|  |  |
| --- | --- |
| etf_logo_skew_shapes copy.tif | **ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ**  **УНИВЕРЗИТЕТ У БАЊОЈ ЛУЦИ** |

Пројектни задатак из предмета „Микропроцесорски системи 2“ ДИГИТАЛНИ ВОЛТМЕТАР

Студент: Небојша Марковић

Број индекса: 1241/11

Предметни наставник: др Златко Бундало

Предметни асистент: др Младен Кнежић

|  |  |
| --- | --- |
| Број бодова: |  |

# САДРЖАЈ :

[САДРЖАЈ : 1](#_Toc516528313)

[Листа фигура : 2](#_Toc516528314)

[1. ДЕФИНИСАЊЕ ПРОЈЕКТА 3](#_Toc516528315)

[2. ЗАХТЈЕВИ ЗА РЕАЛИЗАЦИЈУ ПРОЈЕКТА 4](#_Toc516528316)

[2.1. Увод у израду пројекта 4](#_Toc516528317)

[2.2. Кориснички захтјеви 5](#_Toc516528318)

[2.3. Системски захтјеви 6](#_Toc516528319)

[*2.3.1.* *Спецификација хардвера* 10](#_Toc516528320)

[2.4. Софтверсо рјешење 18](#_Toc516528321)

[3. ТЕСТИРАЊЕ ПРОИЗВОДА 21](#_Toc516528322)

[4. ИЗВОРНИ КОД 23](#_Toc516528323)

[4.1. Аликација 23](#_Toc516528324)

[*4.1.1.* *voltmeter\_main.c* 23](#_Toc516528325)

[*4.1.2.* *voltmeter.h* 24](#_Toc516528326)

[*3.1.3* *voltmeter.c* 26](#_Toc516528327)

[4.2. Module API слој 29](#_Toc516528328)

[*4.2.1.* *display.h* 29](#_Toc516528329)

[*4.2.2.* *display.c* 31](#_Toc516528330)

[*4.2.3.* *ADConv.h* 32](#_Toc516528331)

[*4.2.4.* *ADConv.c* 33](#_Toc516528332)

[*4.2.5.* *spi.h* 34](#_Toc516528333)

[*4.2.6.* *spi.c* 35](#_Toc516528334)

[*4.2.7.* *timer.h* 36](#_Toc516528335)

[*4.2.8.* *timer.c* 38](#_Toc516528336)

[*4.2.9.* *interrupts.h* 40](#_Toc516528337)

[*4.2.10.* *interrupts.c* 41](#_Toc516528338)

[5. ЛИТЕРАТУРА 43](#_Toc516528339)

# Листа фигура :

[*Сл. 2‑1* *Блок шема система* 7](#_Toc516528340)

[*Сл. 2‑2 Предложена хардверса архитектура система* 8](#_Toc516528341)

[*Сл. 2‑3 Развојна плоча Easy8051 v6* 9](#_Toc516528342)

[*Сл. 2‑4 Распоред пинова микроконтролера АТ89Ѕ8253* 10](#_Toc516528343)

[*Сл. 2‑5 LCD дисплеј* 12](#_Toc516528344)

[*Сл. 2‑6 Алгоритам иницијализације LCD-а* 14](#_Toc516528345)

[*Сл. 2‑7 Блок шема MCP3204 А/Д конвертора* 15](#_Toc516528346)

[*Сл. 2‑8 SPI комуникација кориштењем 8-битних сегмената (mode = 0,0 => SCLK на ниском логичком нивоу)* 16](#_Toc516528347)

[*Сл. 2‑9 SPI модул* 17](#_Toc516528348)

[*Сл. 2‑10 Првобитно предложжена софтверска архитектура* 18](#_Toc516528349)

[*Сл. 2‑11 Коначна софтверска архитектура* 19](#_Toc516528350)

*Сл. 3‑1 Мјерен напон од 500 [mV]*

*Сл. 3‑2 Мјерен напон од 5.0 [V]*

*Сл. 3‑3 Мјерен напон од 4.6 [V]*

*Сл. 3‑4 Мјерен напон од 3.4 [V]*

*Сл. 3‑5 Мјерен напон од 2 [V]*

*Сл. 3‑6 Мјерен напон од 1.3 [V]*

# ДЕФИНИСАЊЕ ПРОЈЕКТА

Пројектовати дигитални волтметар коришћењем A/D конвертора MCP3204 и микроконтролера AT89C51/52. Вриједност измјереног напона потребно је приказивати сваке секунде на LCD дисплеју који ради у четверобитном моду.

Потребно је држати се праксе о прегледном писању кода, раздвајања дефиниције и имплементације функција, као и писања одговарајућих коментара. У извјештају потребно је приложити поступак пројектовања, блок шему, електричну шему, изворни програм, те резултате симулације и експерименталног тестирања. Такође, приложити списак коришћених компоненти и њихове основне функционалне спецификације. За симулацију је на располагању *Proteus*, а за тестирње развојно окружење Easy8051.

# ЗАХТЈЕВИ ЗА РЕАЛИЗАЦИЈУ ПРОЈЕКТА

## Увод у израду пројекта

Модули захтјевани задатком (за реализацију дигиталног волтметра) су:

* AT89C51/52 микроконтролер
* MCP3204 A/D конвертор
* LCD дисплеј у 4-битном моду

Но, међутим, да бисмо дошли до закључка шта све треба да има систем који развијамо кренимо од најједноставнијих „**корисничких захтијева**“ који ће нам рећи шта је то што треба да реализујемо и шта све треба да посједује, те какве функционалности наш производ треба да омогући кориснику.

Затим, када се утврди шта тачно треба да реализујемо на ред долази следеће питање – „ Како то да реализујемо?“ - Тада састављамо „**системске захтјеве**“, на основу којих пројектујемо хардверски дио производа. На основу корисничких захтјева, утврђених у претходном кораку, закључујемо које компоненте треба да има систем који ћемо реализовати, те интеракцију која треба да постоји између њих у циљу остваривања жељених функционалности коначног производа. У овом кораку треба преиспитати корисничке захтјеве и префрмулисати их ако за то постоји потреба. С обзиром да у току реализације и даљег развоја производа може доћи до непредвиђених компликација, као и промјене или допуњавања захтјева од стране корисника, препоручљиво је развој радити уз помоћ развојне плоче, уколико је то могуће.

Након што је хардверско окружење познато прелази се на дефинисање „**софтверских захтјева**“, који у случају система који се састоји и од хардверског и од софтверског дијела треба јасно да дефинишу:

* које софтверске модуле треба развити
* какву интеракцију ти модули имају са хардвером
* какву интеракцију имају ти модули медјусобно
* слојевитост софтверског програма, као и евентуално (ако је познато)
* библиотеке које ће бити развијене

Тек сада на ред долази пописивање функција које ће бити потребно развити за реализацију пројекта.

Упоредо са утврђивањем и писањем захтјева на сваком нивоу пројектовања, потребно је описати и тестове (тестове одгварајућег типа на одговарајућем нивоу) који ће потврдити да су жељени захтјеви испуњени. Сем „**софтверских тестова**“, који служе да се потврди да су сви захтјеви наручиоца испуњени, у овом извјештају тестирање и тестови неће бити описани јер није то није тражено у задатку, а одузело би много времена.

## Кориснички захтјеви

Када помислимо на дигитални волтметар прво што закључујемо јесте да је то дигитални уређај који мјери напон. Међутим, из самог тог закључка не можемо извући све функционалности које дигитслни волтметар треба да има, него се морамо поставити у улогу корисника и размислити шта све треба да има један такав уређај.

На овај начин долазимо до корисничких захтјева, који гласе:

|  |  |
| --- | --- |
| **Ознака** | **Опис захтјева** |
| **К.З.1** | Волтметар има сонде за мјерење напона (или прикључке за њих) |
| **К.З.2** | Измјерена вриједнст напона се приказује на дисплеју |
| **К.З.3** | Вриједност која се приказује update-ује се сваке секунде и поново исписује |
| **К.З.4** | Након што се волтметар упали пали се и сигнална лампица која показује да је упаљен и спреман за кориштење |
| **К.З.5** | Аутоматски подешава скалу у којој се приказује мјерена вриједност у зависности од висине измјерене вриједности напона |
| **К.З.6** | Има могућност ручног избора скале |

*Таб. 2.1. Кориснички захтјеви*

Тестови који покривају ове захтјеве су интуитивно јасни – потребно је тестирати готов производ (на овом нивоу никако не хрдвер или софтвер посебно, нити неки од модула, него директно захтјев онакав какав је постављен).

Из *табеле 2.1.* коју смо добили анализом како захтјева корисника, тако и анализом функционалности које волтметар треба да има (и ако нису директно наведене кроз захтјеве корисника), добијамо детаљнију слику шта све производ који правимо треба да има/задовољава, али не и детаље реализације производа. Управо следећим кораком се приближавамо почетку реализације производа. Попишимо шта нам све од хардверских компонената/модула треба и какве везе треба да постоје међу њима да би се могле постићи горе описане функционалности. Приступ писању системских захтјева је следећи корак на путу до циља и у односу на писање корисничких захтјева биће мало детаљније урађен и образложен.

