Ekološko snovanje elektronskih naprav

ENOTA 9: Mikrokrmilniški sistemi 2.del

Ime avtorja: **Andrej Sarjaš**

Vsebina poglavja:

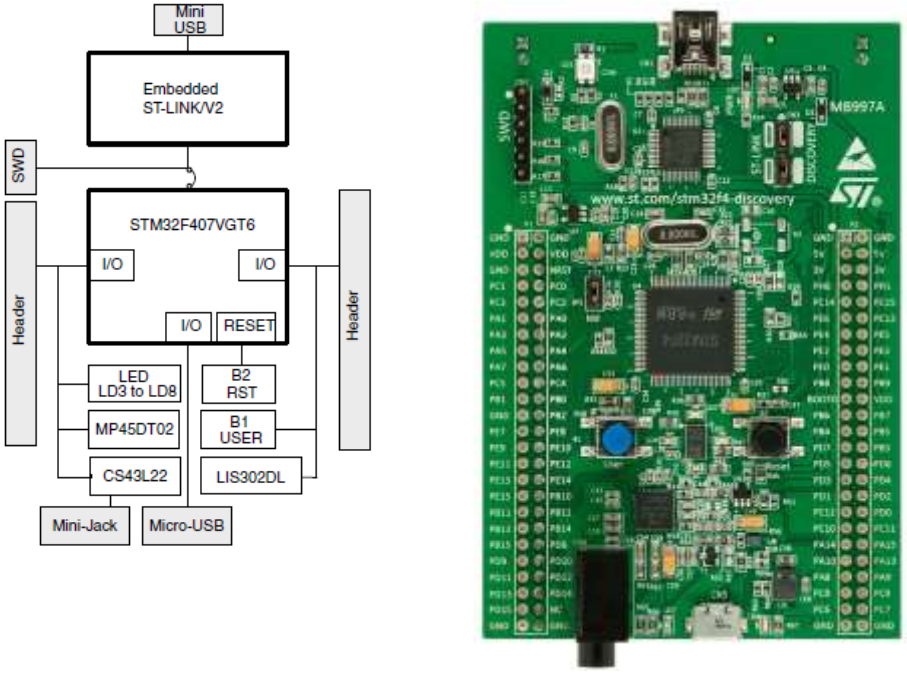
–Sistemi realnega časa
–Komponente realnega časa
–Snovanje programa v sistema realnega časa

Funded by the Erasmus+ Programme of the European Union

Ecoinnovation Skills for European Designers, Številka projekta: 562573-EPP-1-2015-1-SI-EPPKA2-SSA. Opomba: Vsebina publikacije je izključno odgovornost avtorja in v nobenem primeru ne predstavlja stališč Evropske komisije. Komisija ni odgovorna za kakršno koli uporabo gradiva, ki bi lahko bila storjena na podlagi vsebine, ki jih gradivo vsebuje.

9.1. Uporaba mikrokrmilniške plošče SMT32F4discovery

Razvojna plošča **STM32F4Discovery** je nizko cenovna in zmogljiva mikrokrmilniša platforma, primerna za hiter razvoj vgrajenih aplikacij in testiranje zmogljivega mikrokontrolerja ARM STM32F407VG z vgrajeno FPU enoto ('FPU – floating point unit', enota za računanje s števili s plavajočo vejico). Razvojna plošča vsebuje mikrokrmilni ARM STM32F407VG, JTAG programator ST-link V2, pospeškometer, audio kodek, mikrofon, 4 kontrolne led diode, stikalo in mikro-USB priključek. Slika 1. prikazuje razvojno ploščo STM32F4Disocvery.



Slika 1. Razvojna plošča STM32F4Disocvery.

