Ecodesign per dispositivi elettronici

UNITA' 9: Microcontrollori parte 2

Autore: Andrej Sarjaš

Sommario del capitolo:

- Sistemi in tempo reale
- Componenti di sistemi in tempo reale
- Progettazione di programmi in sistemi in tempo reale

Funded by the Erasmus+ Programme of the European Union

Ecoinnovation Skills for European Designers, Project number: 562573-EPP-1-2015-1-SI-EPPKA2-SSA. Disclaimer: The European Commission support for the production of this publication does not constitute endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein

9.1. Utilizzo di scheda a microcontrollereSMT32F4discovery

La scheda di sviluppo **STM32F4Discovery** è una scheda a microcontrolore a basso costo e potente, adatta per lo sviluppo rapido di applicazioni integrate e test del potente microcontrollore ARM STM32F407VG con unità FPU integrata (FPU - unità a virgola mobile, un'unità per il calcolo con numeri virgola mobile). La scheda di sviluppo contiene un microcontrollore ARM STM32F407VG, programmatore JTAG, ST-link V2, accelerometro, codec audio, microfono, 4 diodi LED di controllo, interruttore e connettore micro-USB. L'immagine 1 presenta la scheda di sviluppo STM32F4Discovery.

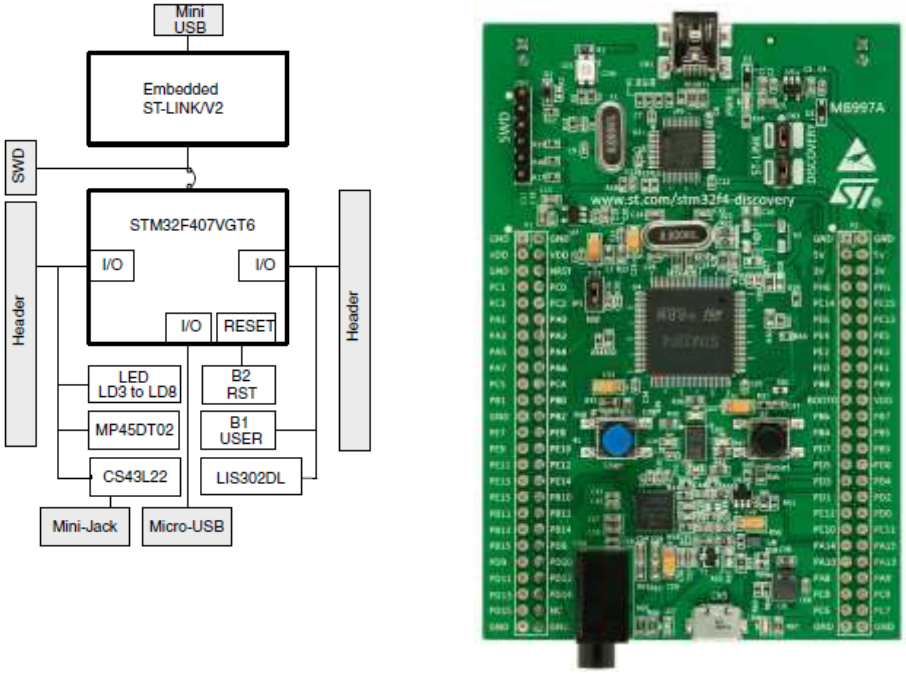

IMMAGINE 1: SCHEDA DI SVILUPPO STM32F4DISOCVERY

Il microcontrollore STM32F407 è basato su Cortex-M4 e può funzionare alla frequenza massima di 168MHz. Core Cortex-M4 offre unità processore a 32 bit e FPU meccaniche. FPU è destinato all'elaborazione digitale del segnale e ha funzionalità del processore DSP. Alla massima frequenza, raggiunge 210 DMIPS (Dyrestone-milioni di istruzioni al secondo). Il microcontrollore contiene anche una vasta gamma di dispositivi periferici:

- 2x USB OTG (On the Go).
- Loop bloccato audio di fase (loop di blocco della fase PLL).
- Contiene 15 interfacce di comunicazione (4 x UART, 2 x USART, 3 x SPI, 3 x I2C, 2x CAN, SDIO).
- Contiene 2x 12-bit DAC in 3 x 12-bit ADC velocità 2.4Msps.
- Ha 17 timer. Tutti i timer sono a 16 bit tranne due che sono a 32 bit.
- Interfacce per sistemi di memoria aggiuntivi, SRAM, NAND, etc.

Il controller ha integrato la memoria SRAM da 192kByte e la memoria FLASH da 1MByte.

Per la programmazione del controller, utilizzeremo lo strumento professionale KEIL uVision utilizzato in molti centri di sviluppo. Per la programmazione, useremo il linguaggio del programma C.

9.2. Impostazione ingressi e uscite -GPIO

GPIO è una abbreviazione per piedini ingresso uscita di tipo generico. I piedini sotto questo acronimo presentano ingressi o uscite al microcontrollore. Nei sistemi di microcontrollori, conosciamo diversi tipi di ingressi / uscite che sono generalmente suddivisi in discreti e analogici. Gli ingressi / uscite digitali presentano livelli logici (0 o 1) in cui gli ingressi / uscite analogici presentano una risoluzione del segnale multilivello. I segnali analogici sono limitati dalla tensione di alimentazione dei microcontrollori, precisamente dall'unità di alimentazione dell'unità AD all'interno del chip. Gli ingressi analogici sono generalmente etichettati come segnali AD (analogici in digitale) e analogici come segnali DA (da digitale ad analogico). I piedini del microcontrollore sono divisi in gruppi chiamati porte e sono etichettati con PA, PB, PC, PD, PE, dove ogni gruppo cattura pin fisici da 0 a 15 (ad esempio PA0-PA15, ecc.). L'impostazione del modulo GPIO e tutte le funzionalità corrispondenti devono essere incluse nella libreria ***stm32fxxx_gpio.c***.

