SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2712

IZVEDBA SATA STVARNOG VREMENA NA MIKROKONTROLERU PORODICE MSP430

Deni Petak

Zagreb, lipanj 2012.

Sadržaj

Uvod

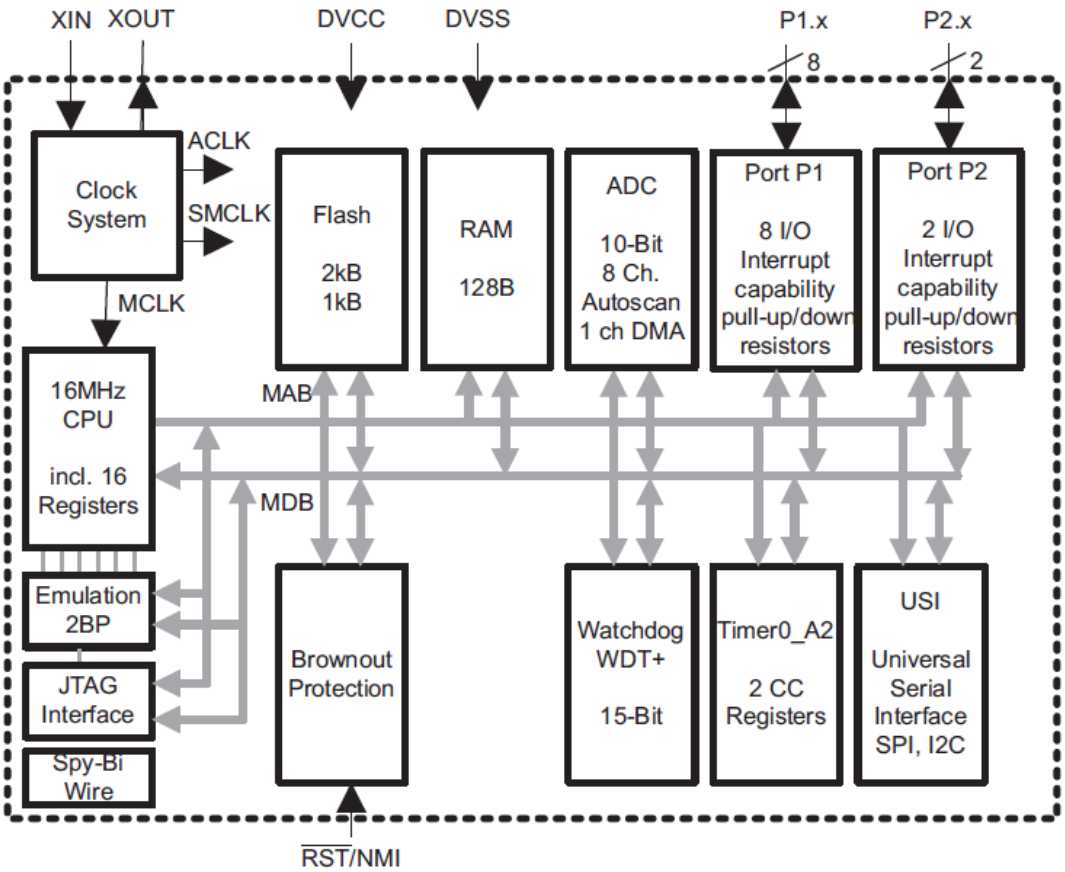
Mikrokontroler je elektronički uređaj, sličan računalu, koji ima uporabu u ugradbenim računalnim sustavima. Ugradbeni računalni sustavi koriste se svuda, često se mikrokontroleri koriste kod kontrole proizvodnog lanca, alarmnim sustavima, regulaciji osvjetljenja, mobitelima, televizorima, danas praktički u svi elektronički uređaji.

U ovom radu je opisana izvedba sata stvarnog vremena na mikrokontroleru porodice MSP430. Također je i opisana sva periferija i zadaća tih perifernih komponenata.

Familija Texas Instruments MSP430 *ultra low power* mikrokontrolera se sastoji od nekoliko uređaja koji predstavljaju različite skupine periferije ciljane za različite primjene. Arhitektura je u kombinaciji s pet nisko energetskih (*low power*) načina rada kako bi se produžio vijek trajanja baterije u prijenosnim mjernim aplikacijama. Uređaj posjeduje snažan 16-bitni RISC procesor, 16-bitne registre i stalni generator koji pridonosi maksimalnom učinkovitošću koda. Digitalno kontrolirani oscilator (DCO) omogućuje buđenje iz *low power* načina u aktivan način rada za manje od 1 μs. MSP430G2x21/G2x31 serija je *ultra low power* sklop s ugrađenim 16-bitnim brojačem, deset I/O pinova, 10-bitni A / D konverter i pri komunikaciji se mogu koristiti sinkroni protokoli (SPI ili I2C).

1. MSP430G2231 Launchpad

MSP430 *Launchpad* je razvojni sustav baziran na 16-bitnom RISC procesoru, sadržanom unutar MSP430G2231 mikrokontroleru. Dvije sabirnice [*Memory Adress Bus*(MAB) i *Memory Data Bus*(MDB)] povezuju procesor, periferije mikrokontrolera i generator takta po principu von Neumannove arhitekture, prilagođene za *low power* aplikacije. Programska memorija (*Flash*) ima kapacitet 2 kilobajta, a podatkovna memorija sadrži 128 bajtova. Ima jedan skup ulazno izlaznih priključaka (*port*) koji služi za komunikaciju mikrokontrolera s vanjskim svijetom. Arhitektura MSP430G2x31 porodice sklopova jer prikazana na slici 1.1



Slika 1.1 Prikaz arhitekture MSP430G2x31 porodice sklopova

Osnovne značajke razvojnog sustava su :

- veoma niska potrošnja energije(*ultra low power*)
- 2 kB Flash
- 128B RAM
- 10 bitni SAR A/D pretvornik
- USI za SPI/I2C komunikaciju
- 14 I/O pinova
- Timer_A s 2 C/C registra

1.1. Centralna procesorska jedinica

MSP430 procesor ima 16-bitnu RISC arhitekturu što je vrlo transparentno za primjenu. Sve operacije, osim programskih instrukcija, su izvedene kao registarske instrukcije sa sedam adresnih izvornih operanada i četiri adresnih odredišnih operanada. Procesor ima 16 integriranih registara koji smanjuju vrijeme izvođenja instrukcija. Svaki registar obavi svoju operaciju u jednom izvornom taktu. Četiri registra, R0 do R3, imaju posebne namjene: programski brojač, pokazivač stoga, status registar i stalni generator, a preostali su registri opće namjene.

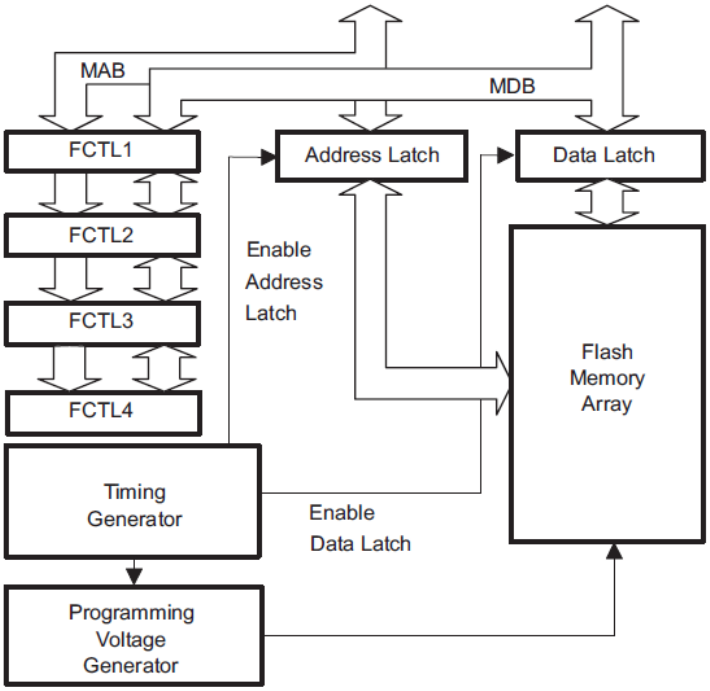
Način rada

MSP430 procesor ima jedan aktivan način rada i pet softwareskih *low power* načina rada. Prekidi mogu probuditi uređaj iz bilo kojeg *low power* načina rada i vratiti ga ponovo u taj *low power* način rada na povratku iz prekidnog potprograma.

1.2. Flash memorija

Flash memorija je u velikoj mjeri zamijenila električki izbrisiv, programabilni ROM (EEPROM). Može se programirati i brisati više puta i to se izvodi električki te je danas daleko najčešći tip memorije. Praktična razlika EEPROM je u tome što može izbrisati proizvoljne lokacije, a *Flash* može brisati samo po blokovima. Većina MSP430 uređaji

koristiti *flash* memoriju. Memorija se može programirati preko *Spy Bi Wire*/JTAG sučelja ili u sustavu od strane procesora. Procesor na flash memoriju može spremati bajt ili riječ(4 bajta ili 32 bita). Sama *flash* memorija ima kapacitet 2 kB. Blok shema *flash* memorije je na slici 1.2



Slika 1.2 Blokovski prikaz Flash memorije

1.3. Oscilator

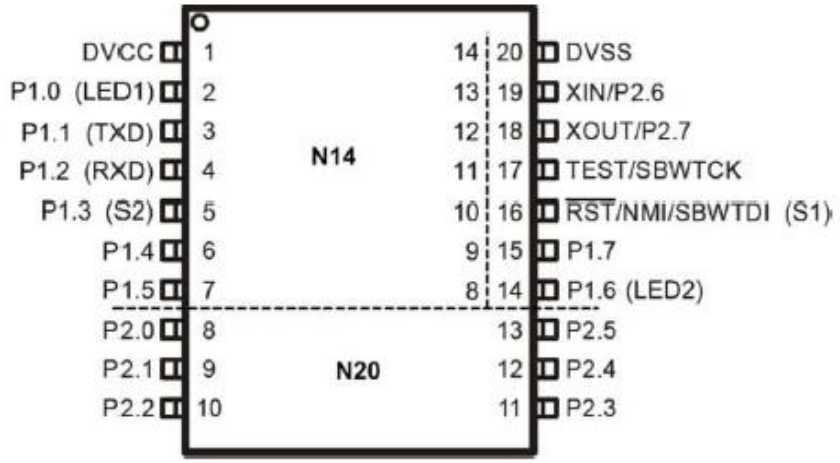
Sam takt procesorske jedinice omogućuje *Basic Clock Module+* koji podržava nisku cijenu sustava i ultra nisku potrošnju energije. Koristeći tri unutarnja takta signala, korisnik može odabrati najbolju ravnotežu performansi i niske potrošnje energije. Može se konfigurirati da radi bez ikakvih vanjskih komponenti, s jednim vanjskim otpornikom, s jednim ili dva vanjska kristala, pod potpunom softwareskom kontrolom. Unutarnji digitalno kontrolirani oscilator (DCO) omogućuje brzo paljenje i stabiliziranje unutar jedne mikrosekunde. DCO radi na frekvenciji od 1 MHz.