Jedro mikrokrmilnik STM32F407 tmemelji na jedru Cortex-M4 in lahko teče na maksimalni frekvenci 168MHz. Jedro Cortex-M4 ponuja 32bit procesorsko enoto ter strojni FPU. FPU je namenjen za digitalno procesiranje signal in ima funkcionalnost DSP procesorja. Pri maksimalni frekvenci doseže 210 DMIPS-ov (DMIPS- 'Dyrestone- Milion Instructions per Second'). Prav tako krmilnik vsebuje velik nabor perifernih naprav.

- 2x USB OTG (On the Go).
- Audio fazno zaklenjeno zanko (PLL- 'Phase lock loop').
- Vsebuje 15 komunikacijskih vmesnikov (4 x UART, 2 x USART, 3 x SPI, 3 x I2C, 2x CAN, SDIO).
- Vsebuje 2x 12bit DAC in 3 x 12bit ADC hitrosti 2.4Msps.
- Ima 17 časovnikov. Vsi časovniki so 16biti razen dveh, ki sta 32bitna.
- Vmesnik za dodatne pomnilniške sisteme, SRAM, NAND itd..

Krmilnik ima integrirano 192kByte SRAM pomnilnika in 1MByte FLASH pomnilnika.

Za programiranje krmilnika bomo uporabili profesionalno orodje KEIL uVision, ki je prisotno v mnogih razvojnih centrih. Za programiranje bomo uporabili programski jezik C.

9.2. Nastavitev diskretnih vhodov in izhodov-GPIO

GPIO –general purpose Inputs/Outputs pins. Pini pod akronimom GPIO predstavljajo izhode ali vode v mikrokrmilnik. Na mikrokrmilniških sistemih poznamo več različni tipov izhod/vhodov, katere na splošno delimo na diskretne in analogne. Diskretni vhodi/izhodi predstavljajo logična nivoja (0 ali 1), kjer analogni vhodi/izhodi predstavljajo več nivojsko resolucijo signala. Analogni signali so omejeni z napajalno napetostjo mikrokontrolerja, natančneje z napajanjem AD-enote znotraj čipa. Analogne vhode ponavadi označimo, kot AD-signali ('Analog to Digital'), analogne izhode kot DA-signali ('Digital to Analog'). Pini mikrokontrolerja so razporejeni po skupinah, ki jih imenujemo porti in so označeni s PA,PB,PC,PD,PE. Kjer vsaka skupina zajema fizične pine od 0-15 (npr. PA0-PA15 itd.). Nastavitev GPIO modula in vseh pripadajočih funkcionalnosti je potrebno v projekt vključiti knjižnico ***stm32fxxx_gpio.c***.

- **Nastavitev diskretnega GPIO izhoda**

Določitev diskretnega vhoda:

```
GPIO_InitTypeDef GPIO_InitStructure; //Struktura

//GPIO out
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13|
                              GPIO_Pin_14 | GPIO_Pin_15;//Konfiguracija pinov 12-15
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;           // Definicija izhoda
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       // Hitrost modula (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 upor ni aktiven
GPIO_Init(GPIOD, &GPIO_InitStructure);                  // Nastavitev porta D
```

Pomen oznak:

```
GPIO_Pin_xx    -  Zaporedna številka pina 1-15 na portu -(A,B,C,D,E)
GPIOx          - Ime porta A,B,C,D…
```