- **Impostazione dell'uscita GPIO discreta.**

Determinazione dell'ingresso discreto:

```
GPIO_InitTypeDef GPIO_InitStructure; //Structure

//GPIO out
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13|
                              GPIO_Pin_14 | GPIO_Pin_15;// Configuration of pins 12-15
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;           // Output definition
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       // Module speed (2/10/50 MHz)
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;   //  push / pull (opposed to open drain)
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pulldown resistor is not
active
GPIO_Init(GPIOD, &GPIO_InitStructure);                  // Port D setting
```

Significato dell'etichetta:

```
GPIO_Pin_xx     -  Sequential pin number 1-15 on port -(A,B,C,D,E)
GPIOx           - Port name A,B,C,D…
```

- **Impostazione dell'ingresso GPIO discreto**

Determinazione dell'ingresso discreto:

```
  GPIO_InitTypeDef GPIO_InitStructure; //Structure
  //GPIO IN
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;         //  Configuration of pin 0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;      // GPIO as input
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Speed of GPIO module(2/10/50 MHz)
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;    // push/pull (opposed to open drain)
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pullup resistor is not
active
  GPIO_Init(GPIOA, &GPIO_InitStructure);           // Port A setting
```

- **ESEMPIO di uso dell'ingresso/uscita discreti:**

Sulla scheda controller, abbiamo quattro diodi che si trovano sulla porta D e sui piedini 12-15. Abbiamo anche un pulsante utente collegato alla porta A e al piedino 0. L'immagine 2 presenta lo schema di cablaggio dei diodi led e un pulsante.

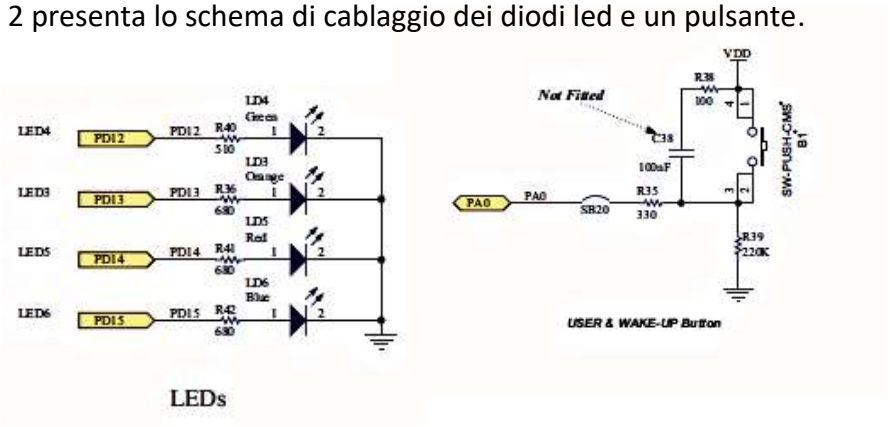

IMMAGINE 2: SCHEMA DI CABLAGGIO DI DIODI E PULSANTI.

Esempio di programma:

```
//MACRO
#define LED1           GPIO_Pin_12
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)

void RCC_Configuration(void);
void GPIO_Configuration(void);
int  button=0;


int main(void)
{

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();

    //Programm
  GPIO_SetBits(GPIOD, GPIO_Pin_12);
  GPIO_SetBits(GPIOD, GPIO_Pin_13);
  GPIO_ResetBits(GPIOD, GPIO_Pin_14);
  GPIO_ResetBits(GPIOD, GPIO_Pin_15);
```

```
    //Infinite loop
    while(1)
    {

            button = GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0); //Read input bit(Button
value)

                    if(button == 1)
                    {
                            GPIO_SetBits(GPIOD, GPIO_Pin_14);
                            GPIO_SetBits(GPIOD, GPIO_Pin_15);
                            LEDon(LED1); //MACRO function
                     }else
                     {
                            GPIO_ResetBits(GPIOD, GPIO_Pin_14);
                            GPIO_ResetBits(GPIOD, GPIO_Pin_15);
                            LEDoff(LED1); //MACRO function
                     }

                }//Infinite loop


}//End MAIN
```

```
//Function RCC_Configuration
void RCC_Configuration(void)
{
  /* GPIO clock enable */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
}

//Function GPIO_Configuration
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;


    //LED OUT
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
                                  GPIO_Pin_14 | GPIO_Pin_15; // configurate all I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;            // GPIO as output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;        // Speed of GPIO module
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // push / pull(opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pullup resistor is not
active
    GPIO_Init(GPIOD, &GPIO_InitStructure);               // Port D setting

     //BUTTON
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;         // PINA 0 configuration
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;      // GPIO as input
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //Speed of GPIO module (2/10/50 MH
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;   //push/pull (opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;  // pullup / pullup resistor is not
active
    GPIO_Init(GPIOA, &GPIO_InitStructure);           // Port A setting

}
```

9.3. Comunicazione UART

La comunicazione UART è più breve per il ricevitore-trasmettitore asincrono universale. È più spesso usato come comunicazione RS232. Per RS232 è necessario un circuito adattativo periferico che converta i livelli di segnale per la logica di livello TTL con un chip (MAX232) o utilizzi il modulo USB-Com-Port che emula la connessione standard RS232 tramite interfaccia USB (FTDi, CP2102, ecc.). Questo può essere visto nell'immagine 3. La scheda di sviluppo contiene quattro moduli USART interni che si trovano nelle seguenti porte:

UART 1 - Rx: PA10 Tx: PA9
UART 2 - Rx: PA3 Tx: PA2
UART 3 - Rx: PB11 Tx: PB10
UART 4 - Rx: PA1 Tx: PA0

- **Eesempio di uso USART2 :**

Nel progetto, includiamo la libreria stm32fxxx_usart.c. Colleghiamo PIN PA3 con pin RX sul modulo CP2012 e PIN PA2 con modulo pin TX CP2012

