Wybrane listingi z opisu ćwiczenia 2

Strona 41

```
#include "stm32f10x.h" // definicja typu uint16_t i stalych GPIO_Pin_X

#define LED1 GPIO_Pin_8
#define LED2 GPIO_Pin_9
#define LED3 GPIO_Pin_10
#define LED4 GPIO_Pin_11
#define LED5 GPIO_Pin_12
#define LED6 GPIO_Pin_13
#define LED7 GPIO_Pin_14
#define LED8 GPIO_Pin_15
#define LEDALL (LED1|LED2|LED3|LED4|LED5|LED6|LED7|LED8)
enum LED_ACTION { LED_ON, LED_OFF, LED_TOGGLE };

void LED(uint16_t led, enum LED_ACTION act);
```

```
void LED(uint16_t led, enum LED_ACTION act) {
    switch(act){
        case LED_ON: GPIO_SetBits(GPIOB, led); break;
        case LED_OFF: GPIO_ResetBits(GPIOB, led); break;
        case LED_TOGGLE: GPIO_WriteBit(GPIOB, led,
            (GPIO_ReadOutputDataBit(GPIOB, led) == Bit_SET?Bit_RESET:Bit_SET));
    }
}
```

Strona 42

```
SysTick_Config(9000); // (72MHz/8) / 9000 = 1KHz (1/1KHz = 1ms)
SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
```

```
void SysTick_Handler(void);
```

Strona 43

```
void DelayTick(void){
    // dekrementacja licznika (o ile jest wiekszy od 0)
}
```

```
void Delay(unsigned int ms){
    // zmiana wartosci licznika
    // testowanie czy jest on wiekszy od 0
}
```

```
NVIC_SetPriority(SysTick_IRQn, 0);
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_X);
```

Strony 44-45

```
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
TIM_OCInitTypeDef TIM_OCInitStructure;

TIM_TimeBaseStructure.TIM_Prescaler = 7200-1;               // 72MHz/7200=10kHz
TIM_TimeBaseStructure.TIM_Period = 10000;                   // 10kHz/10000=1Hz (1s)
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; // zliczanie w gore
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;            // brak powtorzen
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);             // inicjalizacja TIM4
TIM_ITConfig ( TIM4, TIM_IT_CC2 | TIM_IT_Update, ENABLE );  // wlaczenie przerwan
TIM_Cmd(TIM4, ENABLE);                                      // aktywacja timera TIM4

// konfiguracja kanalu 2 timera
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Timing;         // brak zmian OCxREF
TIM_OCInitStructure.TIM_Pulse = 2000;                       // wartosc do porownania
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; // wlaczenie kanalu
TIM_OC2Init(TIM4, &TIM_OCInitStructure);                    // inicjalizacja CC2



NVIC_InitTypeDef NVIC_InitStructure;

NVIC_ClearPendingIRQ(TIM4_IRQn);                    // wyczyszczenie bitu przerwania
NVIC_EnableIRQ(TIM4_IRQn);                          // wlaczenie obslugi przerwania
NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;     // nazwa przerwania
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2; // priorytet wywlaszczania
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;  // podpriorytet
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;     // wlaczenie
NVIC_Init(&NVIC_InitStructure);                     // inicjalizacja struktury



void TIM4_IRQHandler(void){
    if(TIM_GetITStatus(TIM4,TIM_IT_CC2) != RESET){
        LED(LED2,LED_TOGGLE);
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
    } else if(TIM_GetITStatus(TIM4,TIM_IT_Update) != RESET){
        LED(LED3,LED_TOGGLE);
        TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
    }
}
```

Strona 47

```
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0); // PA0 -> EXTI0_IRQn
EXTI_InitStructure.EXTI_Line = EXTI_Line0;                  // linia : 0
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;         // tryb  : przerwanie
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;     // zbocze: opadajace
EXTI_InitStructure.EXTI_LineCmd = ENABLE;                   // aktywowanie konfig.
EXTI_Init(&EXTI_InitStructure);                             // inicjalizacja

NVIC_ClearPendingIRQ(EXTI0_IRQn);                           // czyszcz. bitu przerw.
NVIC_EnableIRQ(EXTI0_IRQn);                                 // wlaczenie przerwania
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;            // przerwanie EXTI0_IRQn
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;   // prior. wywlaszczania
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;          // podpriorytet
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;             // aktywowanie konfig.
NVIC_Init(&NVIC_InitStructure);                             // inicjalizacja



TIM_TimeBaseStructure.TIM_Prescaler = 7200-1;               // 72MHz/7200=10kHz
TIM_TimeBaseStructure.TIM_Period = 350;                     // 10kHz/350~=29Hz(35ms)
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; // zliczanie w gore
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;            // brak powtorzen
TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);             // inicjalizacja TIM3
TIM_ITConfig ( TIM3, TIM_IT_Update, DISABLE );              // wylaczenie przerwan
TIM_Cmd(TIM3, DISABLE);                                     // wylaczenie timera

NVIC_ClearPendingIRQ(TIM3_IRQn);                            // czyszcz. bitu przerw.
NVIC_EnableIRQ(TIM3_IRQn);                                  // wlaczenie przerwania
NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;             // nazwa przerwania
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;   // prior. wywlaszczania
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;          // podpriorytet
```

```
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;            // aktywowanie konfig.
    NVIC_Init(&NVIC_InitStructure);                            // inicjalizacja
```

