Sprawozdanie

Systemy wbudowane

<u>Ćwiczenie 7:</u>
Przetwornik analogowo-cyfrowy.

Wykonanie:

Busłowski Tomasz

Suchwałko Tomasz

Skrouba Kamil

Zawadzka Magdalena

(Grupa PS3)

Prowadzący zajęcia: **dr inż. Adam Klimowicz**

SW, semestr VI, 22-03-2017, Wydział Informatyki, Politechnika Białostocka

Zakres Materiału

1. Budowa i zasada działania przetwornika A/C w STM32
2. Konfiguracja przetwornika A/C
3. Obsługa przerwań od przetwornika
4. Pomiar wyzwalany timerem

Zadania do wykonania

1. Napisz program wyświetlający poziom napięcia przy pomocy linijki diodowej wyskalowanej co 1/8 maksymalnej wartości napięcia (3,3V). Przetwornik A/C ma pracować w trybie ciągłym z maksymalną częstotliwością próbkowania (czyli 1MHz).

2. Napisz program wyświetlający wartość napięcia na potencjometrze na wyświetlaczu LCD w zakresie od 0 do 3,3V z dokładnością do 1mV. Przetwornik ma być wyzwalany sygnałem od Timera TIM1 co 0,5s.

3. Napisz program do regulacji częstotliwości sygnału wydawanego przez sygnalizator dźwiękowy w zakresie od 50Hz do 5kHz. Wartość częstotliwości ma być wyświetlana na wyświetlaczu LCD.

4. Napisz program monitorujący wartość napięcia na potencjometrze. Wyjście wartości napięcia poza zakres 1V-2V ma być sygnalizowane świeceniem diody i sygnałem akustycznym.

Zadanie 1

Zadanie 1 i 2 zostały połączone w jedno zadanie.

main.c

```
int main(void)
{
	volatile unsigned long int i;
	RCC_Config();
	GPIO_Config();
	ADC_Config();

	NVIC_Config();
	TIM_Config();
	LCD_Initialize();
	LCD_WriteCommand(LCD_CLEAR);
```

```
      while (1) {
            ADC_SoftwareStartConvCmd(ADC1, ENABLE);
            while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));
            wartoscADC1  = ADC_GetConversionValue(ADC1);

            GPIO_ResetBits(GPIOB, GPIO_Pin_8 | GPIO_Pin_9 | GPIO_Pin_10 |
GPIO_Pin_11 | GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15);
            if (wartoscADC1 >> 4 >= 1) GPIO_WriteBit(GPIOB, GPIO_Pin_8,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_9, Bit_RESET);
            if (wartoscADC1 >> 4 >= 32) GPIO_WriteBit(GPIOB, GPIO_Pin_9,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_10, Bit_RESET);
            if (wartoscADC1 >> 4 >= 64) GPIO_WriteBit(GPIOB, GPIO_Pin_10,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_11, Bit_RESET);
            if (wartoscADC1 >> 4 >= 96) GPIO_WriteBit(GPIOB, GPIO_Pin_11,
Bit_SET);
            if (wartoscADC1 >> 4 >= 128) GPIO_WriteBit(GPIOB, GPIO_Pin_12,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_12, Bit_RESET);
            if (wartoscADC1 >> 4 >= 160) GPIO_WriteBit(GPIOB, GPIO_Pin_13,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_13, Bit_RESET);
            if (wartoscADC1 >> 4 >= 192) GPIO_WriteBit(GPIOB, GPIO_Pin_14,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_14, Bit_RESET);
            if (wartoscADC1 >> 4 >= 224) GPIO_WriteBit(GPIOB, GPIO_Pin_15,
Bit_SET);
            else  GPIO_WriteBit(GPIOB, GPIO_Pin_15, Bit_RESET);

            for (i=0;i<100000ul;i++);
                  ADC_ExternalTrigConvCmd(ADC1, ENABLE);
  };

      while (1);
      return 0;
}

void TIM1_UP_IRQHandler(void)
{
      //przeladowanie licznika
      if (TIM_GetITStatus(TIM1, TIM_IT_Update) != RESET)
      {
            TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
            LCD_WriteCommand(LCD_CLEAR);
            LCD_WriteCommand(LCD_DDRAM_SET);
            wartoscADC1 = wartoscADC1/4095.0 * 3300;
            LCD_WriteData((wartoscADC1/1000)%10+'0');
            LCD_WriteString(",", 1);
            LCD_WriteData((wartoscADC1/100)%10+'0');
            LCD_WriteData((wartoscADC1/50)%2*5+'0');
            LCD_WriteString("V", 1);
            ADC_ExternalTrigConvCmd(ADC1, ENABLE);
       }
}
```

Zadanie 3

```
#include "stm32f10x.h"
#include "lcd_hd44780_lib.h"
void GPIO_Config(void);
void setFreq(uint16_t val);
void RCC_Config(void);
void ADC_Config(void);
void NVIC_Config(void);
void TIM_Config(void);
char string[15];
int main(void)
{
 volatile unsigned long int i;
 volatile unsigned long int lel;
 unsigned long int wartoscADC1 = 0;
 RCC_Config();
 GPIO_Config();
 ADC_Config();
 TIM_Config();
 LCD_Initialize();
 for (i=0;i<10000000ul;i++);
 while (1) {
 // regulacja częstotliwości sygnału wydawanego przez sygnalizator
dźwiękowy
 //oraz wyświetlanie wartości na wyświetlaczu
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
  while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));
  wartoscADC1 = ADC_GetConversionValue(ADC1);
  wartoscADC1 = wartoscADC1 * 1.225;
  TIM_SetAutoreload(TIM4,60000/wartoscADC1);
  TIM_SetCompare3(TIM4,60000/wartoscADC1/2);
  sprintf(string, "%d", (wartoscADC1));
  LCD_WriteTextXY(string,0,0);
  for (i=0;i<1000000ul;i++);
 };
 return 0;
}
```