## Системски захтјеви

У овом кораку анализираћемо хрдверске компоненте које наш систем треба да има (на нивоу модула за потребе овог рада – у право пројектовање хардвера се нећемо упуштати због обима посла који превазилази обим једног пројектног задатка), те везе и интеракцију које треба да постоје између њих.

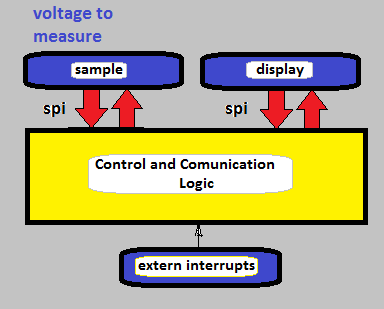
Овдје треба да пројектујемо дигитални волтметар, а знамо да је напон аналогни сигнал. На основу тога закључујемо да улазну величину треба прилагодити за даљу обраду у систему да би његов исправан рад био могућ. На овом мјесту се можемо одлучити како за А/Д конвертор, тако и за нпр. напонски сензор који као излазну величину даје сигнал чија фреквенција зависи од висине улазног напона. Како је задатком директно дефинисано да се користи А/Д конвертор MCP3204, узећемо да је то прва хардверска компонента система која је позната(за коју смо одлучили да ћемо је користити у систему).

Даље, знамо да мора бити могуће приказати на дисплеју вриједности мјерене величине и задатком је дефинисано да то буде LCD дисплеј који ради у четверобитном моду. Дисплеј који може да задовољи овај захтјев је нпр. Hitachi-јев HD44780U, али није наведено да мора овај дисплеј да се користи, па тако може да се изабере и било који други који може да ради у 4-битном моду.

Прије самог исписа на дисплеј информација која долази са А/Д конввертора мора да се обради и прилагоди за испис на дислеју. Због саме поставке задатка, гдје је речено да на располагању имамо микроконтролер AT89C51/52, изабраћемо да микроконтролер за ову обраду и прилагођење података за испис на дисплеј буде нпр. баш АТ89S8253.

Да би кориснички захтеви били испуњени од хардверских компонената потрбени су нам још тастери за ручну промјену скале и сигнална диода, док се аутоматско подешавање скале и освјежавање дисплеја сваке секунде може ријешити софтверски.

Да би ово могло бити функционално потребно је још да знамо везе између изабраних/предложених компонената, испланирамо размјену података, интеракцију и контролу у систему.



*Сл. 2‑1* *Блок шема система*

Не упуштајући се дубље у анализу шта, како и зашто смо тако поставили даћемо предлог како наш склоп треба да изгледа (*Сл.2.1. Блок шема система*и *Сл.2.2. Предложена хардверса архитектура система*). И ако овакву анализу треба спровести када се приступа пројектовању система – овдје ћемо рећи да се до предложене шеме дошло консултујући datasheet-ове одаабраних компонената.

Оно што је очигледно да смо заборавили напоменути (мада је интуитивно јасно ипак мора да буде наведено као захтјев) јесте да нам за исправан рад склопа требају како стабилан такт сигнал, тако и напон напајања Vcc и референтни напон(потребан за рад одабраног А/Д конвертора).

Ако је већ јасно да је неке корисничке захтјеве немогуће испунити или ако видимо да смо направили пропусте при пројектовању система, сада је право вријеме да се то поправи. До ријешења се ријетко када долази из првог покушаја. Уобичајно је да се грешке и пропусти уоче тек на следећим корацима израде проекта, па се тако сви захтјеви и ријешења надопуњују и мијењају све до краја пројекта(мада је наравно добро када постоји што мање потребе за тим).



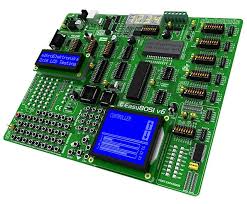
*Сл. 2‑2 Предложена хардверса архитектура система*

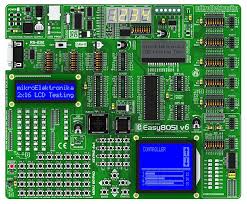
Оваквом анализом долазимо од системских захтјева које наш систем треба да испуњава (*Табела 2.2*) до предлога изгледа система приказаног на слици изнад.

|  |  |
| --- | --- |
| **Ознака** | **Опис захтјева** |
| **С.З.1** | Претварање аналогног напона у дигиталну величину погодну за обраду |
| **С.З.2** | Слање вриједности микроконтролеру на обраду (предложен SPI) протокол |
| **С.З.3** | Могућност управљања сигналном LED диодом |
| **С.З.4** | Могућност конфигурације и управљања дисплејом који ради у 4-битном моду |
| **С.З.5** | Тастери за избор скале и веза са системом за прилагођење података – микроконтролером |
| **С.З.6** | Стабилан такт сигнал |
| **С.З.7** | Напон напајања Vcc |
| **С.З.8** | Стабилан референтни напон |

*Таб. 2.2. Системски захтјеви*

Поређењем предложеног система са расположивим развојним плочама закључујемо да се овакав систем може добити кориштењем развојне плоче „Easy8051 v6“ (*Сл.2.2. Easy8051 v6*), произвођача „Микроелектроника“, па је дата плоча и изабрана за даљи развој ројекта.





*Сл. 2‑3 Развојна плоча Easy8051 v6*

### *Спецификација хардвера*

* **Mикроконтролер AT89S8253**

AT89S8253 је 8-битни микроконтролер високих перформанси, који је израђен у CMOS технологији, а има следеће карактеристике:

* Припада фамилији микроконтролера 8051
* 12 kB флеш меморије
* 2 kB EEPROM меморије
* 256 B интерног е-а за смијештање промјенљивих
* Напон напајања 4-6 [V]
* Радна фреквенција до 24 [MHz]
* 32 улазно-излазне линије распоређене у 4 порта са по 8 пинова
* Три 16-битна тајмера/бројача
* 9 извора прекида
* Прграмабилну UART сеијску комуникацију
* Програмабилни Watch Dog тајмер
* 2 додатна мода рада за смањење потрошње (Idle и Power-down мод)
* Тростепену заштиту уписаног програма

Приказаћемо пинове микроконтролра и њихов распоред у циљу описивања самог контролера и функције његових пинова (*Сл. 2.3. Распоред пинова микроконтролера АТ89Ѕ8253*) .



*Сл. 2‑4 Распоред пинова микроконтролера АТ89Ѕ8253*

* Vcc – напон напајања (4-6V).
* GND – маса
* Порт 0 (P0.0 – P0.7)– Када се дефинишу као излази, на сваки од ових пинова може се прикључити до 8 ТТЛ улазних кола. Када се одреди да раде као улази, тада су то улази бесконачне унутрашње отпорности - односно такви улази се понашају као да „висе у ваздуху“. Ако се користи додатна меморија, на овим изводима се наизмјенично појављују подаци и адресе (А0-А7) намјењени за приступ овом додатном меморијском чипу. Сигнал на ALE пину, притом, одређује шта ће и у ком тренутку бити прослијеђено на порт
* Порт 2 (P2.0 – P2.7) – Када се ови изводи користе као улази или излази, њихове особине су по свему сличне порту P1. Ако се користи спољна меморија, овде ће се појавити виши адресни бајт (А8-А15) за адресирање додатног чипа.
* Порт 3 (P3.0 – P3.7) – Као и на порту P1, изводи се и овде користе као универзални улази или излази али имају и неку додатну улогу
* RST – позитиван напонски импулс на овом пину ресетује микроконтролер.
* ALE/PROG – У нормалном раду, овај пин емитује поворку импулса чија је фреквенција једнака 1/6 фреквенције главног осцилатора. Ако се користи додатна меморија, сигнал на овом пину управља додатним регистром у којем се привремено смјешта нижи адресни бајт (А0-А7). За време уписивања програма у микроконтролер, овај пин такође служи као контролни улаз.
* PSEN – Сигнал са овог пина користи се за читање спољне програмске ROM меморије.
* EA/VPP – Када се овај пин споји на потенцијал масе, микроконтролер ће узимати програмске инструкције из спољне програмске меморије.
* XTAL1 – Улаз интерног осцилатора.
* XTAL2 – Излаз интерног осцилатора.

**ROM меморија** је капацитета 12 кB и изведена је у FLASH технологији, што омогућава велики број уписа/брисања садржаја меморије. Програмирање се врши преко уграђеног SPI модула (Serial Peripheral Interface). Могуће је да се ROM „догради“ спољним чипом.

**RAM меморија** се састоји од 3 блока са по 128 регистара чија структура одговара 8051 стандарду:

* 128 регистара опште намјене
* 128 меморијских локација резервисаних за SFR регистре. И ако су само неке од њих стварно искориштене, празне локације не треба да се користе за смјештање промјенљивих.
* 128 додатних регистара који су слободни за употребу (немају неку посебну намјену). Пошто имају исте адресе као и SFR регистри њима се приступа индиректним путем.

**EEPROM меморија** - Ово је посебна врста меморије која има осбине и RAM и ROM меморије - подаци се уписују и бришу у току рада али остају сачувани и после нестанка напона напајања. Овај микроконтролер има укупно 2 кB (2048 локација) EEPROM меморије.