Sastoji se od tri izvora takta:

- ACLK- izvor je 32768 Hz kristal kvarca
- MCLK –koristi takt procesora
- SMCLK-koristi periferne podsustave za takt

1.4. Vanjski priključci

Vanjski priključci mikrokontrolera korisniku mogu biti na raspolaganju kao ulazno-izlazni ili kao priključci posebne funkcije. Svakom priključku možemo individualno zadati dali je ulazni ili izlazni(I/O). Imaju mogućnost postavljanja prekida (*interrupt*) na rastući ili padajući brid signala takta. Dva priključka služe za napajanje i masu. Za generiranje signal takta, potrebno je izvana priključiti kristal kvarca na XIN i XOUT (koji ujedno mogu biti i I/O). Jedan priključak služi za reset sustava i tu je još TEXT priključak koji ulaze tijekom programiranja mikrokontrolera. Shema priključka MSP430G2231 prikazana je slikom 1.3.



Slika 1.3 Shema priključaka mikrokontrolera MSP430G2231

Tu je jedan 8-bitni I/O *porta* P1 i 2 bita od I/O *porta* P2 :

- svi pojedinačni I/O bitovi se mogu zasebno programirati
- bilo koja kombinacija ulaza, izlaza i prekida je moguća
- svakom I/O je moguće individualno programirati *pull -up/pull-down* otpornik
- nezavisni ulazni i izlazni podatkovni registri

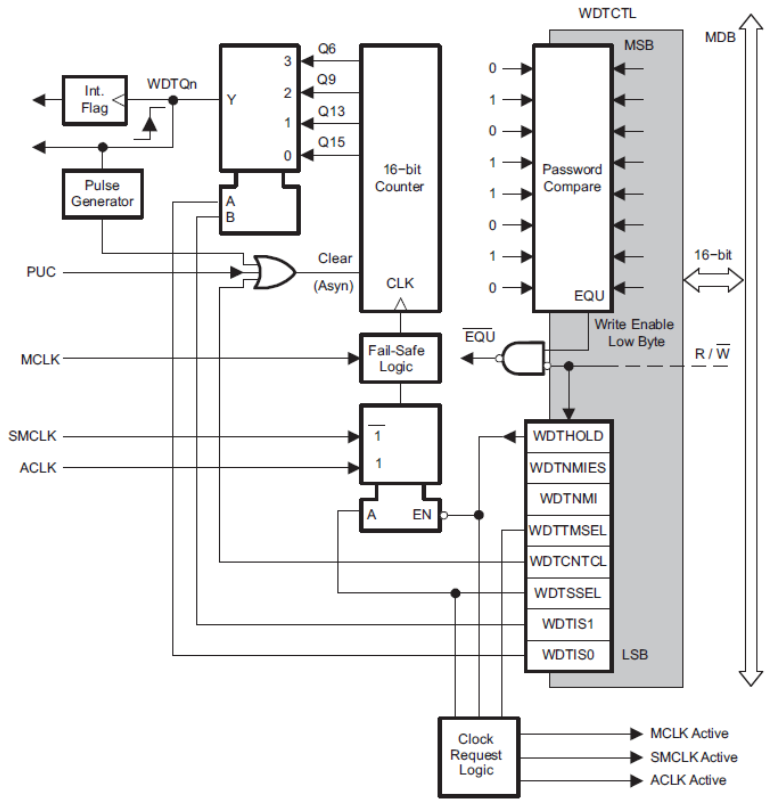
1.5. ADC10

ADC10 modul je 10 bitni analogno-digitalni pretvornik ugrađen u MSP430 Launchpad. Jezgra ADC10 sklopa pretvara analogne ulaze u njihove 10 bitne digitalne oblike i sprema ih u ADC10MEM registar. Jezgra za izbor ima dvije moguće referentne razine napona za

definiciju visoke i niske razine pri pretvorbi. ADC10 jezgra se konfigurira pomoću 2 registra, ADC10CTL0 i ADC10CTL1.

Generator signala se bira pomoću ADC10SSELx bitova između ACLK, SMCLK i MCLK izvora, te može biti podijeljen s 2, 4 i 8.

1.6. Sklopovlje za detekciju pogrešnog rada(WDT+)



Slika 1.4 Blok shema WDT+ sklopa

U praksi postoje slučajevi kada se traži velika pouzdanost elektroničkog uređaja. Kad se sklop nalazi u radnim uvjetima često je izložen nekakvim smetnjama, kao što su šiljci u napajanju (spike), elektromagnetsko zračenje i slično, pa sklop zbog tih smetnji može početi raditi neregularno. Da bi se to izbjeglo koristi se sklopovlje za detekciju pogrešnog rada (*watchdog*). I porodica MSP430 mikrokontrolera ima svoj watchdog timer (WDT+) i njegova blok shema je na slici 1.4 . WDT+ modul se može konfigurirati ili kao watchdog timer ili kao klasični timer, ovisno o tome što se upiše u WDTCTL registar. WDTCTL registar uz to sadržava kontrolne bitove za konfiguraciju RST/NMI priključka. Bilo kakvo upisivanje u WDTCTL mora u gornjim bitovima sadržavati 0x05A, ili WDT pokreće potpuni sistemski reset. Kada radi kao brojač, može ga se koristiti za periodično pokretanje *interruptova*. WDTTMSEL bit se postavlja u jedinicu, što pretvara WDT+ u klasični brojač. Brojač

odbrojava od postavljene vrijednosti, te se WDTIFG zastavica postavlja u jedinicu na kraju svakog odbrojavanja, time pokrećući prekid.

1.7. Timer_A

Timer_A je 16-bitno brojilo s 2 *capture/compare* registra. Podržava više simultanih *capture/compare* operacija, može se koristiti za PWM izlaze, te svakakve operacije koje se moraju obavljati periodično. Svaki od *capture/compare* registara, te i sam *timer* mogu pozvati vlastiti prekid. 16-bitno brojilo smješteno u TAR registru dekrementira ili inkrementira ovisno o jednom od 4 moda rada (STOP, UP, CONTINUOUS, UP/DOWN), na svako povisivanje razine generatora takta.

Za generator signala se mogu izabrati ACLK, SMCLK i MCLK izvori takta, te se svaki može podijeliti s 2, 4 i 8 preko IDx bitova. Ovisno o modu rada, mogu se postaviti generalna TAIFG *interrupt* zastavica, te dvije TACCIE zastavice ako se koriste *capture/compare* registri s time da se mora paziti na to da TACCIE0 zastavica ima najveći prioritet.

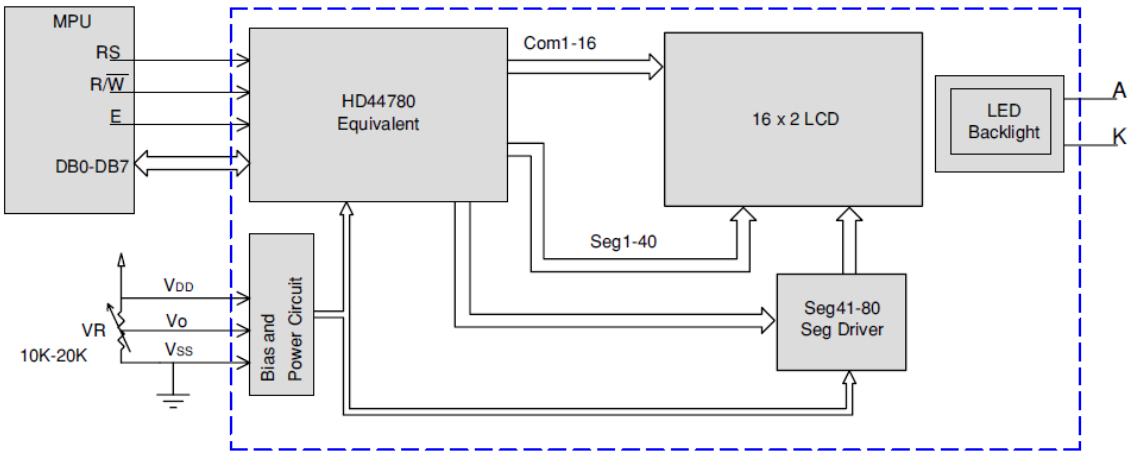
1.8. USI

USI modul podržava osnovnu funkcionalnost za sinkronu serijalnu komunikaciju. U osnovi USI je 8 ili 1 bitni posmačni registar koji se može koristiti za izlazni prijenos te uz manje softwareske preinake može se koristiti za serijalnu komunikaciju, pošto moduli ugrađeni u MSP430 porodicu sklopova, imaju već ugrađene hardwareske dodatke za olakšavanje SPI i I2C komunikaciju. Dodatno, USI moduli uključuju i mogućnost *interrupta*.

