Implementacija hardverske korekcije grešaka u NAND fleš memorijama

Vukašin Ristić, Mirjana Stojilović, Ivan Todorović
Institut Mihajlo Pupin
Univerzitet u Beogradu
Beograd, Srbija
vukasin.ristic@pupin.rs

Sadržaj — **U ovom radu opisana je upotreba NAND fleš memorije u namenskim računarskim sistemima, uključujući hardverski ubrzanu korekciju grešaka. Računarski sistem iz ovog rada realizovan je tako da koristi NAND fleš memoriju kao stalni skladišteni prostor sa koga se učitava operativni sistem sa Linux jezgrom. Dat je prikaz prednosti i mana upotrebe ovakve fleš memorije. Prikazana je struktura NAND memorija kao i njen hardverski i softverski inferfejs. Opisan je način pristupa memorijskom čipu, razvoj softverske podrške u vidu upravljačkih programa, kako u jezgru operativnog sistema, tako i u softveru za pokretanje operativnog sistema. (Abstract)**

Ključne reči - NAND; memorija; embedded; linux; u-boot; ecc; hamming; DM8168 (key words)

I. UVOD

NAND poluprovodničke memorije danas predstavljaju jedan od najpopularnijih tipova memorija za dugoročno čuvanje podataka. Sastavni su deo memorijskih sistema poput memorijskih kartica, USB fleš memorija i SSD diskova. Ipak, ove memorije najveću primenu imaju u namenskim računarima i mobilnim uređajima zahvaljujući svojoj mehaničkoj robusnosti i malim dimenzijama. Kada se koriste u svom osnovnom obliku, bez naprednijih kontrolera koji se brinu o detekciji grešaka, računanju korekcionih kodova, zameni loših stranica i podjednakom korišćenju dostupnog kapaciteta, ove memorije pružaju najveću slobodu pri razvoju novih uređaja, ali obavezuju projektante na samostalno rešavanje pomenutih problema.

U ovom radu opisana je integracija jedne ovakve NAND memorije u računar specijalne namene koji radi pod operativnim sistemom sa Linux jezgrom. U drugom poglavlju prikazane su ključne karakteristike tog računara. Treće poglavlje govori o konkretnom primerku NAND memorije ugrađenom u računar. U četvrtom poglavlju opisan je asinhroni režim rada sa NAND memorijom, iskorišćen na računaru. Peto poglavlje govori o komunikaciji sa NAND čipom koristeći dostupan skup komandi, dok je u šestom obrađen memorijski kontroler računara koji je povezan na NAND. U sedmom poglavlju opisana je podrška za implementirani NAND čip sa stanovišta operativnog sistema, dok osmo poglavlje sumira izloženu materiju i navodi informacije od ključnog značaja pri razvijanju jednog ovakvog računarskog sistema sa NAND-om.

II. REALIZACIJA RAČUNARA NA QSEVEN MODULU

Računar specijalne namene na kome je integrisana jedna ovakva memorija je computer-on-module sistem napravljen po specifikaciji i formatu Qseven® i sa sistemom na čipu Texas Instruments DaVinci DM8168 [1]. Računar je projektovan i proizveden u Institutu Mihajlo Pupin [2] (Sl. 1). Ovaj modul se instalira u standardizovano Qseven podnožje na nekoj od komercijalno dostupnih nosećih ploča sa fizičkim konektorima i eventualnim dodatnim periferijama, zavisno od konkretne primene. Tako su fizičke periferije odvojene od samog računara i ne moraju se obavezno implementirati na svakoj specijalizovanoj platformi. Na računarskom modulu ostaju DM8168, memorije i dodatni kontroleri i prateća logika čije su linije izvedene na Qseven konektor. Sistem na čipu DM8168 sadrži procesorsko jezgro arhitekture ARM Cortex-A8, DSP jedinicu TMS320C674x (floating-point VLIW), OpenGL-kompatibilni 3D grafički akcelerator SGX530, HDVPSS jedinicu za hardversku obradu videa, PCI Express port sa dve linije, SATA kontroler, USB 2.0, Ethernet 1GB/s i drugo. Da bi ovakav sistem mogao da funkcioniše bez potrebe za dodatnim spoljnim memorijama poput hard diskova i memorijskih kartica, potreban mu je skladišteni prostor na samoj ploči koji treba da bude dovoljno velik za kompletan operativni sistem baziran na Linuxu, uključujući upravljačke programe, sistemske biblioteke i korisničke programe. Odabran je NAND fleš čip kapaciteta dva gigabajta koji u sebi sadrži bazični interfejs za prenos komandi i podataka.

Slika 1. Računar EPP-Q7-DM8168 na Qseven modulu razvijen u Institutu Mihajlo Pupin.

