МАТЕМАТИЧКА ГИМНАЗИЈА

У БЕОГРАДУ

МАТУРСКИ РАД

из предмета

Програмирање и програмски језици

на тему

Програмирање микроконтролера

Ученик

Коста Бизетић, IV$_д$

Ментор

Јелена Хаџи-Пурић

Београд, јун, 2016.

Садржај

1 Увод

Употреба рачунара у животу данашњег човека веома је велика. Реч рачунар људи најчешће асоцирају са персоналним рачунарима или лаптоповима, а ретко помисле на микроконтролере. Ово јесте логично када узмемо у обзир чињеницу да су управо персонални рачунари они са којима је човек најчешће у контакту, али може бити и нелогично с обзиром да су у свету миктоконтролери много више заступљени.

Велики део индустрије заснива се на аутоматизованој производњи која ја направила велики корак у развоју човечанства. Иза сваке врсте аутоматизације стоји неки облик рачунара и веома често је то микроконтролер.

Не морамо чак ни разматрати глобалани план. Просечни дом има два до три рачунара, али и 20 до 30 микроконтролера. Многи нису свесни шта је то што контролише фрижидере, телевизоре, микроталасне, даљинске управљаче и све остале мало „паметније“ кућне уређаје.

Из свих наведених разлога сматрам да је основно познавање рада, како миркоконтролера тако и рачунара генерално, веома корисно за боље разумевање света у коме живимо. Мислим да ће једног дана ако није већ данас то постати једно од основних делова информатичке писмености.

Главни циљ овог рада је да објасни основне концепте програмирања микроконтролера као што су регистри, улазно-излатни пинови, интерапт итд. као и да читаоцу створи слику о могућностима примене оваквих рачунара које су застрашујуће велике.

2 Микропроцесор и микроконтролер

Микроконтролер је минијатурни рачунар смештен на јединственом чипу који садржи процесорско језгро, меморију и подесиве улазно-излазне пинове за комуникацију са спољашњим светом.

И поред тога што се знатно разликују, најбољи начин да се објасни шта је тачно микроконтролер је да се упореди са микропроцесором.

Микропроцесор је централни део рачунара у коме се великом брзином извршавају инструкције. Дизајниран је да буде универзалан и да може да покреће разноврсне програме и тиме задовољи све потребе корисника. Зато се најчешће користи у серверима и персоналним рачунарима. Иако је микропроцесор веома снажна рачунарска компонентна да би га користили неопходно му је обезбедити РАМ меморију, осцилатор, чипсет, разне улазно-излазне контролере преко којих он комуницира са спољашњим светом.

Са друге стране, микроконтролер је уређај који садржи све што му је потребно како би самостално радио. Најбоље је замислити га као целокупан рачунар смештен на један мали чип површине око једног квадратног центиметра. Овај уређај није ни изблиза толико моћан као персонални рачунар. Он нема никакву графику, знатно је спорији и садржи много мање меморије. Његови главни адути су величина и веома ниска цена. Кад ово комбинујемо са чињеницом да је микроконтролер врло прилагодљив захваљујући великом броју пинова коју по потреби могу бити улазни или излазни врло је лако схватити зашто се они користе у многим електричним уређајима.

Микроконтролери се користе као контролни центри у уграђеним системима односно системима посебне намене. Дизајнирани су да након што се угради заувек извршавају један програм и да врше тачно одређену функцију у неком уређају. Нпр. регулација температуре у фрижидеру. Овај посао не захтева брз рачунар већ неки који можемо да сместимо у фрижидер. Довољно је да сваке секунде очитава температуру помоћу сензора и упали хлађење уколико та вредност пређе неку критичну.

Још један пример би био одржавање путање модела ракете која полеће вертикално. Како би ово решили повежемо микроконтролер за жироскоп и закрилца ракете. Испрограмирамо га да довољно често мери смер кретања ракете и коригује и најмање промене управљањем закрилцима.

Примера оваквих проблема који не захтевају много процесорцке моћи, али су критични за правилан рад уређаја има јако много, а самим тим и микроконтролера. Један податак који о томе сведочи је да се у модерном аутомобилу користи више од 50 микроконтролера.

Први микрокотролер, TMS 1000, направили су 1971. године инжињери из компаније Texas Instruments. То је био 4-битни процесор који је радио брзином од 0.4MHz , садржао 8 бајтова РАМ меморије, 128 бајтова програмабилне РОМ меморије и 28 пинова.

Данас постоји широк дијапазон микроконтролера прилагођених различитим потребама. Узмимо за пример компанију Microchip. Њихова понуда почиње са 8-битним микроконтролером PIC10F200 који има само 6 пинова , 375 бајтова програмске меморије, 16 бајтова РАМ меморије и ради на 4MHz, а завршава се са 8-битним PIC18F97J94 који има чак 100-пинова, 128 килобајта програмске меморије, 4 килобајта РАМ меморије и ради брзинама до 64MHz.

3 Архитектура микроконтролера

Као и код сваког рачунара и овде вршимо поделу на хардвер и софтвер. Иако се тема овог рада првенствено односи на стварање софтвера неопходно је да се добро упознамо са његовим хардвером како бисмо умели да испрограмирамо микроконтролер да ради оно што желимо.

3.1 Основне структуре

Структуре које су основа рада и које су присутне у сваком микроконтролеру:

1. Централно процесорска јединица (CPU)
2. Осцилатор
3. Регистри
4. Меморија
   4.1. Радна меморија
   4.2. Програмска меморија
5. Улазно-излазни портови

3.1.1 Централно процесорска јединица (CPU)

Као што се може закључити ова јединица је центар свих збивања у микроконтролеру. Састоји се из неколико мањих целина међу којима су:

1. Декодер инструкција – место где инструкције стижу из програмске меморије. Овде се оне дешифрују и на основу тога се остала кола у процесору подешавају ради њиховог извршења.
2. Аритметичко логичка јединица – обавља све математичко логичке операције над подацима.
3. Акомулатор – место где се чувају актуелни резултати или међурезултати операција које аритметичко логичка јединица тренутно извршава.