Ugrađeni brojač bitova dekrementira vrijednost uzorkovanih bitova spremnih za prijenos, te kada USICNTx registar stigne do nulte vrijednosti, sklop postavlja *interrupt* zastavicu USIIFG. Posmični registar i brojač bitova rade ovisni o istom generatoru takta. Pri promjeni razine takta, s niske na visoku, USICNTx dekrementira, a USISR uzorkuje sljedeći bit, dok se na padajućoj razini signala takta dotad uzorkovani bitovi prenose prema izlazu.

2. LCD prikaznik (*lcd display*)

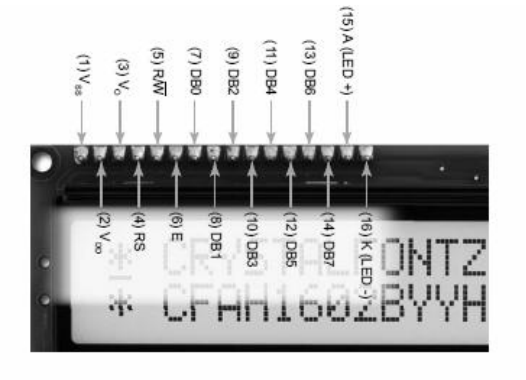
- 16 znakova po 2 linije veličina lcd zaslona
- 4-bitna ili 8-bitni paralelno sučelje
- Standardni Hitachi HD44780 ekvivalentni kontroler
- Prikazuje bijele svjetlosne znakove na plavoj podlozi
- Temperaturno područje: -20°C do +70°C



Slika 2.1 Sistemski blok dijagram LCD-a

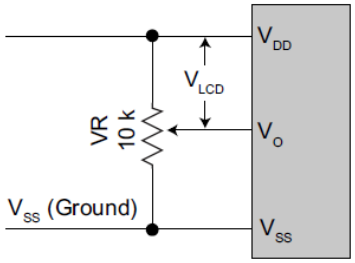
2.1. Vanjski priključci

Lcd ima 16 vanjskih priključaka preko kojih je moguće komunicirati s mikrokontrolerom. Za komunikaciju moguće je birati između 4-bitnog ili 8 bitnog paralelnog sučelja. Na slici 2.2 prikazani su valjnski priključci lcd prikaznika.



Slika 2.2 Vanjski priključci

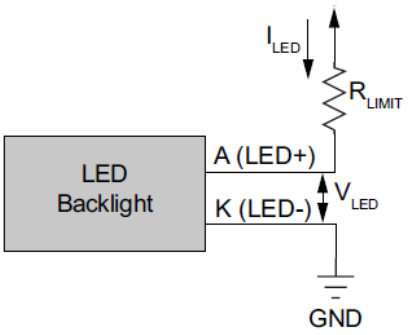
Napajanje *lcd* zaslona je izvedeno preko priključaka V_{SS}, V_{DD} i V_O. Koristi standardni DC izvor od 3.3 V te je specificirano da može radi u granicama od 2.7 do 4.5 V. Struja koja je potrebna za prikaz znakova je 1.2 mA. Često se V_{SS} priključuje na masu, a V_{DD} na izvor napajanja. Priključak V_O služi za podešavanje kontrasta te se na njega spaja 10 k potenciometar kako bi se prilagodio zaslon. Shema spajanja potenciometra na vodove napajanja je na slici 2.3.



Slika 2.3 Spajanje potenciometra na vodove napajanja

Optimalna vrijednost za V_O će se promijeniti s temperaturom, varijacijom V_{DD} i kutom gledanja. U idealnom slučaju, prilagodba V_O bi trebala biti dostupna korisniku tako da svaki korisnik može prilagoditi prikaz na optimalnu razinu kontrasta. Prilagođavanje kontrasta moguće je i pomoću mikrokontrolera koji ima PWM izlaz.

Koristi *LED* pozadinsko osvjetljenje koje je moguće podešavati pomoću priključaka A (LED +) i K (LED -). U idealnom slučaju za *LED* osvjetljenje bi se koristio strujni izvor no u praksi se pokazalo da je jednostavnije imati naponski izvor koji se spaja na priključak A i između njih se spaja otpornik kojeg je lako izračunati. Slika 2.4 prikazuje nam spajanje LED pozadinskog osvjetljenja.



Slika 2.4 Spajanje LED pozadinskog osvjetljenja

Računjanje R_{LIMIT} : $R_{LIMIT} = \frac{V_{DD}-V_{LED}}{I_D}$ (minimalna vrijednost otpora)

Računjanjem ovog izraza dobivamo da je R_{LIMIT} =48 Ω (I_D kod ovog *lcda* iznosi 25 mA).

Ovaj *lcd* prikaznik ima dvije vrste sučelja: 4-bitno i 8-bitno paralelno sučelje. Priključci za prijenos podataka su DB0 do DB7. Kod 4-bitnog paralelnog sučelja koriste se DB4 do DB7 i 8-bitni podatak šalje se u dva navrata prikazniku tako da se prvo pošalje viših četiri bita te kasnije nižih četiri bita. Tu su još priključci za omogućavanje skeniranja podatkovne sabirnice (E priključak), za biranje između podatkovnog registra za čitanje i pisanje i instrukcijskog registra služi priključak RS. Priključak za biranje između pisanja i čitanja sa *lcd* prikaznika omogućuje nam R/W.

Tablica 2.1 Pozicija *displaya* DDRAM adrese

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0xA	0xB	0xC	0xD	0xE	0xF
1	0x40	0x41	0x42	0x43	0x44	0x45	0x46	0x47	0x48	0x49	0x4A	0x4B	0x4C	0x4D	0x4E	0x4F

Opis osnovnih instrukcija za rad sa *lcd* prikaznikom:

Tablica 2.2 Brisanje prikaznika

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1

Tablica 2.3 Povratak pokazivača *lcda* na početnu adresu (X nije bitno dali je 1 ili 0)

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	X

Tablica 2.4 Pomicanje pokazivača u lijevo ili desno

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	I/D	S

I/D: - "1" pomakne se za jedan u desno i DDRAM adresa se poveča za jedan dok kod "0" se pomakne uljevo i DDRAM adresa se smanji za jedan

S: "1" pomak u desno ,"0" pomak u lijevo

Tablica 2.5 Uključenje *displaya*

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	1	D	C	B

D: "1" uključenje i "0" isključenje

C: "1" pokazivač vidljiv , "0" nevidljiv pokazivač

B: "1" pokazivač *blinka* "0" pokazivač ne *blinka*

Tablica 2.6 Funkcijski set

RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	1	DL	N	F	X	X

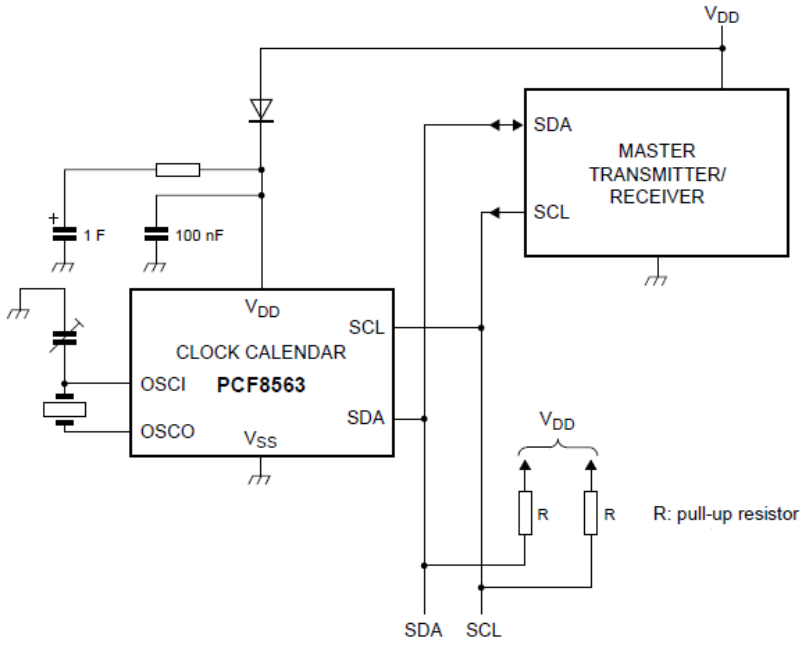
DL: "1" za 8-bitno sučelje i "0" za 4-bitno sučelje paralelne veze

N: "1" dvo-linijski prikaz, "0" jedno linijski prikaz

F: "1" 5x11 točkasti format znakova, "0" 5x8 točkasti format znakova

3. Real time clock (RTC) PCF8563

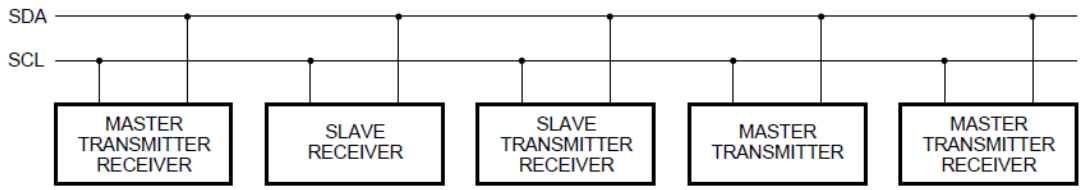
PCF8563 je CMOS *Real-time clock* i kalendar za veoma nisku potrošnju energije. *Real-time clock* je računalni sat koji prati stvarno vrijeme i dok sklop nije uključen na napajanje. Koristi 2-bitno serijsko sučelje za komunikaciju s mikrokontrolerom temeljeno na protokolu I2C. Maksimalna brzina sabirnice je 400 kbit/s. Sadrži šesnaest 8-bitnih registara s automatskim *increment* registrom adrese, na čipu je 32 768 Hz oscilator s jednim integriranim kondenzatorom, djelilo frekvencije koje daje izvorni sat za *Real-time clock* i kalendar, programabilni sat, *timer*, alarm, detektor malog napona i 400 kHz I2C sabirničko sučelje. Svi registri su dizajnirani kao adresabilni 8-bitni paralelni registri, iako svi bitovi nisu implementirani. Prva dva registra (memorijska adresa 00h i 01h) koriste se kao kontrolni i /ili status registri . Memorijske adrese od 02h do 08h se koriste kao brojila za sat (sekunde do godine). Adresna lokacija od 09h do 0Ch sadrži podatke o alarmu i uvjete kada se alarm treba oglasiti. Adresa 0Dh kontrolira frekvenciju CLKOUT izlaz. Registar *CLKOUT-control* može koristiti standardnu frekvencije od 32.768 kHz te 1.024 kHz, 32 Hz ili 1Hz za generiranje sustavnog sata, sata mikrokontrolera ili za kalibraciju oscilatora. 0Eh i 0Fh su *Timer_control* i *timer* registri. Slika 3.1 Prikaz spajanja RTC-a na I2C sabirnicu.



