene inizialmente dichiarata una variabile Ton e successivamente, entrando nello switch se il segnale è attivo (variabile statica SelectedSignal = 1), viene posta uguale al valore del registro TIM2_ARRvalueON che sarebbe il valore di ON dell'Auto Reload Register del TIM2. Il valore di TIM2_ARRvalueON viene posto uguale al valore del registro TIM2_ARRvalueON più quello del registro TIM2_ARRvalueOFF moltiplicato per il valore del duty diviso 100 perché il valore del duty è espresso in percentuale. Infine, il valore del TIM2_ARRvalueOFF viene posto uguale a sé stesso più Ton meno TIM2_ARRvalueON. Finito ciò la variabile statica SelectedSignal che tiene conto se il segnale è attivo o meno viene settata a 0 e poi si ritorna alla funzione menù. Lo stesso procedimento vale per gli altri tre casi.

SetFreq

```
void SetFreq(int NewFreq) {           //Set New Frequency Value
  int Duty=0;                               //In percentage
//  char a[100];

  switch (SelectedSignal) {

    case 1 :                                //Modify Signal 1 Frequency
      Duty = 100*(int)TIM2_ARRvalueON / ((int)TIM2_ARRvalueON + (int)TIM2_ARRvalueOFF);
      TIM2_ARRvalueON = 10000 * Duty / NewFreq;
      TIM2_ARRvalueOFF = ( 100 - Duty ) * 10000 / NewFreq;
//      sprintf(a, "Duty = %d\n Ton = %d\n Toff = %d\n", Duty, TIM2_ARRvalueON, TIM2_ARRvalueOFF);
//      printstring(a);
      break;

    case 2 :                                //Modify Signal 2 Frequency
      Duty = 100*(int)TIM3_ARRvalueON / ((int)TIM3_ARRvalueON + (int)TIM3_ARRvalueOFF);
      TIM3_ARRvalueON = 10000 * Duty / NewFreq;
      TIM3_ARRvalueOFF = ( 100 - Duty ) * 10000 / NewFreq;
      break;

    case 3 :                                //Modify Signal 3 Frequency
      Duty = 100*(int)TIM4_ARRvalueON / ((int)TIM4_ARRvalueON + (int)TIM4_ARRvalueOFF);
      TIM4_ARRvalueON = 10000 * Duty / NewFreq;
      TIM4_ARRvalueOFF = ( 100 - Duty ) * 10000 / NewFreq;
      break;


    case 4 :                                //Modify Signal 4 Frequency
      Duty = 100*(int)TIM5_ARRvalueON / ((int)TIM5_ARRvalueON + (int)TIM5_ARRvalueOFF);
      TIM5_ARRvalueON = 10000 * Duty / NewFreq;
      TIM5_ARRvalueOFF = ( 100 - Duty ) * 10000 / NewFreq;
      break;

  }
  SelectedSignal=0;
}
```

La funzione SetFreq ha lo scopo di permettere all'utente di settare il nuovo valore della frequenza; quindi, la funzione ci restituisce il valore NewFreq inteso proprio come nuovo valore della frequenza scelta. Viene inizialmente dichiarata una variabile Duty e successivamente, entrando nello switch se il segnale è attivo (variabile statica SelectedSignal = 1), viene posta uguale al valore del registro TIM2_ARRvalueON

moltiplicato per 100 (perché il valore del Duty è espresso in percentuale) diviso il valore del registro TIM2_ARRvalueON più il valore del registro TIM2_ARRvalueOFF. Successivamente, TIM2_ARRvalueON viene posto uguale al valore di Duty diviso il valore NewFreq il tutto moltiplicato per 10000 perché entrambi i valori sono espressi in percentuale. Poi, il valore di TIM2_ARRvalueOFF viene posto uguale a 100 meno Duty ovvero il complementare del valore precedente diviso NewFreq e il tutto moltiplicato per 10000 sempre per lo stesso motivo. Finito ciò la variabile statica SelectedSignal che tiene conto se il segnale è attivo o meno viene settata a 0 e poi si ritorna alla funzione menù. Lo stesso procedimento vale per gli altri tre casi.

