Univerzalan algoritam za rad sa EEPROM memorijom velikog memorijskog kapaciteta

Rajs Vladimir, Živorad Mihajlović, Vladimir Milosavljević, Goran Tanasić, Miloš Živanov
Departman za Elektroniku,Energetiku i Telekomunikacije
Fakultet Tehničkih Nauka
Novi Sad, Srbija
rajs.vladimir@gmail.com, zivorad@uns.ac.rs, schutki@gmail.com, goran.taske@gmail.com, zivanov@uns.ac.rs

***Sadržaj*—U ovom radu je prikazan univerzalan algoritam za rad sa EEPROM memorijom kapaciteta 16Mbit. Kako proizvođač memorije definiše koliko jedna memorijska lokacija može da ima pouzdanih ciklusa upis/čitanje, samim tim životni vek same memorije i uređaja koji sadrži ovu memoriju je ograničen. Algoritam koji je opisan u ovom radu omogućava da period pouzdanog rada memorije bude reda veličine oko 10 godina Pristup i upravljanje memorijom vrši mikorkontroler putem SPI komunikacije. Sam upis i čitanje podataka realizovani su periodično u zavisnosti od potrebe korisnika.**

Ključne riječi-EEPROM, memorija; algoritam; mikrokontoler;

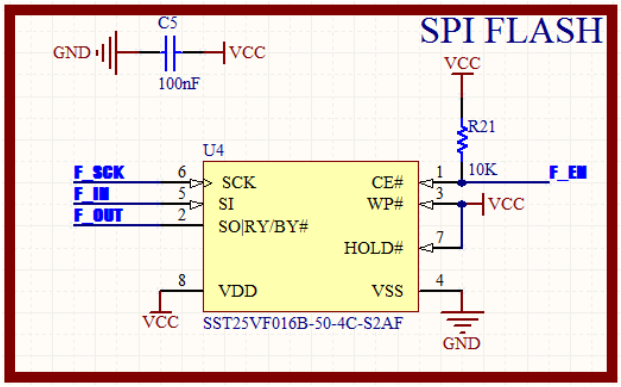
I. UVOD

Jedan od osnovih problema koji se javljaju prilikom upotrebe EEPROM memorije jeste njen životni vek. Svaka memorija ima minimalnu jedinicu u koju može biti upisan neki podatak. Veličina te jedinice se uglavnom kreće od jednog do nekoliko bajtova. Proizvođač memorije definiše koliko minimalna jedinica određene memorije može da ima pouzdanih ciklusa, odnosno koliko puta se može upisati i iščitati neki podatak na jednoj memorijskoj lokaciji. Prilikom konstantnog upisivanja nekog podatka na jednu lokaciju u vremenskom intervalu od jedan sekund, memorijska lokacija bi postala neupotrebljiva posle 27h kontinualog rada. Ovaj konstantacija bi važila za memoriju koja podržava 100000 ciklusa upisa po jednoj lokaciji. Ovakvom upotrebom memorije uređaj bi bio potpuno nepouzdan i jako kratkog životnog veka. Zato je potrebno naći najbolji način organizacije upisa i iščitavanja podataka kako bi smo produžili životni vek nekog uređaja na period od minimum 10 godina.

II. HARDVERSKA STRUKTURA

A. Mikrokontroler

Model mikrokontolera koji smo koristili prilikom izrade projekta je PIC24FJ128GA106[1]. Jedan od osnovnih razloga za izbor ovog modela je njegova količina RAM memorije koja iznosi 16384 bajtova. Velika količina RAM memorije je potrebna da bi kontroler mogao da vrši operacije sa nizovima podataka koji mogu dostizati dužinu čak i do 512 elemenata. Pored velike količine RAM memorije kontroler raspolaže sa 128KB FLASH memorije, četiri UART modula, tri SPI i I2C modula, kao i mnogim drugim ugrađenim periferijalnim modulima. Takođe jedna bitna osobina ovog mikrokontrolera je u tome što se SPI moduli mogu multipleksirati na velik broj pinova. Komunikacija između mikrokontrolera i EEPROM memorije je realizovana pomoću SPI protokola.

