УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧНИ ФАКУЛТЕТ

Катедра за електронику

32-битни микроконтролери и примјена

Регулација температуре лемилице

Пројекат

Ментор:
проф. др. Драган Васиљевић

Кандидат:
Слободан Жерајић 12/3269
slobodanzerajic@yahoo.com

Београд, фебруар 2013.

Садржај

Слике

Табеле

1 Увод

Потреба за регулацијом температуре се јавила веома давно, у праисторији. Људи су као интелигентна бића знали за најпростије начине регулације, затварали су и отварали отворе на склоништима, користили ватру, одјећу итд. У данашње вријеме се свакодневно срећу многи температурни процеси које је потребно регулисати, почевши од уређаја у домаћинству, преко индустријских процеса, па све до процеса који се користе у научно-истраживачке и војне сврхе.

У практичној реализацији електронских кола димензије компонената варирају од најситнијих до доста крупних, користи се велики број материјала са различитим електричним и термичким особинама као и компоненте са резличитим температурним опсезима. Стога постоји потреба за регулацијом температуре како би се лемилицом могле залемити све компоненте без обзира на димензије пинова, материјал и тд. Сви произвођачи лемне опреме поред осталог производе и лемне станице са регулацијом температуре и баве се њиховим развојем и усавршавањем како би могли да удовоље све захтјевнијем тржишту. У овом раду је приказано пројектовање система за регулацију температуре лемилице помоћу микроконтролера на бази *ARM Cortex-M3.*

1.1 ARM Cortex-M3

Cortex-M3 је једна грана *ARMv7* фамилије, пројектован је за коришћење у системима са високим перформансама у комбинацији са малом потрошњом енергије. Такође, и цијена је довољно ниска да ови контролери могу да конкуришу осмобитним и шеснаестобитним. *Cortex-M3* је *Harward* архитектуре, са 32-битном *CP* јединицом, редукованим сетом инструкција и тростепеним *pipeline*-ом. Једна од кључних компонената овог контролера је контролер прекида (*NVIC*) који је пројектован за веома брзо реаговање на прекиде и може да прати до 240 периферија. Систем прекида подржава режим ниске потрошње, па је могуће подесити *CPU* тако да након изласка из прекида језгро аутоматски одлази на спавање до наредног прекида. Подржан је рад са оперативним системима у реалном времену што је од изузетног значаја за примјену у многим електронским системима [1].

