Ecodesign-ul dispozitivelor electronice

UNIT 9: Sisteme de microcontroler partea 2

Autor: Andrej Sarjaš

Rezumatul capitolului:

–Sisteme în timp real
–Componente ale timpului real
–Proiectarea programelor de sisteme în timp real

Funded by the Erasmus+ Programme of the European Union

Competențe Ecoinnovation pentru designeri europeni, numărul proiectului: 562573-PPE-1-2015-1-SI-EPPKA2-SSA. Acest proiect a fost finanțat cu sprijinul din partea Uniunii Europene. Această publicație (comunicare) este responsabilitatea exclusivă a autorului său. Comisia nu este responsabilă pentru eventuala utilizare a informațiilor pe care le conține

9.1. Utilizarea plăcii de microcontroler SMT32F4

Dezvoltarea plăcii STM32F4 la prețuri reduse și performantă, potrivită pentru dezvoltarea rapidă a aplicațiilor încorporate și testarea unui microcontroler puternic ARM STM32F407VG cu unitatea FPU încorporată (unitatea cu puncte variabile FPU, unitatea de calcul cu numere variabile de virgulă). Placa de dezvoltare conține microcontroler ARM STM32F407VG, programator JTAG, ST-link V2, accelerometru, codec audio, microfon, 4 LED-uri de control, comutator și conector micro-USB. Imaginea 1 prezintă schema plăcii STM32F4.

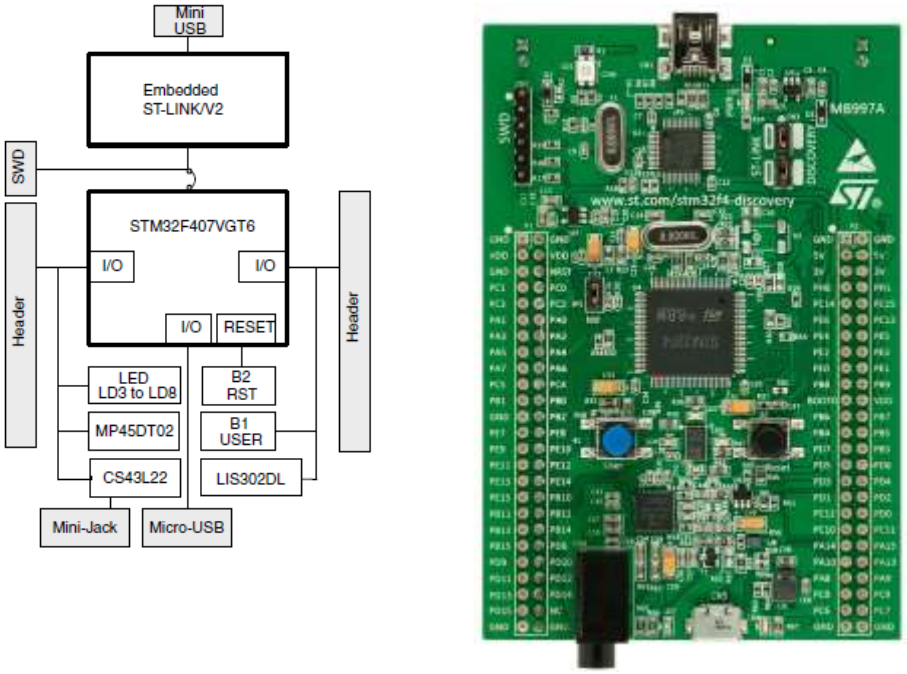


IMAGE 1: DEVELOPMENT BOARD STM32F4DISOCVERY

Microcontrolerul STM32F407 se bazează pe core Cortex-M4 și poate funcționa pe frecvență maximă de 168MHz. Core Cortex-M4 oferă un procesor pe 32 de biți și un FPU mecanic. FPU este destinat procesării digitale a semnalului și are funcționalități ale procesorului DSP. La frecvența maximă, atinge 210 DMIPS (instrucțiuni Dyrestone-milioane de instrucțiuni pe secundă). Microcontrolerul conține, de asemenea, o gamă largă de dispozitive periferice:

- 2x USB OTG (Pe traseu).
- Bucle cu fază audio blocată (buclă de blocare fază PLL).
- Conține 15 interfețe de comunicație (4 x UART, 2 x USART, 3 x SPI, 3 x I2C, 2x CAN, SDIO).
- Conține 2x DAC pe 12 biți în viteză ADC de 3 x 12 biți 2.4Mps.