Могућности неког CPU-а су одређене његовим сетом инструкција. То је скуп свих инструкција које он може да изврши. Примери инструкција су сабирање, логичко не, премештање података са једне локације на другу, условни скокови итд.

3.1.2 Осцилатор

Део микроконтролера који обезбеђује уједначене импусле односно такт којим процесор ради се назива осцилатор. Он такође служи да уједначи ритам и синхронизује рад свих делова уређаја. У основи осцилатора најчешће се користи кристал кварца или керамички резонатор.

Важно је напоменути да фреквенција коју осцилатор ствара није једнака фреквеницији извршавања инструкција. Инструкцијама је најчешће потребно 4 такта процесора да би се извршиле мада некада то може бити и 8 или 16. Ово значи да ако је брзина процесора нпр. 16MHz онда је брзина извршавања инстуркција 4MHz.

3.1.3 Регистри

Електронски склопови који могу да памте стања једног бајта се у пракси називају регистри. Они могу бити опште или специјалне намене (СФР регистри). Регистри опште намене немају унапред предодређену намену већ се могу произвољно користити. СФР регистри имају унапред одабрану улогу. Они су физички везани за разне склопове унутар микроконтролера и служе за конфигурисање њиховог рада. Нпр. регистар који контролише брзину рада осцилатора. Он је спојен са осцитлатором и његово стање директно одређује фреквенцију рада процесора.

3.1.4 Меморија

Меморија је део микроконтролера у коме се чувају подаци. Она се састоји од много меморијских локација од којих свака има јединствену адресу. Приступ меморији се обавља тако што јој кажемо да ли желимо да пишемо или читамо из ње и са које адресе. Постоје разне врсте меморије у зависности од начина на који се складиште подаци. Главна подела меморије зависи од тога шта се чува у њој:

Програмска меморија (ROM)

У њој се трајно чува програм који микроконтролер извршава. Данашњи микроконтролери најчешће користе 16-битно адресирање, а како једна меморијска локације памти један бајт то је укупно 64 килобајта програмске меморије. Ако за пример узмемо да су инструкције дуге 16 битова наш код може имати до 32 хиљаде инстуркција.

Иако њен назив (Read only memory – меморија која се само може читати) неговештава да се њен садржај не може мењати то најчешће није случај. Данас се за програмску меморију угловном користи FLASH меморија по којој се може писати и брисати практично неограничен број пута. Оваква меморија идеална је за експериментисање и развијање програма. Постоји и ОТП (OTP) меморија (One time programmable – једном програмабилна) она права ROM меморија, која је јефтинија и користи се у масовној производњи.

Радна меморија (RAM)

Садржај ове меморије брише се након нестанка напајања и она служи за привремено чување важних података који се тренутно користе у извршавању података.

ЕЕПРОМ (EEPROM) меморија

Овај назив је скраћеница од „Electrically erasable programmable ROM“ што у преводу означава меморију чији се садржај такође може мењати у току рада. Разликује се од RAM-а по томе што се подаци не губе при нестанку напона. Наравно уз ову предност долазе и неке мане. Брзина уписа је знатно нижа у односу на RAM, а такође постоји и ограничен број уписа. Идеална је за податке који се морају трајно чувати али који се ретко мењају као на пример лозинка која се чува у електронској брави.

3.1.5 Улазно-излазни портови

Да би рад микроконтролера имао икаквог смисла он мора да комуницира са додатном електроником, односно са спољашњим светом. Ова функција се врши преко улазно-излазних портова. То су специјални регистри чији је сваки бит повезан за један пин односно извод на кућишту. Пинове даље повезујемо жицама за неку електронску компоненту и нај тај начин остварујемо комуникацију. Најпростији пример тога јесте пин повезан за LED диоду. Постављањем стања тог пина на логичку јединицу палимо диоду.

Назив улазно-излазни означава да исти пин може служити или за унос или за извоз података. Подешавање ове карактеристике се обавља софтверски за шта је опет задужен регистар специјалне намене (SFR) . Ово значи да се улога пина може мењати током извршавања програма.

3.2 Специјалне структуре

Ове структуре нису увек неопходне за извршавање функција микроконтролера, али знатно повећавају његову функционалност. Неке од њих су:

1. Интерапт
2. Тајмери/Бројач
3. Вочдог тајмер
4. АД конвертор
5. Модул за серијску комуникацију
   5.1. I^2C (Inter Integrated Circuit)
   5.2. SPI (Serial Peripheral Interface Bus)
   5.3. UART (Universal Asynchronous Receiver/Transmitter)

3.2.1 Интерапт

Посао микроконтролера се углавном своди на реаговање на неке промене у спољашњем свету. На пример притисак тастера за позивање лифта ће покренути читав низ процеса који доводе лифт до вас. Често је важно да се та реакција догоди брзо. На пример електронски систем за стабилизацију возила мора што пре да се укључи уколико дође до проклизавања точкова. Један од начина да се ово реши је да се микроконтролер врти у бесконачној петљи и стално проверава да ли је све у нормали. Ово решење уопште није оптимално јер контролер обавља доста непотребног посла.

Интерапт је решење за овај проблем. То је асинхрони хардверски механизам који се активира кад дође до некакве промене која тражи реаговање. Интерапт прекида главни ток програма и на кратко преузима контролу. Након интервенција која је пожељно да буде што краћа контрола над уређајем се враћа на главни ток и рад се наставља као да се ништа није догодило.

3.2.2 Тајмер/Бројач

Осцилатор микроконтролера је задужен за усклађени рад целог уређаја. Он то постиже стварањем тачно одређене и веома стабилне фреквинције. Бројањем ових тактова веома презисно можемо мерити време. Управо то је задатак тајмера.

Тајмер је најобичнији регистар чија се вредност увећава сваки инструкцијски циклус. Ако је за једну инстуркцију потребно на пример 250 микросекунди да се изврши, а тајмер изброји 4000 извршених инструкција знамо да је од почетка мерења протекла једна секунда.