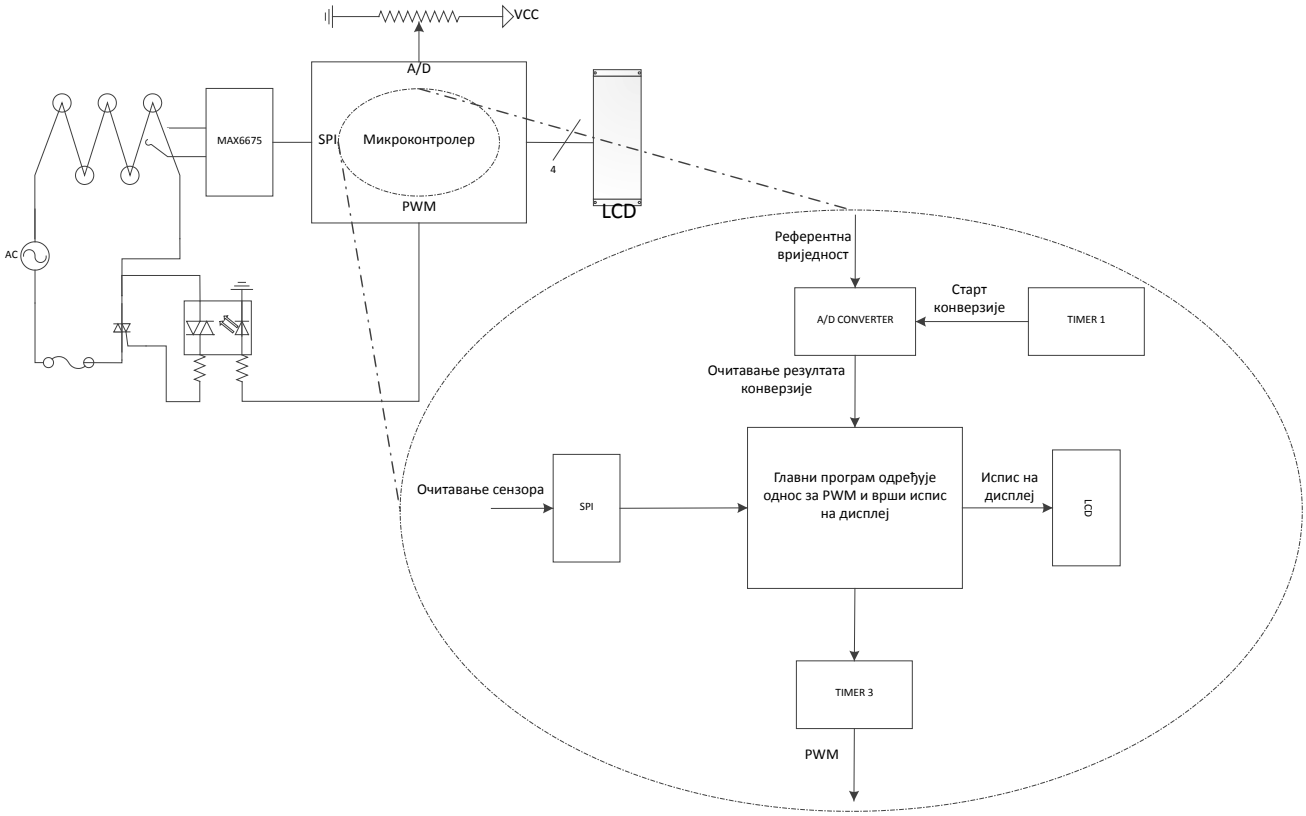
1.2 Развојна плоча STM32VLDISCOVERY

Развојна плоча *STM32VLDISCOVERY* садржи микроконтролер *STM32F100RBT6B* у 64-пинском кућишту, са 128 KB флеш меморије и 8 KB *RAM,* интерфејс за дебаговање, два тастера и четири *LE* диоде.

Диода *LD1* је индикатор *USB* комуникације, *LD2* је индикатор напајања, а преостале двије диоде су везане на пинове *PC8* и *PC9* и могу се користити од стане корисника. Плоча се напаја преко *USB* кабла, а постоји и могућност напајања споља. Један тастер служи за ресетовање контролера, а други је кориснички [2].

2 Пројектовање

Да би се реализовао пројекат потребно је испројектовати додатни хардвер и његово повезивање са развојном плочом, као и софтвер који ће да управља процесом. На слици 2.1. је приказана блок шема система, укључујући и унутрашњост микроконтролера.

