Ecodesign of Electronic Devices

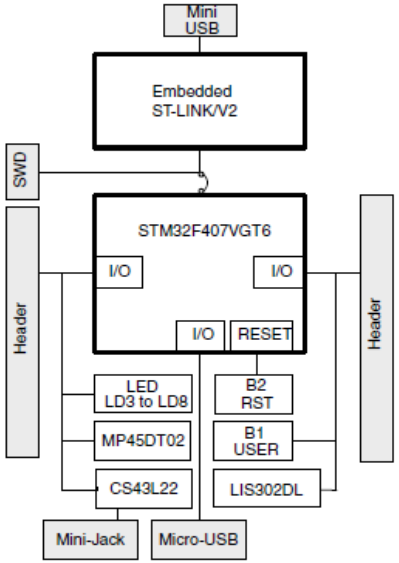
UNIT 9: Microcontroller systems part 2

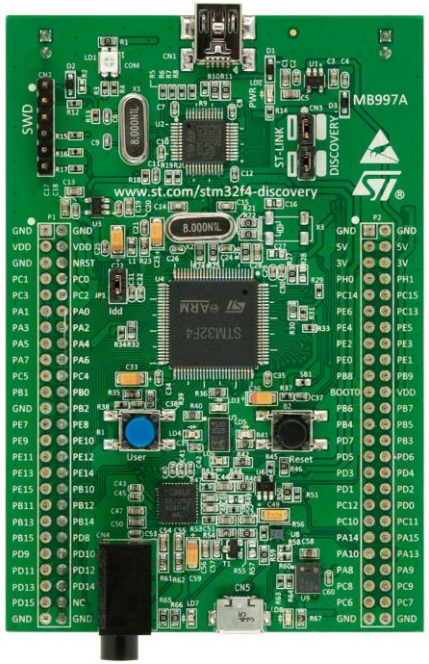
Funded by the Erasmus+ Programme of the European Union

Ecoinnovation Skills for European Designers, Project number: 562573-EPP-1-2015-1-SI-EPPKA2-SSA.

Use of microcontroller board SMT32F4discovery

- Development board **STM32F4Discovery** is low priced and powerful microcontroller board, suitable for fast development of built-in applications and testing of powerful microcontroller ARM STM32F407VG with built-in FPU unit (FPU – floating point unit, a unit for computing with floating comma numbers). Development board contains microcontroller ARM STM32F407VG, JTAG programmer, ST-link V2, accelerometer, audio codec, microphone, 4 control LED diodes, switch and micro-USB connector.





Funded by the Erasmus+ Programme of the European Union

Use of microcontroller board SMT32F4discovery

Microcontroller STM32F407 is based on core Cortex-M4 and can run on maximum frequency 168MHz. Core Cortex-M4 offers 32-bit processor unit and mechanical FPU. FPU is intended for digital processing of signal and has functionalities of DSP processor. At maximum frequency, it reaches 210 DMIPS (Dyrestone-million instruction per second). The microcontroller also contains a wide array of peripheral devices:

- 2x USB OTG (On the Go).
- Audio phase locked loop (PLL- phase lock loop).
- Contains 15 communication interfaces (4 x UART, 2 x USART, 3 x SPI, 3 x I2C, 2x CAN, SDIO).
- Contains 2x 12-bit DAC in 3 x 12-bit ADC speed 2.4Msps.
- Has 17 timers. All timers are 16-bit except two that are 32-bit.
- Interfaces for additional memory systems, SRAM, NAND, etc.

Funded by the
Erasmus+ Programme
of the European Union

Setting discrete inputs and outputs -GPIO

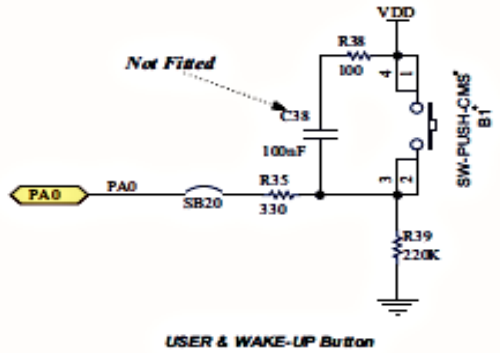
- Microcontroller pins are divided into groups that are named ports and are labeled with PA, PB, PC, PD, PE where every group captures physical pins from 0 to 15 (for example PA0-PA15, etc.). The setting of GPIO module and all corresponding functionalities need to be included in the library ***stm32fxxx_gpio.c***.