Како су регистри најчешће дуги 8 или 16 битова самим тим можемо избројати највише 2^8 – 1 = 255 односно 2^16 – 1 = 65535 извршених инструкција. У случају са да имамо 8-битни тајмер мерење једне секунде на уређају из прошлог примера би захтевало да 16 пута бројимо до 250 и сваки пут правимо интервенцију где вредност вратимо на нула. Овај посао је знатно олакшан коришћењем прескалера. То је коло које се уметне између осцилатора и тајмерског регистра. Његова функција је да ефективно смањи фреквенцију извора тако што ће уместо сваког пута импулс преносити сваки 2, 4, 8, 16... пут. У нашем случају постављање прескалера на однос 1:16 би нам омогућило да само једном бројимо до 250 како би измерили секунду.

Шта ако не желимо да стално проверавамо тајмерски регистар чекајући да прође одређено време већ желимо да обавимо неке друге послове док чекамо. Овај проблем решавамо коришћењем тајмерског интерапта. До њега долази када тајмерски регистар прекорачи (overflow) максималну вредност односно кад се увећа са 255 на 0. Након тога у интерапт рутини обавимо посао који желимо и вратимо контролу програма главном току. На овај начин постижемо да на сваку секундо радимо одређену акцију уз минимално ометање главног програма.

Ако уместо унутрашњег осцилатора на тајмер доведемо неки спољашњи извор импулса добијамо бројач импулса. На овај начин не можемо мерити време зато што тај импулс

углавном неће бити периодичан. Уместо тога можемо бројати неке догађаје. На пример повежемо дугме од лифта на бројач и на тај начин меримо број позива лифта. Коришћење прескалера и интерапта функционише исто као и код тајмера.

3.2.3 Вочдог тајмер

Ово је специјални тајмер који је повезан на засебан осцилатор. Сам назив објашњава његову улогу (watchdog – пас чувар). Он тако рећи надгледа систем и уколико дође до прекорачења регистра рестартује цео микроконтролер. Ово је јако корисно када програм упадне у неку бесконачну петљу међутим да би избегли нежељени рестарт морамо довољно често постављати његову вредност на нула.

3.2.4 А/Д конвертор

Када желимо да измеримо неку вредност као на пример температуру користимо сензор који одређену вредност температуре претвара у одређен напон. Како су пинови дигитални и комуницирају само јединицама и нулама неопходан нам је аналогно-дигитални конвертор. То је коло претвара аналагну вредност напона у дигитални број који микроконтролер може да разуме и користи у даљим израчунавањима.

3.2.5 Модул за серијску комуникацију

Комуницирање са спољашњим светом преко улазно-излазних пинова је идеално решење за мала растојања. Међутим, када треба остварити комуникацију на већим удаљеностима или када паралелна комуникације није могућа једина опција је коришћење серијске везе. Серијска значи да се сигнали шаљу један за другим у серији.

Најважнија ствар код овакве комуникације је строго придржавање Протокола. Протокол је јасно дефинисан скуп правила (заједнички језик) која обе стране морају да поштују како би се разумели и разменили податке.

Најчешће коришћени модули за серијску комуникацију су:

1. $\mathrm{I}^2\mathrm{C}$ (Inter Integrated Circuit)
2. SPI (Serial Peripheral Interface Bus)
3. UART (Universal Asynchronous Receiver/Transmitter)

$\mathbf{I^2C}$ (Inter Integrated Circuit)

Ово је систем који се користи између уређаја на истој штампаној плочи. Веза се остварује помоћу два проводника: један за пренос података и други за усклађивање такта. Комуникација може тећи у оба смера, али не истовремено. Код овакве везе увек је један уређај главни (master) и при остваривању комуникације врши се прозивка међу подређеним (slave) уређајима. Након што се прозове уређај податке размењују искључиво он и главни уређај.

SPI (Serial Peripheral Interface Bus)

Овај систем је веома сличан $\mathrm{I}^2\mathrm{C}$ вези по томе што се комункација одвија прозивком од стране главног уређаја. Разлика је у томе што се користе четири проводника: један за слање, други за примање података и трећи за давање такта. Четврти проводник служи за одабир подређеног уређаја тако што је сваки од њих повезан посебним проводником. Оваква веза се назива „full duplex“ што значи да подаци могу да се шаљу и примају истовремено. Брзина размене је већа него код $\mathrm{I}^2\mathrm{C}$.

UART (Universal Asynchronous Receiver/Transmitter)

Оваква веза је асинхрона, што значи да се не користи посебна линија за одређивање такта. Пример такве комуникације је даљински управљач. Синхронизација се обавља тако што уређаји податке шаљу унапред одређеном брзином. Цела комуникација се своди на конвертовање паралелног бајта у серијске битове.

4 Детаљнији примери коришћења микроконтролера

У овом поглављу ћемо навести три примера једноставних уређаја. Детаљно ћемо објаснити њихову намену и начин рада како би умели да напишемо програме за њих. Програмирање сва три примера биће објашњено у наредном поглављу.

4.1 Пример 1: Бројач

У овом примеру ћемо правити уређај који памти највише двоцифрени број и испусује га истовремено у децималном и бинарном облику. Мењање вредности броја биће омогућено притискањем дугмића којих ће бити два: један за повећавање и други за смањивање. Такође ће постојати један физички прекидач којим ће се контролисати вредност за коју се исписани број умањује или увећава.

Памћење броја се веома лако имплементира чувањем променљиве у меморији.

Мењање исписаног броја

Како не би морали стално да проверамо да ли су думићи притиснути свако дугме ће бити везано на свој пин који има могућност интерапта при промени улазног сигнала. При притиску активираће се интерапт рутина и вредност броја ће се променити без већег прекидања главног тока. Иако овај уређај у главном току неће радити ништа замислићемо да је ово само део неког компликснијег уређаја и да нам је важно да су прекиди главног тока што краћи.