* **LCD дисплеј - Hitachi HD44780**

LCD дисплеј служи за исписивање порука. Садржи контролер HD44780 фирме Hitachi. Састоји се из 2 реда са по 16 линија у којима се исписују карактери док се свако од ових поља састоји од матрице величине 5х8 пиксела као што је приказано на слици (*Сл.2.4. LCD дисплеј*)



*Сл. 2‑5 LCD дисплеј*

Могучности дисплеја су: приказује сва слова абецеде, грчка слова, знакове интерпукције,

математичке симболе итд. Поред тога могуће је приказати и знакове које корисник сам дефинише. Такође посједује и аутоматско помјерање порука преко екрана (шифтовање у лијево и удесно), појављивање курсора, позадинско освјетљење и слично. Контраст на екрану зависи од напона напајања и од тога да ли се поруке исписују у једном или два реда. Због тога се на извод означен са Vee прикључује променљиви напон од 0 – Vdd (обично се за ово користи тример потенциометар).

Унутар дисплеја се налазе 3 меморијска блока:

* DDRAM – Display Data RAM,
* CGROM – Kараkтер Генератор ROM,
* CGRAM – Kараkтер Генератор RAM.

**DDRAM меморија** - У ову меморију смјештају се карактери који треба да буду приказани на дисплеју. Величина ове меморије је довољна за смјештање 80 знакова, а један дио ових локација има директне везе са пољима на екрану.

**CGROM меморија** - У ову меморију фабрички је уписана мапа са изгледом свих карактера које дисплеј може да прикаже - сваком карактеру одговара једна меморијска локација. Адресе меморијских локација ЦГРОМ-а се поклапају са стандардним ASCII вриједностима карактера. То значи да, уколико се у програму који микроконтролер извршава захтјева исписивање неког карактера, на излазу ће се појавити бинарна вриједност карактера. Када се бинарни број учита у LCD дисплеј, приказаће се симбол који се налази на меморијској локацији која одговара бинарном броју у CGROMU-у. Ово важи за сва слова абецеде (велика и мала) али не и за бројеве.

**CGRAM меморија** - Поред тога што приказује све стандардне карактере, LCD дисплеј може да прикаже и ознаке које корисник сâм дефинише. Тиме је омогућено исписивање ћириличних фонтова као и других симбола који стају у оквир величине 5х8 пиксела. Све то омогућава мала RAM меморија (CGRAM) величине 64 бајта. Величина регистра у овој меморији је 8 бита али се користи само нижих 5. Логичка јединица у сваком регистру представља затамњен пиксел, док осам локација узетих

заједно предтављају један знак. Симболи се дефинишу на почетку програма а њихово приказивање се врши позивањем адресе (прва колона у CGROM мапи).

* **Команде, повезивање и иницијализација LCD дисплеја**

Сви подаци који се преносе на LCD дисплеј преко извода D0 – D7 биће тумачени као команде или подаци у зависности од логичког стања на пину RS - RS = 1 – Битови D0 – D7 су адресе карактера који треба да се прикажу на дисплеју. Уграђени процесор адресира уграђену мапу карактера и приказује одговарајући знак. Место појављивања је одређено DDRAM адресом. RS = 0 – Битови D0 – D7 су команде које одређују начин рада дисплеја. Списак свих команди које LCD дисплеј подржава су наведене у табели:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Команда | RS | R/W | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Обриши дисплеј | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Курсор на почетак | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Х |
| Унос карактера | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S |
| On/off | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | U | B |
| Помјерање | 0 | 0 | 0 | 0 | 0 | 1 | D/C | R/L | X | X |
| Начин рада | 0 | 0 | 0 | 0 | 1 | DL | N | F | X | X |
| CGRAM адреса | 0 | 0 | 0 | 1 | CGRAM адреса | | | | | |
| DDRAM адреса | 0 | 0 | 1 | DDRAM адреса | | | | | | |
| Читање BUSY fleg-а (BF) | 0 | 1 | BF | DDRAM адреса | | | | | | |
| Упис у CGRAM или DDRAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Читање из CGRAM /DDRAM-a | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

*Таб. 2.2. Команде за LCD дисплеј*

I/D 1 = Инкремент

0 = Декремент

S 1 = Укључен шифт регистра

0 = Искључен шифт регистра

D 1 = Укључен дисплеј

0 = Искључен дисплеј

U 1 = Видљива линија курсора

0 = Без линије курсора

B 1 = Курсор трепери

0 = Курсор не трепери

R/L 1 = Помjерање удесно

0 = Померање улијево

DL 1 = 8-битно повезивање

0 = 4-битно повезивање

N 1 = Испис у две линије

0 = Испис у једној линији

F 1 = Формат 5x10

0 = Формат 5x8

D/C 1 = Померање дисплеја

0 = Померање курсора

У 4-битном моду рада дисплеја за комуникацију се користе само 4 виша бита (D4 – D7) док се осталимогу оставити неповезани. У том случају, сваки податак се LCD-у шаље у два корака: прво се шаљу 4 виша бита а затим се шаљу 4 нижабита.

Одмах након довоñења напона напајања LCD дисплеј се аутоматски ресетује. Након тога дисплеј је спреман за рад и налази се у фабрички подешеном начину рада што значи:

1. Дисплеј је обрисан;

2. Начин рада:

DL = 1, комуникација се изводи 8-битном везом;

N = 0, поруке се исписују у једном реду;

F = 0, користе се фонтови формата 5x8 тачака;

3. Укључено/искључено:

D = 0, дисплеј је искључен;

U = 0, не види се линија курсора;

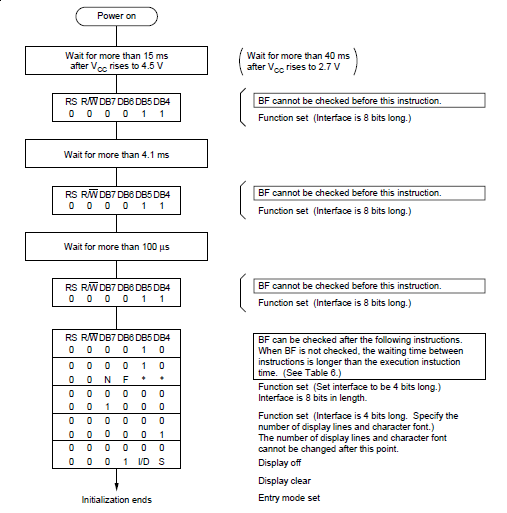
B = 0, искључено је блинкање курсора;

4. Унос карактера:

ID = 1, адресе на екрану се аутоматски увећавају за 1

S = 0 ,искључено померање (shift)

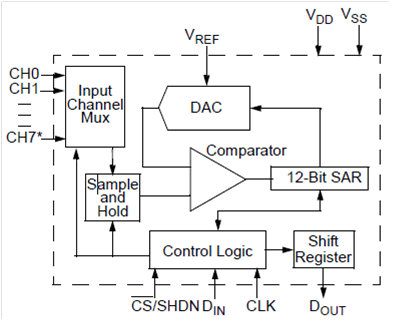
Алгоритам по коме се обавља иницијализација у случају 4-битног повезивања приказан је на слици:



*Сл. 2‑6 Алгоритам иницијализације LCD-а*

* **MCP3204 А/Д конвертор**

MCP3204 је 12-битни А/Д конвертор са сукцесивном апроксимацијом (*Сл.2.3. Блок шема MCP3204 А/Д конвертора*).

****

*Сл. 2‑7 Блок шема MCP3204 А/Д конвертора*

Приказани А/Д конвертор чине:

* Мултиплексер за улазне канале
* Sample and Hold коло
* Д/А конвертор
* Компаратор
* 12-битни SAR (Successive Aproximatio Register)
* Помјерачки регистар
* Контролна логика

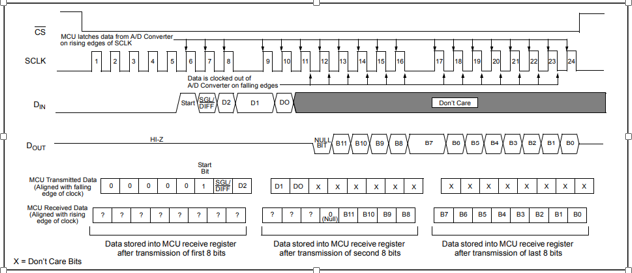
Основне карактеристике овог конвертора су:

* MCP3204 посједује четири канала који се могу користити као четири независна канала или као два диференцијална пара.
* Комуникација са А/Д конверртором остварује се кориштењем серијског интерфејса а са SPI протоколом.
* Референтни напон одређује аналогни улазни опсег. Вриједност бита најмање вриједности (LSB) директно је пропорционална са вриједношћу референтонг напона.
* Дигитални излазни код добија се као функција референтног напона и мјерене вриједности напона као:

Комуникација са MCP3204 А/Д конвертором се обавља помоћу стандардног SPI компатибилног серијског интерфејса. Иницијализација комуникације са било којим уређајем се врши довођењем логичке нуле на пин CS. Први примљени такт импулс, заједно са сигналом CS на нули и сигналом Din на јединици, генерисаће стартни бит за серијску комуникацију. Затим, иза стартног бита следи SGL/DIFF бит и њиме се одређује да ли ће се за конверзију користити један канал у независном или два канала у диференцијалном моду рада. Наредна три бита (D2, D1 и D0) одређују одговарајућу конфигурацију улазних канала при чему вредност бита D2није битна за овај А/Д конвертор. Са семпловањем аналогног сигнала почиње се на четвртој растућој ивици тактног сигнала након проласка стартног бита а завршетак семплавања је на петој опадајућој ивици такта. За потребе овог задатка користи се канал CH0 у независном моду рада (за детаље о раду овог А/Д конвертра консултујте [1])

Након проласка бита D0 потребно је да прође још један такт да би завршио sample and hold. На опадајућој ивици наредног такта на Dout излазу се јавља NULL бит. Наредних 12 битa на излазу Dout означавају резултат конверзије при чему се први јавља MSB бит (Most Significant Bit). Подаци се на излаз увijек шаљу на опадајућој ивици такта.