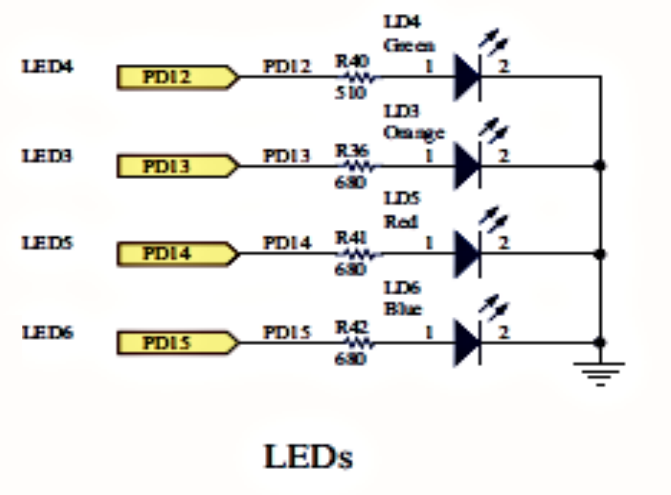
GPIO - output setting:

```
GPIO_InitTypeDef GPIO_InitStructure; //Struktura

//GPIO out
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13|
                              GPIO_Pin_14 | GPIO_Pin_15;

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO Init(GPIOD, &GPIO InitStructure);
```

GPIO - input setting:

```
//GPIO IN
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);
```



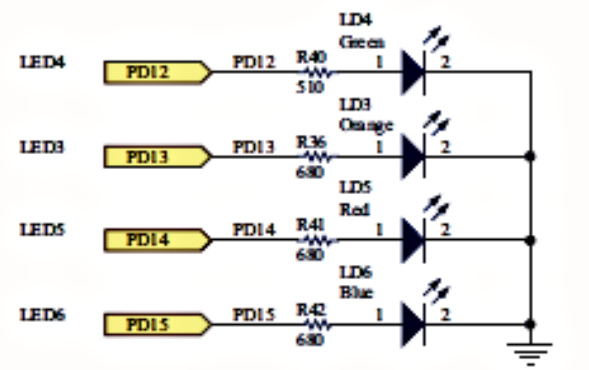


Funded by the
Erasmus+ Programme
of the European Union

Setting discrete inputs and outputs -GPIO

Program example:

```
//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
```



LEDs

Funded by the
Erasmus+ Programme
of the European Union

UART communication

- USART‘s gpio ports:
  - UART 1 - Rx: PA10 Tx: PA9
  - **UART 2 - Rx: PA3 Tx: PA2**
  - UART 3 - Rx: PB11 Tx: PB10
  - UART 4 - Rx: PA1 Tx: PA0

GPIO setting for UART module:

```
   // 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


}
```

```
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

}
```

Call function:

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


Funded by the Erasmus+ Programme of the European Union

ADC conversion

- Development board enables 16 external analog inputs and two internal (sensor for temperature and battery voltage). The microcontroller contains three separate AD converters, where each AD1-3 has separately adjustable resolution from 6, 8, 10 to 12-bit (default value is 12-bit). Three AD converters can be configurated to different GPIO units and pins with additional library **stm32fxxx_adc.c**
- Names of ports and pins that are in AD converters:

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

Funded by the
Erasmus+ Programme
of the European Union

ADC conversion

EXAMPLE of ADC converter settings:

```
/ 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);

}
```

ADC read:

```
//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;
}
```


Funded by the
Erasmus+ Programme
of the European Union

Pulse-width modulation PWM

- Pulse width modulation PWM is modulation type with which we change duty cycle at the constant signal frequency. Pulse width modulation is related to the microcontroller timer.
- Each timer can be physically bound to GPIO where we determine output pin on port A, B, C, D, E when setting PWM with timer and channel choice.