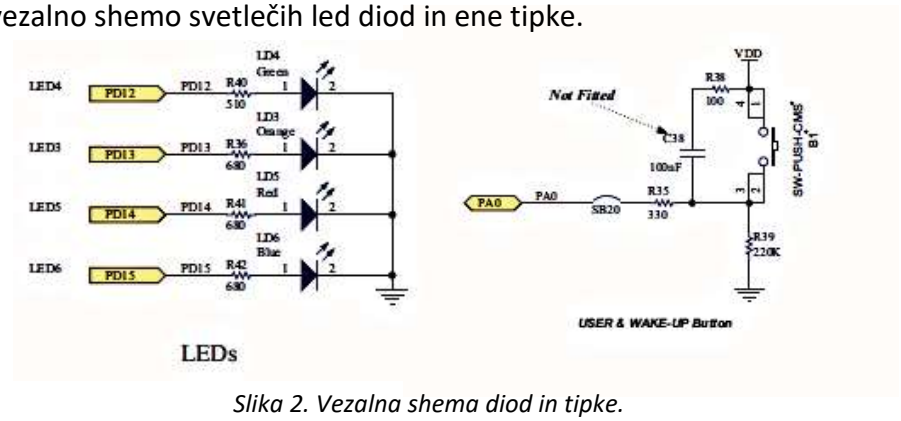
- **Nastavitev diskretnega GPIO vhoda**

Določitev diskretnega vhoda:

```
GPIO_InitTypeDef GPIO_InitStructure; //Struktura
//GPIO IN
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;         //  konfiguracija PINA 0
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;      // GPIO kot vhod
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; // Hitrost GPIO modula(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 upor ni aktiven
GPIO_Init(GPIOA, &GPIO_InitStructure);            // Nastavitev porta A
```

- **PRIMER uporabe diskretnega vhoda/izhoda**

Na krmilniški ploščici imamo štiri diode, ki se nahajajo na portu D in pinih od 12-15. Prav tako imamo uporabniško tipko, ki je povezana na port A in pin 0. Slika 2. prikazuje vezalno shemo svetlečih led diod in ene tipke.



Slika 2. Vezalna shema diod in tipke.

Primer programa:

```
//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  key=0;


int main(void)
{

    //Nastavitev MCU-ja in GPIO
  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)
    {

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

                if(key == 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


}//Konec 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; // konfiguiramo vse I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;            // GPIO kot izhod
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;        // Hitrost  GPIO modula
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // push / pull(opposed to open drain)
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // pullup / pullup upor ni aktiven
    GPIO_Init(GPIOD, &GPIO_InitStructure);               // Nastavitev porta D

     //BUTTON
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;        // Konfiguracija PINA 0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;     // GPIO kot vhod
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //Hitrost  GPIO modula (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 upor ni aktiven
    GPIO_Init(GPIOA, &GPIO_InitStructure);           // Nastavitev porta A

}
```

9.3. UART komunikacija

UART komunikacija krajše ali - serijsko asinhrono prejemanje in pošiljanje podatkov. UART najpogosto uporabljamo, kot komunikacijo RS232. Za RS232 potrebujemo periferno prilagoditveno vezje, ki nam pretvori nivoje signalov za TTL nivojsko logiko s čipom (MAX232) ali uporabimo USB-Com-Port modul, ki nam emulira standardno RS232 povezavo preko USB vmesnika (FTDi, CP2102 itd..), slika 3. Razvojna plošča vsebuje 4 interne USART module, ki se nahajajo na sledečih portih:

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

- **Primer uporabe USART2:**

V projekt vključimo knjižnico **stm32fxxx_usart.c**. PIN PA3 povežemo z RX pinom na modulu CP2012 in PIN PA2 s TX pinom modula CP2012.

Slika 3. Modul 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  key=0;
char buffer[200];
int j,i=0;



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

  //Nastavitev MCU-ja in GPIO
  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
```

```
}//Konec 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; // konfiguiramo vse I/O pine za LED
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;          // GPIO kot izhod
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;   // Hitrost  GPIO modula (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 upor ni aktiven
    GPIO_Init(GPIOD, &GPIO_InitStructure);                 // Nastavitev porta D

  //BUTTON
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;          //  konfiguracija PINA 0
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;           //  GPIO kot vhod
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;    //  Hitrost  GPIO modula (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 upor ni aktiven
  GPIO_Init(GPIOA, &GPIO_InitStructure);                 // Nastavitev porta A

     // 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. AD pretvorba

Razvojna plošča omogoča 16 zunanjih analognih vhodov ter dva notranja (senzor temperature in napetost baterije). Mikrokrmilnik vsebuje tri ločene AD pretvornike, kjer vsak AD1-3 ima ločeno nastavljivo resolucijo od 6, 8, 10 do 12bit-ov (Privzeta vrednost AD je 12bitov). Tri AD pretvornike je možno programsko konfigurirati na različne GPIO enote in pine z dodano knjižnico **stm32fxxx_adc.c**