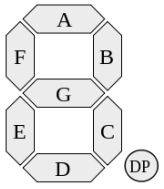
Постоје две врсте дугмића и сходно томе две врсте интерапта. Уколико дугме пропушта напон, а након притиска га зауставља потребан нам је интерапт на силазној ивици односно при паду напона на том пину. Уколико се дугме понаша супротно користићемо интерапт при узласној ивици односно скоку напона.

Мењање вредности за коју се исписани број мења ће бити одређена стањем посебног прекидача на уређају. Прекидач је веома једноставно коло које може бити у два стања. Кад је укључен он ће пропуштати напон, а кад је искључен неће. Стање прекидача лако одређујемо тако што га вежемо на улазни порт и при свакој промени броја проверимо његово стање. Уколико прекидач није укључен број ћемо мењати за један, а уколико јесте за пет.

Приказивање броја у бинарном облику

Најлепше решење за овај проблем било би да користимо ЛЦД (LCD) екран. Међутим, писање драјвера за такав уређај би непотребно отежало наш посао. С обзиром да ћемо једино исписивати нуле и јединице за приказ броја користићемо низ ЛЕД (LED) диода. Упаљена диода означаваће јединицу, а угашена нулу. Како је највећи број који ћемо представљати $99 = 2^0 + 2^1 + 2^5 + 2^6$ биће нам потребно укупно седам ЛЕД диода. Свака диода ће бити везана за свој излазни пин. Да би исписали неки број довољно је да упалимо сигнал на одређеним пиновима и угасимо на осталим.

Приказивање броја у децималном облику



За ову сврху користићемо компоненту која се зове 7-сегментни дисплеј. То је пластично кућуште у коме су распоређене ЛЕД диоде тако да паљењем одређених можемо приказати цифре 0, 1, 2...9. Уређај се састоји из седам ЛЕД диода за цифре и једне која служи за приказивање децималне тачке (видети слику). Важно је напоменути да се свака ЛЕД диода независно контролише односно да овај уређај нема никакву унутрашњу логику која ће исписивати бројеве већ ће тај посао обављати микроконтролер. Нама је неопходно да испишемо две цифре и уколико желимо да микроконтролер директно управља свим диодама мораћемо да издвојимо укупно 16 пинова. Ово није нимало ефикасно, али овај проблем можемо лако решити коришћењем шифт регистра (shift register) .

Шифт регистар је електронска компонента која памти један бајт и сваки бит одржава напон на свом излазу који ћемо у нашем случају повезати на укупно осам ЛЕД диода 7-сегментног дисплеја. Мењање вредности бајта у шифт регистру функционише на следећи начин. Микроконтролер шаље регистру сигнал који може бити нула или један. Шифт регистар шифтује свој бајт удесно и тиме губи вредност на најнижем биту, док се на највиши уписује примљени бит од микроконтролера. Слање тог бита иде преко два проводника са два одвојена пина: први се назива дата (data) пин, а други клок (clock) пин. Први се поставља на вредност коју желимо да пошаљемо, а затим се промени вредност другог. При промени другог пина шифт регистар узима ону вредност која је тренутно на првом проводнику. На овај начин можемо изделити бајт који желимо да пошаљемо регистру и преко само два пина серијкси послати ту вредност. Такође постоји и трећа линија којом се може обрисати вредности бајта односно поставити све битове на нула, али она није неопходна.

Сада знамо како да испишемо два броја користећи два 7-сегментна дисплеја, два шифт регистра и четири пина, међутим исту ствар можемо урадити користећи само два пина. Један је дата пин и он је везан за први шифт регистар. Други је клок и он ће бити везан за оба шифт регистра. Уместо да спојимо дата пин из контролера ка другом регистру спојићемо најнижи излазни бит првог регистра на дата улаз у другом. На овај начин уписивањем бита у први регистар његов најнижи бит се не губи већ се прослеђује другом регистру. Овако добијамо еквивалент 16-битном шифт регистару са два 8-битна.

Уписивање цифара ће функционисати тако што ћемо имати маску за сваку цифру од осам битова и уколико желимо да испишемо нпр. 32 прво ћемо регистрима послати маску броја 2, а затим маску броја 3.

4.2 Пример 2: Мерење Температуре

У овом примеру правимо уређај који мери сопствену температуру помоћу сензора и исписује њену вредност на 7-сегментни дисплеј. Уређај ће такође проверавати да ли је температура безбедна нпр. мања од 30 степени, умерена нпр. до 40 степени или критична преко 40 степени. У зависности од тога палиће се зелена, жута или црвена диода.

Мерење температуре

Сензор који ћемо користити температуру мери на следећи начин. Он прави напон који је линеарно сразмеран температури по формули:

$$\mathrm{U} = \mathrm{T} * \mathrm{c} + \mathrm{U}_0$$

где су U_0 и c неке фиксне константне које зависе од сензора, а T температура коју меримо. Како би наш микроконтролер умео да измери овај напон ми га доводимо на пин који има могућност А/Д конверзије. Овај конвертор упоређује напон који меримо са референтним напоном који му такође морамо довести на одвојен пин и у зависности од њиховог односа уписује број X у одређени регистар микроконтролера и то по формули:

$$\frac{\mathrm{U}}{\mathrm{U}_{\mathrm{ref}}} = \frac{\mathrm{X}}{2^{\mathrm{b}}},$$

где је број $\mathrm{U}_{\mathrm{ref}}$ референтни напон, а b резолуција А/Д конвертора која је у нашем случају 10-битна. Ако је нпр. референтни напон 2V, а ми желимо да измеримо напон од 500mV уз помоћ 10-битног конвертора вредност X ће бити 256. Вредност конверзије се увек заокружује на најближи цео броји јасно је да што више битова ковертор има то прецизније може измерити вредност.

Исписивање

Једино што нам преостаје је да преко ове две формуле израчунамо температуру и њену вредност испишемо на дисплеј на исти начин као у прошлом примеру. Након проверавања да ли је безбедна, умерена или критична упалићемо одговарајућу диоду.