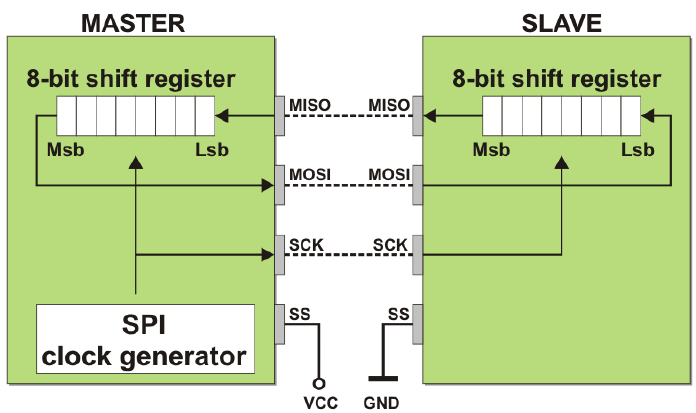
У комуникацији са микроконтролерима обично се шаљу и примају 3 бајта. Такође, потребно је да SPI интерфејс микроконтролера буде подешен тако да тактује податке на опадајућој ивици а да их лечује на растућој. На слици *Сл. 2.4*  приказана је комуникација између микроконтролера и А/Д конвертора у SPI моду 0,0 - режиму рада за који је потребно да сигнал SCLK буде на логичкој нули, те да се пребаци на логичку јединицу при старту комуникације.



*Сл. 2‑8 SPI комуникација кориштењем 8-битних сегмената (mode = 0,0 => SCLK на ниском логичком нивоу)*

* **SPI интерфејс**

SPI je систем који омогућава брзу синхрону везу измеñу контролера и једног или више периферијских уређаја који подржавају SPI. Код овакве везе увијек је један уређај главни (мастер). Он одређује брзину, смјер преноса података (да ли се они шаљу или примају) и формат података. На другом крају налази се “slave”, који је у подређеном положају, што значи да не може покренути размјену података и мора да се прилагођава мастер страни. На слици *Сл. 2.6*  приказан је SPI систем.



*Сл. 2‑9 SPI модул*

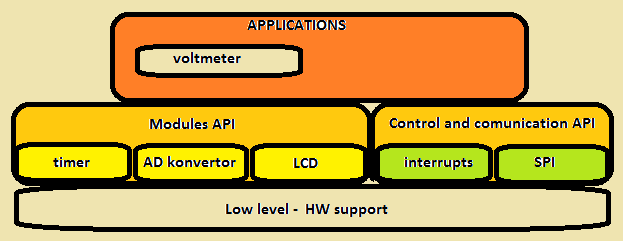
Подаци се преносе full-duplex везом помоћу 3 проводника који се повезују са изводима MISO, MOSI и SCK (P1.6 , P1.5 и P1.7 пинови микроконтролера у нашем случају). Четврти конторлни пин SS се на страни мастера не користи и може се бити искоришћен као улаз/излаз док на страни слејва мора бити на ниском логичком нивоу. Ако се на SS извод слејва доведе сигнал логичке јединице његов SPI систем се деактивира и MOSI пин може да се искористи као улаз за неку другу намjену.

## Софтверсkо рјешење

Када смо задовољили системске захтјеве и на основу њих испројектовали хардверски дио(с обзиром да наш систем садржи микроконтролер) на ред долази да се обезбједи софтверска подршка.

Вратимо се опет на корисничке захтјеве и погледајмо како се они односе на софтвер који треба развити. Речено је да мјерени резултат треба да се исписује на дисплеју, да скала може ручно да се мијења, да се садржај дисплеја освјежава сваке секунде. На основу овога можемо закључити да требамо обезбједити иницијализацију, управљање свим модулима, те комуникацију између одоварајућих модула. Потребно је осмислити софтверску архитектуру – тj. софтверске слојеве, модуле, односе и комуникацију између њих, па на крају и функције које нам требају да би све функционисало.

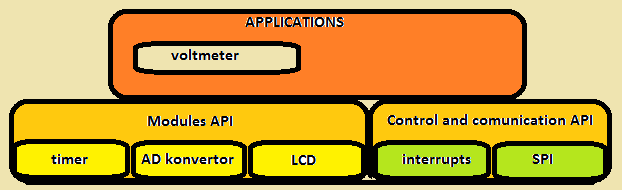
С обзиром да се за реализацију волтметра користе модули који се могу користити и за реализацију много других апликација, циљ је да се обезбједи добра софтверска подршка за рад са било којим од модула у виду библиотека, гдје би се при развоју било које нове апликације те библиотеке лако користиле, а развијале само функције које су специфичне баш за ту апликацију. Тако је првобитно планирано да се софтвер развије у три слоја (*Сл. Првобитно предложжена софтверска архитектура*).



*Сл. 2‑10 Првобитно предложжена софтверска архитектура*

Међутим, и ако гледано са стране даљег развоја и надоградње оваква организација софтвера изгледа веома погодна, кроз фазу тестирања показало се као непрактично рјешење за изабрану хардверску архитектуру. Наиме, добре стране овакве организације су те да ако додамо нови дисплеј, тајмер или А/Д конвертор довољно би било да обезбједимо основне функције за рад са њима (чији је прототип већ установљен) те би се оне само позивале у Модул АПИ функцијама (које већ постоје). Тако би врло лако било замјенити било који од модула које се користи за нашу апликацију и сама замјена не би уопште утицала на саме апликације или на апликативни слој за рад са модулима, као ни на комуникационе библиотеке. Развој нових функција и на апликативном и на хардверском нивоу било би неопходно тек када би се додавао потпуно нови модул у систем, док би за сваку промјену модула или оптимизацију постојећих функција за рад са њим, само функције HW support слоја требало мијењати. У овом случају за развој апликација имали бисмо стандардизоване Modul API функције на које смо навикли, те би развој апликација уз кориштење истих био знатно олакшан.

Међутим, показало се да је при раду са одабраним микроконтролером непрактично развијати софтвер који има три нивоа. При раду са датим контролером имамо могућност избора три различита меморијска модела: small, compact и large. Small модел је модел који све промјенљиве, као и све остале податке, аутоматски смјешта у интеру меморију којој се има директан приступ (користи се директно адресирање). Овај модел и јесте најбржи модел који користи интерну меморију микроконтролера и то је модел који желимо користити (како због брзине, тако и због тога што екстерну меморију нисмо предвидјели при хардверском планирању ). Критична тачка овдје јесте позив функције из функције гдје се RAM меморија брзо „троши“, па да би заобишли проблеме који се ту јављају морали смо избацити слој за хардверску подршку и у склопу Module API функција реализовати и дио који пружа хардверску подршку (*Сл.2.10.* *Коначна софтверска архитектура*)



*Сл. 2‑11 Коначна софтверска архитектура*

Базираћемо се само на развијену апликацију, тј. волтметар. Волтметар треба да може да измјери и прикаже измјерену вриједност напона, што му је примарна функција. Поред најосновнинјих функционалности, у циљу да уређај буде једноставан за кориштење и да пружи кориснику могућности које нису директно захтијеване, али су примамљиве за корисника (и утицаће да поред конкурентних производа на тржишту изабере баш наш производ), потребно је омогућити још неке функционалности. Примјер таквих функција су нпр. промјена скале у којој се мјерена вриједност приказује, сигнална лампица која означава да је волтметар укључен и спреман за рад итд.

Тако би APPLICATIONS слој садржао фајлове са декларацијама и дефиницијама функција везаних за нашу апликацију, те фајл у коме је main програм који позива наведене функције. Добра пракса јесте да се у главном програму апликације користе само функције дефинисане на APPLICATIONS нивоу.

Главни програм за волтметар апликацију имао би следећи изглед:

int main()

{

/\* mesuredVal is variable used to accept mesured val sent from ADC \*/

float idata mesuredVal;

static unsigned long temp;

static unsigned long count;

/\* Initialise all modules used for voltmeter \*/

initVoltmeter();

while(1)

{

/\* do the mesuring \*/

/\* if REFRESH\_TIME is passed since last display refres - refresh the disp\*/

mesuredVal = mesProccess(&scale);

temp = milis();

if (1000 <= (temp - count))

{

count = temp;

dispVoltage(mesuredVal, scale);

}

}

return 0;

}

Видимо да је прво потребно извршити иницијализацију волтметра, а затим се у while обезбјеђује да се сво вријеме за које је инструмент укључен врши мјерење напона и освјежавање приказа на дисплеју сваке секунде.