Imena portov in pinov, kjer se nahajajo AD-pretvorniki:

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

- **PRIMER nastavitve AD pretvornika**

Branje analogne vrednosti na kanalu 1 ,4 in 5 s pomočjo dveh ločenih AD pretvornikov (AD1 in 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


}//Konec 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. Pulzno širinska modulacija PWM

Pulzna širinska modulacija PWM ('Pulse Width Modulation') je način modulacije s katero spreminjamo prevajalno razmerej ('Duty Cycle') pri konstantni frekvenci signala. Pulzno širinska modulacija je vezana na časovnik (Timer) mikrokrmilnika. Mikrokrmilnik STM32F407VG vsebuje 14 časovnikov (TIM1 – TIM14) s časovno resolucijo 16-bitov in dva z 32-bitno resolucijo (TIM2 in TIM5). Z mikrokrmilnikom je možno generirati 14 različnih neodvisnih PWM signalov z različno frekvenco ter 32 PWM signalov, kjer določene skupine imajo enako frekvenco ter neodvisni nastavljivi čas prevajanja ('Duty Cycle').

Vsak časovnik je lahko fizično vezan na GPIO, kjer pri nastavitvi PWM-a z izbiro časovnika ('Timer') in kanala ('Channel'), določimo izhodni pina na portu A,B,C,D,E, (tabela 1).

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			

Tabel 1: Tabela izhodnih pinov za določitev PWM modulacije glede na časovnik in kanal.

Določitev periode PWM modulacije (frekvence) in nastavitev vrednosti ARR registra (Auto Reload Register):

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

Glede na nastavitev projekta je frekvenca časovnikov enaka polovici glavne ure 168MHz, kar znaša 84MHz. Vrednost izračunanega ARR je vrednost maksimalnega prevajalnega razmerja.

- **PRIMER nastavitve PWM modulacije**

Nastavitev 10kHz PWM signala na pinu PD14 in PD15.

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

    //Nastavitev MCU-ja in GPIO
  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


}//Konec MAIN


//PWM konfiguracija
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; // konfiguiramo I/O
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;       // GPIO kot izhod (Alternate
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  // Hitrost  GPIO modula (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. Časovne prekinitvene rutine

Prekinitvene rutine so pomemben del sistemov realnega časa. Na sistemih realnega časa in odvisno od mikrokrmilniškega sistema poznamo različne tipe prekinitvenih rutin, ki se prožijo na različne dogodke. Naj naštejemo nekaj najpogosteje uporabljenih; prekinitev časovnikov, zunanje prekinitve (ob spremembi vrednosti na vhodnem ali izhodnem pinu), prekinitev AD pretvornika, prekinite ob komunikacijskih dogodkih (sprejemanje podatkov, oddajanje podatkov SPI, I2C, UART, napaka na komunikacijskem vodilu itd..). V našem primeru bomo predstavili časovne prekinitve, zunanje prekinitev ter prekinitev ob prejemanju podatkov.

- **Nastavitev prekinitve na časovniku 3 (TIM3)**

Časovna periode časovnika je enaka:

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

V našem primeru uporabljamo skalirni faktor 2, kar pomeni, da je osnovna frekvenca časovnika enaka polovici glavne ure MCU-ja. Osnovna frekvenca ure časovnikov znaša 84MHz.

Primer

Nastavimo periodo časovnika na 10ms. Izberemo vrednost TimerX_prescaler=209, TimerX_preioda=4000;

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

Čas proženja prekinitve določimo s frekvenco časovnika.