III. MICRON NAND MEMORIJSKI ČIP

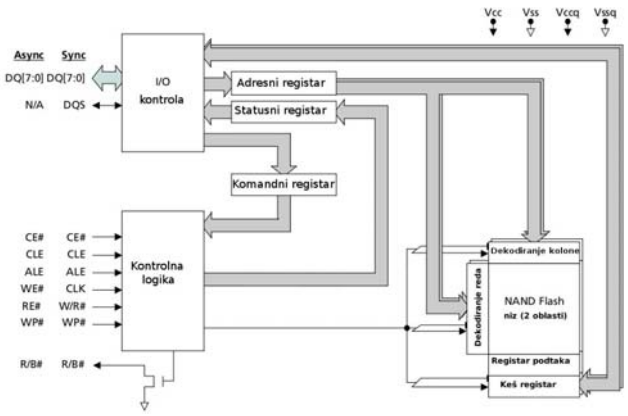
U računar na Qseven modulu ugrađen je NAND čip Micron MT29F16G08ABACA [3] kapaciteta 2GB (16 gigabita), u 48-pinskom TSOP pakovanju. Čip nudi mogućnost rada u sinhronom i asinhronom režimu. Kako bi postigli najviše performanse pri prenosu podataka, korišćen je asinhroni režim.

Memorijski NAND čip poseduje visoko multipleksiranu 8-bitnu magistralu za podatke, komande i adrese. Asinhroni interfejs čini pet kontrolnih linija: CE# (Chip Enable), CLE (Command Latch Enable), ALE (Address Latch Enable), WE# (Write Enable) i RE# (Read Enable). Dodatno, čipovi većeg kapaciteta interno su izvedeni kao skup memorijskih uređaja sa više linija za izbor radne oblasti unutar čipa (*chip enable*). Svaki od ovako odabranih uređaja je nezavisan, a ako je potrebno obezbediti da se više ovakvih uređaja tretira kao jedna velika memorija sa kontinualnim lokacijama, tu funkcionalnost mora obezbediti upravljački program operativnog sistema. Po ONFI standardu [4] za NAND memorije, jedan ovakav nezavisan uređaj na čipu naziva se logička jedinica (eng. *Logical Unit*, LUN). LUN uređaj u čipu je minimalna jedinica koja može nezavisno izvršavati komande i prijavljivati svoj status.

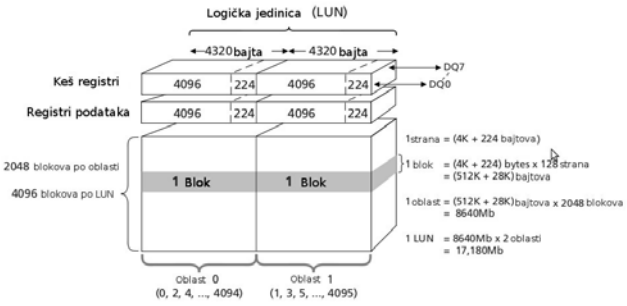
A. Arhitektura logičke jedinice (LUN)

Procesor DM8168 ima interni memorijski kontroler opšte namene (General Purpose Memory Controller) na koji se može povezati NAND čip sa osmobitnom ili šesnaestobitnom magistralom za podatke. Podaci, komande i adrese se multipleksiraju na istim priključcima i dolaze na ulazno-izlazno kontrolno kolo. Komande koje se prime preko ulazno-izlaznog kontrolnog kola, čuvaju se u komandnom registru i prenose do kontrolnog logičkog kola koje generiše interne signale koji upravljaju radom memorijskog uređaja. Adrese se skladište u adresnom registru i šalju dalje na dekoder redova ili kolona kako bi se izabrala adresa kolone (Sl. 2). Podaci se šalju od ili do NAND fleš memorijskog niza, bajt po bajt kroz registar podataka ili keš registar.

NAND fleš memorojski niz je programabilan i čitljiv na nivou stranica, dok se briše na nivou blokova (koji su sastavljeni od većeg broja stranica, reda veličine nekoliko stotina). Za vreme normalnih operacija sa stranicama, registar za podatke i keš registar se ponašaju kao jedan registar.



Slika 2. Prikaz toka podataka unutar NAND memorije.



Slika 3. Struktura NAND memorije.

Za vreme keš operacija, registar podataka i keš registar rade nezavisno da bi povećali propusnost podataka. Statusni registar javlja status operacije logičke jedinice podataka.

B. Organizacija memorije

Jedna logička jednica podataka (LUN) podeljena je na dve oblasti. Svaka od oblasti je dalje podeljena na blokove, koji se sastoje od stranica. Blok (eng. *eraseblock*) je minimalna jedinica podataka koja može biti prebrisana (Sl. 3). Jedna stranica unutar bloka kod konkretnog Micron čipa sadrži 4096 bajtova namenjenih za korisničke podatke i dodatnih 224 bajtova za metapodatke (eng. *out of bounds area*, *OOB*).