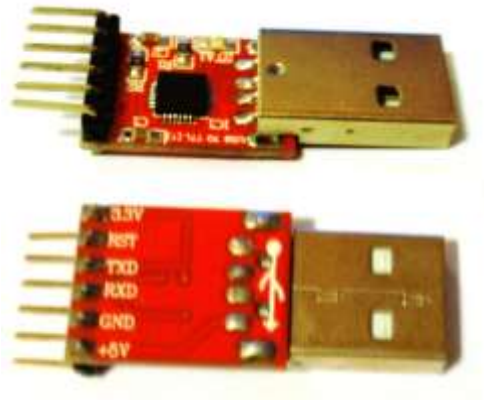

IMMAGINE 3: MODULO CP2012

```
  //MACRO

#define LED1 GPIO_Pin_15
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)


/******Functions*******************************/
void Delay(int ms);
void RCC_Configuration(void);
void GPIO_Configuration(void);
void USART2_Configuration(void);
void NVIC_Configuration(void); //Interrupt controller
int  button=0;
char buffer[200];
int j,i=0;



/*******************************************************************************
* Function Name  : Main
* Description    : Main function
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
int main(void)
{

  // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  NVIC_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON

    USART_Send_Str(USART2, "Hello from StmF4Discovery!\n\r" );


    //Infinite loop
    while(1)
    {
                if(Button==1)
                 {
                     LEDon(LED1);
                 }else
                 {
                      LEDoff(LED1);
                 }

     }//Infinite loop
```

```
}//End MAIN

/*******************************************************************************
* Function Name  : RCC_configuration
* Description    : Clock enable
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RCC_Configuration(void)
{
  /* --------------------------- System Clocks Configuration -----------------*/


   /* USART2 clock enable */
     RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
   /* GPIO clock enable */
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
}


/*******************************************************************************
* Function Name  : GPIO_Configuration
* Description    : Configures the different GPIO ports.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;


    //LED OUT
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 |
                                  GPIO_Pin_14 | GPIO_Pin_15; // Configurate all I/O pins for LED
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;          // GPIO as output
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;   // HitrSpeed of GPIO module (2/10/50 MHz)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      // push / pull (opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;     // pullup / pullup resistor is not active
    GPIO_Init(GPIOD, &GPIO_InitStructure);                 // Port D setting

  //BUTTON
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;         //  Configuration of PIN 0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;          //  GPIO as input
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;    //  Speed of GPIO module(2/10/50 MHz)
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;    //  push / pull (opposed to open drain)
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;    // pullup / pullup resistor is not active
  GPIO_Init(GPIOA, &GPIO_InitStructure);                // Port A setting

     // UART2 GPIO
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2| GPIO_Pin_3;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* Connect USART pins to AF */
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource2, GPIO_AF_USART2); // USART2_TX
  GPIO_PinAFConfig(GPIOA, GPIO_PinSource3, GPIO_AF_USART2); // USART2_RX


}


/*******************************************************************************
* Function Name  : USART_Configuration
* Description    : Configures the USAR module.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USART2_Configuration(void)
{
    USART_InitTypeDef USART_InitStructure;


  //USART2 configuration
  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
```

```
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  USART_Init(USART2, &USART_InitStructure);
  USART_Cmd(USART2, ENABLE);
  USART_ITConfig(USART2, USART_IT_RXNE, ENABLE); // Receive Interrupt enable

}
```

9.4. Conversione AD

La scheda di sviluppo consente 16 ingressi analogici esterni e due interni (sensore per la temperatura e la tensione della batteria). Il microcontrollore contiene tre convertitori AD separati, in cui ogni AD1-3 ha una risoluzione regolabile separatamente da 6, 8, 10 e 12 bit (il valore predefinito è 12 bit). Tre convertitori AD possono essere configurati su diverse unità GPIO e piedino con libreria aggiuntiva **stm32fxxx_adc.c**

Nomi di porte e piedini che si trovano nei convertitori AD:

Channel APB	ADC1 2	ADC2 2	ADC3 2
ADC Channel 0	PA0	PA0	PA0
ADC Channel 1	PA1	PA1	PA1
ADC Channel 2	PA2	PA2	PA2
ADC Channel 3	PA3	PA3	PA3
ADC Channel 4	PA4	PA4	PF6
ADC Channel 5	PA5	PA5	PF7
ADC Channel 6	PA6	PA6	PF8
ADC Channel 7	PA7	PA7	PF9
ADC Channel 8	PB0	PB0	PF10
ADC Channel 9	PB1	PB1	PF3
ADC Channel 10	PC0	PC0	PC0
ADC Channel 11	PC1	PC1	PC1
ADC Channel 12	PC2	PC2	PC2
ADC Channel 13	PC3	PC3	PC3
ADC Channel 14	PC4	PC4	PF4
ADC Channel 15	PC5	PC5	PF5

- **ESEMPIO di impostazione del convertitore AD:**

Lettura dei valori analogici sui canali 1, 4 e 5 con l'aiuto di due convertitori AD separati (AD1 e AD2).

```
int main(void)
{
   uint16_t read_AD[3];

  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  ADC_init();

  USART_Send_Str(USART2,"ADC test!\n\r");


    while(1)
     {

          read_AD[0] = Read_ADC(ADC1,1); //Read ADC value with ADC1 on channel 1 - PA1
          read_AD[1] = Read_ADC(ADC2,4); //Read ADC value with ADC2 on channel 4 - PA4
          read_AD[2] = Read_ADC(ADC2,5); //Read ADC value with ADC2 on channel 5 - PA5

      //Convert number to string
      sprintf(buffer,"PA1: %d   PA4: %d  PA5: %d
\n\r",read_AD[0],read_AD[1],read_AD[2]);

          USART_Send_Str(USART2,buffer);

          Delay(300);
          GPIO_ToggleBits(GPIOD, GPIO_Pin_12);

     }//Infinite loop


}//End MAIN

// ADC configuration
void ADC_init()
{
   GPIO_InitTypeDef        GPIO_InitStructure;
   ADC_InitTypeDef         ADC_InitStructure;
   ADC_CommonInitTypeDef   ADC_CommonInitStructure;

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1  | RCC_APB2Periph_ADC2 ,  ENABLE);

    //Setup GIPO pins
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_1 | GPIO_Pin_4 | GPIO_Pin_5;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOA, &GPIO_InitStructure);


    //Common settings of ADC
  ADC_CommonInitStructure.ADC_DMAAccessMode    = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_Mode             = ADC_Mode_Independent ;
  ADC_CommonInitStructure.ADC_Prescaler        = ADC_Prescaler_Div4;
    ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_10Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);


    //Common settings of ADC
  ADC_InitStructure.ADC_Resolution             = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode           = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode     = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge   = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_DataAlign              = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion        = 1;

    //Connect GPIO to the ADC
  ADC_Init(ADC1, &ADC_InitStructure);
  ADC_Init(ADC2, &ADC_InitStructure);
```

```
   //Enable ADC1 and ADC2
  ADC_Cmd(ADC1, ENABLE);
  ADC_Cmd(ADC2, ENABLE);

}

//READ ADC function
uint16_t Read_ADC(ADC_TypeDef* ADCx, uint8_t channel)
{
        uint16_t timeout = 0xFFF;

            ADC_RegularChannelConfig(ADCx, channel, 1, ADC_SampleTime_28Cycles);

            /* Start software conversion */
            ADCx->CR2 |= (uint32_t)ADC_CR2_SWSTART;

            /* Wait till done */
            while (!(ADCx->SR & ADC_SR_EOC)) {
                if (timeout-- == 0x00) {
                    return 0;
                }
            }

            /* Return result */
            return (uint16_t)ADCx->DR;
}
```