1. Omogočimo sistemsko uro MCU na določen časovnik (***RCC_APB1ENR.TIM2EN = 1***)

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

2. Določimo deljenje ure 'prescaler'. Frekvenco ure delimo MCU (168MHz) z vrednostjo v registru ***TIMx_PSC***

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

3. Določimo število preštetih vrednosti ***TIMx_ARR (auto reload register)***, katerih frekvenca je odvisna od vrednosti v registru ***TIMx_PSC*** in frekvenco MCU-ja. ARR register je enako periodi časovnika.

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

4. Omogočimo način štetja navzgor ali navzdol ter strukturo zapišemo v registre časovnika TM3.

```
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //Counter mode up
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);     //Zapis v strukturo
```

5. Omogočimo časovnik in priredimo prekinitveno rutino.

```
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
```

Za nastavitev proženja prekinitve nastavimo še prioriteto v NVIC (Nested vector interrupt controller).

```
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);
```

Prioritete je možno izbrati od nivoja od 0-4, kjer velja nižja številka prioritete pomeni, da je prekinitev pomembnejša. Podprogram prekinitve se izvaja v predpripravljeni funkciji

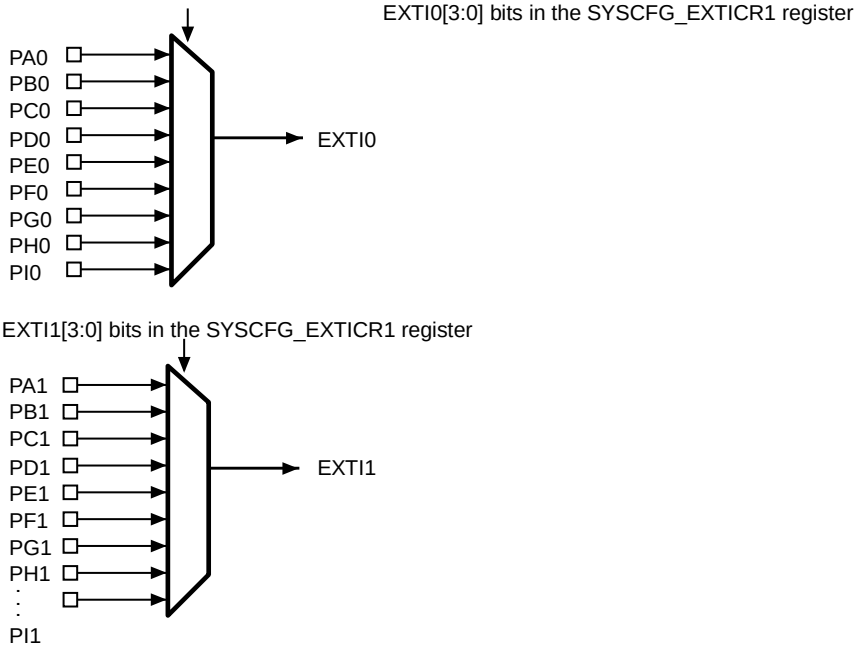
TIM3_IRQHandler(void). Imena funkcij za prekinitve so prednastavljene in zapisane v dokumentu **startup_stm32f40_41xxx.s** .

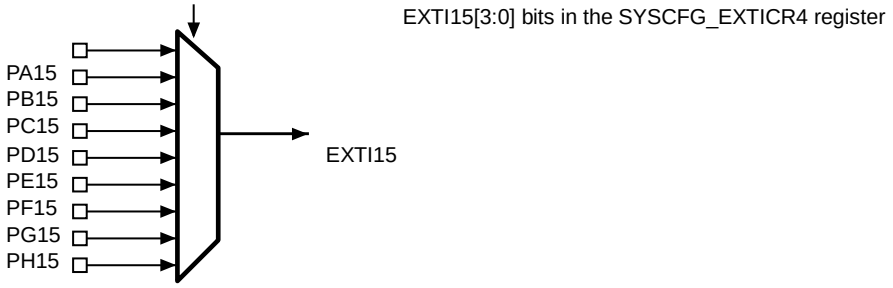
```
//MACRO
#define  LED1 GPIO_Pin_15
#define  LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define  LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define  Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)


int main(void)
{

    //Nastavitev MCU-ja in GPIO
    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


}//Konec 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);


    }
}
```