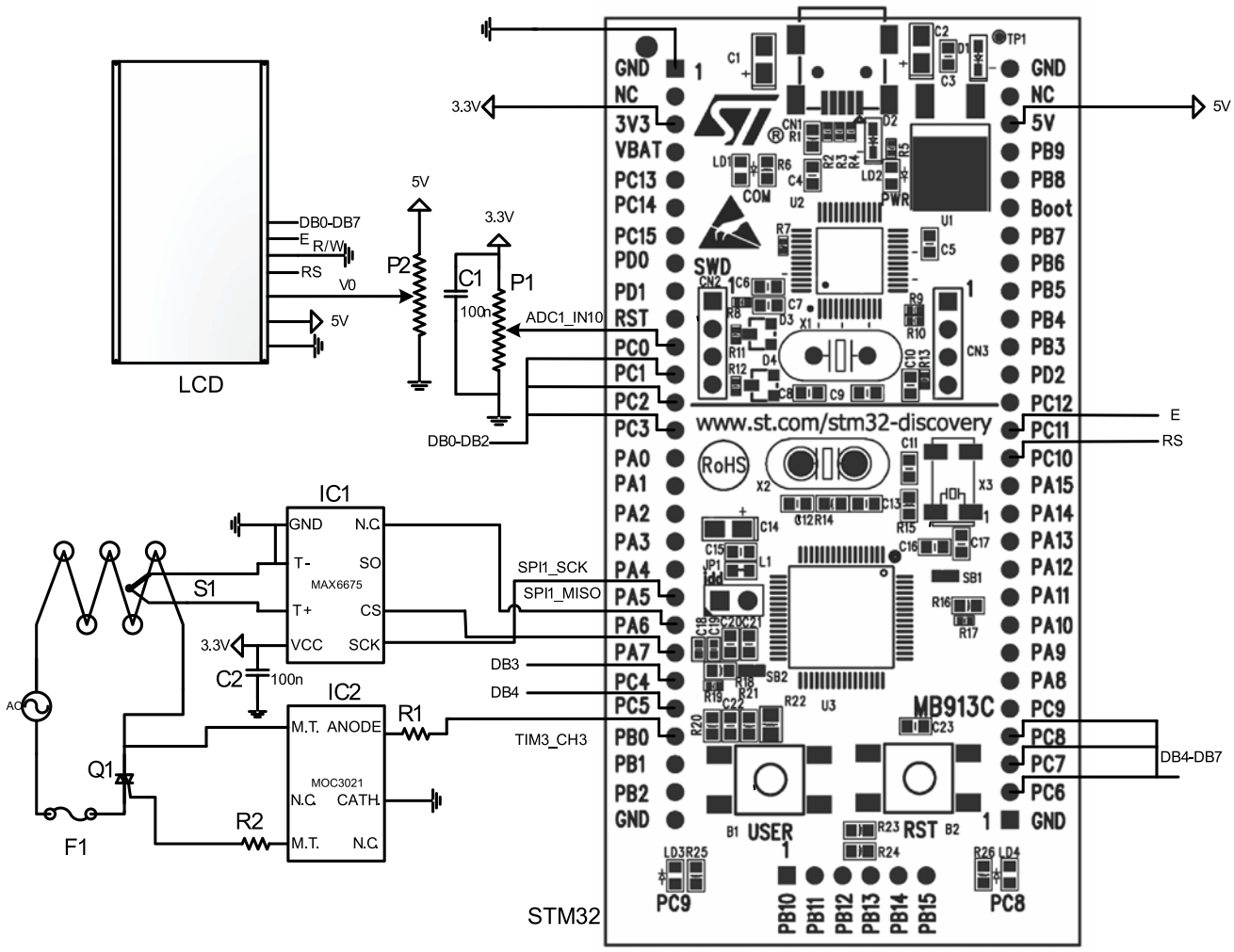


Слика 2.1: Блок шема система.

Жељена вриједност температуре се задаје потенциометром, а алалогно-дигитални конвертор дигитализује ту референцу. Подаци о тренутној температури се добијају од сензора преко *SPI*. Програм микроконтролера упоређује жељену и тренутну вриједност и на основу тога примјењује управљање. Гријачем се управља путем импулсне ширинске модулације, а сигнале који контролишу гријач генерише тајмер 3. Како би корисник имао увид у стање система, вриједности задате и тренутне температуре се исписују на *LC* дисплеј.

2.1 Хардвер

Пројектовање хардверског дијела се своди на избор компонената које одговарају захтјевима и реализацију њихових веза, као и избор периферија микроконтролера, односно, његових пинова. На слици 2.2. је приказана електрична шема система.



Слика 2.2: Електрична шема система.

С обзиром на температурни опсег врха лемилице, за читање температуре се користи *K* термопар [3] са чипом *MAX6675* [4] који врши дигитализацију и компензацију утицаја хладног споја, а преко *SPI* комуницира са микроконтролером. Контрола гријача је реализована помоћу тријака који је оптички галвански одвојен од плоче са микроконтролером и контролише се *PWM* излазом тајмера 3. За приказ података се користи стандардни 2x16 *LCD* [5] који се напаја са 5*V* и комуницира са микроконтролером преко десет паралелних линија.

2.2 Софтвер

Комплетан софтвер је реализован у развојном окружењу *IAR Embedded Workbanch for ARM 6.40* који садржи *Text editor, C/C++* компајлер, линкер и *C-SPY® debugger* са симулатором. Ово окружење подржава рад са *CMSIS* [6] библиотекама што је од изузетне важности с обзиром на сложеност контролера.

2.2.1 Иницијализација

Да би се испрограмирао микроконтролер за неку намјенску апликацију прво је потребно иницијализовати све што ће бити коришћено. Иницијализација система обухвата довођење такта периферијама, конфигурисање портова тако да њихова функција одговара намјени и иницијализацију периферија. Први и трећи тајмер су подешени тако да раде у *PWM* моду, тајмер 1 се користи за покретање аналогно/дигиталне конверзије, а тајмер 3 за контролу гријача. *DMA* контролер је подешен да пребацује податке из регистра података *AD* конвертора у циркуларни бафер у меморији. *AD* конвертор је подешен тако да дозвољава прекид тајмера и *DMA* приступ. *SPI* ради у мастер моду у режиму пријема, податке чита на силазну ивицу такта а први прочитани бит је најстарији. У прилогу 6.1.2 се налази комплетан код функције за иницијализацију.

2.2.2 Читање жељене температуре

Референтна температура се задаје потенциометром који је везан на канал 10 аналогно/дигиталног конвертора. Тајмер 1 ради у *PWM* режиму и његов регистар *CC1* покреће конверзију. Након завршене конверзије, *DMA* контролер преноси податке из регистра *DR, AD* конвертора у бафер у меморији. Пошто се температура регулише на доста широком опсегу, од 200°C до 450°C референтна температура може да се задаје у корацима од 10°C.

2.2.3 Читање тренутне температуре

Температура лемилице се чита путем *SPI,* са чипа *MAX6675* који дигитализује вриједности очитане са термопара. Постављањем бита *CS* на нулу покреће се читање и очитава се 16-битни податак чији је формат приказан у табели 2.1.

Табела 2.1: Формат података са *MAX6675*.

Dummy Sign bit	*12-bit temperature reading*												*Thermocouple reading*	*Device id*	*state*
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	*MSB*										*LSB*		0	*Three-state*

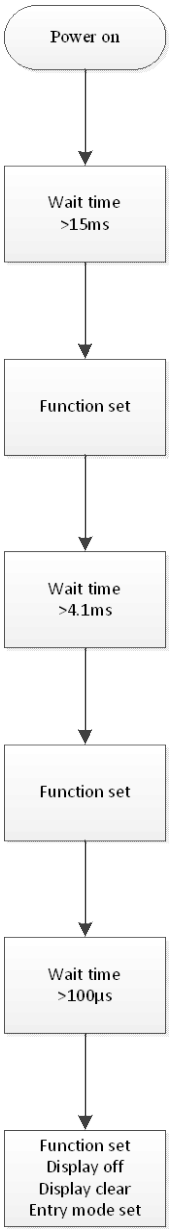
Очитани податак представља температуру у резолуцији од 0.25°C, али се користи резолуција од 1°C. Нова конверзија се покреће постављањем бита *CS* на логичку јединицу и након тога је потребно сачекати да се конверзија заврши [4]. Комплетан код функције за читање температурног сензора се налази у прилогу 6.1.4.