4.3 Пример 3: НФЦ (NFC)

У овом примеру правимо уређај који ће моћи да размењује податке са паметним телефоном или било којим уређајем који користи НФЦ технологију. Он ће то радити користећи ЕЕПРОМ меморију у коју ће податке моћи да уписују и читају и телефон и микроконтролер. Она ће служити као посредник у њиховој комуникацији. Као пример података које контролер шаље телефону користићемо температуру из примера 2. Телефон ће моћи да конфигурише границе интервала безбедне, умерене и критичне температуре и на тај начин демонстрира комуникацију у том смеру.

НФЦ (NFC) комуникација

НФЦ је скраћеница од "Near field communication" односно комуникација на блиској удаљености. Ова технологија је све присутнија у данашњем свету. Њена највећа предност је управо кратак домет који онемогућава било који вид прислушкивања или крађе података. Пошто тема овог рада не залази у програмирање паметних телефона нећемо се бавити комуникацијом са друге стране меморије.

Комуникација између микроконтролера и меморије

Ова комуникације вршиће се по $\mathrm{I^2C}$ протоколу о којем је већ било речи. Веза се састоји из два проводника: први ћемо звати дата (data), а други клок (clock). Размотримо прво случај читања из меморије.

Читање бајта из меморије

Комуникација отпочиње тако што главни (master) уређај изазове почетни услов (start condition) односно пад напона на дата линији док ја напон на клок линији висок. Након тога се сви подређени (slave) уређаји буде и чекају да буду прозвани. У нашем случају је то један уређај али може их бити и много више. Одабир подређеног уређаја врши се слањем његове јединствене адресе и то на следећи начин. Напон на клоку алтернира и при свакој узлазној ивици подређени уређаји очитавају дата линију. Адресе подређених уређаја су најчешће дугачке седам битова тако да се овај процес алтернирања ради седам пута. Осми бит који се шаље означава да ли се врши читање или писање. Иако ми желимо да читамо податке из меморије ово је фаза прозивке и овде се увек шаље нула која означава писање. Након овога се на исти начин шаље адреса у меморији са које желимо да прочитамо бајт. Адреса може имате више бајтова. Затим микроконтролер поново изазива почетни услов међутим овај пут само одабрани уређај слуша и сад му шаљемо адресу праћену јединицом која означава да желимо да читамо из њега. Како сада меморија има све информације о томе шта треба да ради она нам шаље бајт који смо тражили. Уколико смо успешно примили бајт меморији шаљемо бит потврде (acknowledge bit) и правимо стоп услов (stop condition) који означава крај комуникације и ослобађа линију у случају да има више уређаја на њој. Стоп услов је супротан старт услову: узлазна ивица дата линије док је клок линија под напоном.

Уписивање бајта у меморију

Функционише веома слично као и читање. Након стартног услова шаље се адреса са битом за упис затим адреса у коју желимо да упишемо бајт. Након тога уместо поновљања старт услова једноставно пошаљемо бајт који желимо да упишемо и примимо бит од меморије који нам говори да ли је комуникација успешно прошла. Завршавамо везу стварањем стоп услова.

Размена података између микроконтролера и телефона

Како би ово решили одредићемо нека места у меморији преко којих ће ови уређаји комуницирати. Одабраћемо неку меморијску локацију на коју ћемо уписивати вредност температуре. Телефон ће је читати, а он ће границе интервала исто тако уписивати на неку другу адресу са које ће их микроконтролер читати пре сваке провере температуре.

5 Програмирање микроконтролера

У овом поглављу ћемо приказати програме који су описани у четвртој глави, али пре тога морамо навести пар ствари као што су окружење у ком ћемо радити или језик који ћемо користити. Микроконтролер који ћемо програмирати је PIC18F45K20. Овај контролер садржи све потребне структуре и биће више него довољан за наш програм. Уз њега ћемо користити програматор PicKit3. Са рачунаром се повезује преко USB-а, а са микроконтролером преко неколико специјалних пинова. То је све што нам је потребно од хардвера. Као развојно окружење ћемо користити MPLAB X IDE које аутоматски подржава све наше уређаје.

5.1 Разлика програмирања у асемблеру и C-у

Програмирање у асемблеру захтева веома добро познавање сета инструкција процесора као и доста вештине. И поред тога било који дужи програм је ноћна мора за дебаговање и прилично тежак за разумевање уколико га нисте сами куцали. Још један проблеми са асемблером је што је код тешко преносив са уређаја на уређај. Уколико преносите код са једног модела процесора на други често је неопходно променити имена неких пинова или регистара. Овај посао се доста лакши обавља у вишем језику као што је C него у асемблеру коришћем дефиниција (#define).

Из ових разлога језик који ћемо користи јесте C. Иако је доста лакши за употребу он не долази без мана. Код искуцан у асемблеру можете бити доста оптимизованији и иако програмирамо у C-у коришћење неких асемблерских инструкција као што су NOP или SLEEP не можемо заобићи јер C за њих нема замену.

5.2 Пример програма који покреће микроконтролер

Како би програми за сваки од три примера из четвртог поглавља у великом делу били исти у овом поглављу ћемо представити јединствен програм који ће имати све функционалности из тих примера. Да би ипак на неки начин раздвојили примере користићемо прекидаче којих на уређају укупно има четири. У зависности од њихових стања биће активни само делови програма који одговарају примерима.