9.6. Modulazione a larghezza di impulso PWM

Modulazione della larghezza di impulso PWM è un tipo di modulazione con cui cambiamo il ciclo di lavoro alla frequenza del segnale costante. La modulazione dell'ampiezza dell'impulso è correlata al timer del microcontrollore. Il microcontrollore STM32F407VG contiene 14 timer (TIM1-TIM14) con risoluzione temporale 16 bit e due con risoluzione a 32 bit (TIM2 e TIM5). Con un microcontrollore, possiamo generare 14 diversi segnali PWM indipendenti con frequenza diversa e 32 segnali PWM in cui determinati gruppi hanno la stessa frequenza e ciclo di lavoro indipendente.

Ogni timer può essere fisicamente associato a GPIO dove determiniamo il pin di uscita sulla porta A, B, C, D, E quando si imposta PWM con timer e scelta del canale.

PWM Module Pinouts for ST				
	Channel 1	**Channel 2**	**Channel 3**	**Channel 4**
Timer1	_GPIO_MODULE_TIM1_CH1_PA8 _GPIO_MODULE_TIM1_CH1_PE9	_GPIO_MODULE_TIM1_CH2_PA9 _GPIO_MODULE_TIM1_CH2_PE11	_GPIO_MODULE_TIM1_CH3_PA10 _GPIO_MODULE_TIM1_CH3_PE13	_GPIO_MODULE_TIM1_CH4_PA11 _GPIO_MODULE_TIM1_CH4_PE14
Timer2	_GPIO_MODULE_TIM2_CH1_PA0 _GPIO_MODULE_TIM2_CH1_PA5	_GPIO_MODULE_TIM2_CH2_PA1 _GPIO_MODULE_TIM2_CH2_PB3	_GPIO_MODULE_TIM2_CH3_PA2 _GPIO_MODULE_TIM2_CH3_PB10	_GPIO_MODULE_TIM2_CH4_PA3 _GPIO_MODULE_TIM2_CH4_PB11
Timer3	_GPIO_MODULE_TIM3_CH1_PA6 _GPIO_MODULE_TIM3_CH1_PB4 _GPIO_MODULE_TIM3_CH1_PC6	_GPIO_MODULE_TIM3_CH2_PA7 _GPIO_MODULE_TIM3_CH2_PB5 _GPIO_MODULE_TIM3_CH2_PC7	_GPIO_MODULE_TIM3_CH3_PB0 _GPIO_MODULE_TIM3_CH3_PC8	_GPIO_MODULE_TIM3_CH4_PB1 _GPIO_MODULE_TIM3_CH4_PC9
Timer4	_GPIO_MODULE_TIM4_CH1_PB6 _GPIO_MODULE_TIM4_CH1_PD12	_GPIO_MODULE_TIM4_CH2_PB7 _GPIO_MODULE_TIM4_CH2_PD13	_GPIO_MODULE_TIM4_CH3_PB8 _GPIO_MODULE_TIM4_CH3_PD14	_GPIO_MODULE_TIM4_CH4_PB9 _GPIO_MODULE_TIM4_CH4_PD15
Timer5	_GPIO_MODULE_TIM5_CH1_PH10	_GPIO_MODULE_TIM5_CH2_PA1 _GPIO_MODULE_TIM5_CH2_PH11	_GPIO_MODULE_TIM5_CH3_PA2 _GPIO_MODULE_TIM5_CH3_PH12	_GPIO_MODULE_TIM5_CH4_PA3 _GPIO_MODULE_TIM5_CH4_PI0
Timer8	_GPIO_MODULE_TIM8_CH1_PC6 _GPIO_MODULE_TIM8_CH1_PI5	_GPIO_MODULE_TIM8_CH2_PC7 _GPIO_MODULE_TIM8_CH2_PI6	_GPIO_MODULE_TIM8_CH3_PC8 _GPIO_MODULE_TIM8_CH3_PI7	_GPIO_MODULE_TIM8_CH4_PC9 _GPIO_MODULE_TIM8_CH4_PI2
Timer9	_GPIO_MODULE_TIM9_CH1_PA2 _GPIO_MODULE_TIM9_CH1_PE5	_GPIO_MODULE_TIM9_CH2_PA3 _GPIO_MODULE_TIM9_CH2_PE6		
Timer10	_GPIO_MODULE_TIM10_CH1_PB8 _GPIO_MODULE_TIM10_CH1_PF6			
Timer11	_GPIO_MODULE_TIM11_CH1_PB9 _GPIO_MODULE_TIM11_CH1_PF7			
Timer12	_GPIO_MODULE_TIM12_CH1_PB14 _GPIO_MODULE_TIM12_CH1_PH6	_GPIO_MODULE_TIM12_CH2_PB15 _GPIO_MODULE_TIM12_CH2_PH9		
Timer13	_GPIO_MODULE_TIM13_CH1_PA6 _GPIO_MODULE_TIM13_CH1_PF8			
Timer14	_GPIO_MODULE_TIM14_CH1_PA7 _GPIO_MODULE_TIM14_CH1_PF9			

TABELLA 1: TABELLA DEI PIN DI USCITA PER DETERMINARE LA MODULAZIONE PWM A SECONDA DEL TIMER E DEL CANALE.