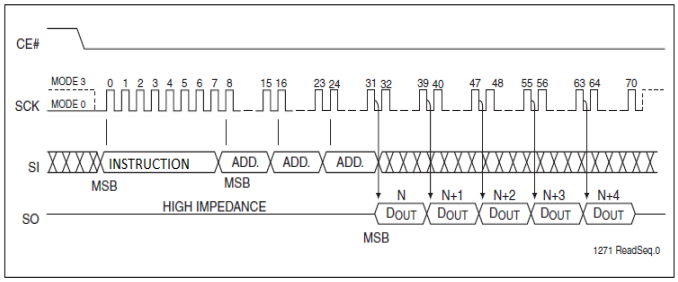


Slika 1 Električna šema EEPROM memorije sa kontrolnim signalima

Za komunikaciju putem SPI protokola mikrokontroler koristi 4 digitlana pina: SCK (serijski takt), SI (serijski ulaz), SO (serijski izlaz) i CE (selektovanje čipa). Na Sl. 1 je dat prikaz memorije sa kontolnim signalima.

B. EEPROM memorija

Model EEPROM memorije koji je korišćen za čuvanje podataka nosi oznaku SST25VF016B. Kapacitet ove memorije je 16Mbita. Proizvođač garantuje 100000 uspešnih ciklusa po jednoj memorijskoj lokaciji. Ova memorija se odlikuje jako dobrom organizacijom, brzinom upisa i iščitavanja, malom potrošnjom i malim dimenzijama kućišta. Adrese memorijskih lokacija se kreću od 000000-1FFFFF heksadecimalno, a veličina jedne lokacije je 1B (jedan bajt). Prilikom slanja jedne instrukcije od strane mikrokontrolera neophodno je prvo selektovati čip spuštanjem signala CE na logičku 0.

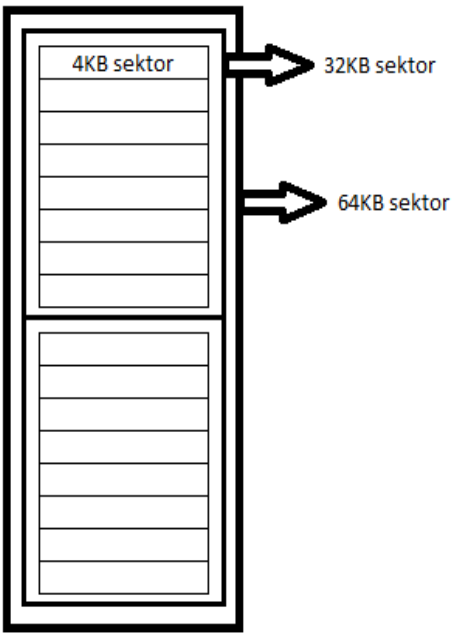


Slika 2 Dijagram kontrolnih signala prilikom izvršavanja naredbe

Zatim, putem signala SI neophodno je poslati naredbu za datu instrukciju i na kraju poslati adresu lokacije koja se sastoji od tri bajta (*high address, midle address, low address*). Na Sl. 2 dat je dijagram na kome su prikazani tokovi signala prilikom izvršavanja jedne instrukcije. Spisak sa intrukcijama je dat u uputstvu za rad sa memorijom [2].

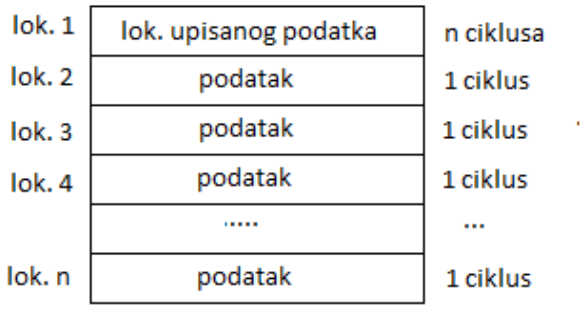
Memorija je organizovana po sektorima veličine 4KB, 32KB i 64KB . Sektori su hijerarhijski organizovani, odnosno manji sektori su sadržani u većim. Organizacija sektora memorije je prikazana na Sl. 3. Ovakva podela je jako korisna kada se koristi za brzo brisanje memorije, ali predstavlja problem prilikom rada zbog osobine da je sektor najmanja jedinica koja se može obrisati. To znači da prilikom brisanja ne možemo obrisati memorijsku lokaciju veličine 1 bajt nego ceo sektor od 4KB. Prilikom brisanja neke memorijske lokacije njoj se dodeljuje vrednost FF heksadecimalno.