Програм се састоји од девет изворних кодова:

1. adc.c – А/Д конверзија и функције за исписивање температуре на дисплеј и диоде
2. cnt.c – Мењање бројача и исписивање на дисплеј и диоде
3. configuration_bits.c – Конфигурациони битови процесора
4. i2c.c – Драјвер за I^2C комуникацију
5. init.c – Главна функција иницијализације и функција за чекање (wait())
6. interrupt.c – Интерапт рутине
7. m24lr.c – Драјвер за комуникацију са ЕЕПРОМ меморијом
8. main.c – Главни део програма
9. sevenseg.c – Драјвер за исписивање цифара на дисплеју

и седам заглавља од којих шест служе за прототипове функција:

1. adc.h
2. cnt.h
3. i2c.h
4. init.h
5. m24lr.h
6. sevenseg.h

и седмо које служи за глобалне променљиве, дефинисање константни и типова:

7. system.h

5.2.1 Конфигурациони битови (configuration_bits.c)

Овај део кода је потпуно независтан од остака и није много важан за објашење рада целокупног програма. Он садржи све специјалне конфигурационе битове процесора којих има доста и којих нема потребе све наводити. Првих неколико, а и све остале линије изгледају овако:

```
// CONFIG1H
#pragma config FOSC = INTIO7
#pragma config FCMEN = OFF
#pragma config IESO = OFF
// CONFIG2L
#pragma config PWRT = OFF
#pragma config BOREN = SBORDIS
#pragma config BORV = 18
...
```

Добра ствар је што што овај део кода није потребно куцати већ га MPLAB може сам генерисати заједно са коментаримо који објашњавају шта свако подешавање ради.

5.2.2 Драјвер 7-сегментног дисплеја

Код који контролише дисплеј се налази у sevenseg.c, а код који следи је из истоименог заглавља:

```
#include <xc.h>
#include "system.h"

#define DCLK RE1
#define DDAT RE2
#define DCLR RC5

#define DCLK_DIR TRISE1
#define DDAT_DIR TRISE2
#define DCLR_DIR TRISC5

void init_sevenseg(void);
void clear(void);
void write_digit(u08 digit,u08 dp);
```

Основни регистри које ћемо користити за рад са улазно-излазним портовима су TRIS и R регистри. TRISE је пример регистра који за сваки од осам пинова са порта E одређује да ли је улазни или излазни. TRISE1 је други бит овог регистра, а TRISE0 први. Уколико је вредност TRISE1 постављена на један овај пин је улазни и тада користимо регистар RE1 како би читали да ли на њему примамо сигнал или не. Уколико је TRISE1 нула тада је пин излазни и уписом јединице у RE1 доводимо напон на тај пин.

Тип u08 означава unsigned промељиву дужине 8 бита и дефинисан је у "system.h".

Микроконтролер са дисплејом односно шифт регистром комуницира преко три пина RE1, RE2 и RC5. Коришћење ових имена које су у ствари замаскиране адресе регистара стања ових пинова нам је омогућено захваљујући заглављу <xc.h> у којем су сви регистри микроконтролера овако дефинисани. Кад не би било ових маски морали би да нпр. за приступ регистру TRISE уместо TRISE пишемо физичку адресу тог регисата 0xF96 што би нам доста закомпликовала посао.

Такође можемо приметити начин редефинисања имена регистара портова. Иако је доста згодно приступати TRISE1 него другом биту са адресе 0xF96, ствари можемо додатно упростити давањем смислених имена тим регистрима. Први разлог је што ће код бити знатно лакши за читање и разумевање ако користимо DCLK (Display Clock) уместо RE1 (други пин порта E). Други разлог је што ако се некада одлучимо да исти код користимо на другом уређају код кога распоред пинова није исти довољно је да за сваки пин који се разликује променимо само једну линију кода. Портове ћемо овако редефинисати током целог програма.

Пошто смо се упознали са основом коришћења портова можемо да пређемо на имплементације функција и изворни код sevenseg.c:

```
#include "sevenseg.h"
#include <xc.h>

static  u08 digit_mask[] = {0xFC, 0x60, 0xDA, 0xF2, 0x66,
                    0xB6, 0xBE, 0xE0, 0xFE, 0xF6};

void init_sevenseg() {
   DCLK_DIR = 0;      // Definise pin linije DCLK kao output
   DDAT_DIR = 0;      // Definise pin linije DDAT kao output
   DCLR_DIR = 0;      // Definise pin linije DCLR kao output
   clear();           // Brise ceo ekran
}

void clear() {
   DCLR = 1;
   nop();
   DCLR = 0;
   nop();
   DCLR = 1;
}

void write_digit(u08 digit,u08 dp) {
   DCLK = 0;
   for(u08 i = 0; i<8 ;i++) {
      DDAT = ((digit_mask[digit]+dp)>>i) & 0x01;
      nop();nop();nop();
      DCLK = 1;
      nop();nop();nop();
      DCLK = 0;
   }
   DDAT = 0;
}
```

init_sevenseg() је функција која се покреће при паљењу уређаја. Она конфигурише пинове за комуникацију са дисплејом као излазне.

Функција clear() ради тако што направи силазну ивицу на линији DCLR која је повезана за оба шифт регистра и на тај начин им даје сигнал да обришу свој садржај.

Функција nop() позива асемблерску функцију NOP која не ради ништо односно служи да уређај мирује један инструкцијски циклус. Овде је коришћена да временски раздвоји промене напона и да да времена шифт регистру да обрише свој излаз.

Функција write_digit() уписује цифру на место десетица на екрану док на место јединица долази стара цифра десетица. Она ради тако што шаље један бајт вишем шифт регистру преко линије DDAT при промени линије DCLK. digit_mask[] је низ дужине 10 у којем се налазе бајтови за приказивање сваке цифре. Уколико је вредност параметра dp једнака један након цифре ће се упалити и децимална тачка. Овде такође користимо функцију nop() како би стабилизовали пренос.

5.2.3 Бројач

Он је имплементиран у изворном коду cnt.c чије заглавље нећемо наводити јер не садржи ништа осим прототипова функција.

```
#include "cnt.h"
#include "sevenseg.h"

u08 counter = COUNTER_START_NUMBER;

void sevenseg_update_counter(s08 a) {
   counter = ((counter+a)%100+100);  // Promeni vrednost brojaca za a
   write_digit(counter%10,0);              // Upisi cifru jedinica]
   write_digit((counter/10)%10,0);      // Upisi cifru desetica
}
void update_led_counter() {
   LED = counter - 100;
}
```

У променљивој counter памтимо вредност бројача и њу иницијализујемо на неки број који означавамо константом COUNTER_START_NUMBER (из system.h) тако што ако желимо да та вредности буде 1 (односно 01 на дисплеју) у променљивој чувамо 101. Уколико би је чували као 1 при смањивању за пет дошло би до underflow-а и вредност би постала 252%100 (односно 52 на дисплеју) док је жељена вредност 96.

update_led_counter() је веома једноставна функција која у регистар PORTD (порт на којем се налазе диоде) овде редефинисан као LED уписује вредност бројача.