• Are 17 cronometre. Toate cronometrele sunt de 16 biți, cu excepția a două care sunt de 32 de biți.

• Interfețe pentru sisteme suplimentare de memorie, SRAM, NAND etc.

Controlerul are integrată memorie SRAM de 192kByte și memorie FLASH de 1MB.

Pentru controlerul de programare, vom folosi instrumentul profesional KEIL uVision, folosit în multe centre de dezvoltare. Pentru programare, vom folosi limbajul programelor C.

9.2. Setarea intrărilor și ieșirilor separate - GPIO

GPIO este o abreviere pentru pini de intrări / ieșiri cu destinație generală. Sub acest acronim prezintă intrări sau ieșiri către microcontroler. În sistemele de microcontrolere, știm mai multe tipuri de intrări / ieșiri care sunt în general împărțite în mod separat și analogic. Intrările / ieșirile separate prezintă nivele logice (0 sau 1), intrările / ieșirile analogice prezintă rezoluția semnalului pe mai multe niveluri. Semnalele analogice sunt limitate de tensiunea de alimentare a microcontrolerelor, exact prin alimentarea cu energie a unității AD în interiorul cipului. Intrările analogice sunt de obicei etichetate ca semnale AD (analog-digital) și ieșiri analogice ca semnale DA (digital la analog). Microfoanele sunt împărțite în grupuri care sunt numite porturi și sunt etichetate cu PA, PB, PC, PD, PE unde fiecare grup captează pinii fizici de la 0 la 15 (de exemplu PA0-PA15 etc.). Setarea modulului GPIO și toate funcționalitățile corespunzătoare trebuie incluse în biblioteca stm32fxxx_gpio.c.

- **Setarea ieșirii separate GPIO.**

Determinarea intrărilor separate:

```
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
```

Semnificația etichetei:

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

- **Setarea intrării separate GPIO**

Determinarea intrărilor separate:

```
  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
```

- **EXEMPLU de utilizare a intrărilor / ieșirilor separate:**

Pe panoul de comandă, avem patru diode care sunt pe portul D și pinii 12-15. De asemenea, avem un buton de utilizator care este conectat la portul A și pinul 0. Imaginea 2 prezintă schema de conectare a diodelor led și un buton.

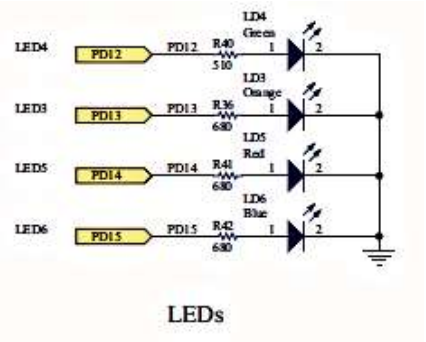


IMAGE 2: WIRING SCHEME OF DIODES AND BUTTONS.

Exemplu de program:

```
//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. Comunicarea UART

Comunicarea UART este mai scurtă pentru receptorul-transmițător universal asincron. Este cel mai adesea folosit ca o comunicare RS232. Pentru RS232 avem nevoie de un circuit adaptiv periferic care convertește nivelurile de semnal pentru logica nivelului TTL cu un cip (MAX232) sau utilizează modulul USB-Com-Port care emulează conexiunea standard RS232 prin interfața USB (FTDi, CP2102, etc.). Acest lucru poate fi văzut în imaginea 3. Bordul de dezvoltare conține patru module interne USART care se află în următoarele porturi:

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

- **Exemplu de utilizare USART2:**

În proiect, includem biblioteca stm32fxxx_usart.c. Conectăm PIN-ul PA3 cu pinul RX pe modulul CP2012 și PIN-ul PA2 cu modulul PIN TX CP2012.

IMAGE 3: MODULE 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. Conversia AD

Placa de dezvoltare permite 16 intrări analogice externe și două interne (senzor pentru temperatură și tensiune baterie). Microcontrolerul conține trei convertoare AD separate, în care fiecare AD1-3 are rezoluție reglabilă separat de la 6, 8, 10 până la 12 biți (valoarea implicită este de 12 biți). Trei convertoare AD pot fi configurate la diferite unități GPIO și ace cu bibliotecă suplimentară stm32fxxx_adc.c

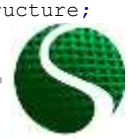
Numele de porturi și ace care se află în convertoarele AD:

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

- **EXEMPLU de setări ale convertorului AD:**

Citirea valorilor analogice pe canalele 1, 4 și 5 cu ajutorul a două convertoare AD (AD1 și AD2) separate.