Kako se u memoriju upisuju nizovi podataka u vremenskom intervalu koji se može menjati u zavisnosti od potrebe za čuvanjem padataka, potrebno je da u svakom trenutku imamo informaciju o tome na koju je memorjsku lokaciju upisan poslednji podatak. Ovu informaciju koristimo kako bi prilikom sledećeg startovanja mogli da nastavimo sa upisivanjem tamo gde smo stali. Ako jednu memorijsku lokaciju odvojimo samo za čuvanje informacije o adresi poslednjeg upisanog podatka onda bi u tom slucaju ta memorijska lokacija bila izložena daleko većem broju ciklusa upisa u odnosu na ostale što bi rezultovalo uništenjem te lokacije.Samim ti ovo rešenje ne bi bilo pouzdano.



Slika 3 Organizacija memorije po sektorima

Na Sl. 4 prikazan je ovakav pristup rešavanju problema.



Slika 4 Neadekvatan algoritam za upotrebu memorije prilikom rada sa velikim brojem ciklusa upisa

Vidimo da je prva memorijska lokacija koja je rezervisana samo za čuvanje informacije o adresi poslednjeg upisanog podatka pretpela n puta više ciklusa upisa u odnosu na druge lokacije

To bi značilo da je životni vek te lokacije n puta manji u odnosu na ostale što implicira i smanjenju životnog veka uređaja koji koristi ovu memoriju. Da bi se produžio životni vek memorije potrebno je pronaći odgovarajući način za čuvanje informacije o lokaciji poslednjeg upisanog podatka. Ovakav pristup ne sme da utiče na povećan broj ciklusa upisa na nekoj memorijskoj lokaciji. Ova zamisao se može postići ako se razvije odgovarajući algoritam pomoću kojeg bi mikrokontroler mogao da ustanovi gde je poslednji podatak sačuvan.

III. ALGORITAM ZA EFIKASAN RAD SA EEPROM I USLOVI KORIŠĆENJA

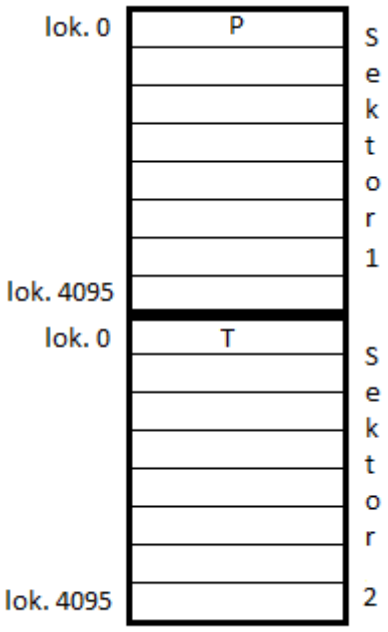
Princip ovog algoritma je zasnovan na činjenici da je memorija podeljena po sektorima različitih veličina od kojih je najmanji sektor ima veličinu od 4KB. Kako data memorija ima ukupni kapacitet od 16MBit to znači da imamo 512 sektora od po 4KB. Proračun je prikazan u jednačini (1)

$$\frac{Kapacitet\ memorije}{Kapacitet\ najmanjeg\ bloka} = \frac{16Mbit}{4KB} = 512 blokova \quad (1)$$

Pošto najmanja količina podataka koja može da se izbriše iznosi 4KB, glavna ideja je da se sektori od značaja obeleže na neki način. Na taj način bi mogli da imamo uvid u informacije koje bi označavale prazne sektore, popunjene sektore i sektore koji su otvoreni za upisivanje. Da bi smo označili neki sektor dovoljno je da upišemo jedan bajt određene vrednosti na prvu adresu sektora. Ako koristimo jedan bajt za označavanje sektora i imamo 512 sektora to znači da ćemo smanjiti kapacitet memorije za 512 B. Ovaj kompromis je prihvatljiv jer je gubitak memorijskog prostora svega 0.024% .