Determinazione del periodo di modulazione PWM (frequenza) e impostazione del valore ARR (auto reload register):

$$ARR(peridoa) = (TimerX\_frequency / PWM\_frequency) - 1$$

A seconda delle impostazioni del progetto, la frequenza del timer equivale alla metà della frequenza di clock principale (168 MHz), che è 84 MHz. Il valore ARR calcolato è il valore del rapporto di conversione massimo.

- **ESEMPIO di impostazioni di modulazione PWM**

Impostazione 10 kHz del segnale PWM su pin PD14 e PD15.

```
int main(void)
{
   int duty1=4200,duty2=0;

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  PWM Configuration();



    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON

    USART_Send_Str(USART2,"PWM test!\n\r" );


    while(1)
     {
          Delay(10);
          duty1+=10;
          duty2+=10;

           if(duty1>=8399)
           {duty1=0;}
           TIM4->CCR3=duty1; //Write duty cycle to register, pin PD14

           if(duty2>=8399)
           {duty2=0;}
           TIM4->CCR4=duty2; //Write duty cycle to register, pin PD15

      }//Infinite loop


}//End MAIN


//PWM configuration
void PWM_Configuration(void)
{

    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    TIM_OCInitTypeDef  TIM_OCInitStructure;


    //PWM on pin PD14 PD15
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_15; // configurate I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;       // GPIO as output (Alternate
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  // Speed of GPIO module (2/10/50)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;      // push / pull
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;     // pullup / pullup
    GPIO_Init(GPIOD, &GPIO_InitStructure);

      /* Connect pins PD14 and PD15 on timer TIM4 */
   GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4); // PD14 on Channel 3
   GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4); // PD14 on Channel 4



     /* TIM4 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    /* Time base configuration 10kHz */
  TIM_TimeBaseStructure.TIM_Period = 8399; // ARR+1=(TIM3
clock/PWM_frequncy)=(84MHz/10kHz)=8400
  TIM_TimeBaseStructure.TIM_Prescaler = 0;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
```

```
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);


  /* PWM1 Mode configuration: Channel3 */
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 4200; //Duty 50%
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OC3Init(TIM4, &TIM_OCInitStructure);
  TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);


    /* PWM1 Mode configuration: Channel4 */
    TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0; //Duty 0%
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OC4Init(TIM4, &TIM_OCInitStructure);
  TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);


    TIM_ARRPreloadConfig(TIM4, ENABLE);

   /* TIM4 enable counter */
   TIM_Cmd(TIM4, ENABLE);
}
```

9.7. Timer per le routine di interruzione

Le routine di interruzione sono una parte importante dei sistemi in tempo reale. Su questi sistemi e in base al sistema del microcontrollore conosciamo diversi tipi di routine di interruzione che vengono attivate da eventi diversi. Alcuni di questi sono: interruzione del timer, interrupt esterni (al cambio dei valori sul pin di input o output), interrupt del convertitore AD, interruzione in caso di eventi di comunicazione (ricezione dati, trasmissione dati SPI, I2C, UART, errore sul bus di comunicazione, eccetera.). Nel nostro caso, presenteremo interrupt di tempo, interrupt e interrupt esterni alla ricezione dei dati.

- **Impostazioni di interrupt sul timer 3 (TIM3)**

Periodi di temporizzazione pari a:

$$Period = (TimerX\_frequency / TimerX\_prescaler + 1)^{-1} \cdot TimerX\_period$$

Nel nostro caso viene utilizzato il fattore di ridimensionamento 2, il che significa che la frequenza di base del timer è uguale alla metà dell'orologio MCU principale. La frequenza di base del timer è di 84MHz.

Esempio

Imposta il periodo del timer su 10 ms. Scegli il valore TimerX_prescaler=209, TimerX_period=4000;

$$Period = (TimerX\_frequency / TimerX\_prescaler + 1)^{-1} \cdot TimerX\_period$$
$$= (84MHz / 209 + 1)^{-1} \cdot 4000 = 0.01s$$

Il tempo di interruzione del trigger è impostato con la frequenza del timer.

1. Abilitare l'orologio di sistema MCU sul timer determinato (***RCC_APB1ENR.TIM2EN = 1***)

```
/* TIMx clock enable */
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
```

2. Determinare il divario dell'orologio 'prescaler'. La frequenza dell'orologio è divisa MCU (168MHz) con il valore nel registro ***TIMx_PSC***

```
/* TIM3 clock prescaler */
TIM_TimeBaseStructure.TIM_Prescaler = 210;
```

3. Determinare il numero di valori contati ***TIMx_ARR (auto reload register)***, la cui frequenza dipende dal valore nel registro ***TIMx_PSC*** e frequenza MCU. Il registro ARR è uguale al periodo del timer.

```
/* TIM3 period*/
TIM_TimeBaseStructure.TIM_Period = 4000;
```

4. Abilitare il conteggio o il conto alla rovescia e scrivere la struttura nei registri del timer TM3.

```
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //Counter mode up
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);     //Structure entry
```

5. Abilitare il timer e modificare la routine di interrupt.

```
TIM_ClearFlag(TIM3, TIM_FLAG_Update);   //Clear flag
TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //Interrupt on auto reload register
TIM_Cmd(TIM3, ENABLE); //Enable timer
```

Per l'impostazione dei trigger di interrupt, è necessario impostare anche la priorità in NVIC (controller di interrupt vettoriale nidificato).

```
NVIC_InitTypeDef NVIC_InitStruct;

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);        //TIM3 Priority
NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

Le priorità possono essere selezionate sui livelli 0-4, dove il numero più basso significa che l'interruzione è più importante. Il sottoprogramma di interrupt viene eseguito nella funzione preimpostata **TIM3_IRQHandler(void).** I nomi delle funzioni di interruzione sono preimpostati e scritti nel documento **startup_stm32f40_41xxx.s**.