2.2.4 Контрола гријача

На основу информација о задатој, и тренутној температури, главни програм у сваком пролазу кроз петљу одређује параметар *D* за ширинску модулацију. *D* физички представља снагу гријача у опсегу од 0 до 1000. Примијењено је пропорционално и интегрално дејство, а константе су одређене експериментално. Тајмер 3 је конфигурисан да ради у *PWM* моду и у његов *CC1* регистар се уписује вриједност *D,* а тај излазни пин контролише прекидач на мрежном напајању лемилице.

2.2.5 Испис на дисплеј

Да би се иницијализовао дисплеј потребно је испоштовати процедуру дефинисану од стране произвођача [5]. На слици 2.3. је приказан дијаграм тока при иницијализацији.



Слика 2.3: Дијаграм тока при иницијализацији LCD.

Током иницијализације је изабрана осмобитна комуникација са дисплејем, укључене су обје линије и дефинисан фонт који се користи. У табели 2.2. су приказане функције појединих пинова.

Табела 2.2: Функције пинова *LC* дисплеја.

Пин	Симбол	Функција
1	*Vcc*	*GND*
2	*Vdd*	Напајање, 5V
3	*V0*	Подешавање контраста
4	*RS*	*Register Select*
5	*R/W*	*Read/Write*
6	*E*	*Enable*
7-14	*DB0-DB7*	Линије за податке

3 Реализација

Спецификација електронских компонената потребних за израду хардвера је приказана у табели 3.1., а списак осталих дијелова у табели 3.2

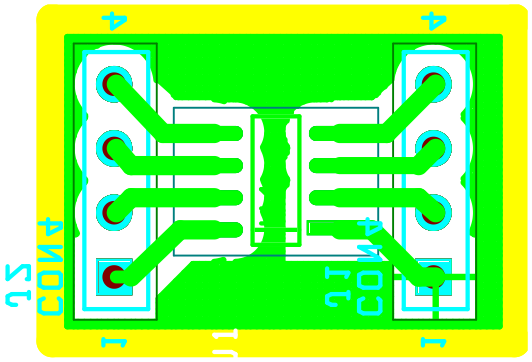
Табела 3.1: Списак електронских компонената.

Ознака на шеми	Компонента	Кућиште	Опис
STM32	*STM32vlDiscovery*		Развојна плоча са микроконтролером
IC1	*MAX6675*	*SOIC 8-pin*	Дигитални конвертор
IC2	*MOC3021*	*DIP 6-pin*	Оптокаплер
R1	Отпорник, 470Ω		1W
R2	Отпорник, 270Ω		1W
P1	Потенциометар, 10kΩ		Једнообртни потенциометар
P2	Тример, 10kΩ		Једнообртни тример, лежећи
C1, C2	Кондензатор, 100nF		Керамички
Q1	*BT136-600E*	*TO220*	Тријак, 4A
S1	K термопар	*1.5x150mm*	
LCD	Дисплеј		*2x16*
F1	Осигурач, 3.15A, T	*5x20mm*	Троми

Табела 3.2: Списак неелектричних компонената.

Компонента	Опис
Матадор плоча	10x10cm
Конектори pin header мушки	40x1
Конектори pin header женски	40x1,
Кућиште осигурача	5x20mm
Једнострани пертинакс	
Одстојници	10mm
Шрафови	M3x20mm
Утичница, мрежна	
Утикач, мрежни	

Проторип уређаја је реализован на испитној матадор плочи димензија 10x10 центиметара. Пошто је *MAX6675* пакован у *SMD SOIC* кућиште, израђен је адаптер *SOIC/DIP* који је приказан на слици 3.1.



Слика 3.1: Адаптер SOIC/DIP.

4 Тестирање

Систем је тестиран у реалним условима, укључена је лемилица снаге 25W, а сензор је причвршћен на њен врх. Утврђено је да сви дијелови система функционишу како је очекивано, и да је могуће остварити регулацију температуре на жељену вриједност. Окретањем потенциометра мијења се жељена температура што се јасно види на дисплеју. Промјене тренутне температуре се такође приказују и уочава се тенденција повећања, односно смањења у зависности од реферeнце.

5 Закључак

Испројектован је систем регулације температуре лемилице који контролише микрконтролер *STM32F100RB*. Исти систем се може користити и за регулацију било ког температурног процеса уз евентуалне модификације како што су коришћење другог сензора, постављање друкчијег опсега референтне температуре, промјена резолуције итд. Поред тога, ресурси контролера дозвољавају и проширење система, било да се ради о сложенијем систему регулације, са више сензора и гријача, било да је ријеч о неком сасвим другом процесу који се одвија паралелно.