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			

Funded by the
Erasmus+ Programme
of the European Union

Pulse-width modulation PWM

- PWM example on port PD14 and PD15 with TIMER 4.

```
//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);
 }
```


Funded by the
Erasmus+ Programme
of the European Union

Timer interrupt routine

Interrupt settings on timer 3 (TIM3)

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

In our case is used scaling factor 2, which means that the basic timer frequency is equal to half of main MCU clock. The basic timer clock frequency is 84MHz.

- **EXAMPLE**

Set timer period to 10ms. Choose value TimerX_prescaler=209, TimerX_period=4000;

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

Funded by the Erasmus+ Programme of the European Union

Timer interrupt routine

- Example of timer interrupt on TIMER 3 with period of 10ms:

```
/**************************************************************************
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)
((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
}
```

Priority setting:

```
/**************************************************************************
 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);

  }
```

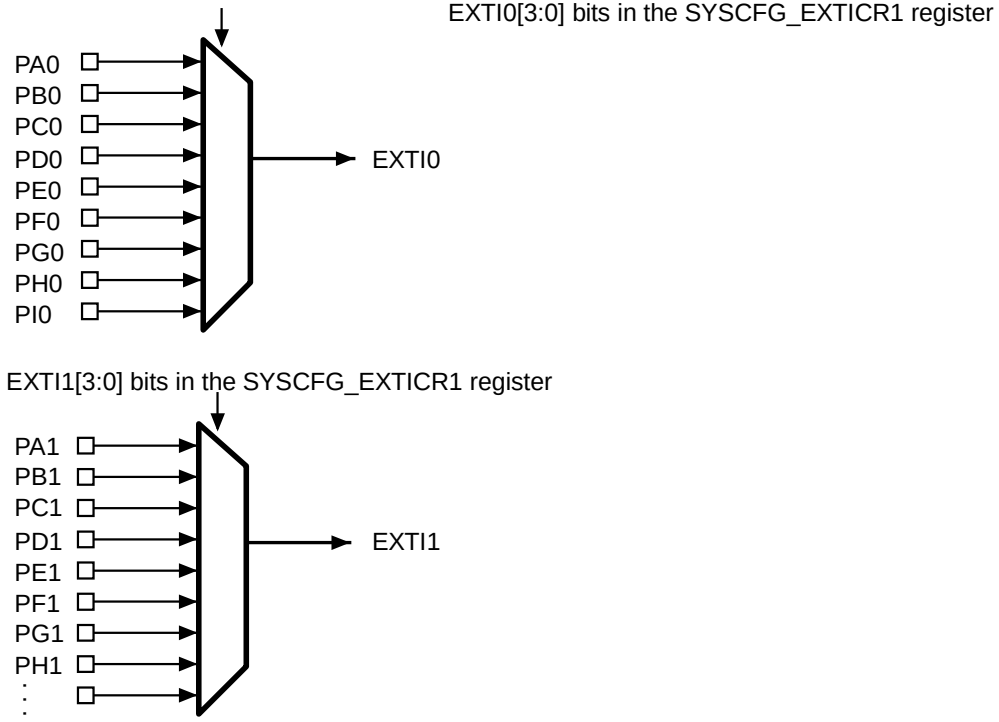
Interrupt call function:

```
/**************************************************************
 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);


    }
}
```


Funded by the Erasmus+ Programme of the European Union

External interrupt

- External interrupts that are triggered by external devices, switches, communication protocols, etc. need to be enabled with a set of entries in certain system registers.
- These interrupts are divided amongst units EXTI0 – EXTI15 and are programmatically determined in control register SYSCFG_EXTICRx.
- Each EXTI0-15 corresponds to a sequential port number 0-15



Funded by the Erasmus+ Programme of the European Union

External interrupt

- External interrupt on pin PA0:

```
/*******************************************************************************
 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);

}
```