Prostor za metapodatke je prvenstveno namenjen za čuvanje indikacije ispravnosti stranice i čuvanje koda za ispravljanje grešaka. Prva dva bajta u ovom prostoru su najbitnija jer označavaju status stranice. Ako je sadržaj prva dva bajta ispunjen logičkim jedinicama, to znači da je stranica ispravna. Upravljački program koji se brine o proveri integriteta zapisa može koristiti proizvoljnu dužinu preostalog prostora, zavisno od tipa koda za ispravljanje grešaka.

Ako se koristi OOB oblast za kontrolu integriteta sadržaja stranice, od ključnog je značaja da se prilikom čitanja i upisa koristi identičan algoritam za kontrolu grešaka. Ako bi se upisao podatak koristeći jednu ECC šemu, a pročitao koristeći neku drugu, upravljački program na računaru tretiraće podatke kao neispravne, a stranica će biti obeležena kao oštećena.

IV. ASINHRONI INTERFEJS

Asinhroni interfejs je po automatizmu aktivan kada se NAND čip priključi na napajanje. Ulazno-izlazna magistrala (pinovi DQ[7:0]) je deljena multipleksiranjem između I/O linija za podatke, adresa i komandi (Sl. 2). DQS signal koji je deo sinhronog interfejsa, ako postoji, u stanju je visoke impedanse kada je aktivan asinhroni interfejs.

A. Asinhroni mirni režim

Chip enable (CE#) se koristi da bi se uključio ili isključio uređaj. Kada je CE# postavljen na nizak nivo, svi signali za uređaj su uključeni. Sa CE# podešenim na nizak nivo, uređaj može primati komande, adrese i podatke. Može biti više od jednog logičkog uređaja (LUN) u jednom NAND fleš pakovanju, a svaki takav uređaj se kontroliše preko sopstvene *chip enable* linije. Prvi takav uređaj se aktivira preko pina CE#, a drugi, ako postoji, preko CE2# i tako dalje. Uređaj je

neaktivan kada je CE# postavljen na visok nivo, čak i ako je uređaj zauzet.

Kada je uređaj neaktivan, svi njegovi signali su isključeni osim CE#, WP# i R/B#. Sve dok je uređaj neaktivan, ostali uređaji mogu koristiti isključene NAND signale koje dele sa njim. LUN je tako u režimu male potrošnje, kada je onemogućen a nije zauzet. Ako je pak zauzet za vreme dok je onemogućen, on će ući u režim manje potrošnje kada svi njegovi delovi (logički blokovi) završe svoj posao. Režim manje potrošnje pomaže smanjenju ukupne potrošnje energije na računaru.

B. Asinhrona magistrala u besposlenom režimu

Magistrala uređaja je besposlena kada je CE# podešen na nizak logički nivo, a WE# i RE# su na visokom. Dok je magistrala besposlena, svi signali su uključeni osim DQS, koji se ne koristi za vreme asinhronog moda rada. U ovom režimu neće doći do skladištenja komandi, adresa i podataka na uređaju, a takođe neće doći ni do slanja podataka od strane uređaja.

C. Asinhrone komande

Asinhrona komanda se upisuje u komandni registar posredstvom magistrale DQ[7:0] na uzlaznu ivicu signala WE#, kada su CE# i ALE na niskom logičkom nivou, a CLE i RE# na visokom. Komande se obično ignorišu kod logičkih blokova koji su zauzeti (RDY=0). Ipak, ima komandi kao što su READ STATUS (70h) i READ STATUS ENHANCED (78h) koje se prihvataju od strane logičkih blokova, čak i ako su zauzeti.

D. Asinhrone adrese

Asinhrone adrese se upisuju u adresni registar posredstvom magistrale DQ[7:0] na uzlaznu ivicu WE# kada su CE# i CLE na niskom logičkom nivou, a ALE i RE# na visokom.

Bitovi koji nisu deo adresnog prostora moraju biti postavljeni na nizak nivo. Broj ciklusa za svaku komandu varira. Za svaku komandu treba pogledati detaljan opis komande, kako bi se odredili zahtevi za adresiranje.

Adresa se obično ignoriše od strane logičkog bloka koji je zauzet (RDY=0). Ipak, neke adrese se prihvataju čak i ako je blok zauzet — primera radi, adresni ciklus koji prati komandu READ STATUS ENHANCED (78h).

E. Asinhroni ulaz podataka

Podaci se upisuju preko DQ[7:0] linija do keš registra izabranog logičkog bloka na uzlaznu ivicu signala WE#, dok su CE#, ALE i CLE na niskom logičkom nivou i RE# na visokom. Ulazni podaci se ignorišu od strane logičke jedinice podataka ako nije izabrana ili ako je zauzeta (RDY=0).