Литература

[1] *Trevor Martin, The Insider's Guide To The STM32 ARM Based Microcontroller, Hitex, 22.10.2009.*

[2] *UM0919 User Manual, ST, September 2010.*

[3] *Sensor, Thermocouple, Type K, Mineral Insulated, Miniature Plug, datasheet*.

[4] *Cold-Junction-Compensated K-Thermocouple-to-Digital Converter, Maxim Integrated Products, 2002*.

[5] *Specification MC21605A6W-SPR, Midas Components Ltd, 06.2006.*

[6] *Joseph Yiu, The Definitive Guide To The ARM Cortex-M3, Elsevier, Oxford, 2010. Ch. 10.*

[7] *Geoffrey Brown, Descovering the STM32 Microcontroller, January 8, 2013.*

6 Прилози

6.1 Програмски код микроконтролера

6.1.1 Главни програм *main.c*

```
#include "stm32F10x.h"
#include "STM32vldiscovery.h"
#include "Sensor_Read.h"
#include "PWM_Out.h"
#include "System_Initialisation.h"
#include "lcd_driver.h"
#include "lcd_out.h"
#include "math.h"


  int referentna_temperatura;
  int ocitana_temperatura;
  int D = 500;                                                    // Однос импулс/пауза
  int k = 50;                                          // Константа пропорционалног дејства
  int ki= 10;                                                  // Интеграциона константа
  int delta;
  #define sign(x) (( x > 0 ) - ( x < 0 ))



//*****************************************************************************//
//              Регулација температуре лемилице, главни програм                //
//*****************************************************************************//

int main()
{
  System_Initialisation();                                    // Иницијализација периферија
  lcd_init();                                                     // Иницијализација екрана

  while(1)
  {
    referentna_temperatura = Pot_Read();                        // Читање жељене температуре
    ocitana_temperatura = Sensor_Read();                       // Читање тренутне температуре
    delta = referentna_temperatura - ocitana_temperatura;

    D = delta*k + sign(delta)*ki;                       // Одређивање односа за ширинску модулацију
    if (D<0) D=0;
    if (D>1000) D=1000;

    PWM_Out(D);                                                // Упис у CC регистар тајмера 3
    lcd_out(referentna_temperatura, ocitana_temperatura);          // Испис података на дисплеј
    }
}
```