Slika 3.1 Spajanje PCF8563 pomoću I2C sabirnice sa ostalim elektroničkim uređajima

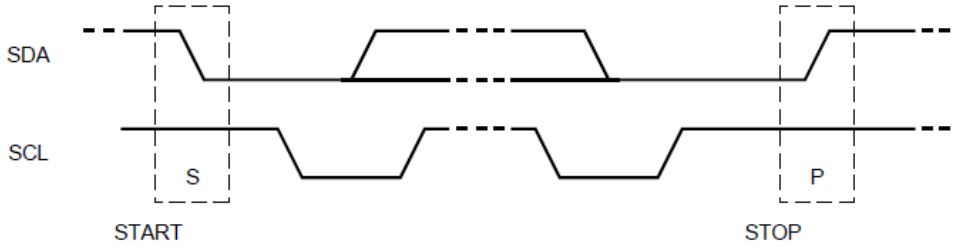
3.1. I2C protokol

I2C sabirnica je sinkrona serijska sabirnica namijenjena povezivanju raznih digitalnih i upravljačkih analognih sustava. Sva komunikacija obavlja se preko dvije linije: SDA(serial Data) kojom se prenose podaci i SCL (serial clock) kojim se prenosi takt. PCF8563 koristi I2C protokol za komunikaciju između jedinica. I2C protokol započinje upravljač (*master*) i tokom komunikacije generira takt. Prozvana jedinica zove se izvršilac (*slave*). Jedinica koja odašilje podatak zove se odašiljač (*transmitter*), ona koja prima podatke naziva se prijemnik (*reciver*). U komunikaciji može sudjelovati više od dvije jedinica, a postupak kojim se dodjeljuje sabirnica zove se arbitracija (*arbitration*).



Slika 3.2 Spajanje jedinica na I2C sabirnicu

Kada sabirnica ne izvršava nikakvu komunikaciju SDA i SCL su u stanju 1 pa je dogovoreno da I2C protokol započinje START bitom i završava STOP bitom. START bit je prelaz SDA iz stanja 1 u stanje 0 kada je SCL u stanju 1, a STOP bit je obrnuti prijelaz. Jedino kod START i STOP bita je dozvoljeno mijenjati SDA kada je SCL u stanju 1. Na slici 3.3 su prikazani START i STOP bitovi



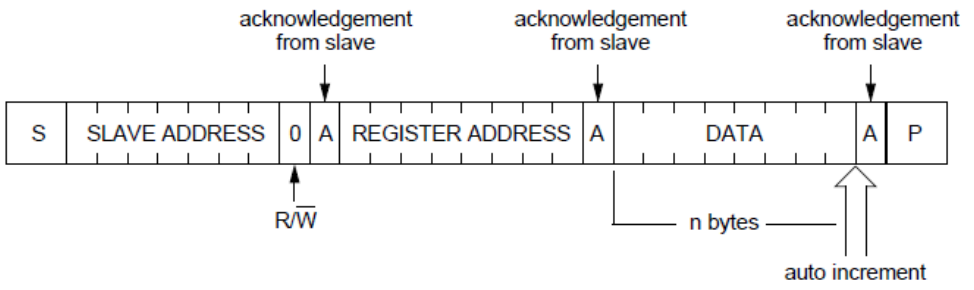
Slika 3.3 START i STOP bit

Broj podatak od 8 bita u komunikaciji je neograničen pa se nakon svakog prenesenog *bajta* treba poslati potvrda A=0 (*Acknowledge*) i to od strane *slavea*. Potvrda treba biti poslana preko SDA linije. SDA je u stanju 0 kada je takt SCL u stanju 1. Nakon svih prenesenih podataka *slave* postavlja A=1 i to je znak da je komunikacija završena te se poslije toga šalje STOP bit.

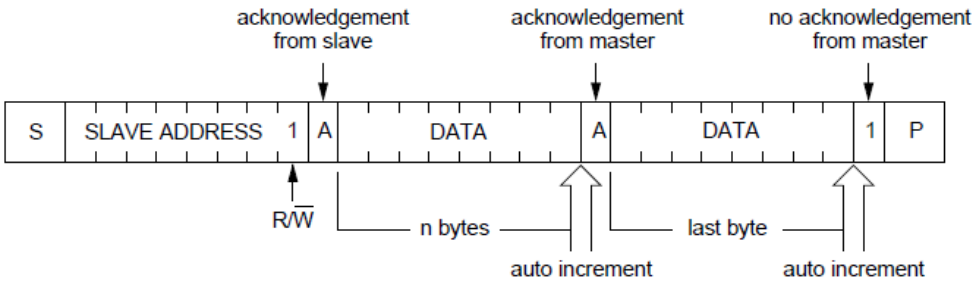
Prije prenošenja podataka treba se poslati adresa s koje se čita ili piše podatak. Adresa je 7 bitna i da bi se popunio bajt treba nam još jedan bit s kojim označavamo dali upisujemo ili čitamo podatke. PCF8563 se može ponašati kao izvršilac (*slave*) prijamnik (*reciver*) ili odašiljač (*transmitter*). SCL linija može samo primati tak, dok je SDA linija dvosmjerna.

Adrese PCF8563:

- za čitanje: A3h(10100011)
- za pisanje: A2h(10100010)



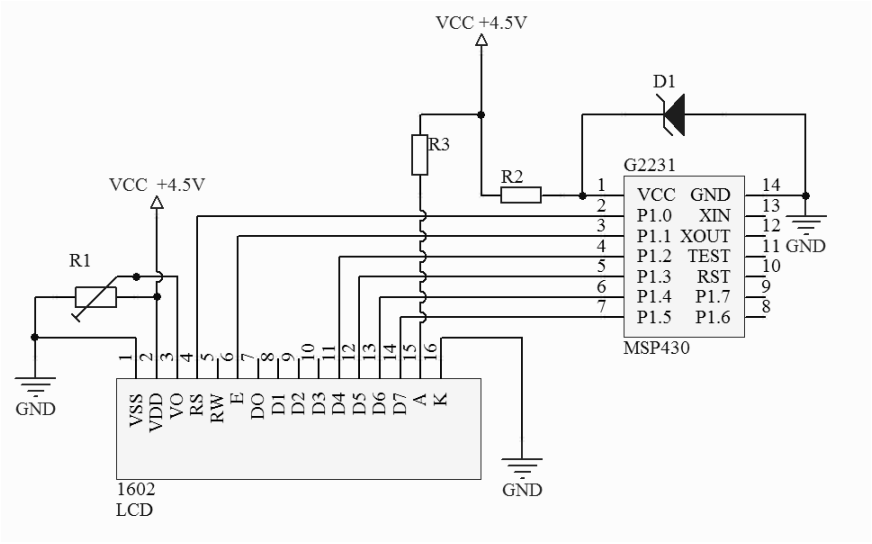
Slika 3.4 Pisanje u PCF8563



Slika 3.5 Čitanje sa PCF8563

4. Izvedba stvarnog sata bez RTC sklopa

Pomoću gore opisanih sklopovlja (mikrokontrolera i *lcd* prikaznika) sastavljen je sat stvarnog vremena. Za napajanje koristi 4.5 V bateriju. Glavnu zadaću, računanje vremena, obavlja mikrokontroler familije MSP430 i šalje podatke, odnosno iznos vremena na *lcd* prikaznik. Shematski prikaz spajanja sklopovlja nalazi se na slici 4.1.