F. Asinhroni izlaz podataka

Podaci mogu biti poslati iz logičke jedinice podataka ako je ona u spremnom stanju (READY). Izlaz podataka je podržan nakon READ operacije iz NAND fleš niza. Podaci se šalju iz keš registra izabrane logičke jedinice podataka, na magistralu DQ[7:0] na silaznu ivicu RE# signala, kada je su CE#, ALE i CLE na niskom logičkom nivou, WE# na visiokom.

Ako kontroler koristi vremensku konstantu (t_{RC}) od 30 ns ili više, host može prihvatiti podatke na uzlaznu ivicu signala RE#. Ako kontroler koristi t_{RC} sa manje od 30 ns, host može prihvatiti podatke na sledeću opadajuću ivicu RE# signala.

Zahtevi za slanje podataka obično se ignorišu od strane logičke jedinice koja je zauzeta (RDY=0). Ipak, moguće je pročitati podatke iz statusnog registra čak i kada je logička jedinica podataka zauzeta, ako je poslata komanda READ STATUS (70h) ili READ STATUS ENHANCED (78h).

G. Signal za zabranu reprogramiranja

Dodatni signal WP# (*write protect*) dozvoljava ili onemogućava operacije programiranja (PROGRAM) i brisanja (ERASE). Kada je WP# na niskom logičkom nivou, programiranje i brisanje su onemogućeni. Kada je WP# na visokom nivou, ove operacije su dozvoljene.

Preporučuje se da WP# bude na niskom nivou tokom podizanja sistema sve dok Vcc i Vccq ne dostignu stabilne vrednosti, kako bi se sprečilo nekontrolisano reprogramiranje. Signal WP# može menjati stanje samo kada logička jedinica nije zauzeta i priprema se za komandnu sekvencu. WP# signal je uvek aktivan ulaz, čak i kada je CE# na visokom nivou, pa bi trebalo izbeći multipleksiranje sa ostalim signalima.

H. Signal za detekciju spremnosti uređaja

Ready/Busy (R/B#) signal nudi hardversku metodu za ispitivanje da li je uređaj zauzet ili spreman. Uređaj je zauzet kada je jedna ili više logičkih jedinica zauzeto (RDY=0). Uređaj je spreman kada su sve logičke jedinice spremne (RDY=1). Kako svaka logička jedinica sadrži statusni registar, moguće je nezavisno odrediti status svake logičke jedinice, poliranjem njihovih statusnih registara umesto korišćenja R/B# signala.

V. SLANJE KOMANDI

Nakon razumevanja fizičkog sloja za slanje komandi, adresa i podataka, jednostavno je implementirati slanje konkretnih komandi uređaju. Svakako, neke komande zahtevaju više a neke manje ciklusa na magistrali. Najbitnije komande su one koje se odnose na čitanje i programiranje stranica, kao i one za brisanje blokova. Ostale komande služe za resetovanje, čitanje statusa, postavljanje i proveru konfiguracije kao i pozicioniranje po adresama.

Ne postoji standardni skup dodatnih komandi, pa one mogu varirati od čipa do čipa, bez obzira na pokušaje standardizacije NAND memorija, kao što je ONFI konzorcijum.

Slanje komande se uglavnom sastoji iz slanja komandne instrukcije, željene adrese i podataka, odnosno njihovog iščekivanja. Postoje komande koje ne očekuju nikakakve adrese niti podatke, poput komande RESET. Postoji širok opseg komandi koje se koriste i svaka od njih je specifična. Zbog toga je prilikom razvoja upravljačkog programa potrebno izdvojiti posebno vreme za svaku komandu. Ipak, nije uvek potrebno razviti softverski interfejs za svaku komandu – mnoge

se uopšte ne moraju koristiti u uobičajenom radu. Tako u ovom slučaju nisu razvijene komande koje se odnose na sinhroni prenos, jer je čip povezan tako da se koristi isključivo u asinhronom režimu.

VI. Povezivanje memorije u sistem

Glavni čip na ploči (DM8168) povezuje se sa NAND memorijom posredstvom memorijskog kontrolera opšte namene (General-Purpose Memory Controller) koji se nalazi u samom čipu, a takođe služi i za povezivanje sa memorijama poput DDR3 SDRAM, NOR itd. Prednost upotrebe ovog kontrolera leži u uniformnom pristupu različitim tipovima memorija.

GPMC poseduje jedinicu za računanje ECC (eng. *Error Code Correction*) "u letu", za vreme samih operacija čitanja ili programiranja (upisa) podataka. Ova jedinica sadrži odgovarajuće registre za ovu namenu i podržava računanje nad blokovima podataka od 512 bajtova, što znači da se u konkretnom slučaju računjanje ECC-a za stranicu od 4 kB mora vršiti u osam koraka. Korisnici mogu birati jedan od dva podržana algoritma različitih mogućnosti.