```
//MACRO
#define  LED1 GPIO_Pin_15
#define  LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define  LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define  Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)


int main(void)
{

    // MCU and GPIO setting
    RCC_Configuration();
    GPIO_Configuration();
    USART2_Configuration();
    TIM_Configuration();
    NVIC_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON

    USART_Send_Str(USART2,"TIMER test!\n\r");


    while(1)
                {



                }//Infinite loop


}//End MAIN

/*****************************************************************************
Timer3 configuration
*****************************************************************************/
void TIM_Configuration(void)
{

    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;



     /* TIM3 clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

    /* Time base configuration */
    TIM_TimeBaseStructure.TIM_Period = 4000;     //Timer3 period =
((TIM3_clock_freq)/(TIM_prescaler+1))^-1 * TIM_Preiod= ((84MHz)/(209+1))^-1 * 4000=10ms
    TIM_TimeBaseStructure.TIM_Prescaler =209;
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;   //Counter mode up
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);


    TIM_ClearFlag(TIM3, TIM_FLAG_Update);   //Clear flag
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE); //Interrupt on auto reload register
    TIM_Cmd(TIM3, ENABLE); //Enable timer
}


/*****************************************************************************
 NVIC-vector
*****************************************************************************/
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStruct;

        NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);        //

      //TIM3 Priority
```

```
        NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);

}

/*******************************************************************************
 Interrupt function
*******************************************************************************/
void TIM3_IRQHandler(void)
{

    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
    {
           TIM_ClearITPendingBit(TIM3, TIM_IT_Update);


              GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
             GPIO_ToggleBits(GPIOD, GPIO_Pin_13);


    }
}
```

- **Impostazione di interrupt sul pulsante A0**

Gli interrupt esterni attivati da dispositivi esterni, switch, protocolli di comunicazione, ecc. devono essere abilitati con una serie di voci in determinati registri di sistema. Questi interrupt sono suddivisi tra le unità EXTI0 - EXTI15 e sono determinati a livello di programmazione nel registro di controllo SYSCFG_EXTICRx. Ogni EXTI0-15 corrisponde a un numero di porta sequenziale 0-15, come mostrato nell'immagine 4.

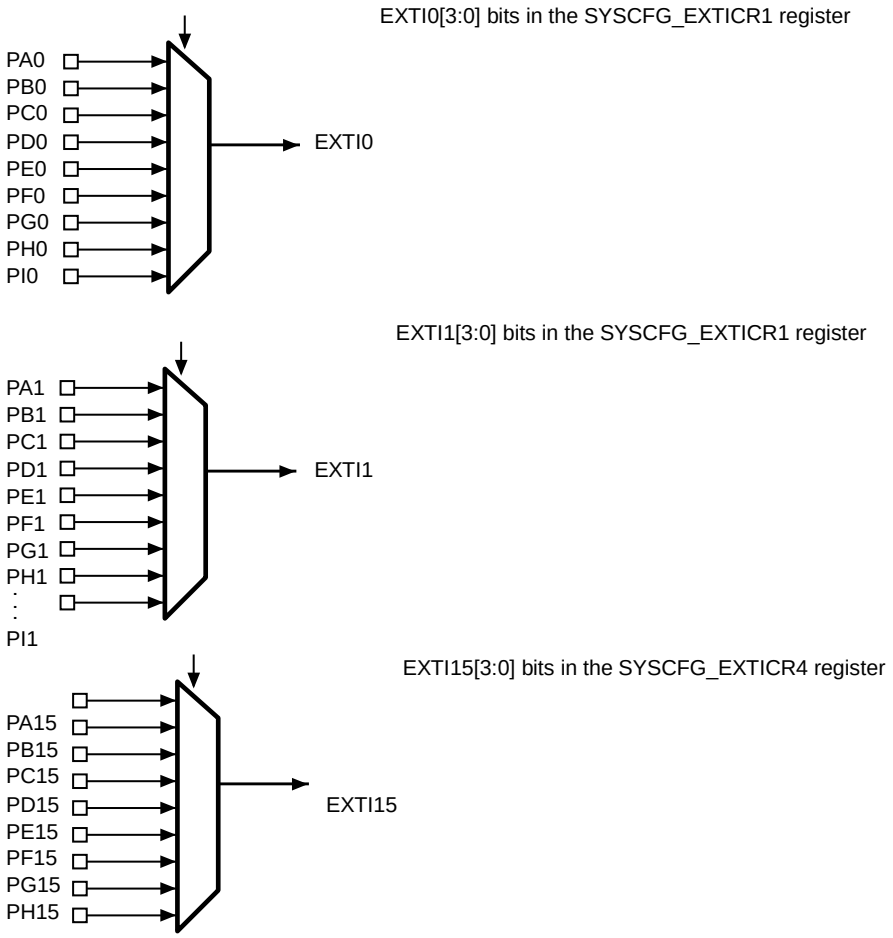

IMMAGINE 4: GRUPPI DI INTERRUPT ESTERNI

Esempio di interrupt esterno tramite il pulsante PA0.

Determina la maschera che inseriamo nel registro SYSCFG_EXTICRx. Useremo il pulsante sulla porta A e il numero sequenziale 0 (PA0). Come visto nell'immagine sopra, dobbiamo usare l'interruzione EXTI0. Impostazione del pin GPIO:

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Connessione pin PA0 a interrupt oculare:

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

Impostazione di interrupt esterno:

```
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
```

A seconda del tipo di trigger (che sia impostato nel registro come fronte positivo o negativo) si determina il tipo di interrupt.