Zadanie 4

```
int main(void)
{
  volatile unsigned long int i;


  //konfiguracja systemu
  RCC_Config();
  GPIO_Config();
  ADC_Config();

    NVIC_Config();
    /*Tu nalezy umiescic ewentualne dalsze funkcje konfigurujace system*/
    TIM_Config();
```

```
    LCD_Initialize();
    LCD_WriteCommand(LCD_CLEAR);
    TIM_Config4();
    LCD_WriteCommand(LCD_CLEAR);

  while (1) {
      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
      //wyzwolenie pojedynczego pomiaru
      while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));       //odczekaj na
zakonczenie konwersji
      wartoscADC1  = ADC_GetConversionValue(ADC1);         //pobiez
zmierzona wartosc
         GPIO_ResetBits(GPIOB, GPIO_Pin_8);
             GPIO_ResetBits(GPIOB, GPIO_Pin_9 | GPIO_Pin_10 | GPIO_Pin_11 |
GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15);
             if (wartoscADC1 >> 4 <= 80 || wartoscADC1 >> 4 >= 155)
             {
             TIM_Cmd(TIM4, ENABLE);
             GPIO_WriteBit(GPIOB, GPIO_Pin_10, Bit_SET);
              }
             else  {
                   GPIO_WriteBit(GPIOB, GPIO_Pin_10, Bit_RESET);
                     TIM_Cmd(TIM4, DISABLE);
             }

             for (i=0;i<100000ul;i++);
                   ADC_ExternalTrigConvCmd(ADC1, ENABLE);
  };

    while (1);
  return 0;
}

void TIM1_UP_IRQHandler(void)
{
      //przeladowanie licznika
      if (TIM_GetITStatus(TIM1, TIM_IT_Update) != RESET)
      {
            TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
            LCD_WriteCommand(LCD_CLEAR);
            LCD_WriteCommand(LCD_DDRAM_SET);
            wartoscADC1 = wartoscADC1/4095.0 * 3300;
            LCD_WriteData((wartoscADC1/1000)%10+'0');
            LCD_WriteString(",", 1);
            LCD_WriteData((wartoscADC1/100)%10+'0');
            LCD_WriteData((wartoscADC1/50)%2*5+'0');
            LCD_WriteString("V", 1);
            ADC_ExternalTrigConvCmd(ADC1, ENABLE);
       }
}

void TIM_Config4() {
  //konfigurowanie licznikow

  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
  TIM_OCInitTypeDef  TIM_OCInitStructure;

  // Konfiguracja TIM4
  // Ustawienia ukladu podstawy czasu
  TIM_TimeBaseStructure.TIM_Prescaler = 1200;// 72MHz/1200 = 60kHz
  TIM_TimeBaseStructure.TIM_Period = 170;      //czestotliwosc PWM = 1440
```

```
Hz 72MHz / wartosc 60KHz
  TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

  TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);

  // Kanaly 1 i 2 nie uzywane
  // Konfiguracja kanalu 3 - uzywamy kanalu 3 poniewaz jego wyjscie jest na
GPIOB8 - gdzie jest LED1
  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 85; //wypelnienie = 50000/25000=50%
  TIM_OCInitStructure.TIM_OCPolarity=TIM_OCPolarity_High;
  TIM_OC3Init(TIM4, &TIM_OCInitStructure);

  TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);//wlaczenie buforowania

  TIM_ARRPreloadConfig(TIM4, ENABLE);//wlaczenie buforowania
  // Wlaczenie timera

}

void TIM_Config(void) {
      //Konfiguracja timerow
      TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
                  TIM_OCInitTypeDef  TIM_OCInitStructure;

      //Konfiguracja licznika 1
      //Ustawienia taktowania i trybu pracy licznika 1
      TIM_TimeBaseStructure.TIM_Prescaler = 7200-1;        //taktowanie
licznka fclk = 72MHz/7200 = 10kHz
      TIM_TimeBaseStructure.TIM_Period = 7000;             //okres
przepelnien licznika = 20000 taktow = 2 sekundy
      TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //dzielnik
zegara dla ukladu generacji dead-time i filtra
      TIM_TimeBaseStructure.TIM_RepetitionCounter=0;       //licznik
powtorzen
      TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //tryb
pracy licznika

      TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);      //Inicjalizacja
licznika

      // Wlaczenie przerwan od licznikow
  TIM_ITConfig(TIM1, TIM_IT_Update , ENABLE);  //wlaczenie przerwania od
przepelnienia
  //TIM_ITConfig(TIM1, TIM_IT_CC1, ENABLE);    //wlaczenie przerwanie od
porownania w kanale 1



      // Wlaczenie timerow
  TIM_Cmd(TIM1, ENABLE);
}
```