1) "Hamming" kod za jednobitnu ispravku grešaka na 8-bitnim i 16-bitnim NAND Flaš memorijama organizovanim u stranama većim od 512 bajtova.
2) BCH (Bose-Chaudhuri-Hocquenghem) kod za robusniju 4-bitnu, 8-bitnu ili 16-bitnu ispravku grešaka.

GMPC ne rukuje direktno kodom za ispravljanje grešaka. Na upravljačkom programu leži odgovornost što se tiče čitanja rezultata ECC računanja, poređenja sa očekivanom vrednošću i preduzimanja odgovarajuće akcije otklanjanja grešaka u skladu sa korišćenom šemom. Tokom upisa, GPMC računa bitove parnosti, a tokom čitanja daje dodatne infornacije procesoru da bi ispravio greške bez ponovnog čitanja bafera podataka.

Hamming kod šema zasnovana je na dvodimenzionalnoj (red - kolona) akumulaciji bita parnosti. Ta akumulacija postignuta je na programiranom broju bajtova ili 16-bitnim rečima pročitanim sa memorijskog uređaja, odnosno zapisanim na memorijski uređaj u režimu toka (*stream*). Pošto ECC jedinica može da podrži samo jednu akumulaciju sadržaja, moguće je računanje greške samo jedne odabrane memorijske jedinice istovremeno.

Hamming kod ECC šema može ispraviti grešku na jednom bitu na svakih 2048 bita (256 bajtova) koristeći 12 bita ECC koda, ili grešku na jednom bitu na svakih 4096 bita (512 bajtova) koristeći 24 bita ECC koda.

S obzirom da je proizvodnja NAND čipa koji nema stoprocentno ispravne stranice daleko jeftinija nego proizvodnja čipa koji je garantovano ispravan u celosti, proizvođač specificira minimum ispravnih stranicaova u svakoj logičkoj jedinici (LUN) unutar čipa. Loša stranica je ona koja ima više loših bita od količine koja se može ispraviti relativno jednostavnom ECC šemom kao što je Hamming kod.

VII. Integracija podrške za NAND u operativni sistem

Da bi se hardverski ubrzana korekcija greški mogla koristiti iz operativnog sistema, potrebno je napraviti adekvatan upravljački program. Dodatan zahtev je podizanje operativnog sistema sa ovakve memorije, sto je takođe bilo neophodno uraditi za ovakav računar, uzevši u obzir odsustvo drugih vrsta stalnih memorija velikog kapaciteta na ovom modulu.

Računari kao što je ovaj ne poseduju sopstveni BIOS, koji je kod klasičnih PC računara tu kako bi inicijalizovao hardver i pokrenuo izvršavanje operativnog sistema. Zato ovakvi računari koriste "loadere" u više nivoa koji pomažu prilikom inicijalizacije operativnog sistema. Ovi *loaderi* faktički imaju istu osnovnu svrhu kao BIOS na PC računarima. Kod ovog računara, kao i kod većine sličnih računara ove svrhe, koriste se program "X-Loader" za inicijalizaciju prvog nivoa i popularni U-Boot, *loader* drugog nivoa. *Loader* prvog nivoa vrši podešavanje multipleksiranih pinova, inicijalizuje klok i dinamičku RAM memoriju, a zatim učitava *loader* drugog nivoa u RAM i pokreće njegovo izvršavanje. Tako pokrenuti *loader* drugog nivoa (U-Boot) počinje svoj rad vršeći dodatne potrebne hardverske inicijalizacije komponenti računara, poput aktiviranja Ethernet kontrolera, izvlačenja dodatnih čipova iz stanja reseta i slično.

U-boot podrazumevano sadrži podršku za različite sisteme datoteka, a tu je o podrška za pristup mreži koristeći TFTP protokol ili mrežni sistem datoteka (NFS). Nakon inicijalizacije svih memorija i sistema datoteka, ovaj *loader* može učitati kernel operativnog sistema sa stalne memorije u RAM i otpočeti njegovo podizanje.

Sistem na čipu DM8168 takođe poseduje sopstveni minijaturni *bootloader* lociran u ROM-u čipa, koji služi da inicijalizuje najosnovnije elemente čipa i prepozna memorijske uređaje. Podržano je podizanje sistema iz različitih izvora kao što su NOR memorija povezana na SPI interfejs procesora, kao i MMC (eksterna memorijska kartica), NAND i NOR memorija povezani direktno na GPMC. Međutim, prilikom implementacije NAND memorijskog čipa kompanije Micron, pokazalo se da ga ROM *bootloader* ne prepoznaje na ispravan način, iako je sam NAND čip funkcionalan.