```
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;

EXTI_Init(&EXTI_InitStructure);

EXTI_GenerateSWInterrupt(EXTI_Line0);
```

Esempio di impostazioni complete del programma:

```
//MACRO
#define LED1 GPIO_Pin_15
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)

int main(void)
{


  // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2_Configuration();
  EXTILine0 Configuartion();
  NVIC_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  //Green LED ON

    USART_Send_Str(USART2,"EXTI interrupt test!\n\r");


    while(1)
        {


        }//Infinite loop


}//End MAIN

/*******************************************************************************
 EXTI0 configuration
*******************************************************************************/
void EXTILine0_Configuartion(void)
{
       EXTI_InitTypeDef   EXTI_InitStructure;
       GPIO_InitTypeDef   GPIO_InitStructure;

        /* Enable SYSCFG clock */
     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

      //BUTTON
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;          //  configuration PIN_A 0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;           //  GPIO as input
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;    //  Speed of GPIO module
(2/10/50 MHz)
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;     //  push / pull (opposed to open
drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;    // pullup / pullup resistor is
not active
    GPIO_Init(GPIOA, &GPIO_InitStructure);                // Setting of port A

     /* Connect EXTI Line0 to PA0 pin */
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

        /* Configure EXTI Line0 */
    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
```

```
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

     /* Generate software interrupt: simulate a rising edge applied on EXTI0 line */
     EXTI_GenerateSWInterrupt(EXTI_Line0);

}

/*******************************************************************************
 NVIC configuration for EXTI0
*******************************************************************************/
void NVIC_Configuration(void)
{

    NVIC_InitTypeDef NVIC_InitStruct;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);         //

   /* Enable and set EXTI Line0 Interrupt  */
  NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
  NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStruct);

}


/*******************************************************************************
EXTI0 Interrupt function
*******************************************************************************/
void EXTI0_IRQHandler(void)
{

     if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {


            GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
            GPIO_ToggleBits(GPIOD, GPIO_Pin_13);

          EXTI_ClearITPendingBit(EXTI_Line0);
    }
```

9.8. Il codificatore incrementale

Il codificatore incrementale è un sensore del dispositivo che rileva spostamenti o rotazioni del sistema. In questo modo possiamo differenziare gli encoder incrementali lineari e rotativi. Molto spesso incontriamo encoder incrementali rotativi, come misuratori angolari, tachimetri o pulsanti di controllo su dispositivi elettronici. I codificatori rotativi sono spesso montati direttamente sull'asse del motore o sull'elemento rotante. La misurazione degli spostamenti e la rotazione con codificatore incrementale è una pratica industriale generale. La maggior parte dei sistemi integrati contiene già moduli per il collegamento del codificatore con il sistema di microcontrollori. Il principio di misurazione si basa sul principio del conteggio degli incrementi causato dallo spostamento del valore misurato. I sistemi più precisi hanno da due a quattro canali e l'indice della rotazione completa (A, B, A ', B', Indice), quelli più semplici solo uno (A). Il funzionamento incrementale del codificatore è presentato nell'immagine 5:

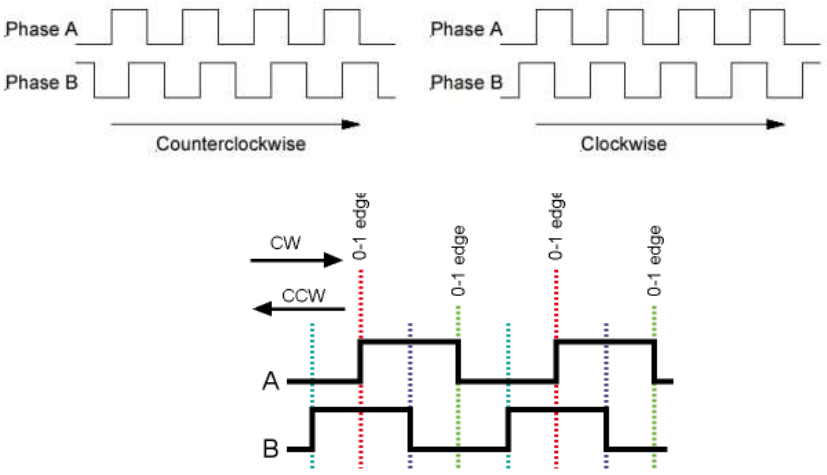

IMMAGINE 5: PRINCIPIO DI FUNZIONAMENTO DEL CODIFICATORE INCREMENTALE

Nell'inizializzazione del codificatore incrementale, è importante collegare il timer, analogamente ai pin PWM e GPIO. In questo caso, è possibile utilizzare solo i pin dei canali 1 e 2 (visualizzati nella tabella 1). I pin dei canali 3 e 4 non possono essere utilizzati nel modo codificatore incrementale. In questo caso, il conteggio del timer non è correlato alla risoluzione e all'ora del timer predefiniti, ma solo al codificatore incrementale. Per questo, il timer deve essere impostato sulla modalità codificatore incrementale con funzione 'TIM_EncoderInterfaceConfig(xxx)'. Il timer può essere utilizzato in un circuito con un interrupt o senza, il che significa che l'interrupt viene attivato a seconda di un certo numero di incrementi preimpostati.

Il seguente esempio di codice mostra le impostazioni dell'encoder sul pin PC6 (A) e PC7 (B) con il timer TIM3 e con l'interrupt con 3200 incrementi contati.

Esempio di codice:

```
 int main(void)
{
  int16_t encoder;
  char Buffer[5];

   RCC_Configuration();
   ENCODER_Configuration();
   NVIC_Configuration();

    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON
    USART_Send_Str(USART2,"ENCODER TEST\n\r");

    //Infinite loop
    while(1)
      {

              encoder = TIM3->CNT; //Read encoder value
              sprintf(Buffer,"%d ",encoder); //Convert to string
              USART_Send_Str(USART2,Buffer);USART_Send_Str(USART2,"\n\r"); //Send data
USART2
              Delay(200);

      }//END - Infinite loop
```

```
}//END - MAIN


/*******************************************************************************
ENCODER_Configuration
*******************************************************************************/
void ENCODER_Configuration()
{

        TIM_TimeBaseInitTypeDef TIM3_TimeBaseStructure;
        GPIO_InitTypeDef GPIO_InitStructure;

        /*ENCODER PIN Configuration ( PC6 & PC7 )*/
        RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC , ENABLE);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7; //PIN PC6-T1(A) PC7-T2(B)
        GPIO_InitStructure.GPIO_Mode =  GPIO_Mode_AF;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
        GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;  //Trigger only on + voltage
        GPIO_InitStructure.GPIO_OType= GPIO_AF_TIM3;
        GPIO_Init(GPIOC, &GPIO_InitStructure);

        /*Connect PIN PC6 and PC7 on TIM 3 (Channel 1&2)*/
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM3); //Enable GPIO PC6 To alternate
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource7, GPIO_AF_TIM3); //Enable GPIO PC7 To alternate