5.2.4 Главна функција иницијализације и функција за чекање (wait())

Главна функција иницијализације се налази у init.c коду и она садржи све функције инизијализације у програму. Позива се на почетку main.c.

```
void init_system() {
   init_oscillator();
   init_tmr0();
   init_interrupts();
   init_ports();
   init_sevenseg();
   init_adc();
   init_i2c();
   m24_init();
}
```

Осим ње init.c такође садржи имплементације многих ових функција иницијализације који овде нећемо наводити јер се у свима само подешавају конфигурациони регистари и нису много интересантне.

Функција који има значаја јесте функција wait() и она служи да стави контролер у режим чекања тачно одређено време. Ово је јако корисно уколико желимо да нешто радимо одређен број пута за неко време. На пример да би сваке секунде мењали стања свих диода користили би wait() тачно једну секунду и онда би променили стања и тако у круг.

```
void init_tmr0() {
   T08BIT = 0;    // Konfigurise Timer0 kao 16-bitan tajmer/brojac
   T0CS = 0;      // Tajmer radi instrukcijskom brzinom (CLKOUT)
   PSA = 1;       // Iskljucuje Preskaler (1:1)
   TMR0ON = 1;    // Ukljucuje Timer0
}

// Cekanje ms milisekundi
void wait(u16 ms) {
   TMR0 = 0xFFFF - MS_AT_FCY;
   elapsed = 0; // Resetuj vrednost elapsed
   TMR0IE = 1; // Ukljuci interapt na tajmeru
   while(elapsed < ms); // Cekaj da se interapt desi ms puta
   TMR0IE = 0; // Iskljuci interapt na tajmeru
   return;
}
```

Овде можемо видети функцију wait() заједно са функцијом иницијализације тајмера. Константа MS_AT_FCY једнака је броју извршених инструкција за време једне милисекунде. У функцији иницијализација видимо да тајмер ради управо том брзином и можемо да закључимо да ће вредност регистра TMR0 након тачно једне секунде достићи 0xFFFF, а након тога 0x0000 што ће направити интерапт:

```
void interrupt inter() {
   if(INT1IF) {
      if(SW0) set_decp(1);
      else sevenseg_update_counter(increment[SW3]); // Uvecaj brojac
      INT1IF = 0; // Ocisti zastavu
   }
   else if(INT0IF) {
      if(SW0) set_decp(0);
      else sevenseg_update_counter(-increment[SW3]); // Umanji brojac
      INT0IF = 0; // Ocisti zastavu
   }
   else if(TMR0IF) {
      elapsed++; // Uvecaj proteklo vreme
      TMR0 = 0xFFFF - MS_AT_FCY;
      TMR0IF = 0; // Ocisti zastavu
   }
   else {
      // Interapt za koji nemamo odredjenu rutinu
   }
}
```

inter() је једина функција из interrupt.c и она се позива сваки пут када дође до инерапта у систему. INT1IF, INT0IF и TMR0IF су заставе интерапта и њих систем поставља на јединицу уколико су оне узрок интерапта. Оваквим низом наредби if и else брзо проверавамо шта је

довело до интерапта и након што урадимо рутину која је пожељно да буде што краћа обавезно „очистимо“ заставу.

Пошто је дошло до интерапта тајмера видимо да се променљива elapsed увећава, регистар тајмера ресетовао и да ће до интерапта долазити сваке милисекунде. Ово ће се понављати ms (параметар функције wait()) пута односно трајати ms милисекунди све док се while петља у функцији wait() не заврши и интерапт на тајмеру искључи.

Заставе INT1IF, INT0IF су интерапти који се активирају притиском на дугмиће. Резултат њиховог рада зависи од регистра SW0 који је везан за први прекидач. Уколико је регистар укључен позива се функција која мења оквир приказивања температуре (приказује цео број степени или цифру јединица и једну децималну). Уколико прекидач није укључен долази до повећавања или смањења вредности бројача за вредност одређену низом increment[] (из system.h) која опет зависи од тога да ли је трећи прекидач укључен.

5.2.5 А/Д конверзија

Аналогно дигитална конверзија је испрограмирана функцијом update_temp() у adc.c:

```
void update_temp() {
   temp_mV = ADRES; // Procitaj rezultat konverzije
   temp_C = temp_mV + temp_mV - 500; // Pretvori rezultat u celzijuse
   sevenseg_write_temp(); // Prikazi temperaturu na displeju
   update_led_temp(); // Prikazi temperaturu na diodama
   if(++cnt2 == 20) // Upisi rezultat u EEPROM memoriju na adresu 4:5
   {
      m24_write_byte(4,HI(temp_C));
      wait(5);
      m24_write_byte(5,LO(temp_C));
      cnt2 = 0;
   }
   GODONE = 1; // Zapocni konverziju
}
```

Ова функција се позива и main.c десет пута у секунди како би температура на екрану била што свежија. Процес конверзије започиње се постављањем бита GODONE и он траје неколико милисекунди након чега се резултат конверзију уписује у регистар ADRES. При следећем уласку у функцију проћи ће довољно времена да се конверзија заврши и тада се резултат памти и исписује на дисплеј и ЛЕД диоде. Уписивање у ЕЕПРОМ се врши једном у 20 пута како меморија не би изгубила могућност уписивања података због превише коришћења. Пауза од пет милисекунди је неопходна како би меморија уписала бајт пре него што јој стигне следећи.