Slika 4.1 Shema stvarnog sata bez RTC sklopa

4.1. Odabir komponenata

Kod ovog odabira komponenata prvenstveno se misli na otpornike i zener diodu. Potenciometar R1 služi za kontrast *lcd* prikaznika i treba uzeti iznos od 10 kΩ, a otpor R3 za napajanje pozadinskog osvjetljenja najmanje od 48 Ω. Otpotrnici R1 i R3 objašnjeni su već u 2.1 poglavlju pa će se ovdje samo spomenuti. Serija otpora R2 i zener diode D1 služi za smanjenje napona napajanja sa 4.5 V na 3.3 V. Zener dioda propušta napon napajanja do 3.3 V i služi kao stabilizator napona. Otpornik u seriji s diodom odvodi ostalih 1.2 V. Za izračun otpora otpornika potrebna nam je struja diode i struja mikrokontrolera. Struja diode iznosi 76 mA a mikrokontorler koristi maksimalno 4 mA kada rade obje LED diode pa smo po formuli ispod teksta (Ohmovom zakonu) dobili da otpor treba iznositi 15 Ω.

$$R_2 = \frac{V_{CC} - V_Z}{I_Z + I_{msp430max}}$$

4.2. Programska emulacija sata

Programiranje i *debugiranje* je rađeno u programu *Code Composer Studio v5*. *Code Composer Studio v5* je program kojeg je izdan od strane Texas Instruments. Cijelo programiranje je izvedeno u C/C++ programskom jeziku. Za prikaz sata najprikladnije je koristiti što manju frekvenciju signal takta za prekid u potprogram. Na MSP430G2231 je potrebno spojiti kristal kvarca s frekvencijom 32.768 kHz koji dolazi u pakovanju s *launchpadom*. Da bi se oscilator normalno radio potrebno je softwarski uključiti kondenzator. Možemo birati između 1 pF, 6 pF, 10 pF ili 12.5 pF te kod ovog slučaja kondenzator iznosi 10 pF.

```
BCSCTL3 |= XCAP_3;          //10 pF

CCTL0=CCIE;                 //omogući prekid

CCR0=4095                   //broji do 4096 i postavlja prekid

TACTL=TASSEL_1+ID_3+MC_1; //koristi ACLK(32kHz),djeli takt sa 8, broji
//prema gore
```

Timer_A je zadano da koristi ACLK odnosno kristal kvarca koji je podijeljen s 8 kako bi se dobilo što duže vremensko razdoblje. MC_1 je način rada Timer_A kada on broji prema gore i omogućili prekid u potprogram s upisivanjem CCIE u CCTL0 registar. Da zna do kud treba brojati i postaviti prekid u registar CCR0 smo upisali vrijednost 4095 što sve zajedno dijeli frekvenciju prekida na 1 Hz odnosno prekidni potprogram dešava se svake sekunde.

U prekidnom vektoru pozivamo funkciju **sat** kako bi povećali broj sekundi. U funkciji **sat** je napisano niz *if* naredbi s kojima se provjerava da li je došlo do prelijevanja (*overflow*). Prva *if* provjerava da li su sekunde došle do 59 ako jesu vrati na 0, druga naredba provjerava isto to samo za minute itd. Naravno naredbe su napisane tako da, ako se ne izvrši prva naredba u druge se i ne ulazi. Kod provjere dana u mjesecu provjerava se da li je godina prijestupna ili nije.

```
...
if(++t->tm_sec > 59) {            //povečaj sekundu i provjeri overflow
        t->tm_sec = 0;            //resetiraj sekunde
        if(++t->tm_min > 59){...   //povečaj minute i provjeri overflow
```

4.3. Programska emulacija *lcd* prikaznika

Da bi *lcd* uopće prikazivao znakove potrebno ga je prvo inicijalizirati. Prvo i osnovno je potrebno definirati koje paralelno sučelje ćemo odabrati. Kod ovog slučaja odabrali smo 4-bitno sučelje zbog toga što mikrokontroler ima samo 10 I/O priključaka od čega smo već dva potrošili na oscilator (port P2 priključci 6 i 7 se koriste kod ACLK načina rada Timer_A). Uz 4 priključka preko kojih se prenosi podatak trebamo još 2, priključak za RS koji selektira čitanje/pisanje ili instrukciju te E priključak koji javlja *lcdu* kad treba čitati instrukcije ili podatak. Za inicijalizaciju redom koristimo funkcije, koje su objašnjene u poglavlju 2.1 Vanjski priključci (11 str.), funkcijski set, uključenje displaya i pomicanje pokazivača lijevo desno. Poslije svake funkcije potrebno je uključiti i isključiti E priključak da *lcd* prihvati funkciju. Sve te funkcije potrebno odnosno niz od 8 bita potrebno je poslati na **posaljibajt** funkciju koja izravno komunicira sa *lcdom*. Najprije postavi sve pinove u 0. Tada 8 bitni podatak pomnoži sa 0xF0 kako bi poslao prvo najznačajnije bitove. Nakon množenja pomakne cijeli podatak za 2 bita u desno kako bi podatak pomaknuo na priključke

```
void posaljibajt(char Bajt, int pod){
        LCD_OUT &= (~LCD_MASK);         // svi pinovu u 0
        LCD_OUT |= (Bajt & 0xF0)>>2;    // gornji 4 bita na sabirnici
        if (pod == TRUE)                   //TRUE:ispis znakova
            LCD_OUT |= LCD_RS;              //RS u 1
        else
            LCD_OUT &= ~LCD_RS;              // RS u 0
        pulsiranje();                          //skeniranje
        LCD_OUT &= (~LCD_MASK);           // svi pinovi u 0
        LCD_OUT |= ((Bajt & 0x0F) << 2);  // donja 4 bita na sabinicu
        if (pod == TRUE)
            LCD_OUT |= LCD_RS;
        else
            LCD_OUT &= ~LCD_RS;
        pulsiranje();

    }
```

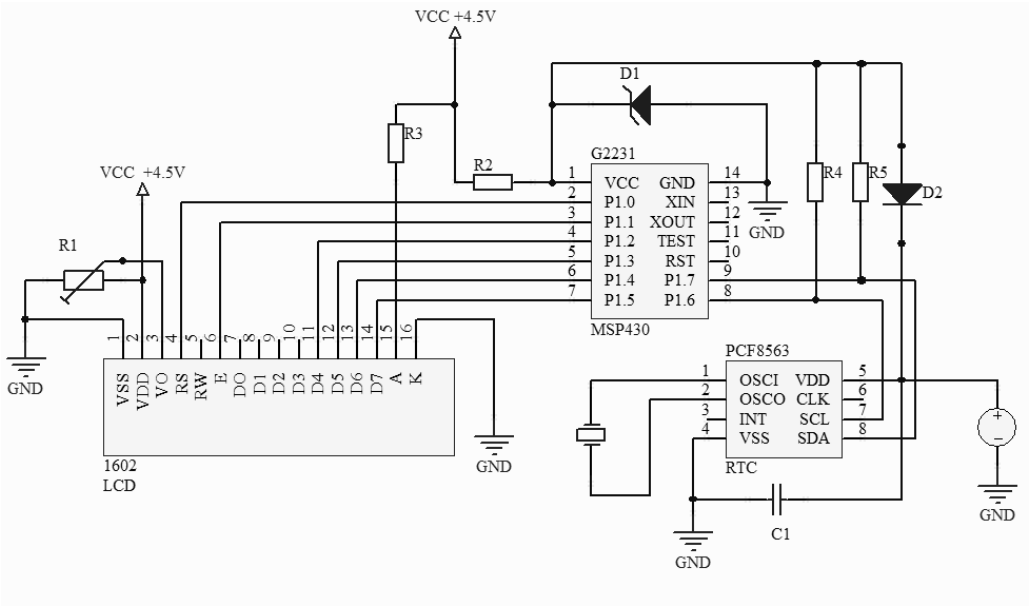
mikrokontrolera koje koristimo za prijenos (P.2 do P.5). Tad provjerava da li treba poslati instrukciju ili ispisati znakove. Nakon poslana 4 bita stavlja E u 1 (pulsiranje) *lcd* čita podatak. Završetkom pulsiranja postupak se ponavlja s niža 4 bita podataka.

Kod inicijalizacije *lcda* potrebno je pričekati oko 40 ms da se uključi što je mnogo više vremena od uključenja mikrokontrolera. Potreban nam je i funkcija koja briše sa *lcd* prikaznika znakove i to činimo tako da pošaljemo 0x01 i funkcija povratak pokazivača na početak slanjem 0x02 pomoću funkcije **posaljibajt**. Za proizvoljno kretanje po zaslonu *lcda* potrebna nam je funkcija **Inicijalizacija** s kojim upisivanjem 0 kao prvog argumenta ostajemo u prvom redu *lcd* zaslona ili bilo kojeg različitog broja za drugi red. Također je moguće birati i poziciju lijevo ili desno. Drugi argument funkcije može biti broj od 0 do 16 i tako da je 0 krajnji lijevi položaj a 16 krajnji desni položaj.

U Dodatku A se nalazi cijeli kod sa detaljnim komentarima.

5. Izvedba stvarnog sata s RTC PCF8563 sklopom

Stvarni sat sa sklopom RTC PCF8563 čuva vrijeme i dok nije spojen na napajanje(potrebno je samo napajanje RTC-a) . Napajanje se jednako spaja kao i kod verzije bez RTC sklopa. Na napon od mikrokontrolera spaja se RTC i ne narušava previše napon jer troši vrlo malo struje (par stotina mikro ampera).



Slika 5.1 Shema stvarnog sata sa RTC PCF8563

Periferne komponente R1, R2, R3 i D1 opisali smo u poglavlju 4.1 Odabir komponenata pa nećemo ovdje trošiti riječi. Dioda D2 propušta struju od napajanja MSP430 mikrokontrolera prema RTC PCF8563 ali ne i u suprotnom smjeru kad je napajanje odspojeno kako mikrokontroler nebi trošio struju baterije koja napaja RTC sklop. Uzeta je klasična dioda za takve primjene 1N4148. Na priključke 1 i 2 RTC sklopa spajamo kvarcni oscilator od 32 768 Hz kako bi imao stalni izvor takta. Otpornik C1 koristimo kao blokadni otpornik. Blokadni otpornik koristimo zato što vodići nisu idealni (imaju konačan otpor, induktivitet i kapacitet). CMOS digitalni sklopovi poput RTC-a u statičkim prilikama *ne disipiraju* snagu, ali troše struju izvora napajanje prilikom promjene logičkog stanja. Kratkotrajni pad napona napajanja na priključku RTC sklopa može dovesti do neispravnog rada, jer se ne može osigurati dovoljno naboja za promjenu logičkog stanja za vrijeme prijelazne pojave. Za blokadni kondenzator uzima se keramički od 100 nF. Otpornici R4 i R5 su pull up otpornici i oni služe

za dizanje I2C sabirnice u stanje logičke jedinice. Iznos otpornika za zaključenje sabirnice dan je izrazom

$$R = \frac{t_r}{C_{BUS}}$$

Gdje je t_r najveće dozvoljeno vrijeme porasta signala na sabirnicu, a C_{BUS} najveći dopušten kapacitet na pojedinim linijama sabirnice. Spomenuti parametri navedini su u tehničkim specifikacijama RTC PCF8563, a iznose t_r=1 μs kod standardne brzine te t_r=0.3 μs kod brzog načina i C_{BUS}=400 pF. Računjanjem ove formule dobivamo R=2.5 kΩ za standardnu i R=750 Ω za brzi način komunikacije. Ovdje korisitmo za *pull up* otpornike R=10 kΩ jer nam nije tolko bitna promjena stanja koliko smanjenje potrošnje energije. Sa većim otpornikom stuja je manja pa zbog toga i potrošnja energije manja, ali smo zbog toga dobili manje strmi prijelaz iz stanja 1 u stanje 0 i obrnuto.