6.1.2 Функција за иницијализацију

```
#include "stm32F10x.h"
#include "STM32vldiscovery.h"
#include "Sensor_Read.h"
#include  "PWM_Out.h"
#include "misc.h"
#include "System_Initialisation.h"
#include "lcd_driver.h"
#include "stm32f10x_conf.h"
```

```
//******************************************************************************//
//                          Дефинисање структура                                //
//******************************************************************************//

  GPIO_InitTypeDef         GPIO_InitStructure;
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseInitStruct;
  TIM_OCInitTypeDef        TIM_OCInitStruct;
  SPI_InitTypeDef          SPI_InitStructure;
  NVIC_InitTypeDef         NVIC_InitStructure;
  ADC_InitTypeDef          ADC_InitStructure;
  DMA_InitTypeDef          DMA_InitStructure;


  #define ADC1_DR_Address    ((u32)0x4001244C)
  vu16 ADC_RegularConvertedValueTab[64];

  extern char state;
  static __IO uint32_t TimingDelay;
  void Delay(__IO uint32_t nTime);


//******************************************************************************//
//                          Довођење такта периферијама                         //
//******************************************************************************//

  void RCC_Init(void)
  {
    RCC_ADCCLKConfig(RCC_PCLK2_Div2);                                  // Конфигурисање такта за ADC
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,     ENABLE);       // Довођење такта за DMA контролер
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,  ENABLE);                       // Порт C
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1,   ENABLE);                       // Тајмер 1
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,   ENABLE);                  // Довођење такта ADC-у
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,  ENABLE);                       // Порт B
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,   ENABLE);                 // Алтернативне функције
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,   ENABLE);                       // Тајмер 3
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,  ENABLE);                       // Порт A
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1,   ENABLE);                       // SPI
  }



//******************************************************************************//
//                          Иницијализација GPIO портова                        //
//******************************************************************************//

  void GPIOPorts_Init(void)
  {
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;                                  // PC0, аналогни улаз
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5 |
GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_10 | GPIO_Pin_11;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;            // PCx пинови за lcd, out push-pull
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;                          // Конфигурација пина за sck
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;               // alternative function push-pull
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;                          // Конфигурација пина за miso
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;                           // Конфигурација пина за cs
```

```
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;                              // out push-pull
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_Init(GPIOA, &GPIO_InitStructure);

      GPIO_StructInit(&GPIO_InitStructure);                                  // Дефинисање порта за pwm
      GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_0;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStructure.GPIO_Mode  =  GPIO_Mode_AF_PP;             //  Конфигурација  за  алтернативну
функцију - Push Pull
      GPIO_Init( GPIOB, &GPIO_InitStructure );
    }


  //*************************************************************************//
  //                        Иницијализација тајмера                          //
  //*************************************************************************//

    void TIM_Init(void)
    {
      TIM_TimeBaseStructInit(&TIM_TimeBaseInitStruct);               // Конфигурација тајмера за adc
      TIM_TimeBaseInitStruct.TIM_Period = 150 - 1;
      TIM_TimeBaseInitStruct.TIM_Prescaler = 0;
      TIM_TimeBaseInitStruct.TIM_ClockDivision = 0x0;
      TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;
      TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStruct);
      TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
      TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
      TIM_OCInitStruct.TIM_Pulse = 150 / 2;
      TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
      TIM_OC1Init(TIM1, &TIM_OCInitStruct);

      TIM_TimeBaseStructInit( &TIM_TimeBaseInitStruct );                    // Тајмер T3, pwm, 100Hz
      TIM_TimeBaseInitStruct.TIM_ClockDivision = TIM_CKD_DIV4;
      TIM_TimeBaseInitStruct.TIM_Period = 1000 - 1;                           // Период 1000
      TIM_TimeBaseInitStruct.TIM_Prescaler = 240 - 1;                        // Div 240 prescaler
      TIM_TimeBaseInit( TIM3, &TIM_TimeBaseInitStruct );
      TIM_OCStructInit( &TIM_OCInitStruct );             // Иницијализација OC структуре тајмера TIM3
      TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
      TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_PWM1;
      TIM_OCInitStruct.TIM_Pulse = 0;                     // 0 .. 1000 (0=Always Off, 1000=Always On)
      TIM_OC3Init( TIM3, &TIM_OCInitStruct );
      TIM_Cmd( TIM3, ENABLE );                                                 // Стартовање тајмера
    }



  //*************************************************************************//
  //                     Иницијализација DMA контролера                      //
  //*************************************************************************//

    void DMACon_Init(void)
    {
      DMA_DeInit(DMA1_Channel1);
      DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;                 // Адреса изворишта
за dma пренос - DATA REGISTER ADC-a
      DMA_InitStructure.DMA_MemoryBaseAddr = (u32)ADC_RegularConvertedValueTab;   // Адреса одредишта
      DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
      DMA_InitStructure.DMA_BufferSize = 64;
      DMA_InitStructure.DMA_PeripheralInc  =  DMA_PeripheralInc_Disable;                 //  Искључено
инкрементирање адресе у периферији
      DMA_InitStructure.DMA_MemoryInc  =  DMA_MemoryInc_Enable;                           // Укључено
инкрементирање адресе у меморији
      DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; // Величина 16 бита
      DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
      DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;                    // Циркуларни режим преноса
      DMA_InitStructure.DMA_Priority = DMA_Priority_High;
      DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
      DMA_Init(DMA1_Channel1, &DMA_InitStructure);
      DMA_Cmd(DMA1_Channel1, ENABLE);
```

```
  }


//*****************************************************************************//
//                        Иницијализација AD конвертора                        //
//*****************************************************************************//

  void ADCCon_Init(void)
  {
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;                         // Једна конверзија
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_NbrOfChannel = 1;
    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_RegularChannelConfig(ADC1, ADC_Channel_10, 1, ADC_SampleTime_1Cycles5);
    ADC_ExternalTrigConvCmd(ADC1, ENABLE);                          // Омогућавање екстерног тригера
    ADC_DMACmd(ADC1, ENABLE);                                      // Омогућавање DMA преноса за ADC
    ADC_Cmd(ADC1, ENABLE);

    ADC_ResetCalibration(ADC1);                                                 // adc конверзија
    while(ADC_GetResetCalibrationStatus(ADC1));
    ADC_StartCalibration(ADC1);
    while(ADC_GetCalibrationStatus(ADC1));
  }



//*****************************************************************************//
//                        Иницијализација SPI периферије                       //
//*****************************************************************************//

  void SPIPer_Init(void)
  {
    SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
    SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;                          // Читање на силазну ивицу
    SPI_InitStructure.SPI_CRCPolynomial = 0;
    SPI_InitStructure.SPI_DataSize = SPI_DataSize_16b;
    SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_RxOnly;              // Ради само пријем
    SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;                          // Први бит је msb
    SPI_InitStructure.SPI_Mode = SPI_Mode_Master;                               // master mode
    SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
    SPI_Init(SPI1, &SPI_InitStructure);
    SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);                            // enable spi
  }



//*****************************************************************************//
//                        Иницијализација система                              //
//*****************************************************************************//

  void System_Initialisation(void)
  {
    RCC_Init();                                                 // Довођење такта периферијама
    GPIOPorts_Init();                                               // Иницијализација портова

    GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);

    TIM_Init();                                                     // Иницијализација тајмера
    DMACon_Init();                                                      // Иницијализација dma
    ADCCon_Init();                                                      // Иницијализација adc

    TIM_Cmd(TIM1, ENABLE);               // Дозвола рада тајмера тек када се конфигуришу DMA и ADC
    TIM_CtrlPWMOutputs(TIM1, ENABLE);                           // Генерисање PWM излаза за тајмер 1
```

```
    SPIPer_Init();                                                          // Иницијализација spi

    if (SysTick_Config(SystemCoreClock / 1000))
    {
      while (1);
    }
  }



//*****************************************************************************//
//                          Читање резултата конверзије                        //
//*****************************************************************************//

  int Pot_Read (void)                                  // Читање потенциометра, резултат конверзије
  {
    return ((ADC_RegularConvertedValueTab[0])/160+20)*10;                  // Заокруживање на xx0
  }



//*****************************************************************************//
//                          Функција за кашњење                                //
//*****************************************************************************//



  void Delay(__IO uint32_t nTime)
  {
    TimingDelay = nTime;

    while(TimingDelay != 0);
  }


  void TimingDelay_Decrement(void)
  {
    if (TimingDelay != 0x00)
    {
      TimingDelay--;
    }
  }
```