Исписивање на ЛЕД диоде изгледа овако:

```
void update_led_temp() {
   if(SW2) {
      if(++cnt1 == 20)
      {
         m24_read_byte(0,&lower_bound);
         m24_read_byte(1,&higher_bound);
         cnt1 = 0;
      }
   }
   else {
      lower_bound = START_LOWER_BOUND;
      higher_bound = START_HIGHER_BOUND;
   }
   if(temp_C < 10*lower_bound) LED = 4;
   else if(temp_C < 10*higher_bound) LED = 2;
   else LED = 1;
}
```

Уколико је трећи прекидач укључен границе за температуру се читају из ЕЕПРОМ меморије и то сваки 20. пут како би се растеретило коришћење меморије од стране микроконтролера јер би иначе пристип телефоном био немогућ (меморија би стално била заузета). Након тога се температура упоређује са границама и пали одговарајућа диода (границе се множе са десет зато што се температура чува умножена за десет како би била цео број нпр. 25.2C се чува као 252).

5.2.6 Драјвер за I^2C комуникацију

То су драјвери за комуникацију са ЕЕПРОМ меморијом и иплементирани су у заглављу i2c.c. Дакле ово заглавље садржи основне функција за I^2C комуникацију као што су оне које стварају старт или стор услов или које шаљу или примају бајт. Све ове функције су имплементиране веома слично:

```
void i2c_repeat_start(void) {
   RSEN = 1;          // Ponovo napravi start uslov
                              // Nakon sto se napravi start uslov RSEN = 0 i SSPIF = 1
   while (!SSPIF);    // Sacekaj da se zavrsi
   SSPIF = 0;         // Ocisti zastavu
}

bit i2c_trm_byte(u08 data) {
   SSPBUF = data;     // Popuni bafer za slanje
   while (!SSPIF);    // Sacekaj da se zavrsi
   SSPIF = 0;         // Ocisti zastavu

   return ACKSTAT;    // Potvrda od podredjenog
}
```

Оне раде тако што поставимо неки бит у зависности од тога шта желимо да урадимо (нпр. RSEN уколико желимо да направимо поновљени старт услов) и чекамо да се укључи застава SSPIF која означава да се процес завршио. Уколико шаљемо податке довољно је да напунимо бафер

за слање и контролер ће знати шта желимо и скоро сав посао ће одрадити сам. Захваљујући овој функционалности нема потребе за компликованим функцијама са пуно проверавања и усклађивања времена.

5.2.7 Драјвер за комуникацију са ЕЕПРОМ меморијом

Начин рада овог драјвера је детаљно објашњен у трећем примеру четврте главе. Он користи функције за I^2C комуникацију и имплементира функције за слање и примање података које су међусабно веома сличне и зато наводимо само једну:

```
bit m24_write_byte(u16 at, u08 data) {
   i2c_start();
   i2c_trm_byte(M24_ADDR);       // 7-bitna adresa sa bitom za upis
   i2c_trm_byte(HI(at));         // Visi bajt adrese
   i2c_trm_byte(LO(at));         // Nizi bajt adrese
   i2c_trm_byte(data);           // Bajt sa podacima
   i2c_stop();
   return 1;
}
```

Комуникација отпочиње старт условом и праћена је слањем адресе уређаја за упис као и адресе у меморији која се шаље у деловима. Након тога се шаље бајт са подацима и веза се завршава стоп условом.

5.2.8 Главни ток програма (main.c)

```
void main() {
   init_system(); // Inicijalizuj sistem
   while(true) {    // Glavni tok programa
      wait(100);                  // Cekaj 100 milisekundi
      if(SW0) {
         update_temp();           // Ispisuj temperaturu
      } else {
         sevenseg_update_counter(0); // Ispisuj brojac
         update_led_counter();
      }
   }
   return;
}
```

Након иницијализације програм улази у бесконачну петљу у којој у зависности од првог прекидача исписује температуру или бројач. Функција за чекање се позива како се не би претерало са освежавањем дисплеја и још важније уписом у меморију.

6 Закључак

Главни задатак овог рада био је да пренесе читаоцу основне идеје програмирања микроконтролера. Иако је овај рад подељен на неколико глава мислим да се поред ње може направити још једна подела и то на два дела:

Први који обухвата прве три главе и говори о хардверу и архитектуру микроконтролера. У њему сам се трудио да једноставним и логичним примерима објасним неке структуре који на први поглед не делују много корисно (Вочдог тајмер). Овај део можемо назвати и теоретским делом јер се говори о основним концептима као што су интерапт и регистри чије је познавање неопходно за било какав облик програмирања.

Други део је централни део рада и он се бави главном темом односно програмирањем. Иако почиње главом о детаљним примерима употребе микроконтролера у којој се још увек не појављују никакави кодови у њему се објашњене многе идеја на које се рад касније позива. Програмом из пете главе, који сматрам носиоцем целог рада, сам покушао да прикажем употребу што више основних структура микроконтролера на што бољи начин. Мислим да сам у томе успео да захваљујући структури програма која је издељена на мање делове чија се функција може лакше објаснити.

Током писања рада научио сам много ствари о микроконтролерима и њиховој примени, али поред тога мислим да сам доста напредовао у програмирању генерално. Имао сам прилику да се боље упознам са асемблеромским језиком користећи неке његове инструкције. Такође сам научио како да код учиним јаснијим за читање и модуларанијим односно лако преносивим са уређаја на уређај коришћењем дефиниција и поделом кода на више извор кодова. Све су ово ствари са којима сам се слабо сретао у такмичарском програмирању.

7 Литература

[1] Wikipedia: “Microcontroller”

https://en.wikipedia.org/wiki/Microcontroller

[2] Wikipedia: “Microprocessor”

https://en.wikipedia.org/wiki/Microprocessor

[3] Milan Verle, PIC mikrokontroleri, Beograd, 2008.

[4] Kenneth J. Ayala, The 8051 Microcontroller, Western Carolina University, 1991.

[5] Microchip, PIC18F45K20 Datasheet, 2015.

http://ww1.microchip.com/downloads/en/DeviceDoc/40001303H.pdf