Programska emulacija za lcd prikaznik jednaka je verziji bez RTC sklopa pa ju u ovom poglavlju nećemo detaljno razrađivati pa je zbog toga samo opisana programska emulacija I2C protokola.

5.1. Programska emulacija I2C protokola

Sam I2C protokol opisan je u poglavlju 3.1 I2C protokol te u ovoj programskoj emulaciji je identičan opis tog protokola samo u C/C++ programskom jeziku. Korisnik koristi I2C sučelje pomoću dvije funkcije: **CitajBajt** i **PisiBajt**.

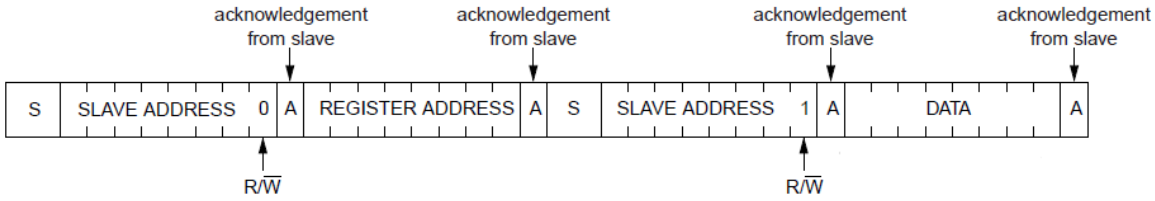
Funkcija **PisiBajt**

U ovoj funkciji opisani je protokol kojim upravljač odašiljač (*master transmitter*) upisuje podatak na adresu od izvršioca prijemnika (*slave reciver*). Komunikacija počinje slanjem START bita koji je također izveden od jedne funkcije **startbit** koja fizički spušta prvo SDA a nakon toga i SCL. Funkcija **PisiBajt** prima tri argumenta tipa *char*. Prvi argument je I2C adresa izvršioca i upisujemo 0x51, a ne 0xA2 kao što piše u poglavlju 3.1 odnosno upisujemo adresu koja je pomaknuta za jedan bit u desno. To radimo zato što kod funkcije **CitajBajt** i pišemo i čitamo iz RTC sklopa pa da ne trebamo dodatni argument u pozivu funkcije. Drugi argument je adresa u koju upisujemo podatak i treći je podatak koji upisujemo na tu prijašnju adresu. Svaki taj argument šalje se redom od prvog do trećeg pomoću funkcije slanje8 i to tako da kroz SDA liniju idu tri niza od 8 bitova. Između svakog se provjerava da li je *slave*

spreman primiti podatak pomoću funkcije **potvrda** koja provjerava da li je A u 0. Nakon što se sva tri argumenta pošalju šalje se **stopbit** koji završava komunikaciju.

Funkcija **CitajBajt**

Ova funkcija dosta je slična prijašnjoj funkciji **PisiBajt** samo što kod ove upisujemo samo dva argumenta tipa *char*, I2C adresu i adresu na kojoj se nalazi podatak. U opisu ove funkcije *master* je sada prvo *transmitter* pa kasnije *reciver*, a *slave* se ponaša obrnuto. MSP430 mikrokontroler započinje komunikaciju slanjem START bita te nakon toga se šalje I2C adresa 0x51 i adresa ovisno koji se podatak traži, npr. za sekunde je to adresa 0x02 pa do 0x08 za godinu. Nakon pozicioniranja *master* zaustavlja komunikaciju i opet je pokreče sa slanjem START bita. Za bolje shvaćanje prikazan je protokol funkcije **CitajBajt** na slici 5.2.



Slika 5.2 I2C protokol funkcije **CitajBajt**

Nakon START bita upisuje se ona ista adresa 0x51 pomaknuta za jedan bit u desno ali umjesto nule dodaje se jedinica koja je znak RTC sklopu da je *master* spreman za čitanje iz adrese koju su prethodno odredili (ako čitamo sekunde tad čita iz 0x02 adrese). Podatak se čita pomoću funkcije **citaj8** koja isto kao i kod slanja čita bit po bit i sprema u internu varijablu koju kasnije kod izlaska iz funkcije **CitajBajt** šalje kao rezultat funkcije. No prije samog izlazka postavlja A=1 što je znak da RTC prestaje slati podatke. Kako bi komunikacija završila šalje se STOP bit i izlazi iz funkcije.

Sve ove funkcije napisane su u Dodatku A

6. Analiza

U ovom radu opisane su dvije verzije sata implementirane pomoću MSP430G2231 mikrokontrolera pa će u ovom poglavlju biti napisane neke prednosti i mane obijuh verzija.

Za početak trebamo reći da ni jedna ni druga verzija nemaju nikakav *meni* s kojim bi se moglo eventualno podešavati vrijeme i datum jer nije bilo dovoljno memorije za implementaciju. *Meni* bi se mogao implementirati pomoću jedne tipke tako da dugi pritisak koristimo za promjenu funkcija a kratki za mijenjanje vremena (sekunde, minute, sate, dane, itd.). U tablici ispod dati su podaci mjerenja napona.

	Napajanje lcda	Napajanje MSP430	Napajanje RTC
Bez RTC-a	4.5 V	3 V	-
S RTC-om	4.5 V	3 V	1.7 V

Verzija sata bez RTC-a

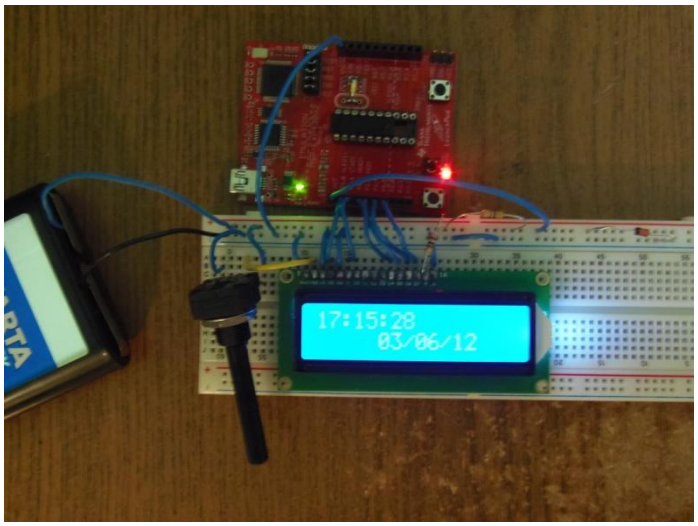
Nakon 4 sata rada verzije sata bez RTC sklopa napon baterije s 4.5 V je pao na 4.3 V što je zabrinjavajuće s obzirom da je to samo sat. Mada ne znamo kako bi se ponašao tokom nekog duljeg vremenskog razdoblja.

Verzija sata s RTC-om

Promatrajući sat s RTC sklopom ulijeva nadu. Smanjili smo potrošnju tako da uključujemo *lcd* i MSP430 samo kad želimo gledati na sat pa gotovo ne koristimo napajanje sata. Napon baterije RTC sklopa pao je sa 1.7 V na 1.68 V što je prihvatljivije u odnosnu na prvu verziju.

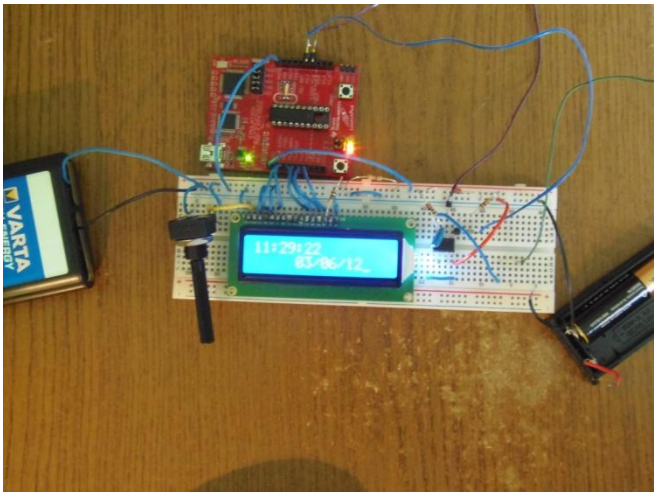
U nastavku slijede nekoliko slika vezanih uz promatranja satova.

Slika 6.1 Sat bez RTC sklopa



Slika 6.2 Sat bez RTC sklopa

Slika 6.3 Sat s RTC sklopom



Slika 6.4 Sat s RTC-om

Zaključak

U ovom radu razrađena je 4-bitna paralelna komunikacija sa lcd prikaznikom i I2C protokol komunikacije između mikrokontrolera i RTC sklopa te periferije.

Mikrokontroleri imaju veliku primjenu u današnjem svijetu i razvoj cijelog svijeta ovisi o njima. Kako bi se poboljšali potrebno je razviti što brži rad sklopa ili brži algoritam izvođenja programa, ali mislim da je najvažniji aspekt što manja potrošnja energije uz solidnu brzinu rada. U ovom radu pokazane su dvije verzije: jedna koja ne štedi potrošnju energije i druga koja štedi energiju.