Функција *mesProccess(&scale)* омогућавада се на основу измјерене вриједности аутоматски одреди и скала у којој се измјерена вриједност напона приказује. Као што је речено, скала у којој се приказује мјерење може и ручно да се бира притиском једног од два тастера која су планирана пројектом. Притиском на тастер дешава се интерапт којим се увећава или смањује вриједност глобалне промјенљиве scale. Одабрано је да се притиском на тастер покреће прекидна рутина из разлога што је промјена скале нешто што корисник захтјева при кориштењу инструмента и при таквом захтјеву требало би да одзив инструмента буде брз.

Функције специфичне за волтметар убачене су на ниво апликације кроз фајлове *voltmeter.h* и *voltmeter.c.* Функције специфичне за кориштене модуле, комуникацију или прекидне рутине дефинисане су и имплементиране кроз Modue API слој. Све развијене функције су при самој декларацији у коду детаљно описане, те неће бити детаљније објашњаване у овом раду.

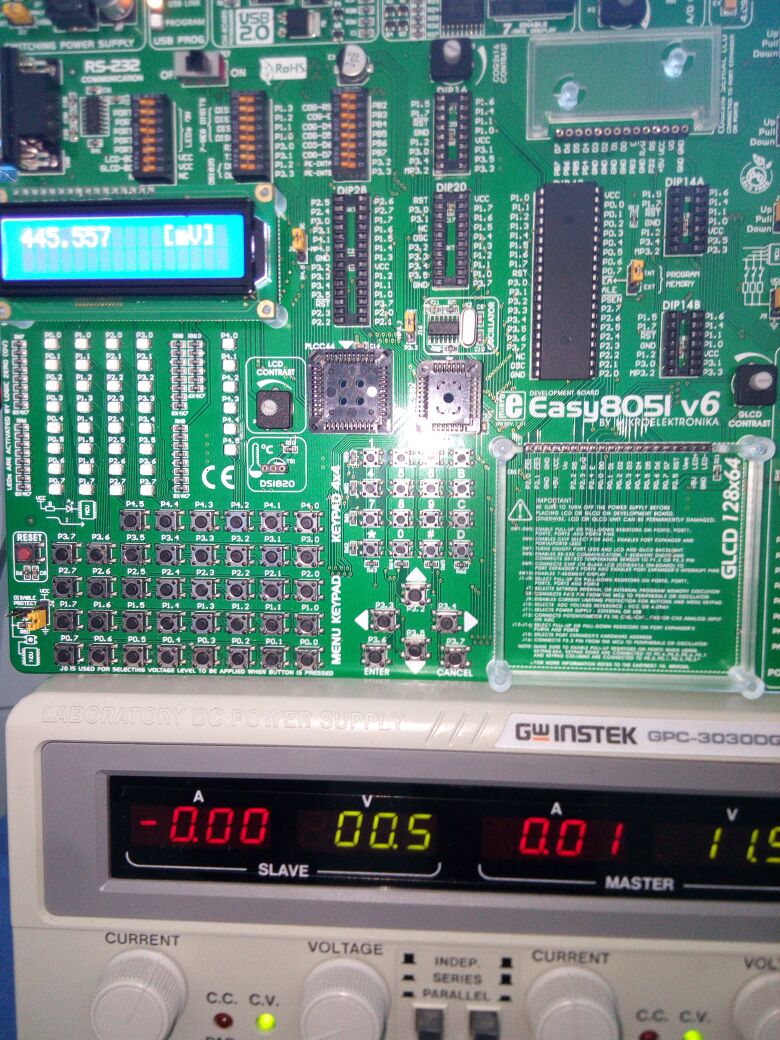
# ТЕСТИРАЊЕ ПРОИЗВОДА

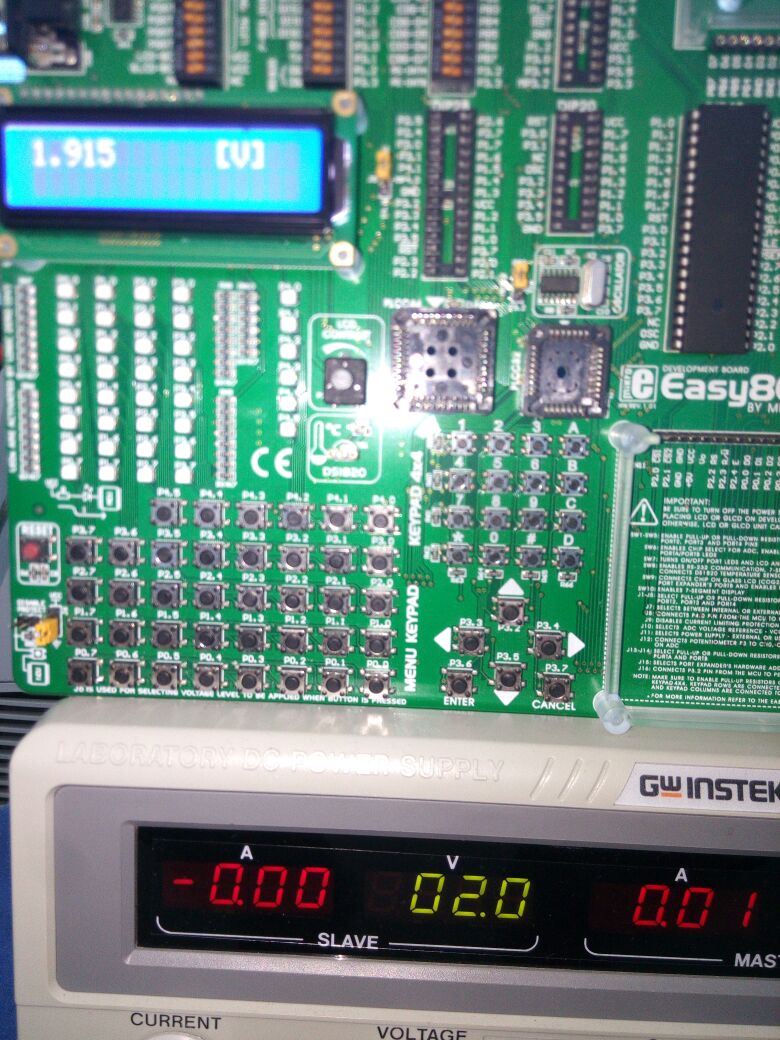
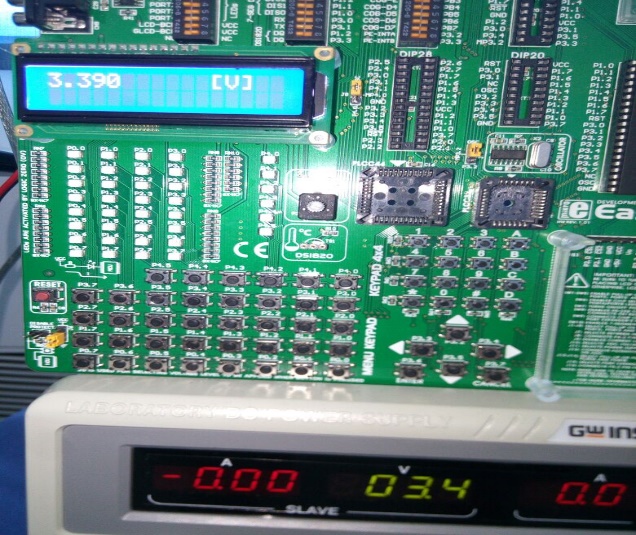
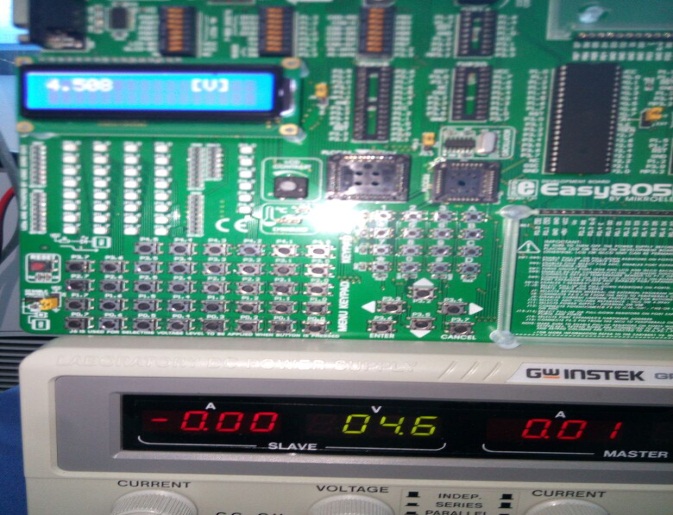
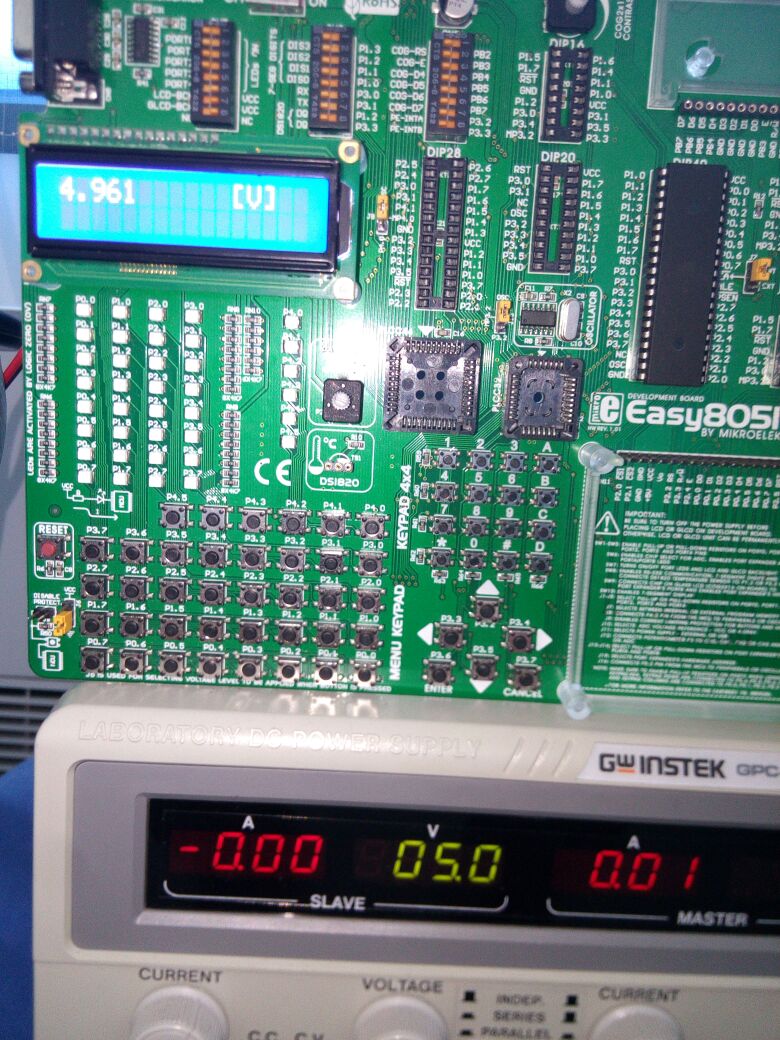
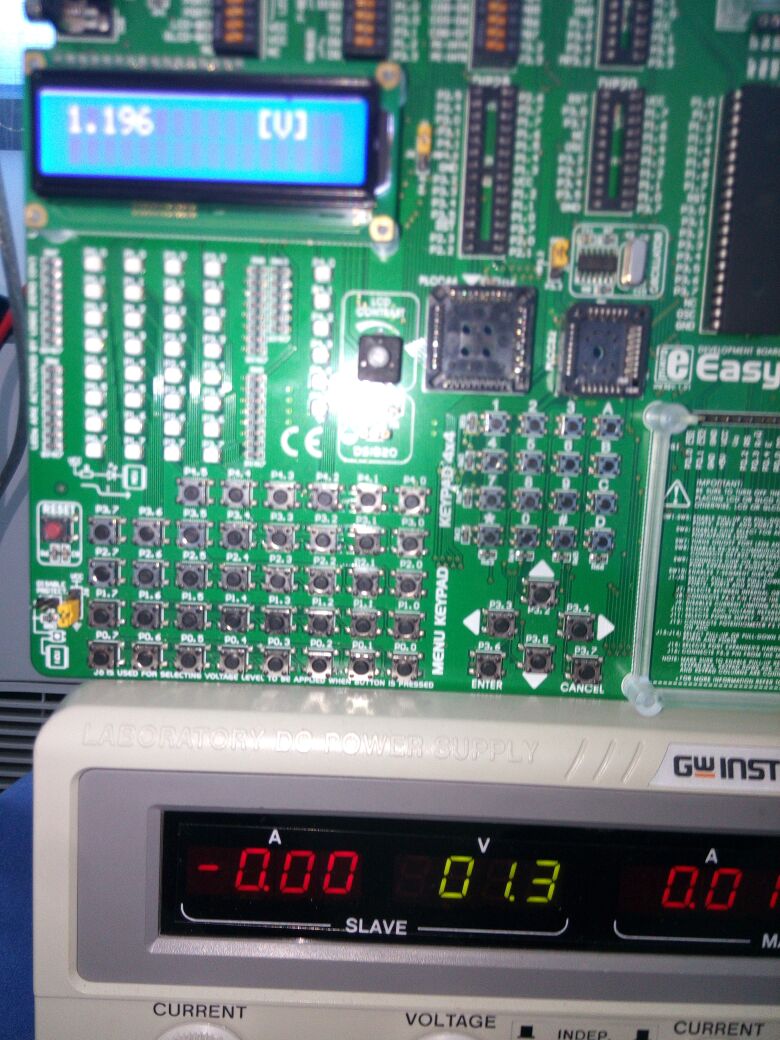
Тестирајући производ добили смо следеће резултате:

|  |  |  |  |
| --- | --- | --- | --- |
| Мјерена вриједност напона | Измјерена вриједност напона | Апсолутна грешка мјерења ΔV [V] | Релативна грешка мјерења |
| 500.0 [mV] | 445.557 [mV] | 0.05443 | 0.1088 |
| 1.3 [V] | 1.196 [V] | 0.10400 | 0.0800 |
| 2.0 [V] | 1.915 [V] | 0.08500 | 0.0425 |
| 3.4 [V] | 3.390 [V] | 0.00100 | 0.0029 |
| 4.6 [V] | 4.500 [V] | 0.10000 | 0.0217 |
| 5.0 [V] | 4.961 [V] | 0.03900 | 0.0078 |

*Таб. 3.1. Резултати добијени при тестирању волтметра*

*Сл. 3‑1 Мјерен напон од 500 [mV]*



**

*Сл. 3‑2 Мјерен напон од 5.0 [V]*

*Сл. 3‑3 Мјерен напон од 4.6 [V]*

*Сл. 3‑4 Мјерен напон од 3.4 [V]*

*Сл. 3‑5 Мјерен напон од 2 [V]*

*Сл. 3‑6 Мјерен напон од 1.3 [V]*

# ИЗВОРНИ КОД

# Аликација

## *voltmeter\_main.c*

#include "voltmeter.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define REFRESH\_TIME (1000) /\* 1000 [ms] \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* extern varivables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern volatile char scale;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* global varivables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

//mesuredVal is variable used to accept mesured val sent from ADC

float idata mesuredVal;

static unsigned long temp;

static unsigned long idata count;

//Initialise all modules used for voltmeter

initVoltmeter();

while(1)

{

//do the mesuring

temp = milis();

//if REFRESH\_TIME is passed since last display refres - refresh the disp

if (REFRESH\_TIME <= (temp - count))

{

mesuredVal = mesProccess(&scale);

count = temp;

dispVoltage(mesuredVal, scale);

}

}

return 0;

}

## *voltmeter.h*

#ifndef \_VOLTMETER\_H\_

#define \_VOLTMETER\_H\_

#include "../../support/API/include/timer.h"

#include "../../support/API/include/interrupts.h"

#include "../../support/API/include/display.h"

#include "../../support/API/include/ADConv.h"

#include "../../support/API/include/spi.h"

/\*\*

\* @brief

\* This function initialises all modules used by Voltmeter. Those

\* modules are: -Display

\* -A/D converter

\* -SPI

\* -Timer

\* -Interrupt

\* -Set Interrupt handlers

\* It should be called before any other API function.

\*

\* @param [IN] - No input params

\*

\* @ret [No return value]

\*/

void idata initVoltmeter(void);

/\*\*

\* @brief

\* The function is used to for messurment process. It gives back messured value

\* and sets automaticly right scale (where the parameter of the function is

\* a pointer to the scale variable)

\*

\* @param [IN] - \*scale - pointer to the scale variable

\*

\* @ret [OUT] - returns value of samples (whole scale take 4096 samples if

\* MCP3204 is used). This is not measured value, but just

\* reference value-where max value is 4096. Real mesured voltage

\* is calculated using this value and Reference Voltage value

\* (reference volltage value on A/D conv)

\*/

float mesProccess(char\* scale);

/\*\*

\* @brief

\* Display float value (measured voltage here), on Display, in scale provided.

\* this function takes num of samples(that are returned by mesProccess function)

\* and displays real measured voltage (measured voltage is function of sample

\* number and reference voltage on A/D convertor)

\*

\* @param [IN] - float mesuredVal - relative voltage level

\* @param [IN] - char scale - scale in which voltage is displayed

\*

\* @ret [No return value]

\*/

void dispVoltage(float mesuredVal, char scale);

/\*\*

\* @brief

\* This is external\_interrupt\_0 handler for Voltmetar app.

\* It is used to be registerd to extern interrupt in voltmetar Init function.

\* This function is used for manual scale changeing

\*

\* @param [IN] - No input parameters

\*

\* @ret [No return value]

\*/

void incrScaleIntrHandler(void);

/\*\*

\* @brief

\* This is external\_interrupt\_0 handler for Voltmetar app.

\* It is used to be registerd to extern interrupt in voltmetar Init function.