- **Nastavitev prekinitev na tipko A0**

Zunanje prekinitve, ki jih prožijo zunanje naprave, stikala, komunikacijski protokoli itd., je potrebno omogočiti z naborom vpisov v določene sistemske registre. Zunanje prekinitve so porazdeljene po enotah EXTI0 – EXTI15 in se programsko določijo v kontrolnem registru SYSCFG_EXTICRx. Vsaka EXTI0-15 pripada zaporedni številki porta 0-15, slika 4.





Slika 4. Skupine za zunanje prekinitve

Primer za zunanjo prekinitev, preko tipke PA0.

Določimo masko prekinitve, ki jo vpišemo v register SYSCFG_EXTICRx. Ker bomo uporabljali tipko na portu A in zaporedno številko 0 (PA0). Moramo iz zgornje slike uporabiti prekinitev EXTI0. Nastavitev GPIO pina:

```
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);
```

Povezava pina PA0 na zunanjo prekinitev:

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

Nastavitev zunanje prekinitve:

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

Glede na tip proženja registra na pozitivno ali negativno fronto določimo način prekinitve.

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

EXTI_Init(&EXTI_InitStructure);

EXTI_GenerateSWInterrupt(EXTI_Line0);
```

Primer celotnega programa nastavitve:

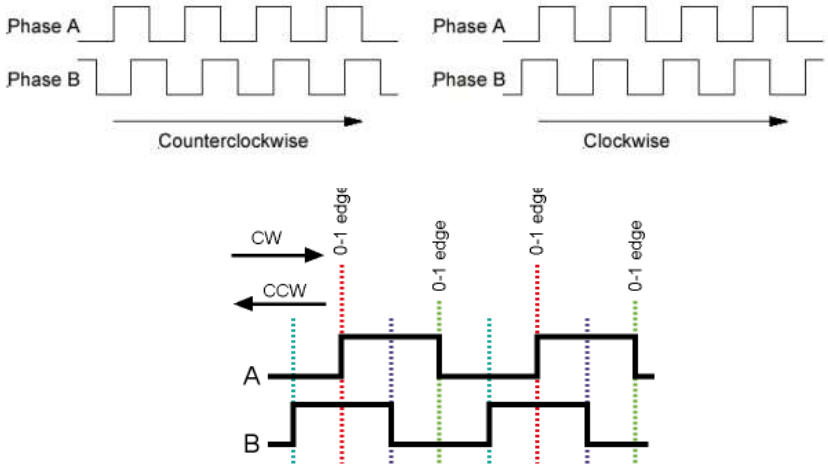
```
//MACRO
#define LED1 GPIO_Pin_15
#define LEDon(LED)    GPIO_SetBits(GPIOD, LED)
#define LEDoff(LED)   GPIO_ResetBits(GPIOD, LED)
#define Button        GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)

int main(void)
{


  //Nastavitev MCU-ja in GPIO
```

```
  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


}//Konec 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;          //  konfiguracija PIN_A 0
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;           //  GPIO kot vhod
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;    //  Hitrost  GPIO modula
(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 upor ni
aktiven
    GPIO_Init(GPIOA, &GPIO_InitStructure);                 // Nastavitev porta 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. Inkrementalni dajalnik

Inkrementalni dajalnik je naprava-senzor, ki zaznava premike ali zasuke sistema. Na ta način ločimo linearne in rotacijske inkrementalne dajalnike. Zelo pogosto rotacijske inkrementalne dajalnike srečamo, kot merilnike kota zasuka, kotne hitrosti motorja ali, kot kontrolne gumbe na elektronskih napravah. Rotacijski merilniki so pogosto montirani direktno na os motorja ali rotacijski element. Način merjenja pomikov in rotacij s pomočjo inkrementalnih dajalnikov je splošna industrijska praksa. Večina vgrajenih sistemov že vsebuje strojne module za povezavo dajalnika z mikrokrmilniškim sistemom. Princip merjenja z inkrementalnim dajalnikom sloni na principu štetja inkrementov, ki jih povzroči pomik merjenca. Natančnejši sistemi imajo dva do štiri kanale in index- celotnega obrata (A,B,A',B',Index), preprostejši pa enega (A). Način delovanja inkrementalnega dajalnika prikazuje slika 5:



Slika 5. Princip delovanja inkrementalnega dajalnika

Pri inicializaciji inkrementalnega dajalnika je potrebno podobno, kot pri načinu PWM, GPIO pine povezati na časovnik. V tem primeru velja, da so lahko uporabljeni pini le iz kanala 1 in kanala2 (Tabela 1). Pini kanala 3 in 4 ne morejo biti uporabljeni v načinu 'enkoder' (inkrementalni dajalnik). V tem primeru štetje časovnika ni vezano na prednastavljeno resolucijo časovnika ter uro, ampak zgolj na inkremente dajalnika. Za ta namen je potrebno nastaviti časovnik v način inkrementalnega dajalnika s funkcijo ' TIM_EncoderInterfaceConfig(xxx)'. Časovnik lahko uporabljamo v sklopu s prekinitvijo ali brez, kar pomeni, da se prekinitev proži na določeno število prednastavljenih inkrementov.

Sledeči primer kode prikazuje nastavitev dajalnika na pinu PC6(A) in PC7(B) s časovnikom TIM3, ter prekinitvijo s 3200 preštetimi inkrementi.

Primer kode:

```
 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);




                    }
    }
```

9.9. SPI komunikacija s pospeškometrom LIS302DL

Na razvojni plošči se nahaja triosni pospeškometer LIS302DL z merilnim razponom od ±2 - ±8g in 8 bitno resolucijo . Senzor je na MCU povezan s SPI vodilom preko pinov PA7(MOSI), PA6(MISO), PA5(SCK), PE3(CS ali SS). SPI (Serial Peripheral Interface) vodilo lahko deluje v načinu 3 ali 4 žičnem načinu. SC/SS (chip select, slave select). Za inicializacijo senzorja bomo uporabili že obstoječe knjižnice, kjer je bistvenega pomena, da senzor pravilno kalibriramo. Nastavimo območje občutljivosti in frekvenčni razpon. Podatki iz knjižnice so že preračunani vrednost mm/s2=mg.

Primer programa:

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

    //Nastavitev MCU-ja in GPIO
  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
```

9.10. Virtualni serijski port VCP

Na dani preizkusni plošči nimamo posebej modula za USART komunikacijo (FTDI, CP2012..), zato bomo uporabili USB 2.0 modul v mikrokrmilniku, katerega bomo emulirali, kot virtualni COM-port. Če želimo uporabljati USB modul, kot VCP, moramo vključiti dodatno knjižnico proizvajalca ST-Microelectronics. V naslednjem primeru je zgled programski kode, ki prikazuje nastavitev USB-VCP vmesnika na razvojni plošči STM32 DiscoveryF407. Mikro-USB vtič, se nahaja na pinih PA11 in PA12, kot podatkovna linija (D- D+). Za lažje razumevanje in uporabo USB vmesnika smo uporabili preoblikovano knjižnico Tilena Majerle, ki je prosto dostopna na http://stm32f4-discovery.com/tag/tilen-majerle/. Program opisuje prejemanje podatkov s končnim znakom '%' ter izpis prejete vsebine s pritiskom na gumb. S funkcijo USBD_Init() in argumentom USE_USB_OTG_FS inicializiramo USB-VCP vmesnik na pinih PA10,PA11,PA12,PA13.

Primer programa:

```
USB_OTG_CORE_HANDLE USB_OTG_dev; //USB VCP structure

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

   //Nastavitev MCU-ja in GPIO
   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
```