```
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;
}
```

9.5. PWM pentru modularea impulsului

Modularea lățimii impulsului PWM este tipul de modulație cu care schimbăm ciclul de sarcină la frecvența semnalului constant. Modularea lățimii impulsurilor este legată de cronometrul microcontrolerului. Microcontrolerul STM32F407VG conține 14 timere (TIM1-TIM14) cu rezoluție de timp 16 biți și două cu rezoluție pe 32 de biți (TIM2 și TIM5). Cu un microcontroler, putem genera 14 semnale independente PWM

independente cu frecvență diferită și 32 de semnale PWM unde anumite grupuri au aceeași frecvență și un ciclu de funcționare independent.

Fiecare cronometru poate fi legat fizic la GPIO unde determinăm pinul de ieșire pe porturile A, B, C, D, E când setăm PWM cu temporizator și alegerea canalului.

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			

TABLE 1: TABLE OF OUTPUT PINS FOR DETERMINING PWM MODULATION DEPENDING ON THE TIMER AND CHANNEL.

Determinarea perioadei de modulare PWM (frecvență) și setarea valorii ARR (registru auto reîncărcare):

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

În funcție de setările proiectului, frecvența temporizatorului este egală cu jumătate din frecvența ceasului principal (168MHz), care este de 84MHz. Valoarea ARR calculată este valoarea raportului de conversie maxim.

- **EXEMPLU de setări de modulare PWM**

Setarea de 10 kHz a semnalului PWM pe pinul PD14 și 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);
}
```

9.6. Întreruperea rutinei temporizatorului

Rutinele de întrerupere reprezintă o parte importantă a sistemelor în timp real. Pe aceste sisteme și în funcție de sistemul de microcontrolere, știm diferite tipuri de rutine de întrerupere declanșate de evenimente diferite. Unele dintre acestea sunt: întreruperea temporizării, întreruperile externe (la schimbarea valorilor pe pinul de intrare sau ieșire), întreruperea conversiei AD, întreruperea evenimentelor de comunicație (primirea datelor, transmiterea datelor SPI, I2C, UART, etc). În cazul nostru, vom prezenta o întrerupere de timp, o întrerupere externă și o întrerupere la primirea datelor.

- **Setările de întrerupere a temporizatorului 3 (TIM3)**

Perioadele de timp ale temporizatorului:

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

În cazul nostru se folosește factorul de scalare 2, ceea ce înseamnă că frecvența de bază a cronometrului este egală cu jumătate din ceasul MCU principal. Frecvența ceasului de bază al ceasului este de 84MHz.

Exemple

Setați perioada cronometrului la 10 ms. Alegeți valoarea TimerX_prescaler=209, TimerX_period=4000;

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

Timpul de întrerupere al declanșatorului este setat cu frecvența temporizatorului.

1. Activați ceasul de sistem MCU pe temporizator determinat (***RCC_APB1ENR.TIM2EN = 1***)

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

2. Determinați divizarea "prescaler" a ceasului. Frecvența ceasului este divizată MCU (168MHz) cu valoarea în registrul TIMx_PSC

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

3. Determinați numărul de valori calculate TIMx_ARR (registrul de reîncărcare automată), a cărui frecvență depinde de valoarea din registrul TIMx_PSC și frecvența MCU. Registrul ARR este egal cu perioada de timp.

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

4. Activați numărarea modurilor și scrieți structura în registrele temporizatorului TM3.

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

5. Activați cronometrul și editați rutina de întrerupere.

```
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
```

Pentru setarea declanșatoarelor de întrerupere, trebuie să stabilim prioritățile în NVIC ("adăpostul" controlerului de întrerupere vectorial).

```
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);
```

Prioritățile pot fi selectate la nivelele 0-4, unde numărul mai mic înseamnă că întreruperea este mai importantă. Întreruperea subprogramului este executată în funcția pre-pregătită TIM3_IRQHandler (liber). Numele funcțiilor de întrerupere sunt presetate și scrise în documentul 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);