\* This function is used for manual scale changeing

\*

\* @param [IN] - No input parameters

\*

\* @ret [No return value]

\*/

void decrScaleIntrHandler(void);

#endif

## *voltmeter.c*

#include "voltmeter.h"

volatile char scale;

char\* idata currScale[] ={"[mV]", "[V]", "[kV]"};

bit signalLed at P0\_0\_bit;

static int data manualChange;

static void helloMsg()

{

char idata txt1[] = "Easy 8051";

char idata txt2[] = "VOLTMETAR";

char idata i;

LcdCmd(\_LCD\_CLEAR);

LcdPutString(1, 16, txt1);

LcdPutString(2, 18, txt2);

LcdCmd(\_LCD\_CURSOR\_OFF);

for(i=0; i<15; i++)

{

LcdCmd(\_LCD\_SHIFT\_LEFT);

Delay\_ms(10);

}

Delay\_ms(1000);

LcdCmd(\_LCD\_CLEAR);

}

void idata initVoltmeter()

{

char IEval;

char masterMode0;

manualChange = 0;

signalLed = 1;

IEval = \_EN\_INTR | //enable interrupts

\_EN\_EX\_1\_INTR | //enable extern 1 interrupt

\_EN\_TC0\_INTR | //enable TC0interrupts

\_EN\_EX\_0\_INTR;

masterMode0 = SPI\_INTR\_EN |

SPI\_EN |

SPI\_MSB |

SPI\_MASTER |

SPI\_SCK\_IDLE\_L |

SPI\_SCK\_PHASE\_L |

SPI\_CLK\_RATE\_1;

initSPI(masterMode0);

MCP3204\_Init(0,0);

initLcd();

intrEn(IEval);

initTC();

registerIntrEx0(incrScaleIntrHandler);

registerIntrEx1(decrScaleIntrHandler);

helloMsg();

signalLed = 0;

}

void dispVoltage(float mesuredVal, char scale)

{

char mesuredVal2txt[5];

float temp;

temp = mesuredVal;

switch (scale)

{

case 0:

temp = mesuredVal\*1000.0;

break;

case 1:

temp = mesuredVal;

break;

case 2:

temp = mesuredVal \* 0.001;

break;

default:

break;

}

sprintf(mesuredVal2txt, "%.3f", temp);

LcdCmd(\_LCD\_CLEAR);

LcdPutString(1, 1, mesuredVal2txt);

LcdPutString(1, 12, currScale[scale]);

LcdCmd(\_LCD\_CURSOR\_OFF);

}

float mesProccess(char\* scale)

{

float sample = 0.0;

MCP3204\_SetChannel(0);

sample = MCP3204\_GetSample();

if (0 == manualChange)

{

if (sample < 1.0)

{

\*scale = 0;

}

else if (sample >= 1.0 && sample < 1000.0)

{

\*scale = 1;

}

else

{

\*scale = 2;

}

}

return sample;

}

void incrScaleIntrHandler(void)

{

scale += (scale < 2)?1:0;

manualChange = 1;

}

void decrScaleIntrHandler(void)

{

scale -=(scale > 0)?1:0;

manualChange = 1;

}

# Module API слој

## *display.h*

#ifndef \_DISPLAY\_H\_

#define \_DISPLAY\_H\_

/\*\*

\* @brief

\* This function initialises the display.

\*

\* @param [IN] - void - No input params

\*

\* @ret [No return value]

\*/

void idata initLcd(void);

/\*\*

\* @brief

\* This function is used to set cursor possition on the display

\*

\* @param [IN] - char row -row coordinate of display where txt is printed

\* @param [IN] - char column -colmun coordinate of display where txt is printed

\*

\* @ret [No return value]

\*/

void LcdSetCursor(char row, char colmun);

/\*\*

\* @brief

\* This function prints character, given as input, on the current cursor

\* possition on the display

\*

\* @param [IN] - char chr -Value that will be displayed

\*

\* @ret [No return value]

\*/

void LcdPutChar(char chr);

/\*\*

\* @brief

\* This function takes prints string, given as input, on the display

\*

\* @param [IN] - char row -row coordinate of display where txt is printed

\* @param [IN] - char column -colmun coordinate of display where txt is printed

\* @param [IN] - char\* txt -Value that will be displayed

\*

\* @ret [No return value]

\*/

void LcdPutString(int row, int column, char\* txt);

/\*\*

\* @brief

\* This function takes commands offten needed when using display. Commands

\* available are : \_LCD\_FIRST\_ROW

\* \_LCD\_SECOND\_ROW

\* \_LCD\_CLEAR

\* \_LCD\_RETURN\_HOME

\* \_LCD\_CURSOR\_OFF

\* \_LCD\_UNDERLINE\_ON

\* \_LCD\_BLINK\_CURSOR\_ON

\* \_LCD\_MOVE\_CURSOR\_LEFT

\* \_LCD\_MOVE\_CURSOR\_RIGHT

\* \_LCD\_TURN\_ON

\* \_LCD\_TURN\_OFF

\* \_LCD\_SHIFT\_LEFT

\* \_LCD\_SHIFT\_RIGHT

\*

\* @param [IN] - char txtCmd -symbolic command

\*

\* @ret [No return value]

\*/

void LcdCmd(char txtCmd);

#endif

## *display.c*

#include "../include/display.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LCD Functions for 4-bit mode

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lcd module connections

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Regisrer\_select pin

// for commands to LCD - set it to be low

// to display data (char) it shuld be set to high voltage level

sbit LCD\_RS at P2\_0\_bit;

// Enable pin

// When writeing data to the LCD it will read the data on the falling

// edge of this pin

sbit LCD\_EN at P2\_1\_bit;

// Data pins

sbit LCD\_D7 at P2\_5\_bit;

sbit LCD\_D6 at P2\_4\_bit;

sbit LCD\_D5 at P2\_3\_bit;

sbit LCD\_D4 at P2\_2\_bit;

void LcdCmd(char cmd)

{

Lcd\_Cmd(cmd);

}

void LcdSetCursor(char row, char colmun)

{

char temp;

temp = (1 == row)?(\_LCD\_FIRST\_ROW + colmun):(\_LCD\_SECOND\_ROW + colmun);

LcdCmd(temp);

}

void initLcd()

{

LCD\_Init();

}

void LcdPutString(int row, int colmun, char \*txt)

{

LCD\_Out(row,colmun, txt);

}

## *ADConv.h*

#ifndef \_ADCONV\_H\_

#define \_ADCONV\_H\_

#define REF\_VOLTAGE (5.0)

#define DELTA\_STEPS (4096.0)

#define DELTA\_VAL (REF\_VOLTAGE / DELTA\_STEPS)

/\*\*

\* @brief

\* Function used to initialize the ADC module

\*

\* @param [IN] - unsigned char mode -mode: 0 - differential,

\* 1 - single

\* @param [IN] - cunsigned char channel -channel: 0-3

\*

\* @ret [No return value]

\*/

void MCP3204\_Init(unsigned char mode, unsigned char channel);

/\*\*

\* @brief

\* Function used to set the mode of the ADC module

\*

\* @param [IN] - unsigned char mode -mode: 0 - differential,

\* 1 - single

\*

\* @ret [No return value]

\*/

void MCP3204\_SetMode(unsigned char mode);

/\*\*

\* @brief

\* Function used to set the channel of the ADC module

\*

\* @param [IN] - cunsigned char channel -channel: 0-3

\*

\* @ret [No return value]

\*/

void MCP3204\_SetChannel(unsigned char channel);

/\*\*

\* @brief

\* Function used to get sample from ADC module.

\* Returns voltage value sampled

\*

\* @param [IN] - void - no input parameters

\*

\* @ret [OUT] - float - voltage value sampled

\*/

float MCP3204\_GetSample(void);

#endif

## *ADConv.c*

#include "../include/ADConv.h"

#include "../include/spi.h"

sbit MCP3204\_CS at P3\_5\_bit;

int cmd;

void MCP3204\_Init(unsigned char mode, unsigned char channel)

{

MCP3204\_CS = 1;

cmd = (mode << 9) | (channel << 6) | 0x0400;

}

void MCP3204\_SetMode(unsigned char mode)

{

cmd &= 0xfdff;

cmd |= mode << 9;

}

void MCP3204\_SetChannel(unsigned char channel)

{

cmd &= 0xfe3f;

cmd |= channel << 6;

}

float MCP3204\_GetSample(void)

{

int readVal;

float retVal;

char read\_h;

char read\_l;

MCP3204\_CS = 1;

MCP3204\_CS = 0;

//variable given as argument is command sent to AD convertor

//value recived rom convertor is always return value from SPI\_BTransfer func

//configuration byte is sent for each transfer

SPI\_ByteTransfer(cmd >> 8);

//AD convertor gives back 12 bit information in two steps.

//First step - 4 higher bits of sample value - 0f=0000 1111

read\_h = SPI\_ByteTransfer(cmd) & 0x0f;

//2nd step - 8 lower bits of information are recived, no command is sent to

read\_l = SPI\_ByteTransfer(0);

MCP3204\_CS = 1;

readVal = (read\_h << 8) | read\_l;

MCP3204\_CS = 0;

retVal = readVal \* DELTA\_VAL;

return retVal;

}

## *spi.h*

#ifndef \_SPI\_H\_

#define \_SPI\_H\_

#define SPI\_INTR\_EN (0X80)

#define SPI\_EN (0X40)

#define SPI\_LSB (0X20)

#define SPI\_MSB (0X00)

#define SPI\_MASTER (0X10)

#define SPI\_SLAVE (0X00)

#define SPI\_SCK\_IDLE\_H (0X08)

#define SPI\_SCK\_IDLE\_L (0X00)

#define SPI\_SCK\_PHASE\_H (0X08)

#define SPI\_SCK\_PHASE\_L (0X00)

#define SPI\_CLK\_RATE\_0 (0X00)

#define SPI\_CLK\_RATE\_1 (0X01)

#define SPI\_CLK\_RATE\_2 (0X02)

#define SPI\_CLK\_RATE\_3 (0X03)

#define MASTER\_MODE\_0 (SPI\_INTR\_EN |SPI\_EN | SPI\_MSB | SPI\_MASTER |\

SPI\_SCK\_IDLE\_L | SPI\_SCK\_PHASE\_L | SPI\_CLK\_RATE\_1)

/\*\*

\* @brief

\* SPI init function - used to set SPI transfer parameters as SPI mode (master /

\* slave), LSB/MSB, phase, idle state of clk signal ...