        /*TIM3 setting*/
        TIM_DeInit(TIM3);
        RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
        TIM3_TimeBaseStructure.TIM_Period = 3200;   //Number of encoder counts
        TIM3_TimeBaseStructure.TIM_Prescaler = 0;
        TIM3_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
        TIM3_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
        //ENCODER MODE
       TIM_EncoderInterfaceConfig(TIM3,
                                  TIM_EncoderMode_TI12, //Count on both channel A in B
                                   TIM_ICPolarity_Rising,
                                   TIM_ICPolarity_Rising);
        TIM_TimeBaseInit(TIM3, &TIM3_TimeBaseStructure);


          TIM3->CNT=0; //Inital value of the encoder timer

        //Enable upadate flag
         TIM_ClearFlag(TIM3, TIM_FLAG_Update);
        //Timer interrupt enable, for one revolution
        TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
        //Start timer TIM3
        TIM_Cmd(TIM3, ENABLE);

}


/*******************************************************************************
*  Interrupt  priority configuration
*******************************************************************************/
void NVIC_Configuration(void)
{
        NVIC_InitTypeDef NVIC_InitStruct;


        //Timer 3 Priority, with encoder
        NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
        NVIC_InitStruct.NVIC_IRQChannelSubPriority = 1;
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
        NVIC_Init(&NVIC_InitStruct);


}



/*******************************************************************************
Encoder interrupt function, occur after 3200 increments
```

```
*******************************************************************************/
    void TIM3_IRQHandler(void)
    {

                    if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET)
                    {
                            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);



                    }
    }
```

9.8. Comunicazione SPI con accelerometro LIS302DL

Sull'asse di sviluppo si trova l'accelerometro a triplo asse LIS302DL con campo di misura tra ± 2 e ± 8g e risoluzione a 8 bit. Il sensore su MCU è collegato al bus SPI tramite i pin PA7 (MOSI), PA6 (MISO), PA5 (SCK), PE3 (CS o SS). SPI (interfaccia periferica seriale) può funzionare in modalità a 3 o 4 fili. SC / SS (selezione chip, selezione slave). Per l'inizializzazione del sensore, utilizzeremo le librerie esistenti, dove è fondamentale che il sensore sia calibrato correttamente. Imposta la gamma di sensibilità e il range di frequenza. I dati della libreria sono già valori calcolati mm / s2 = mg.

Esempi di programma:

```
int main(void)
{
    TM_LIS302DL_LIS3DSH_t Axes_Data;
    char Buffer[200];

    // MCU and GPIO setting
  RCC_Configuration();
  GPIO_Configuration();
  USART2 Configuration();
  NVIC_Configuration();


  GPIO_SetBits(GPIOD, GPIO_Pin_12);  //Green LED ON

  USART Send Str(USART2,"Hello from StmF4Discovery!\n\r");


     /*Initialization of LIS302DL*/
     TM_LIS302DL_LIS3DSH_Init(TM_LIS3DSH_Sensitivity_2G, TM_LIS3DSH_Filter_800Hz);

    //Infinite loop
    while(1)
    {
         TM_LIS302DL_LIS3DSH_ReadAxes(&Axes_Data); //READ data from SPI
         /*Write to string*/
         sprintf(Buffer,"x: %d  y: %d   z: %d ",Axes_Data.X, Axes_Data.Y, Axes_Data.Z);
         /*Send string*/
         USART_Send_Str(USART2,Buffer); USART_Send_Str(USART2,"\n\r");

        Delay(200);
    }//Infinite loop


}//Konec MAIN
```

9.9. Porta seriale virtuale VCP

Sulla scheda di test fornita non abbiamo un modulo speciale per la comunicazione USART (FTDI, CP2012 ..), quindi useremo l'USB 2.0 come modulo nel microcontrollore, che emuleremo come una porta COM virtuale. Se si desidera utilizzare il modulo USB come VCP, è necessario includere una libreria aggiuntiva per produttore ST-Microelectronics. Nell'esempio seguente è riportato il codice programma che mostra l'impostazione dell'interfaccia USB-VSP sulla scheda di sviluppo STM32 DiscoveryF407. La spina Micro USB si trova sui pin PA11 e PA12 e funziona come linea dati (D-D +). Per facilitare la comprensione e l'utilizzo dell'interfaccia USB, abbiamo preparato una libreria riprogettata di Tilen Majerle che è liberamente accessibile al link [http://stm32f4-discovery.com/tag/tilen-majerle/.](http://stm32f4-discovery.com/tag/tilen-majerle/) Il programma descrive i dati ricevuti con il segno finale '%' e una visualizzazione del contenuto ricevuto con un clic sul pulsante. Con la funzione USBD_Init () e l'argomento USE_USB_OTG_FS inizializziamo l'interfaccia USB-VCP sui pin PA10, PA11, PA12, PA13.

Esempio di programma:

```
USB_OTG_CORE_HANDLE USB_OTG_dev; //USB VCP structure

int main(void)
{
   uint8_t b;
   uint8_t i=0;
   char USB_read[100];

   //MCU and GPIO setting
   RCC_Configuration();
   GPIO_Configuration();


    GPIO_SetBits(GPIOD, GPIO_Pin_12);  //  LED ON
    GPIO_SetBits(GPIOD, GPIO_Pin_13);  // LED ON


      /* Initialize USB VCP */
       USBD_Init(  &USB_OTG_dev,
                   USE_USB_OTG_FS
                   USB_OTG_FS_CORE_ID,
                   &USR_desc,
                   &USBD_CDC_cb,
                   &USR_cb);

    //Infinite loop
    while(1)
    {


            if(Button==1)//Send received data on the button press
            {
                   if (TM_USB_VCP_Getc(&b) == TM_USB_VCP_DATA_OK) {


                           USB_read[i]=(char)b;
                           i++;

                              if(b=='%')//Terminal character
                               {
                                    USB_read[i-1]=' ';
                                    TM_USB_VCP_Puts(USB_read);TM_USB_VCP_Puts("\n\r");
//Send data
```

```
                                    for(int j=0;j<=i;j++) //Clear out buffer
                                    USB_read[j]=' ';

                                    i=0;  //Clear array index
                                }
                        }

                }


        }//Infinite loop-END


}//MAIN - END
```