Priority setting:

```
/*******************************************************************************
 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);

}
```

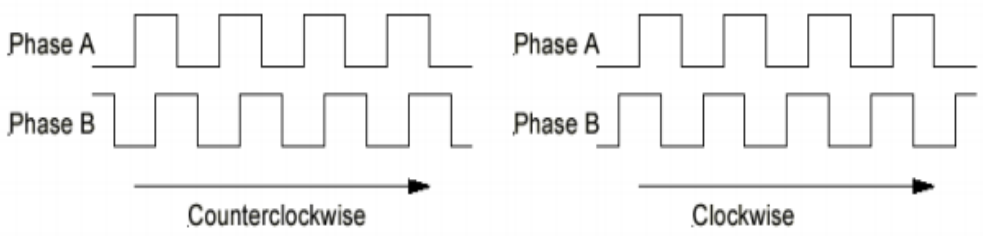
Interrupt call function :

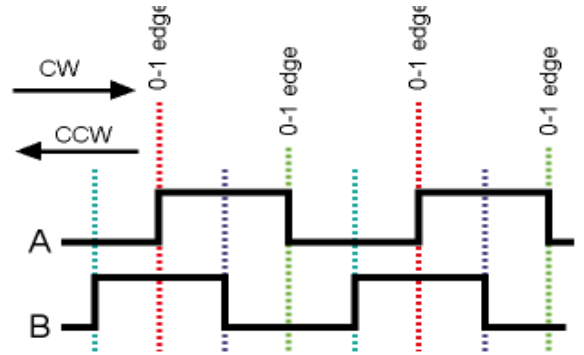
```
/*******************************************************************************
EXTI0 Interrupt function
*******************************************************************************/
void EXTI0_IRQHandler(void)
{

     if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {


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

          EXTI_ClearITPendingBit(EXTI_Line0);
    }
```

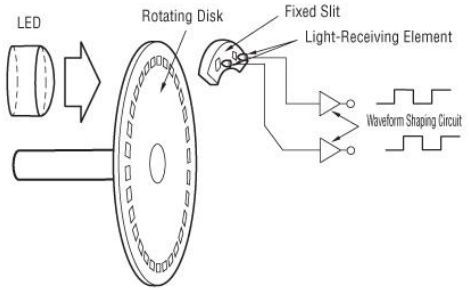

Funded by the Erasmus+ Programme of the European Union

Incremental encoder

- Incremental encoder is a device-sensor that detects system shifts or rotations. This way we can differentiate linear and rotary incremental encoders.
- Very often we meet rotary incremental encoders, such as angle gauges, speedometers or control buttons on electronic devices.
- The measurement principle is based on the principle of increment counting that is caused by the shift of measured value.







Funded by the
Erasmus+ Programme
of the European Union

Incremental encoder

- Encoder example on pins PC6(A) and PC7(B) with timer TIM3 and 3200 pulses.

```
/*******************************************************************************
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);
}
```


Funded by the Erasmus+ Programme of the European Union

Incremental encoder

Priority setting:

```
/*******************************************************************************
*  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);


}
```

Interrupt call function, continuously triggered after 3200 counts :

```
/*******************************************************************************
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);




                    }
    }
```

Encoder read, direct from timers register:

```
encoder = TIM3->CNT; //Read encoder value
```


Funded by the Erasmus+ Programme of the European Union

Incremental encoder

- Code example:

```
/*****************************************************************************
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);
}
```

Encoder read:

```
encoder = TIM3->CNT; //Read encoder value
```


Funded by the
Erasmus+ Programme
of the European Union

Virtual serial port VCP

- In the following example is program code that shows the setting of USB-VSP interface on development board STM32 DiscoveryF407. Micro USB plug is located on pins PA11 and PA12 and operates as data line (D- D+).
- The program describes data receiving with the final sign ‘%’ and a display of received content with a click on the button. With function USBD_Init() and argument USE_USB_OTG_FS we initialize USB-VCP interface on pins PA10,PA11,PA12,PA13.

```
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
```


Funded by the Erasmus+ Programme of the European Union