\*

\* @param [IN] - char spcrVal - value to write in SPCR register - use defines

\* and logiccal "or" to get proper values

\*

\* @ret [No return value]

\*/

void idata initSPI(char spcrVal);

/\*\*

\* @brief

\* This function is used to send or recive data via SPI interface. Data for

\* sending are passed as argument to the function and data to recive are given as

\* return value from function

\*

\* @param [IN] - char \_data - Data for sending

\*

\* @ret [out] - char - Recived data

\*/

unsigned char SPI\_ByteTransfer(char \_data);

#endif

## *spi.c*

#include "../include/spi.h"

void idata initSPI(char spcrVal)

{

SPCR = spcrVal;

}

char SPI\_ByteTransfer(char \_data)

{

SPDR = \_data;

while(SPIF\_bit == 0);

return SPDR;

}

## *timer.h*

#ifndef \_TIMER\_H\_

#define \_TIMER\_H\_

#include "interrupts.h"

//TMOD register

#define ENABLE\_TC1 (0x80)

#define ENABLE\_TC0 (0x08)

// timer modes

#define T1\_MODE\_0 (0x00)

#define T1\_MODE\_1 (0x10)

#define T1\_AUTO\_RELOAD (0x20)

#define T1\_MODE\_3 (0x30)

#define T0\_MODE\_0 (0x00)

#define T0\_MODE\_1 (0x01)

#define T0\_AUTO\_RELOAD (0x02)

#define T0\_MODE\_3 (0x03)

//TCON register

#define ENABLE\_T1 (0x40)

#define ENABLE\_T0 (0x10)

#define C1\_INTR\_ON\_FALLING\_EDGE (0x04)

#define C0\_INTR\_ON\_FALLING\_EDGE (0x01)

#define TIMER\_100u (0x88)

/\*\*

\* @brief

\* The function is used to init timer/counter

\* Currently it is written to set Timer to work in auto reload mode

\* this function should be repaired for further usage

\*

\* @param [No input params]

\*

\* @ret [No valureturne]

\*/

void initTC(void);

/\*\*

\* @brief

\* The function is used to set timer mode if the module is used as timer. It

\* writes coresponding values in TMOD register. Values that should be used are

\* defined in this file

\*

\* @param [IN] - char TCmode - timer mode

\*

\* @ret [No return value]

\*/

void setTCMode(char TCmode);

/\*\*

\* @brief

\* The function returns value of global static variable that is updated after

\* each timer interrupt

\*

\* @param [No input params]

\*

\* @ret [OUT] - unsigned long - value updated after each timer interrupt

\*/

unsigned long milis(void);

/\*\*

\* @brief

\* The function updates TCON register

\*

\* @param [IN] - char TCONreg - valu to write in TCON register

\*

\* @ret [No return value]

\*/

void TCCntrl(char TCONreg);

/\*\*

\* @brief

\* The function sets timer/counter values - useful in autoreload mode

\*

\* @param [IN] - char timer - timer or 1

\* @param [IN] - char timerVal - value to write in timer

\*

\* @ret [No return value]

\*/

void setTimerVal(char timer,char timerVal);

/\*\*

\* @brief

\* The interrupt handler functions that should be registerd to timer interrupts

\*

\* @param [No input params]

\*

\* @ret [No valureturne]

\*/

void T0IntrHandler(void);

void T1IntrHandler(void);

#endif

## *timer.c*

#include "../include/timer.h"

#define PRESCALER (8)

static long tcount =0;

unsigned long milis()

{

return tcount;

}

void initTC()

{

char idata TCmode;

char idata TCONreg;

TCmode = ENABLE\_TC0 |

T0\_AUTO\_RELOAD;

TCONreg = ENABLE\_T0 |

C1\_INTR\_ON\_FALLING\_EDGE |

C0\_INTR\_ON\_FALLING\_EDGE;

// Set initial count value

tcount = 0;

// Set timer 0 (gate disable, interval timer, auto-reload mode)

setTCMode(TCmode);

// Set auto-reload value to get 100us base tick

TH0 = TIMER\_100u;

//

TCCntrl(TCONreg);

// Enable interrupts (sets EA and ET0)

//register timer/counter 0 intr handler function

registerIntrTC0(T0IntrHandler);

}

void setTCMode(char TCmode)

{

TMOD = 0;

TMOD = TCmode;

}

void TCCntrl(char TCONreg)

{

TCON = TCONreg;

}

void setTimerVal(char timer,char timerVal)

{

if(0 == timer)

{

TH0 = timer;

}

else if(1 == timer)

{

TH1 = timer;

}

}

void T0IntrHandler(void)

{

static unsigned int ticks = 0;

//increases tcount after 1[ms]

if (ticks++ >= PRESCALER)

{

tcount++;

ticks = 0;

}

}

void T1IntrHandler()

{

//write this function when needed

}

## *interrupts.h*

#ifndef \_INTERRUPTS\_H\_

#define \_INTERRUPTS\_H\_

// IE register defines

#define \_CLEAR\_INTR\_REG (0X00)

#define \_EN\_INTR (0X80)

#define \_EN\_SERIAL\_INTR (0X10)

#define \_EN\_TC1\_INTR (0X08)

#define \_EN\_EX\_1\_INTR (0X04)

#define \_EN\_TC0\_INTR (0X02)

#define \_EN\_EX\_0\_INTR (0X01)

typedef void (\*InterruptFxn)(void);

/\*\*

\* @brief

\* The function is used to enable interrupts globali. It sets value of IE reg

\*

\* @param [IN] - char IEval -value to be written into IE reg

\*

\* @ret [No return value]

\*/

void idata intrEn(char IEval);

/\*\*

\* @brief

\* Interrupt register functions - used to register applications specific

\* functions to interrupt routines available

\*

\* @param [IN] - InterruptFxn arg - function to be called when interrupt happends

\*

\* @ret [No return value]

\*/

void idata registerIntrEx0(InterruptFxn arg);

void idata registerIntrEx1(InterruptFxn arg);

void idata registerIntrTC0(InterruptFxn arg);

void idata registerIntrTC1(InterruptFxn arg);

#endif

## *interrupts.c*

#include "../include/interrupts.h"

static InterruptFxn pt2Handler[4] = {(void\*)0, (void\*)0, (void\*)0, (void\*)0};

void idata registerIntrEx0(InterruptFxn arg)

{

pt2Handler[0] = arg;

}

void idata registerIntrEx1(InterruptFxn arg)

{

pt2Handler[1] = arg;

}

void idata registerIntrTC0(InterruptFxn arg)

{

pt2Handler[2] = arg;

}

void idata registerIntrTC1(InterruptFxn arg)

{

pt2Handler[3] = arg;

}

void idata intrEn(char IEval)

{

IE = IEval;

//set priorities

IP = 0x0a;

}

void extIntr0() iv IVT\_ADDR\_EX0 ilevel 0 ics ICS\_AUTO

{

char regVal;

regVal = IE;

IE = regVal & 0b01111111;

pt2Handler[0]();

IE = regVal | 0b10000000;

}

void extIntr1() iv IVT\_ADDR\_EX1 ilevel 0 ics ICS\_AUTO

{

char regVal;

regVal = IE;

IE = regVal & 0b01111111;

pt2Handler[1]();

IE = regVal | 0b10000000;

}

void tc0() iv IVT\_ADDR\_ET0 ilevel 0 ics ICS\_AUTO

{

char regVal;

regVal = IE;

IE = regVal & 0b01111111;

pt2Handler[2]();

IE = regVal | 0b10000000;

}

void tc1() iv IVT\_ADDR\_ET1 ilevel 0 ics ICS\_AUTO

{

char regVal;

regVal = IE;

IE = 0;

pt2Handler[3]();

IE = regVal;

}

# ЛИТЕРАТУРА

[1] MCP3204 datasheet: “ <http://pdf1.alldatasheet.com/datasheet-pdf/view/74936/MICROCHIP/MCP3204.html>”

[2] display datasheet: “<http://pdf1.alldatasheet.com/datasheet-pdf/view/63663/HITACHI/HD44780U.html>”

[3] ucontroler datasheet: “<http://pdf1.alldatasheet.com/datasheet-pdf/view/162929/ATMEL/AT89S8253.html>”

[4] Easy8051 datasheet: “<https://download.mikroe.com/documents/full-featured-boards/easy/easy8051-v6/easy8051-v6-manual-v100.pdf>”

[5] interfaceing spi: “<http://extremeelectronics.co.in/avr-tutorials/interfacing-12-bit-spi-adc-mcp3204-with-avr-micro/>”

[6] Угљеша Јовановић, Марко Стојановић: “Дигитални волтметар реализован помоћу микроконтролера АТ89Ѕ253“

[7] Микропроцесорски системи Скрипта