    }
}
```

- **Setarea întreruperii cu butonul A0**

Întreruperile externe care sunt declanșate de dispozitive externe, comutatoare, protocoale de comunicații etc. trebuie activate cu un set de intrări în anumite registre de sistem. Aceste întreruperi sunt împărțite între unitățile EXTI0 - EXTI15 și sunt determinate programabil în registrul de control SYSCFG_EXTICRx. Fiecare EXTI0-15 corespunde unui număr de port secvențial 0-15, așa cum se vede în imaginea 4.

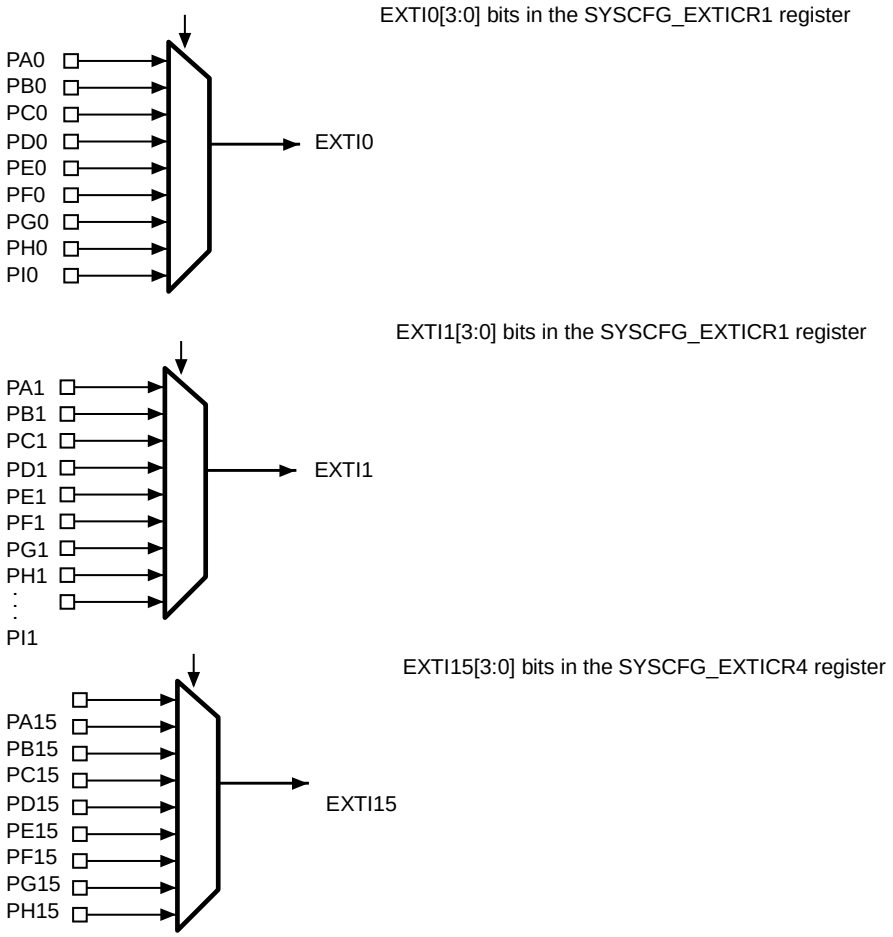


IMAGE 4: GROUPS FOR EXTERNAL INTERRUPTS

Exemplu pentru întreruperea externă prin butonul PA0.

Determinați masca pe care o introducem în registrul SYSCFG_EXTICRx. Vom folosi butonul pe portul A și numărul secvențial 0 (PA0). După cum se vede în imaginea de mai sus, trebuie să folosim întreruperea EXTI0. Setarea pinului 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);
```

PA0 conexiune pin pentru întrerupere externă:

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

Setarea întreruperii externe:

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

În funcție de tipul declanșatorului (dacă acesta este setat în registru ca margine pozitivă sau negativă), determinăm tipul de întrerupere.

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

EXTI_Init(&EXTI_InitStructure);

EXTI_GenerateSWInterrupt(EXTI_Line0);
```

Exemplu de setări complete ale programului:

```
//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);
    }
```

9.7. Codificator incremental

Codificatorul incremental este un senzor de dispozitiv care detectează schimbările de sistem sau rotațiile. În acest fel putem diferenția codificatorii incrementali liniari și rotativi. Foarte adesea întâlnim codificatoare incrementale rotative, cum ar fi unghiometre, vitezometre sau butoane de control pe dispozitive electronice. Dispozitivele rotative de codificare sunt adesea montate direct pe axa motorului sau pe elementul rotativ. Măsurarea mișcărilor și rotației cu codificatoare incrementale este o practică industrială generală. Cele mai multe sisteme încorporate conțin deja module pentru conectarea codificatorului cu sistemul de microcontroler. Principiul de măsurare se bazează pe principiul numărării incrementului care este cauzat de schimbarea valorii măsurate. Sistemele mai precise au două până la patru canale și un indice al rotației complete (A, B, A ', B', Index), cele mai simple doar una (A). Funcționarea incrementală a codificatorului este prezentată în imaginea 5:

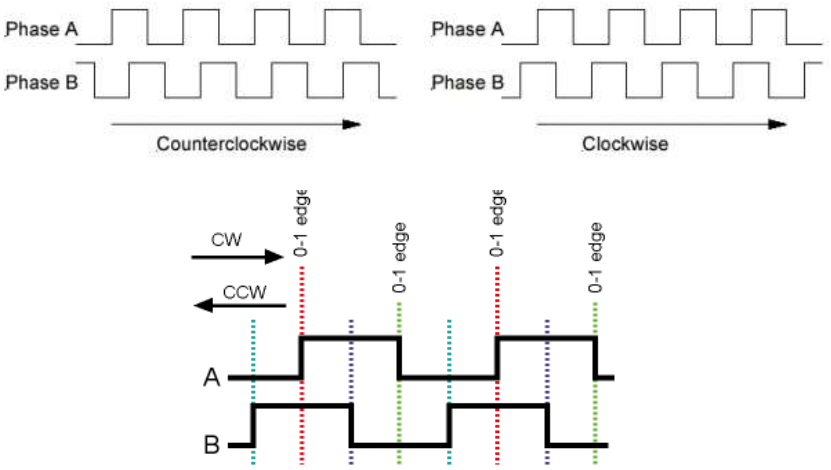


IMAGE 5: INCREMENT ENCODER FUNCTIONING PRINCIPLE

La inițializarea codificatorului incremental, este important să conectați temporizatorul, la fel ca pinii PWM și GPIO. În acest caz, pot fi utilizate numai pini din canalele 1 și 2 (văzute în tabelul 1). Pini din canalele 3 și 4 nu pot fi utilizați în codificatorul incremental al modulului. În acest caz, numărarea temporizatorului nu este legată de rezoluția predefinită a temporizatorului și de ceas, ci numai de codificatorul incremental. Pentru aceasta, temporizatorul trebuie să fie setat la modul codificatorului incremental cu funcția 'TIM_EncoderInterfaceConfig (xxx)'. Temporizatorul poate fi utilizat într-un circuit cu o întrerupere sau fără, ceea ce înseamnă că întreruperea este declanșată în funcție de un anumit număr de incremente prestabilite.

Următorul exemplu de cod arată setările codorului pe pinul PC6 (A) și PC7 (B) cu temporizatorul TIM3 și cu întrerupere cu 3200 incremente numărate.

Exemplu de cod:

```
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. Comunicarea SPI cu accelerometrul LIS302DL

Pe placa de dezvoltare este localizat accelerometrul tripolar LIS302DL cu interval de măsurare între ± 2 și ± 8g și rezoluție pe 8 biți. Senzorul de pe MCU este conectat la magistrala SPI prin pinii PA7 (MOSI), PA6 (MISO), PA5 (SCK), PE3 (CS sau SS). SPI (interfața periferică serial) poate funcționa în mod 3 sau 4 fire. SC / SS (selecție cip, selecție slave). Pentru inițializarea senzorilor, vom folosi bibliotecile existente, unde este esențial ca senzorul să fie calibrat corect. Setați intervalul de sensibilitate și intervalul de frecvență. Datele bibliotecii sunt deja calculate la valoarea mm / s2 = mg.

Exemplu de program:

```
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. Port serial VCP virtual

Pe placa de testare dată nu avem un modul special pentru comunicarea USART (FTDI, CP2012 ..), prin urmare, vom folosi USB 2.0 ca modul în microcontroler, pe care îl vom emula ca un port virtual COM. Dacă vrem să folosim modulul USB ca VCP, atunci trebuie să includem o bibliotecă suplimentară de către producătorul ST-Microelectronics. În următorul exemplu este codul de program care arată setarea interfeței USB-VSP pe placa de dezvoltare STM32 DiscoveryF407. Conectorul micro USB este situat pe pinii PA11 și PA12 și funcționează ca linie de date (D-D +). Pentru înțelegerea și utilizarea mai ușoară a interfeței USB, am pregătit o bibliotecă redesenată de Tilen Majerle, care este accesibilă gratuit la http://stm32f4-discovery.com/tag/tilen-majerle/. Programul descrie primirea datelor cu semnul final '%' și afișarea conținutului primit cu un clic pe buton. Cu funcția USBD_Init () și argumentul USE_USB_OTG_FS inițializăm interfața USB-VCP pe pinii PA10, PA11, PA12, PA13.

Exemplu de program:

```
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
```