6.1.3 Драјвер за дисплеј

```
#include "lcd_driver.h"
#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_gpio.h"
#include "System_Initialisation.h"



//*****************************************************************************//
//                          Слање нижих 8 бита на порт C                       //
//*****************************************************************************//

  void Port_Write (unsigned char podatak)
  {
    unsigned char maska = 0x01;
    unsigned int pin=0x0002;                                               // Почиње од пина PC.1
    int i;

    for (i = 0; i<=7; i++)
    {
```

```
      if ((podatak & maska) != 0x00)
        GPIO_WriteBit(LCD_PORT, pin, Bit_SET);
      else GPIO_WriteBit(LCD_PORT, pin, Bit_RESET);

      maska = maska << 1;
      pin = pin << 1;
    }
  }



//*****************************************************************************//
//                        Иницијализација LCD-a                                //
//*****************************************************************************//

  void lcd_init(void)
  {                                        // Испис у 2 линије, осмобитни приступ, фонт 5x8 пиксела
    Delay(15);
    lcd_write_command(Function_Set);
    Delay(5);
    lcd_write_command(Function_Set);
    Delay(1);
    lcd_write_command(Function_Set);
    lcd_write_command(Function_Set);
    lcd_write_command(Display_Off);
    lcd_write_command(Clear_Display);
    lcd_write_command(Entery_Set_Mode);
  }



//*****************************************************************************//
//                        Упис команде у LCD контролер                         //
//*****************************************************************************//

  void lcd_write_command(unsigned char command)
  {
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_RS, Bit_RESET);                                 // RS = 0
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_RESET);                                 // E = 0
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_SET);                                   // E = 1
    Port_Write(command);
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_RESET);                                 // E = 0
  }



//*****************************************************************************//
//                        Упис података у LCD контролер                        //
//*****************************************************************************//

  void lcd_write_data(unsigned char char_data)
  {
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_RS, Bit_SET);                                   // RS = 1
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_RESET);                                 // E = 0
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_SET);                                   // E = 1
    Port_Write(char_data);
    Delay(1);
    GPIO_WriteBit(LCD_PORT, lcd_EN, Bit_RESET);                                 // E = 0
  }
```

```
//*****************************************************************************//
//                         Испис карактера на дисплеју                         //
//*****************************************************************************//

  void display_char(unsigned char position, unsigned char char_data)
  {
    unsigned char komanda;

    lcd_write_command(Display_On);                        // Displey ON, Cursor ON, Blink Cursor OFF
    Delay(10);
    if(position >= 0x10) komanda = position + 0xB0;                    // Пребацивање у други ред
    else  komanda = position + 0x80;
    lcd_write_command(komanda);
    lcd_write_data(char_data);
  }



//*****************************************************************************//
//                         Брисање сегмента дисплеја                           //
//*****************************************************************************//

  void delete_part(unsigned char position, unsigned char length)
  {
    unsigned char komanda;
    unsigned char data=' ';
    int j;

    lcd_write_command(Display_On);                        // Displey ON, Cursor ON, Blink Cursor OFF
    Delay(10);
    for (j=0; j<length; j++)
    {
      if(position >= 0x10) komanda = position + 0xB0;                    // Пребацивање у други ред
      else  komanda = position + 0x80;

      lcd_write_command(komanda);
      lcd_write_data(data);
      position=position+1;
    }
  }



//*****************************************************************************//
//               Испис стринга на LCD од прве позиције колоне цол              //
//*****************************************************************************//

  void display_string(unsigned char col, unsigned char string_data[16])
  {
    unsigned char col_tem,i;                                  // Испис од прве позиције колоне col
    unsigned char karakter;
    unsigned char * data=string_data;
    col_tem=col<<4;
    i=0;
    while((data[i] != 0x00)&&(i<16))
    {
      karakter=data[i];
      display_char(col_tem,karakter);
      col_tem=col_tem+1;
      i=i+1;
     }
  }



//*****************************************************************************//
//                   Испис реалног броја на позицији pos                       //
//*****************************************************************************//
```

```
void display_real(unsigned char pos, float real_num)
{
  float broj = real_num;
  int ceo_br;
  unsigned char karakter = 0x30;
  unsigned char i = 0;
  unsigned char n = 0;
  ceo_br = (int)broj;

  while(ceo_br!=0)
  {
    ceo_br=ceo_br/10;
    n++;
  }

  ceo_br = (int)broj;

  while(ceo_br!=0)
  {
    karakter = (unsigned char)(ceo_br%10) + 0x30;
    display_char(pos+n-i-1,karakter);
    ceo_br = ceo_br/10;
    i++;
  }

  ceo_br = (int)broj;
  broj = broj-ceo_br;
  if (broj!= 0)
  {
    karakter = '.';
    display_char(pos+i,karakter);
    i++;
  }

  while (broj > 0.01)
  {
    broj = broj*10;
    ceo_br = (int)broj;
    broj = broj-ceo_br;

    if(broj>0.99)
    {
      ceo_br=ceo_br+1;
      broj=1-broj;
    }

    karakter = (unsigned char)(ceo_br%10) + 0x30;
    display_char(pos+i,karakter);
    i++;
  }
}
```