Strona 48

```
    void EXTI0_IRQHandler(void){
        if(EXTI_GetITStatus(EXTI_Line0) != RESET){      // sprawdzenie przyczyny
            EXTI_ClearITPendingBit(EXTI_Line0);         // wyczyszczenie bitu przerwania

            TIM_SetCounter(TIM3, 0);                    // reset licznika timera
            TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE ); // aktywacja przerwania
            TIM_Cmd(TIM3, ENABLE);                      // aktywacja timera TIM3
        }
    }


    void TIM3_IRQHandler(void){
        if(TIM_GetITStatus(TIM3,TIM_IT_Update) != RESET){ // sprawdzenie przyczyny
            TIM_ClearITPendingBit(TIM3, TIM_IT_Update);   // wyczyszczenie bitu przerw.
            if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == Bit_RESET) // jesli wcisniety
                LED(LED5,LED_TOGGLE);                     // zrob cos (przelacz LED5)
            TIM_ITConfig (TIM3, TIM_IT_Update, DISABLE ); // deaktywacja przerwania
            TIM_Cmd(TIM3, DISABLE);                       // deaktywacja timera TIM3
        }
    }
```

Strony 49-50

```
    char KB2char(void){
        unsigned int GPIO_Pin_row, GPIO_Pin_col, i, j;
        const unsigned char KBkody[16] = {'1','2','3','A',\
                                          '4','5','6','B',\
                                          '7','8','9','C',\
                                          '*','0','#','D'};
        GPIO_SetBits(GPIOD, GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13);
        GPIO_Pin_row = GPIO_Pin_10;
        for(i=0;i<4;++i){
            GPIO_ResetBits(GPIOD, GPIO_Pin_row);
            Delay(5);
            GPIO_Pin_col = GPIO_Pin_6;
            for(j=0;j<4;++j){
                if(GPIO_ReadInputDataBit(GPIOD, GPIO_Pin_col) == 0){
                    GPIO_ResetBits(GPIOD, GPIO_Pin_10|GPIO_Pin_11|
                      GPIO_Pin_12|GPIO_Pin_13);
                    return KBkody[4*i+j];
                }
                GPIO_Pin_col = GPIO_Pin_col << 1;
            }
            GPIO_SetBits(GPIOD, GPIO_Pin_row);
            GPIO_Pin_row = GPIO_Pin_row << 1;
        }
        GPIO_ResetBits(GPIOD, GPIO_Pin_10|GPIO_Pin_11|GPIO_Pin_12|GPIO_Pin_13);
        return 0;
    }
```

Strony 51-52

```
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;

    TIM_TimeBaseStructure.TIM_Prescaler = 7200-1;
    TIM_TimeBaseStructure.TIM_Period = 5000; // 2Hz -> 0.5s
    TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
```

```
TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

// konfiguracja kanalu timera
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_Pulse = 1; // minimalne przesuniecie
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OC2Init(TIM2, &TIM_OCInitStructure);
TIM_ITConfig ( TIM2, TIM_IT_CC2 , ENABLE );
TIM_Cmd(TIM2, ENABLE);



ADC_InitTypeDef  ADC_InitStructure;

ADC_DeInit(ADC1);                                   // reset ustawien ADC1
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  // niezalezne dzialanie ADC 1 i 2
ADC_InitStructure.ADC_ScanConvMode = DISABLE;       // pomiar pojedynczego kanalu
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;  // pomiar automatyczny
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_CC2; // T2CC2->ADC
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right; // pomiar wyrownany do prawej
ADC_InitStructure.ADC_NbrOfChannel = 1;             // jeden kanal
ADC_Init(ADC1, &ADC_InitStructure);                 // inicjalizacja ADC1

ADC_RegularChannelConfig(ADC1, ADC_Channel_16, 1, ADC_SampleTime_41Cycles5); // konf.
ADC_ExternalTrigConvCmd(ADC1, ENABLE);
ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);

ADC_Cmd(ADC1, ENABLE);                              // aktywacja ADC1

ADC_ResetCalibration(ADC1);                         // reset rejestru kalibracji ADC1
while(ADC_GetResetCalibrationStatus(ADC1));         // oczekiwanie na koniec resetu
ADC_StartCalibration(ADC1);                         // start kalibracji ADC1
while(ADC_GetCalibrationStatus(ADC1));              // czekaj na koniec kalibracji

ADC_TempSensorVrefintCmd(ENABLE);                   // wlaczenie czujnika temperatury
```

Strona 53

```
void ADC1_2_IRQHandler(void){
    if(ADC_GetITStatus(ADC1, ADC_IT_EOC) != RESET){
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
        sprintf((char*)bufor,"%2d*C",((V25-ADC_GetConversionValue(ADC1))/Avg_Slope+25));
    }
}



const uint16_t V25 = 1750;     // gdy V25=1.41V dla napięcia odniesienia 3.3V
const uint16_t Avg_Slope = 5;  // gdy Avg_Slope=4.3mV/C dla napięcia odniesienia 3.3V
```