Problem postoji zbog nepostojanja definitivnog standarda za NAND memorije, a autori ROM *bootloadera* nisu planirali upotrebu čipova sa stranicama većim od 2 kB. Kako ovakav problem nije mogao biti prevaziđen jer ROM *bootloader* nije moguće menjati nakon izlaska čipa iz proizvodnje, pristupilo se pronalaženju alternativnog rešenja. Na ploču je ugrađena mala količina NOR memorije povezane na SPI linije čipa DM8168. Ideja je da se u SPI fleš memoriji uskladište *loaderi* prvog i drugog nivoa (X-Loader i U-Boot), a da U-Boot obezbedi podršku za pristup Micron NAND memoriji sa koje bi se potom učitao konkretan operativni sistem.

A. Upravljanje memorijom

Za razliku od poluprovodničkih memorija poput USB fleš uređaja i SD kartica, koje pored NAND memorije imaju i odgovarajući „pametan" kontroler koji se brine o ravnomernom iskorišćenju prostora, obeležavanju loših blokova i slično, osnovni NAND čipovi nemaju takav kontroler u sebi. To znači

da upravljački softver *loadera* i operativnog sistema mora da vodi računa o lošim blokovima i istrošenosti memorije (zbog konačnog broja upisa po bloku, pre nego što isti počne da gubi osobinu zadržavanja stabilnog stanja), kao i da učestvuje u otkrivanju grešaka. Iz tog razloga, NAND memorije zahtevaju posebne upravljačke programe i sisteme datoteka.

Prilikom razvoja upravljačkih programa unutar U-Boot *loadera* i operativnog sistema sa Linux jezgrom, izuzetno je važno voditi računa o njihovoj međusobnoj kompatibilnosti, odnosno da podaci koji su upisani u NAND iz *loadera* budu čitljivi iz Linuxa i obrnuto. Konkretno, kod za ispravljanje grešaka u upravljačkim programima oba sistema treba da ima identičan rezultat, inače dolazi do grešaka. Recimo, Linux će pokušati da pročita podatke upisane iz *loadera*, zateći će „pogrešan" ECC kod i tretirati stranicu kao pogrešno upisanu, iako su konkretni podaci ispravni. Dalje, svi podaci moraju biti zapisani na istim mestima u okviru za to predviđenog prostora. U ovom slučaju se za korekciju grešaka koristi Hamming kod koji zahteva 24 bita (3 bajta) na 512 bajtova korisnih podataka. To ukupno čini 24 bajtova, koliko je potrebno za kod za ispravku grešaka kod stranice od 4 kilobajta. Proizvođač je na ovom NAND čipu obezbedio dodatni prostor od 224 bajtova uz svaku stranicu od 4 kB za podatke, što znači da većina prostora neće biti iskorišćena u celosti. Ključno je da kod za ispravljanje grešaka bude zapisan uvek na tačno određenom i uvek istom mestu unutar prostora namenjenog za metapodatke stranice.

NAND memorije ne poseduju klasične tabele particija pa se struktura željenih particija proizvoljno definiše putem U-Boota i Linuxa. I u ovom slučaju je bitno da se logičke particije poklapaju u pogledu početne lokacije i dužine, kako bi sve metastrukture sistema datoteka bile pravilno identifikovane. S obzirom da Linux standardno podržava prihvatanje opcionih argumenata u formi „komandne linije" koju mu *loader* šalje, na ovaj način je moguće logički raspored particija iz U-Boota direktno proslediti Linuxu prilikom njegovog podizanja.

Upravljački programi koji su realizovani na nivou operativnog sistema Linux služe da realizuju pristup memoriji na nivou blokova, sto uključuje pristup bloku za čitanje, upis ili brisanje sadržaja bloka. Inače, blokovi su hardverski definisani u toku same izrade NAND memorije, a upravljački softver apstrahuje i realizuje pristup bloku na nivou operativnog sistema. Pristup blokovima podataka vrši se posredstvom komandi univerzalnog memorijskog kontrolera (GPMC), kao i realizacijom komandi koje su specifične samo za tu memoriju.

Upravljački softver takođe vodi računa u računanju koda za ispravljanje grešaka (u ovom slučaju „Hamming" koda) kao i o eventualnoj korekciji grešaka koje su detektovane prilikom čitanja. Tokom računanja koda za ispravljanje grešaka, koristi se specijalizovani hardver koji se nalazi u memorijskom kontroleru opšte namene (GPMC). Memorijskom čipu se pristupa kroz asinhron hardverski interfejs, slanjem odgovarajućih komandi za čitanje, upis ili za brisanje celog bloka.

Posebna klasa uređaja na Linuxu naziva se MTD (eng. *Memory Technology Device*) [5] i služi za objedinjavanje načina pristupa različitim tipovima fleš memorija, uključujući NAND i NOR, i to bez obzira na koji su hardverski interfejs povezani. Na Linuxu, MTD služi kao međusloj između upravljačkog programa konkretnog uređaja i gornjeg sloja koji obezbeđuje organizaciju podataka koristeći konkretan sistem datoteka.