IRQHandler

```
void USART2_IRQHandler () {

  ReceivedCommand = USART2_DR;
}
//

void TIM2_IRQHandler() {              //TIM2 controls PWM signal on PA0
  TIM2_SR &= (~(1<<0));                     //TIM2 Interrupt Flag set to 0
  GPIOA_ODR ^= (1<<0);

  if (GPIOA_ODR & 1)
    TIM2_ARR    = TIM2_ARRvalueON;
  else
    TIM2_ARR    = TIM2_ARRvalueOFF;
}
//

void TIM3_IRQHandler() {              //TIM3 controls PWM signal on PA1
  TIM3_SR &= (~(1<<0));                     //TIM3 Interrupt Flag set to 0
  GPIOA_ODR   ^= (1<<1);

  if (GPIOA_ODR   & (1<<1))
    TIM3_ARR    = TIM3_ARRvalueON;
  else
    TIM3_ARR    = TIM3_ARRvalueOFF;
}

void TIM4_IRQHandler() {              //TIM4 controls PWM signal on PA4
  TIM4_SR &= (~(1<<0));                     //TIM4 Interrupt Flag set to 0
  GPIOA_ODR   ^= (1<<4);

  if (GPIOA_ODR   & (1<<4))
    TIM4_ARR    = TIM4_ARRvalueON;
  else
    TIM4_ARR    = TIM4_ARRvalueOFF;
}
//

void TIM5_IRQHandler() {              //TIM5 controls PWM signal on PA5
  TIM5_SR &= (~(1<<0));                     //TIM5 Interrupt Flag set to 0
  GPIOA_ODR   ^= (1<<5);

  if (GPIOA_ODR   & (1<<5))
    TIM5_ARR    = TIM5_ARRvalueON;
  else
    TIM5_ARR    = TIM5_ARRvalueOFF;
}
```

La funzione IRQHandler viene eseguita nel momento in cui il timer 2 genera l'interrupt ossia quando termina il conteggio. All'interno del file "startup_stm32f401xe.s" è specificato che la funzione da eseguire nel momento in cui la periferica precedentemente menzionata genera l'interrupt deve avere questo nome particolare.

Utilizzare gli interrupt consente di ottimizzare il codice in quanto gli interrupt sono gestiti a livello hardware e non software. Per quanto riguarda l'USART2, questa funzione pone semplicemente il valore della variabile statica ReceivedCommand uguale al valore del registro USART2_DR, mentre per quanto riguarda i Timer viene settata l'Interrupt Flag del Tim2 a 0, successivamente viene posto 0 al bit 0 del registro GPIOA_ODR e poi si entra nel comando if – else in cui se l'operazione "and" tra il valore del registro GPIOA_ODR e 1 è verificata allora viene posto il valore del registro TIM2_ARR uguale al valore del registro TIM2_ARRvalueON; se non è verificata la condizione allora viene posto il valore del registro TIM2_ARR uguale al valore del registro TIM2_ARRvalueOFF.

ModifyBaud

```
void ModifyBaud () {
  submenu = 3;
  printstring("Selezionare il nuovo valore da impostare per il Baud Rate (BR): \n");
  printstring("- Digitare 1 per: BR = 2400\n");
  printstring("- Digitare 2 per: BR = 4800\n");
  printstring("- Digitare 3 per: BR = 9600\n");
  printstring("- Digitare 4 per: BR = 14400\n");
  printstring("- Digitare 5 per: BR = 19200\n\n");
}
```

La funzione ModifyDuty setta la variabile statica submenù al valore 3 e si serve della funzione printstring per mettere a video su Termite ciò che è illustrato in figura.

MCUIDread

```
void MCUIDRead () {
  int i;
  char ID[200];
  char* point = &ID[0];

  printstring("L'ID del microcontrollore è :\n");

  for(i=31;i>=0;i--) {
    if(MCUID3 & (long)(1<<i))
      sprintf(point, "1");
    else
      sprintf(point, "0");

    point++;

    if(! ( i % 4 ) ) {
      sprintf(point, " ");
      point++;
    }
  }


    for(i=31;i>=0;i--) {
    if(MCUID2 & (long)(1<<i))
      sprintf(point, "1");
    else
      sprintf(point, "0");

    point++;
```

```
    if(! ( i % 4 ) ) {
      sprintf(point, " ");
      point++;
    }
  }


    for(i=31;i>=0;i--) {
    if(MCUID1 & (long)(1<<i))
      sprintf(point, "1");
    else
      sprintf(point, "0");

    point++;

    if( (! ( i % 4 ) ) && (i!=0) ) {
      sprintf(point, " ");
      point++;
    }
  }

  sprintf(point, "\n \n \0");
  printstring(ID);
  menu();
}
```

La funzione MCUIDread ha lo scopo di mettere a video su Termite l'ID del microcontrollore che è una sequenza binaria di 96 bit contenuta 3 registri ciascuno formato da 32 bit, ovvero UID1, UID2 e UID3. Inizialmente vengono dichiarati una variabile "i" utile per i passi successivi, una stringa di caratteri chiamata ID di dimensioni 200 caratteri per comodità e poi viene dichiarato un puntatore che "punta" al primo carattere della stringa chiamata ID. Dopo la funzione printstring viene eseguito un ciclo for al contrario altrimenti verrebbe letto il l'ultimo bit invece del primo e così via. All'interno del ciclo for viene esegui un comando if – else che "printa" 1 se l'operazione "and" tra il registro MCUID3 e la stringa con 1 al posto di "i" è verificata, altrimenti 0. Vi è infine un ulteriore comando if che serve a spaziare i bit "printati" ogni 4 bit. Le stesse operazioni vengono svolte per i registri MCUID2 e MCUID1; viene infine stampato l'ID completo e spaziato ogni 4 bit con la funzione printstring e si viene riportati alla funzione menù.