Usprkos rezultatima mjerenja takvi satovi koji se mogu kupiti sa sličnim lcd prikaznicima nemaju baterijsko napajanje već koriste ispravljeni napon gradske mreže pa ni ove potrošnje koje se čine velikim možda i nisu toliko kritične.

Literatura

[1] *Character lcd module specifications*, Crystalfontz America Incorporated, 2008.

[2] *PCF8563Real-time clock/calendar*, NXP Semiconductors, 2011.

[3] John H. Davies, *MSP430 Microcontroller basics*, Burlington, Newnes, 2008.

[4] *MSP430x2xx Family User's Guide*, Texas Instruments Incorporated, 2008.

[5] *MSP430G2x31*, Texas Instruments Incorporated, 2011.

[6] Mladen Vučić, Upotreba mikrokontrolera u ugradbenim računalnim sustavima, Zagreb, 2007.

[7] Cache, *Interface MSP430 Launchpad with LCD Module (LCM) in 4 bit mode*, 27 lipanj 2011., http://cacheattack.blogspot.com/2011/06/quick-overview-on-interfacing-msp430.html , 15 travanj 2012.

[8] oPossum, *Software Real Time Clock (RTC) - two methods*, 22 ožujak 2012., http://www.43oh.com/forum/viewtopic.php?f=10&t=2477&p=18305&hilit=clock#p18305 , 20 travanj 2012

IZVEDBA SATA STVARNOG VREMENA NA MIKROKONTROLERU PORODICE MSP430

Sažetak

U ovom radu opisana je implementacija sata pomoću MSP430G2231 mikrokontrolera i lcd prikaznika. Razrađene su dvije verzije sata. U prvoj verziji razrađen je sat pomoću mikrokontrolera i lcd prikaznika. Mikrokontroler koristi 4-bitno paralelno sučelje za vezu sa lcdom. Nema mogućnost pamćenja vremena pri promjeni baterijskog napajanja. Druga verzija uz mikrokontroler i lcd koristi i RTC PCF8563 sklop koji ima funkciju računanja stvarnog vremena. RTC i mikrokontroler za komunikaciju koriste I2C protokol. RTC troši malo energije pa ima zasebno napajanje te sat može pamtiti vrijeme.

Ključne riječi: sat, MSP430G2231 mikrokontroler, lcd prikaznik, RTC PCF8563, 4-bitno paralelno sučelje, I2C

IMPLEMENTING A REAL-TIME CLOCK ON THE MSP430 MICROCONTROLLER

Abstract

This paper describes the implementation of the clock using a microcontroller MSP430G2231 and LCD display. Two version elaborated. The first version was developed using a microcontroller clock and LCD display. The microcontroller uses 4-bit parallel interface for connection to LCD. There is no ability to remember the time when changing the battery power. Second version of the microcontroller and LCD uses PCF8563 Real-time clock and calendar. For communication RTC and the microcontroller are using the I2C protocol. RTC spends low energy and has a separate power supply that makes clock remember time.

Keywords: clock, microcontroller MSP430G2231, LCD display, PCF8563 RTC, 4 bit parallel interface, I2C

Dodatak A

U ovom dodatku nalazi se cijeli programski kod izvedbe sata bez RTC sklopa

lcd.h

```
#include"msp430x20x2.h"
void pulsiranje(void);
void posaljibajt(char Bajt, int pod);
void pozicija(char redak, char stupac);
void Obrisi(void);
void Inicijalizacija(void);

#define LCD_DIR P1DIR
#define LCD_OUT P1OUT

#define LCD_RS BIT0 // P1.0
#define LCD_E BIT1 // P1.1
#define LCD_D7 BIT5 // P1.5
#define LCD_D6 BIT4 // P1.4
#define LCD_D5 BIT3 // P1.3
#define LCD_D4 BIT2 // P1.2



#define LCD_MASK ((LCD_RS | LCD_E | LCD_D7 | LCD_D6 | LCD_D5 | LCD_D4))

#define FALSE 0
#define TRUE 1

void pulsiranje(){                      //skeniranje podatkovne sabirnice
        LCD_OUT &= ~LCD_E;       //niska razina EN
        __delay_cycles(200);
        LCD_OUT |= LCD_E;        // viska razina EN
        __delay_cycles(200);
        LCD_OUT &= (~LCD_E);     //niska razina EN
        __delay_cycles(200);
    }
void posaljibajt(char Bajt, int pod){
        LCD_OUT &= (~LCD_MASK);        // svi pinovu u 0
        LCD_OUT |= (Bajt & 0xF0)>>2;   // gornji 4 bita na sabirnici
        if (pod == TRUE)                  //TRUE:ispis znakova
            LCD_OUT |= LCD_RS;             //RS u 1
        else
            LCD_OUT &= ~LCD_RS;              // RS u 0
        pulsiranje();                          //skeniranje
        LCD_OUT &= (~LCD_MASK);          // svi pinovi u 0
        LCD_OUT |= ((Bajt & 0x0F) << 2);    // donja 4 bita na sabinicu
        if (pod == TRUE)
            LCD_OUT |= LCD_RS;
        else
            LCD_OUT &= ~LCD_RS;
        pulsiranje();
    }
void pozicija(char redak, char stupac){ // redak: 0 za 1. red ostalo za
2.red, //stupac je za stupac pokazivača
```

```
        char address;
        if (redak == 0)                 //1. red  lcd-a
            address = 0;
        else
            address = 0x40;            //2. red
        address |= stupac;
        posaljibajt(0x80 | address, FALSE);
    }
void Obrisi(){
        posaljibajt(0x01, FALSE);          // brisi lcd
        posaljibajt(0x02, FALSE);          //pokacivac na nulu
    }
void Inicijalizacija(void){
        LCD_DIR |= LCD_MASK;
        LCD_OUT &= ~(LCD_MASK);
        __delay_cycles(100000);           //cekanje da se lcd uključi
        LCD_OUT &= ~LCD_RS;
        LCD_OUT &= ~LCD_E;
        LCD_OUT = 0x08;                 //4-bitno sučelje
        pulsiranje();
        posaljibajt(0x28, FALSE);             //4-bitno sučelje,2 linije
//displaya,5x8 format znakova
        posaljibajt(0x0C, FALSE);          //display on,pokazivač on
        posaljibajt(0x06, FALSE);  //automatsko zbrajanje,pomak u desno
//pokazivača
    }
```

sat.h

```
#include <time.h>
  void sat(struct tm *t);

static int prestupna(const int y) // provjera prijestupne godine
{
    if(y%4!=0)
       return 0;
    else return 1;
}

void sat(struct tm *t)
{
    static const signed char dum[13][2] = {              // dani u mjesecu
        0,      0,
        31,     31,                                      // siječanj
        28,     29,                                      // veljača
        31,     31,                                      // ožujak
        30,     30,                                      // travanj
        31,     31,                                      // svibanj
        30,     30,                                      // lipanj
        31,     31,                                      // srpanj
        31,     31,                                      // kolovoz
        30,     30,                                      // rujan
        31,     31,                                      // listopad
        30,     30,                                      // studeni
        31,     31                                       // prosinac
    };

    if(++t->tm_sec > 59) {    //povečaj sekundu i provjeri overflow
```

```
        t->tm_sec = 0;                                      //resetiraj sekunde
        if(++t->tm_min > 59) {       //povečaj minute i provjeri overflow
            t->tm_min = 0;                                  //resetiraj minute
            if(++t->tm_hour > 23) {                         //povečaj sat
                t->tm_hour = 0;                             //resetiraj

                if(++t->tm_mday > dum[t->tm_mon][prestupna(t->tm_year)]){
//povecaj dan i provjeri overflow
                    t->tm_mday = 1;
                    if(++t->tm_mon > 12) {
                        t->tm_mon = 1;
                        ++t->tm_year;
                    }
                }
            }
        }
    }
}
```

main.c

```
#include <time.h>
#include "sat.h"
#include "msp430x20x2.h"
#include "lcd.h"

#define TRUE 1

void prikazi_sat(const struct tm *t){//prikazivanje vremena i datuma na
//lcd-u
               pozicija(0,0);            //pozicija pokazivača lcd-a
               posaljibajt(t->tm_hour/10+48,TRUE);
               posaljibajt(t->tm_hour % 10+48,TRUE);
               posaljibajt(':',TRUE);
               posaljibajt(t->tm_min/10+48,TRUE);
               posaljibajt(t->tm_min % 10+48,TRUE);
               posaljibajt(':',TRUE);
               posaljibajt(t->tm_sec/10+48,TRUE);
               posaljibajt(t->tm_sec % 10+48,TRUE);

               pozicija(1,5);             // drugi red lcd-a
               posaljibajt(t->tm_mday/10+48,TRUE);
               posaljibajt(t->tm_mday % 10+48,TRUE);
               posaljibajt('/',TRUE);
               posaljibajt(t->tm_mon/10+48,TRUE);
               posaljibajt(t->tm_mon % 10+48,TRUE);
               posaljibajt('/',TRUE);
               posaljibajt(t->tm_year%100 / 10+48,TRUE);
               posaljibajt(t->tm_year % 10+48,TRUE);
      }
```

```
    struct tm ts;                              // struktura vremena

#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void) // prekid koji se dešava svake sekunde
{
        sat(&ts);                               //povecava vrijeme
        prikazi_sat(&ts);                       //prikazuje vrijeme
    __bic_SR_register_on_exit(LPM0_bits);//vrati se u low-power nacin rada
}

    void main(void){

            WDTCTL = WDTPW + WDTHOLD;        //zaustavi watchdog

            BCSCTL3 |= XCAP_3;               //10 pF
            CCTL0 = CCIE;                    //omogući prekid
            CCR0 = 4095;       //broji do 4096 i postavlja prekid
      TACTL = TASSEL_1 + ID_3 + MC_1;  //koristi ACLK(32kHz),djeli takt sa
//8,broji prem gore

            Inicijalizacija();                  //inicijalizacija lcda
            Obrisi();                           // obrisi lcd

                  ts.tm_hour = 17;                                  // sat
                  ts.tm_min = 15;                                   // minute
                  ts.tm_sec = 0;                                    // sekunde
                  ts.tm_mon = 6;                                    // mjesec
                  ts.tm_mday = 3;              // dan u mjesecu
                  ts.tm_year = 2012;                                // godina

__bis_SR_register(LPM0_bits + GIE); //low-power način rada i omoguci
prekid
}
```