Prilikom planiranja softvera, potrebno je realizovati i uključiti podršku za ovu klasu uređaja na nivou operativnog sistema Linux. Prilikom planiranja i realizacije jezgra operativnog sistema, potrebno je uključiti u jezgro podršku za MTD uređaje, kao i particije i sisteme datoteka. Realizovani upravljački programi moraju omogućiti korektan pristup podacima i odgovarajuće ispravljanje grešaka. Potrebno je obezbediti odgovarajuće datoteke koje definišu izgled particija MTD uređaja. Operativni sistem Linux drugačije obeležava particije MTD uređaja u odnosu na ostale blok uređaje. Tako particija */dev/mtd1* može biti druga particija na prvom memorijskom čipu, ali i prva particija na drugom memorijskom čipu.

Veličina svake particije strogo mora biti umnožak veličine bloka za brisanje (u slučaju konkretne Micron memorije, blok za brisanje velik je 512 kB). Ako to nije slučaj, operativni sistem će automatski pomeriti početak particije na prvu sledeću odgovarajuću adresu, ali neće pomeriti i podatke koji su eventualno upisani iz U-Boota. Linux kao operativni sistem nudi veliki broj alata za manipulaciju blokovima, particijima i sistemima datoteka MTD uređaja. Solidnu podršku nudi i U-Boot *loader* (naravno, uzevši u obzir da se radi o bazičnom *loaderu* a ne punokrvnom operativnom sistemu), koji takođe može prikazati sadržaj svake stranice pojedinačno, uključujući njenu oblast za metapodatke.

B. Sistem datoteka JFSS2

Upotreba NAND memorija zahteva specijalni softverski tretman. Ovi memorijski uređaji zbog svoje specifične strukture tehnički ne spadaju ni u jednu od dve osnovne klase uređaja koje Linux podržava (*character* i *block*, odnosno uređaji sa kojih se čita isključivo po FIFO principu i uređaji po kojima je moguće pomerati pokazivač za čitanje). Zbog toga je na Linuxu razvijen dodatni softverski sloj za apstrakciju ovih uređaja, pomenuti MTD. Njegova uloga je da gornjem sloju (implementaciji sistema datoteka) prezentuje NAND memoriju kao uređaj kompatibilan sa klasičnim blok uređajima, kao što su hard diskovi.

Za razliku od blok uređaja poputa hard diskova koji su zasnovani na sektorima kao najmanjoj jedinici podataka, ovi uređaji su zasnovani na blokovima koji se brišu (eng. *eraseblock*). Postoje tri glavne operacije koje se mogu izvršiti nad ovim uređajima, a to su: čitanje iz bloka, upis u blok i brisanje bloka. Blokovi su znatno veći od sektora, reda nekoliko stotina. Njihov životni vek je ograničen, pa su osmišljeni algoritmi za ravnomerno raspoređivanje podataka duž celog kapaciteta NAND memorije, kako bi se izbegao slučaj da pojedini blokovi prerano otkažu zbog preteranog korišćenja. Međutim, za razliku od klasičnih hard diskova, zamenu oštećenih jedinica podataka ne radi hardver. Svaki blok je dostupan softveru, bez obzira da li je on oštećen ili ispravan. Zbog svega toga potrebno je koristiti specijalni sistem datoteka koji je svestan strukture NAND memorije i raspolaže adekvatnim algoritmima [6].

Jedan od takvih sistema datoteka novijeg datuma jeste JFFS2 (eng. *Journalling Flash File System version 2*), unapređena verzija JFFS sistema koji izvorno nije podržavao NAND uređaje i kompresiju podataka [7]. Sistem datoteka JFFS2 realizovan je koristeći strukturu dnevničkih zapisa (eng. *log-structured filesystem*). Najmanja jedinica ovog sistema datoteka je pomenuti blok za brisanje (*eraseblock*). Svaka izmena na nekoj datoteci u NAND-u smešta se u drugi fizički blok, dok se stara lokacija obeležava kao slobodna umesto da se uzastopno prebrisuje. Na taj način se sistem datoteka brine o ravnomernoj eksploataciji svih blokova, a samim tim i dužem ispravnom funkcionisanju celog NAND čipa. Ako se prilikom upisa zaključi da je blok oštećen, on će biti preskočen i dodat listi takvih blokova. Ova lista se obično nalazi u operativnoj memoriji i popunjava se svaki put kada se podiže operativni sistem. Alternativno, određeni broj blokova može biti proglašen oblašću za trajno čuvanje mape loših blokova.