6.1.4 Функција за читање сензора

```
#include "stm32F10x.h"
#include "STM32vldiscovery.h"
#include "stm32f10x_spi.h"
#include "stm32F10x_conf.h"
#include "System_initialisation.h"

  int i=0;
```

```
  //*****************************************************************************//
  //              Читање температурног сензора преко SPI                         //
  //*****************************************************************************//

    int Sensor_Read(void)
    {
      GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);
      Delay(1);
      GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_SET);   // Покретање конверзије температуре постављањем CS
на један
      Delay(300);                                                    // Чекање док траје конверзија

      SPI_Cmd(SPI1, ENABLE);
      GPIO_WriteBit(GPIOA, GPIO_Pin_7, Bit_RESET);    // Поставља се CS на нула како би се покренуло
читање
      while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);

      SPI_Cmd(SPI1, DISABLE);
      SPI_I2S_ClearFlag(SPI1, SPI_I2S_FLAG_RXNE);
      return (SPI1->DR)/32 ;                            // Функција враћа температуру у степенима.
Вриједност се налази од 14 до 3 бита, али у резолуцији од 0,25
    };
```

6.1.5 Функција за испис на дисплеј

```
  #include "stm32F10x.h"
  #include "STM32vldiscovery.h"
  #include "lcd_driver.h"



  //*****************************************************************************//
  //              Испис жељене и тренутне температуре на LCD                     //
  //*****************************************************************************//

    void lcd_out(int ref, int temp)
    {
      display_char(0,'T');
      display_char(1,'=');
      display_real(2, temp);                                        // Испис температуре лемилице
      display_char(5,0xDF);                                    // Код који представља знак за степен
      display_char(6,'C');

      display_char(7,' ');
      display_char(8,' ');

      display_char(9,'R');
      display_char(10,'=');
      display_real(11, ref);                                          // Испис жељене температуре
      display_char(14,0xDF);                                   // Код који представља знак за степен
      display_char(15,'C');

      display_char(16,'M');
      display_char(17,'i');
      display_char(18,'n');

      if (ref>=225) display_char(19,0xFF); else display_char(19, 0xDB);            // Исписује празан
оквир или пуно поље, у зависности да ли је температура већа од дате вриједности. Корак је 25 степени
      if (ref>=250) display_char(20,0xFF); else display_char(20, 0xDB);
      if (ref>=275) display_char(21,0xFF); else display_char(21, 0xDB);
      if (ref>=300) display_char(22,0xFF); else display_char(22, 0xDB);
      if (ref>=325) display_char(23,0xFF); else display_char(23, 0xDB);
      if (ref>=350) display_char(24,0xFF); else display_char(24, 0xDB);
      if (ref>=375) display_char(25,0xFF); else display_char(25, 0xDB);
      if (ref>=400) display_char(26,0xFF); else display_char(26, 0xDB);
```

```
    if (ref>=425) display_char(27,0xFF); else display_char(27, 0xDB);
    if (ref>=450) display_char(28,0xFF); else display_char(28, 0xDB);

    display_char(29,'M');
    display_char(30,'a');
    display_char(31,'x');
    }
```

6.1.6 Функција за упис у *CC* регистар тајмера 3

```
#include "stm32F10x.h"
#include "STM32vldiscovery.h"



//*****************************************************************************//
//              Упис односа импулс/пауза у тајмер 3                            //
//*****************************************************************************//

  int dute_ratio;

  void PWM_Out(int dute_ratio)
  {
    TIM3->CCR3 = dute_ratio;
  }
```