Slijedeći kod je sat sa RTC sklopom i ovdje nećemo staviti kod lcd.h jer je identičan prijašnjem.

i2c.h

```
#include "msp430x20x2.h"
int SDA_vrijednost (void);
void startbit (void);
void slanje8 (unsigned char bajt );
unsigned char citaj8 (void);
char potvrda (void);
char CitajBajt(unsigned char i2c , unsigned char Adresa);
unsigned char PisiBajt(unsigned char i2c ,unsigned char Adresa,
unsigned char podatak);
void stopbit(void);

int SDA_vrijednost (void){
      int rez;
      rez=BIT7&P1IN;       //vrati vrijednost koju je SDA primio
      return rez;
}

void startbit (void)
```

```
{
      P1OUT|=BIT7+BIT6;          // postavi SDA i SCL u 1
      __delay_cycles (100);      //cekaj
       P1OUT^=BIT7;               //SDA = 0
      __delay_cycles (100);
      P1OUT^=BIT6;               //SCL =1
      __delay_cycles (100);
}

void slanje8 (unsigned char bajt){
      char j;
         for (j=8;j>0;j--){
            if (bajt&(1<<(j-1))){
                  P1OUT|=BIT7;          // SDA je bit u poslanom bajtu
            }
            else{
                  P1OUT&=0x7F;              //SDA = 0
            }
            __delay_cycles (100);
            P1OUT|=BIT6;                 //SCL = 1
            __delay_cycles (100);
            P1OUT^=BIT6;                 //SCL = 0
            __delay_cycles (100);

      }
      P1OUT|=BIT7;             // SDA = 1
}

unsigned char citaj8 (void){
      unsigned char j,podatak=0;
      P1DIR&=0x7F;                           // SDA je input
      for (j=0;j<8;j++)
      {
            P1OUT|=BIT6;                       //SCL = 1
            __delay_cycles (100);
            if (SDA_vrijednost()){  //upisi vrijednost u podatak ako je u
1
                  podatak=podatak+(0x80>>j);
            }
            P1OUT^=BIT6;                  //SCL = 0
            __delay_cycles (100);
      }
      P1DIR|=BIT7;                     //prebaci u output SDA
    return (podatak);
}
char potvrda (void){
      char s=0;
      P1DIR&=0x7F;                 // SDA je input
      P1OUT|=BIT6;                 //SCL = 1
      __delay_cycles (100);
      s=SDA_vrijednost();          //upisi vrijednost SDA u s
      P1OUT^=BIT6;                 //SCL =0
      __delay_cycles (100);
      P1DIR|=BIT7;                 //SDA je output
      return s;                    //vrati potvrdu
}

char CitajBajt(unsigned char i2c, unsigned char Adresa){
      char pod=0 ;
      startbit();                 // započni komunikaciju
      slanje8 ((i2c<<1)+0);       //7 bitna adresa + bit R
```

```
      if (potvrda())              // cekaj potvrdu
      {
            stopbit();              // ako je A = 1 prekini
            return (0xFF);          // komunikaciju
      }
      slanje8 (Adresa);          //adresa sa koje se cita
      if (potvrda()){
            stopbit();
            return (0xFF);
      }
      startbit();                // zapocni komunikaciju
      slanje8 ((i2c<<1)+1);      //7 bitna adresa + bit W
      if (potvrda()){            // cekaj potvrdu
            stopbit();
            return (0xFF);
      }

      pod=citaj8();              //citaj podatak
      P1OUT|=BIT7;               //postavi A u 1
      __delay_cycles (100);
      P1OUT|=BIT6;
      __delay_cycles (100);
      P1OUT^=BIT6;
      __delay_cycles (100);
      stopbit();                //zaustavi komunikaciju
      return (pod);
}
void stopbit(void){
      P1OUT&=0x7F;            //SDA u 0
      __delay_cycles (100);
      P1OUT|=BIT6;            //SCl u 1
      __delay_cycles (100);
      P1OUT|=BIT7;            //SDA u 1
      __delay_cycles (100);
}

unsigned char PisiBajt (unsigned char i2c, unsigned char Adresa, unsigned
char podatak){
      startbit();
      slanje8 ((i2c<<1)+0);            //7 bitna adresa + bit R
      if (potvrda()){                  //cekaj potvrdu
            stopbit();                   // ako je A = 1 prekini
            return (0xFF);
      }
      slanje8 (Adresa);                //adresa na koju se pise podatak
      if (potvrda()){
            stopbit();
            return (0xFF);
      }
      slanje8 (podatak);               //posalji podatak
      if (potvrda()){
            stopbit();
            return (0xFF);
      }
      stopbit();                      // zaustavi komunikaciju
      return 0;
}
```

main.c

```
#include <time.h>
#include "i2c.h"
#include "msp430x20x2.h"
#include "lcd.h"

#define TRUE 1
static unsigned bcd2bin (char n){     //pretvori iz BCD u binarni
     return ((((n >> 4) & 0x0F) * 10) + (n & 0x0F));
}
static unsigned char bin2bcd (unsigned int n){ //pretvori iz bniarnog u
BCD
      return (((n / 10) << 4) | (n % 10));
}

void prikazi_sat(const struct tm *t){ //prikazivanje vremena i datuma na
lcd-u

               pozicija(0,0);         //pozicija pokazivača lcd-a
               posaljibajt(t->tm_hour/10+48,TRUE);
               posaljibajt(t->tm_hour % 10+48,TRUE);
               posaljibajt(':',TRUE);
               posaljibajt(t->tm_min/10+48,TRUE);
               posaljibajt(t->tm_min % 10+48,TRUE);
               posaljibajt(':',TRUE);
               posaljibajt(t->tm_sec/10+48,TRUE);
               posaljibajt(t->tm_sec % 10+48,TRUE);

               pozicija(1,5);             // drugi red lcd-a
               posaljibajt(t->tm_mday/10+48,TRUE);
               posaljibajt(t->tm_mday % 10+48,TRUE);
               posaljibajt('/',TRUE);
               posaljibajt(t->tm_mon/10+48,TRUE);
               posaljibajt(t->tm_mon % 10+48,TRUE);
               posaljibajt('/',TRUE);
               posaljibajt(t->tm_year%100 / 10+48,TRUE);
               posaljibajt(t->tm_year % 10+48,TRUE);

    }


   struct tm ts;                   // struktura vremena
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void) // prekid koji se dešava svake sekunde
{
      char sec,min,sat,dan,mj,god;
                  sec=CitajBajt(0x51,0x02); //cita podatke sa RTCa
                  ts.tm_sec=bcd2bin(sec&0x7F);    // i pretvara ih
                  min=CitajBajt(0x51,0x03);
                  ts.tm_min=bcd2bin(min&0x7F);
                  sat=CitajBajt(0x51,0x04);
                  ts.tm_hour=bcd2bin(sat&0x3F);
                  dan=CitajBajt(0x51,0x05);
                  ts.tm_mday=bcd2bin(dan&0x3F);
                  mj=CitajBajt(0x51,0x07);
                  ts.tm_mon=bcd2bin(mj&0x1F);
                  god=CitajBajt(0x51,0x08);
                  ts.tm_year=bcd2bin(god);

        prikazi_sat(&ts);                  //prikazuje vrijeme
```

```
__bic_SR_register_on_exit(LPM0_bits);  //vrati se u low-power nacin rada
}

void main(void){

            WDTCTL = WDTPW + WDTHOLD;

            BCSCTL3 |= XCAP_3;                //10 pF
            CCTL0 = CCIE;                     //omogući prekid
            CCR0 = 4095;    //broji do 4096 i postavlja prekid
            TACTL = TASSEL_1 + ID_3 + MC_1;   //koristi ACLK(32kHz),djeli
//takt sa 8,broji prem gore
            Inicijalizacija();                //inicijalizacija lcda
            Obrisi();                         // obrisi lcd
            P1DIR|=BIT6+BIT7;
            char min,sat;
            int s,m,h,d,mje,godina;
            s=0;              //postavljanje sata, sekunde
            m=28;             // minute
            h=11;             // sati
            d=3;              // dan
            mje=6;            // mjesec
            godina=12;        // godina(RTC ima mogucnost samo od 0 do 99)
  min=CitajBajt(0x51,0x03);        // procitaj minute s RTCa
  ts.tm_min=bcd2bin(min&0x7F);      //pretvori u binarni
  sat=CitajBajt(0x51,0x04);         //citaj sate
  ts.tm_hour=bcd2bin(sat&0x3F);

     if(ts.tm_min==0){         //ako je pročitani podatak 0 minuta
        if(ts.tm_hour==0){                    // i 0 sati
      PisiBajt(0x51,0x02,bin2bcd(s));     // upisi nase vrijednosti
      PisiBajt(0x51,0x03,bin2bcd(m));
      PisiBajt(0x51,0x04,bin2bcd(h));
      PisiBajt(0x51,0x05,bin2bcd(d));
      PisiBajt(0x51,0x07,bin2bcd(mje));
      PisiBajt(0x51,0x08,bin2bcd(godina));
                  }
          }
__bis_SR_register(LPM0_bits + GIE);//low-power način rada i omoguci prekid
}
```