Ako je memorija potpuno prazna označićemo prvi sektor sa "P", a sledeći sa "T", tako što ćemo upisati ASCII vrednosti navedenih slova na prve memorijske lokacije u svakom sektoru. Sektor koji je označen sa slovom "P" nazivaćemo "prethodni" a sektor označen sa slovom "T" nazivaćemo "tekući". Prethodih sektora ne može biti više od jedan. Ovaj sektor će se naviše koristiti prilikom isčitavanja podataka iz memorije. Ako posmatramo memoriju koja sadrži od 0-511

sektora prethodni sektor će se uvek nalaziti ispred prvog tekućeg sektora.



Slika 5 Memorijske lokacije sa označenim sektorima

Tekućih sektora može biti maksimalno 510, a za naš algoritam od posebnog značaja će biti prvi tekući i poslednji tekući sektor. Sektori koji su potpuno prazni će prilikom brisanja biti označeni sa FF heksadecimalno. Označeni sektori su prikazani na Sl. 5

Osnovne funkcije algoritma su upisivanje podataka i čitanje podataka. Takođe postoji i funkcija za pretraživanje sektora koja nam obezbeđuje ključne informacije vezane za prethodne i tekuće sektore. Primenom ovog algoritma EEPROM memorija se može posmatrati kao kružni bafer. Na taj način postiže se ravnomerno upisivanje podataka kroz celu memorju, odnosno svaka lokacija će imati ravnomerni broj ciklusa i pri tome ćemo uvek imati informaciju o tome koji je podatak upisan prvi, a koji poslednji

1) Upisivanje podataka

Prilikom startovanja mikrokontroler pretražuje prethodne i tekuće sektore. Prilikom ove pretrage brojimo koliko imamo tekućih sektora. Ako ne pronađe nijedan prethodni i tekući sektor to znači da je memorija prazna i tada se prvi sektor inicijalizuje kao prethodni a drugi kao tekući sektor. Ako postoji tekući sektor, a nema prethodnog tada prvi prazan sektor, koji se nalazi pre prvog tekućeg koji smo pronašli, označavo kao prethodni.

Ako upisujemo niz od 32 bajta to znači da u jedan sektor možemo upisati 127 nizova. Umesto niza koristićemo termin "blok", a broj elemenata niza nazivaćemo "dužina bloka". Podrazumeva se da je svaki element niza dužine 1B. Određivanje broja blokova u jednom sektoru je prikazano u jednačini (2). Algoritam važi za bilo koju dužinu bloka manjeg od 4095

$$\frac{Kapacitet\ jednog\ sektora}{Dužina\ bloka} = \frac{4095B}{32B} = 127blokova \quad (2)$$

Podrazumevaćemo da nijedan blok za svoj prvi element ne sme imati vrednost "0", kao i da se u bloku ne sme naći niz od pet uzastopnih elemenata koji imaju vrednost "0".

Kada smo odredili koji je sektor prethodni i broj tekućih sektora možemo pronaći redni broj poslednjegi tekućeg sektora. Određivanje broja poslednjeg tekućeg sektora je prikazano u jednačini (3)

$$RP+UT = RTn \quad (3)$$

gde su:
RP- redni broj prethodnog sektora
UT- ukupan br.tekućih sektora
RT- redni broj poslednjeg tekućeg sektora

Poslednji tekući sektor predstavlja sektor koji je spreman za upisivanje podataka. Kada smo odredili redni broj poslednjeg tekućeg sektora potrebno je da proverimo koliko je blokova upisano u njega. Pošto znamo koliko blokova može da se nađe u jednom sektoru pretraživaćemo poslednji tekući sektor tako što ćemo proveravati samo memorijske lokacije na kojima može da se nađe prvi element nekog bloka. Ako se na pretraženoj lokaciji nalazi element koji različit od "0" to znači da imamo upisan blok u tom sektoru. Kada nađemo prvu lokaciju čija je vrednost "0" to znači da na toj lokaciji možemo početi sa upisivanjem novog bloka. Na ovaj način umesto 4095 lokacija pretražićemo samo 117 lokacija što znatno ubrzava ceo proces.