Zbog veličine *erase* bloka, koja je značajno veća od veličine sektora na hard diskovima, NAND nije povoljan za skladištenje jako malih i retko popunjenih datoteka, pa je potrebno da aplikacije koriste veće bafere u RAM memoriji, tako da se upisivanje vrši sa što manjom fregmentacijom. podaci o slobodnom prostoru na disku ne mogu biti precizni i relevantni. Naposletku, JFFS2 podržava razne tipove kompresije (uključujući često korišćeni *zlib*), što kao posledicu ima nemogućnost tačnog određivanja veličine preostalog slobodnog prostora na uređaju (pošto će količina podataka koji se mogu upisati direktno zavisiti od njihovog tipa, odnosno mogućnosti da se kompresuju u manjoj ili većoj meri).

VIII. ZAKLJUČAK

NAND žrtvuje slučajan pristup i prednosti izvršavanja direktno iz stalne memorije (eng. *in-place execution*) koje nudi NOR. Sa druge strane, NAND je izrazito pogodan za sisteme koji zahtevaju veliki kapacitet. On nudi veću gustinu, veći kapacitet i značajno manju cenu po bitu informacije. U poređenju sa NOR memorijama, NAND karakterišu brzo brisanje, sekvencijalni upis i sekvencijalno čitanje.

Uzevši u obzir izložene informacije, može se zaključiti da je NAND memorija jako pogodna za pojedine primene, ali i prilično komplikovana za integraciju u namenske računarske sisteme. Problem se usložnjava kada je potrebno koristiti NAND memoriju kao stalnu memoriju za čuvanje samog operativnog sistema. Tada je, pored osnovne implementacije upravljačkih programa u operativnom sistemu, neophodno razviti i upravljački program u *loaderu* koji treba da učita glavni operativni sistem. Naročitu pažnju treba obratiti na usaglašavanje algoritma za detekciju i korekciju grešaka u *loaderu* i samom operativnom sistemu.

Proces računanja ECC-a nije jeftina procesorska operacija, pa pojedini procesori i kontroleri sadrže hardverske jedinice zadužene samo za to. Na konkretnom računaru iz ovog rada dostupno je hardversko ubrzanje generisanja i provere ECC-a u okviru memorijskog kontrolera procesora, što je iskorišćeno u upravljačkom softveru *bootloadera* i Linuxa.

U namenskim sistemima novijeg datuma, umesto NAND memorije u bazičnom obliku sve se češće koristi eMMC (eng. *embedded MultiMedia Card*), koji je takođe sastavljen od NAND memorije i suštinski funkcioniše kao SD kartica, ali je spakovan u čip i samim tim se odlikuje većom robusnošću i životnim vekom u poređenju sa SD karticama. eMMC u sebi sadrži napredniji kontroler koji automatski vrši upravljanje blokovima i korekciju grešaka, bez potrebe da se time bavi upravljački softver operativnog sistema. Na taj način skupe softverske operacije zamenjuje specijalizovani hardver, čime se donekle gubi na fleksibilnosti i mogućnosti implementacije specijalnih softverskih funkcija ukoliko za njima postoji potreba, ali se može drastično uštedeti na vremenu i resursima, naročito ako se pojavi potreba za zamenom memorijskog čipa zbog prestanka tržišnog života starog modela.

ZAHVALNICA

Finansiranje rada je delimično obezbeđeno iz budžeta projekata Ministarstva prosvete, nauke i tehnološkog razvoja Republike Srbije, oznaka TR32043.

LITERATURA

[1] Texas Instruments TMS320DM816x DaVinci Video Processors, http:// www.ti.com/product/tms320dm8168

[2] Karakteristike računara specijalne namene IMPT EPP-Q7-DM8168, http://www.embedded.rs/products/ti-davinci-dm8168-qseven-module

[3] NAND Flash Memory, MT29F16G08ABACA datasheet, http://www. micron.com/

[4] Open NAND Flash Interface, http://www.onfi.org/

[5] Memory Technology Device, http://www.linux-mtd.infradead.org/

[6] Seung-Ho Lim, Kyu-ho Park, „An efficient NAND flash file system for flash memory storage," Computers, IEEE Transactions on Volume: 55, Issue:7, pp. 906–912 , July 2006.

[7] D. Woodhouse „JFFS: The Journalling Flash File System", in Proceedings of Ottawa Linux Symposium, 2001.

ABSTRACT

This paper describes the use of NAND flash memories in dedicated computer systems, with particular focus on hardware-accelerated error correction. A computer system running a Linux-based operating system and using this kind of memory was developed. We discuss the advantages and disadvantages of using NAND memories in such systems. Moreover, we explain the structure of NAND memories and their hardware and software inferfaces, the ways to access the memory chip, a software development support in the form of device drivers, and a specialized filesystem for NAND flash.

AN IMPLEMENTATION OF HARDWARE-ACCELERATED ECC IN NAND FLASH MEMORIES

Vukašin Ristić, Mirjana Stojilović, Ivan Todorović