Error

```
void error () {
  printstring("Errore: Comando Non Valido\n\n");
  menu();
}
```

La funzione Error viene richiamata ogni qual volta viene digitato un carattere non permesso dal software sviluppato; viene così messo a video tramite la funzione printstring che è stato digitato un comando non valido e si viene riportati alla funzione menù.

Main

```
int main(void){

  RCC_APB1ENR |=  0x2000F;                    //USART2, TIM2, TIM3, TIM4, TIM5 clock enable
  RCC_AHB1ENR |=  (1<<0);                     //GPIOA clock enable
  GPIOA_MODER |=  0x5A5;                      //Set PA2 e PA3 as AF, PA0/1/4/5 as Output
  GPIOA_AFRL  =   0x7700;                     //Set PA2 e PA3 as AF07
  USART2_CR1  |=  0x202C;                     //USART2 enable, Rx Tx enable, RNEIE (Read register not empty interrupt enable)
  USART2_BRR  =   0x683;                      //USART2 baud rate set to 9600

  TIM2_DIER   |= (1<<0);                      //TIM2 Update Interrupt Enable
  TIM3_DIER   |= (1<<0);                      //TIM3 Update Interrupt Enable
  TIM4_DIER   |= (1<<0);                      //TIM4 Update Interrupt Enable
  TIM5_DIER   |= (1<<0);                      //TIM5 Update Interrupt Enable

  TIM2_PSC    = 0xF;                          //TIM2 Prescaler set to 16    so fCount = 16 MHz / 16 = 1 MHz
  TIM3_PSC    = 0xF;                          //TIM3 Prescaler set to 16
  TIM4_PSC    = 0xF;                          //TIM4 Prescaler set to 16
  TIM5_PSC    = 0xF;                          //TIM5 Prescaler set to 16

  TIM2_ARR    = 0;                            //TIM2 ARR set to 0
  TIM3_ARR    = 0;                            //TIM3 ARR set to 0
  TIM4_ARR    = 0;                            //TIM4 ARR set to 0
  TIM5_ARR    = 0;                            //TIM5 ARR set to 0

  TIM2_CR1    |= 0x5;                         //TIM2 Counter Enable, UG Interrupt Disable, TIM2_ARR Not Buffered
  TIM3_CR1    |= 0x5;                         //TIM3 Counter Enable, UG Interrupt Disable, TIM3_ARR Not Buffered
  TIM4_CR1    |= 0x5;                         //TIM4 Counter Enable, UG Interrupt Disable, TIM4_ARR Not Buffered
  TIM5_CR1    |= 0x5;                         //TIM5 Counter Enable, UG Interrupt Disable, TIM5_ARR Not Buffered


  NVIC_ISER1  |=  (1<<6)  | (1<<18);          //USART2 and TIM5 NVIC Interrupt Enable
  NVIC_ISER0  |= 0x70000000;                  //TIM2/3/4        NVIC Interrupt Enable

  printstring("Benvenuto nel programma per la generazione di segnali PWM tramite la scheda STM32F401RE\n");
  menu();
```

La funzione Main è la parte essenziale di tutto il programma in cui vengono abilitati i clock dell'USART e dei vari Timer, il clock della GPIOA, vengono settati i vari pin della scheda alcuni come Alternative Function (PA2 e PA3), in particolare come Alternative Function 07 e altri come Output (PA0, PA1, PA4 e PA5), viene abilitata l'USART2, l'Rx e il Tx e viene settato il baud rate a 9600 di default. Vengono poi abilitati gli interrupt sui vari Timer utilizzati tramite il registro TIMx_DIER, vengono impostati i prescaler dei vari Timer a 16, vengono settati gli Auto Reload Register dei vari Timer a 0 e vengono attivati i contatori dei vari Timer, disabilitati gli interrupt update generation dei vari Timer, vengono abilitati gli NVIC interrupt dei vari Timer e dell'USART2 ed infine viene utilizzata la funzione printstring e si viene poi dopo riportati alla funzione menù.