Pre upisa novog bloka treba proveriti da li je to poslednja lokacija u sektoru na koju neki blok može biti upisan. Ako to zaista jeste poslednja lokacija to znači da smo taj sektor napunili i da moramo preći na sledeći sektor. To ćemo uraditi tako što ćemo sledeći sektor proglasiti tekućim i povećati broj tekućih sektora. Prilikom sledećeg upisa mikrokontroler će za poslednji tekući sektor uzeti baš taj sektor koji smo upravo proglasili za tekući

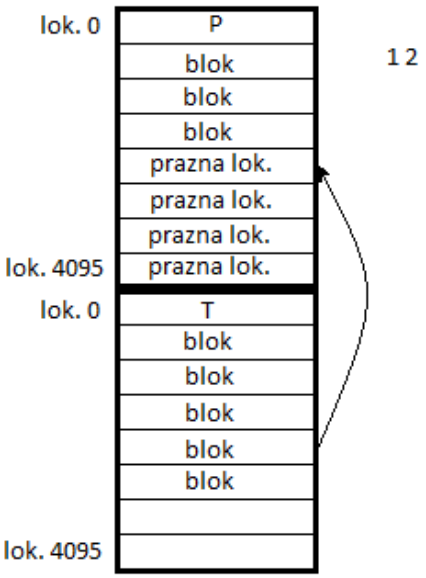
Kada se redni broj poslednjeg tekućeg sektora izjednači sa rednim brojem predhodnog sektora to znači da je memorija puna i tada se blokira svaki dalji upis sve dok se ne oslobodi barem jedan sektor u memoriji.

2) Čitanje podataka

Prilikom isčitavanja podataka od značaja su nam redni broj prethodnog sektora i redni broj prvog tekućeg sektora. Da bi smo došli do ovih informacija potrebno je samo da skeniramo prve lokacije u sektorima i na taj način nađemo redni broj predhodnog sektora. Kada smo našli redni broj prethodnog sektora onda možemo sa lakoćom izračunati redni broj prvog tekućeg sektora. Način pomoću kojeg određujemo taj broj prikazan je ujednačini (4)

$$RP+1=RT_1 \quad (4)$$

gde su:
RP- redni broj prethodnog sektora
UT- ukupan br.tekućih sektora
RT_1-redni broj prvog tekućeg sektora



Slika 6 Kopiranje bloka iz prvog tekućeg sektora na istu lokaciji u prethodni sektor

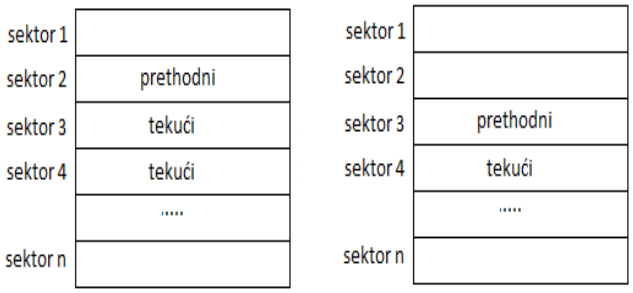
U prvom tekućem sektoru se nalaze blokovi koji su prvi upisani u memoriju i oni će prvi biti isčitani

To znači da će memorija raditi po principu FIFO bafera (first in first out). Prvo je potrebno odrediti koliko upisanih blokova se nalazi u prethodnom sektoru i zatim naći prvu praznu lokaciju na koju može da se upiše neki novi blok. Treba napomenuti da će svaki isčitan blok iz prvog tekućeg sektora biti upisan u prethodni sektor na indentičnu lokaciju.

Ovo se radi kako bi se uvek moglo odrediti koji blok treba da se isčita, a ujedno predstavlja mehanizam za oporavljanje od greške prilikom iznenadnog otkaza.

Kada je određen broj upisanih blokova u prethodnom sektoru, tada znamo da je prva sledeća prazna lokacija identična lokaciji bloka koji treba da se isčita iz prvog tekućeg sektora. Onda kopiramo blok iz prvog tekućeg sektora koji se nalazi na toj lokaciji u prethodni sektor, ali na tu istu lokaciju. Taj proces je prikazan na Sl. 6. Prilikom kopiranja mikrokontroler smešta blok u privremeni niz i tada može da raspolaže sa traženim blokom

Kada se prothodni sektor napuni do kraja to znači da je ceo prvi tekući sektor isčitan. Takođe postoji mogućnost da prvi tekući sektor nije u potpunosti popunjen, pa se može desiti da prepisujemo prazan blok u prethodni sektor. Zato se prilikom svakog isčitavanja bloka proverava da li u njemu postoji niz od pet uzastopnih "0". Ako postoji takv niz to znači su svi podaci isčitani iz memorije. Na taj način dobijamo informaciju o kraju isčitavanja i sprečavamo kopiranje praznih nizova u prethodni sektor. Kada je ceo prvi tekući sektor isčitan, pod uslovom da je bio skroz napunjen, tek tada možemo da oslobodimo memorijski prostor jer nam ti podaci više nisu potrebni. To se radi tako što obrišemo prvi tekući i predhodni sektor i zatim za prethodni označimo sektor koji je bio prvi tekući.



Slika 7 Stanje memorije pre i posle oslobađanja prostora

Na taj način dobili smo jedan potpuno prazan sektor i smanjili smo broj tekućih sektora za jedan. Na Sl. 7 je prikazano stanje memorije pre i posle oslobađanja prostora

IV. ZAKLJUČAK

Ovaj projekat je realizovan u okviru projektovanja uređaja za praćenje parametara životne sredine [3]-[5]. Pošto realizovani uređaji predstavljaju mobilne merne stanice neophodno je bilo poslati podatke koje sadrže trenutnu GPS poziciju i vrednosti parametara životne sredine. Podaci su se slali putem GPRS protokla, ali prilikom gubitka signala svi podaci su se morali skladistiti u EEPROM. Tek, nakon dobijanja signala podaci su se slali, čitanjem iz memorije.

Implementacijom opisanog algoritma u radu sa memorijomuspešno je produžen vek trajanja same mobilne stanice i otklonjen je problem koji nastaje prilikom gubitka signala.

ZAHVALNICA

Ovaj rad finansijski je podržan od strane Ministarstva zanauku Republike Srbije u okviru projekta pod brojem III43008 kao i Pokrajinskog sekretarijata za nauku i tehnološkirazvoj AP Vojvodine pod brojem 114-451-3266.

LITERATURA

[1] Uputstvo za rad sa mikrokontrolerom PIC24FJ128GA106.Dostupno na:http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en532133,, februar 2014

[2] Uputstvo za rad sa memorijom SST25VF016B. Dostupno na: https://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en548647, februar2014

[3] Mihajlović Živorad, Milosavljević Vladimir, Rajs Vladimir, Miloš, Živanov Miloš, "Application of GPRS Modules in Data Acquisition and Control of Devices for Air Quality Monitoring" Telekomunikacioni forum TELFOR 20 ; Beograd ; 2012,

[4] Tomić Josif, Rajs Vladimir, Milosavljević Vladimir, Mihajlović Živorad," Realizacija instrumenta za merenje parametara životne sredine" ETRAN (57 ; Zlatibor ; 2013).

[5] Mihajlović Živorad, Milosavljević Vladimir, Rajs Vladimir, Miloš, Živanov Miloš, "Remote Environmental Monitoring System for Application in Industry and Landfill Sites" Mediterranean Conference on Embedded Computing - MECO (2 ; Budva ; 2013) "

Universal algorithm for the EEPROM with large capacity

Rajs Vladimir, Živorad Mihajlović, Vladimir Milosavljević, Goran Tanasić, Miloš Živanov

Abstract-This paper presents a universal algorithm for work with EEPROM memory capacity of 16Mbit. Manufacturer of memory is defined as a memory location have a reliable cycle read / write. The lifetime of the memory and device which containing this memory is limited. The algorithm described in this paper allows the period of reliable operation of memory is on the order of 10 years. Access and management of memory is done using microcontroller via SPI communication. Read and write data are realized periodically depending on the needs of users.