Univerzitet u Kragujevcu
Fakultet tehničkih nauka u Čačku

Uroš Pešović

Mikrokontrolerski sistemi

Čačak, 2023. godine

UNIVERZITET U KRAGUJEVCU
FAKULTET TEHNIČKIH NAUKA U ČAČKU

MIKROKONTROLERSKI SISTEMI

Uroš Pešović

Čačak, 2023.

Autor: dr Uroš Pešović

Naslov: Mikrokontrolerski sistemi

Recenzenti:

prof. dr Siniša Ranđić

prof. dr Predrag Petrović

Lektor: Đorđe Đurđević

Nastavno naučno veće Fakulteta tehničkih nauka u Čačku odobrilo je štampanje ovog udžbenika odlukom broj 012-86-237/23 od 22.2.2023.

Izdavač: Fakultet tehničkih nauka u Čačku

CIP - Каталогизација у публикацији

Народна библиотека Србије, Београд

004.42:004.383/.384(075.8)

ПЕШОВИЋ, Урош, 1982-

Mikrokontrolerski sistemi / Uroš Pešović. - Čačak : Fakultet tehničkih nauka, 2023 (Čačak : Fakultet tehničkih nauka, Grafički centar). - 128 str. : ilustr. ; 25 cm

Na vrhu nasl. str.: Univerzitet u Kragujevcu. - Tiraž 150. - Bibliografija: str. 127-128.

ISBN 978-86-7776-265-0

а) Микроконтролери -- Програмирање

COBISS.SR-ID 109761033

PREDGOVOR

U današnje vreme retkost je pronaći kompleksan uređaj ili sistem čija kontrola nije zasnovana na računaru. Savremeni računarski sistemi proizvode se u različitim veličinama, počevši od minijaturnih računara koji se nazivaju mikrokontroleri, personalnih računara i računara visokih performansi. Mikrokontroleri predstavljaju minijaturne računare u formi integrisanog kola, koja su najbrojniji tip računarskih sistema sa desetinama milijardi proizvedenih primeraka godišnje. Njihova cena višestruko je niža u odnosu na personalne računare, zahvaljujući čemu postaju nezaobilazni u uređajima koji nas okružuju. S obzirom da komercijalni uređaji moraju biti kompetetivni na globalnom tržištu, projektovanje mikrokontrolerskih sistema osim niza tehničkih ograničenja zahteva od inženjera da se prilagodi i ekonomskim ograničenjima kako bi uređaj bio što konkurentniji na tržištu.

Mikrokontrolerski sistem predstavlja hardversko/softversku platformu čiji je zadatak upravljanje uređajem u koji se ugrađuje. Projektovanje mikrokontrolerskih sistema zahteva širok spektar znanja kako bi inženjer bio u stanju da projektuje hardverski deo, koji je u interakciji sa sistemom, i softver kako bi uređaj ispravno funkcionisao.

Cilj ovog udžbenika jeste upoznavanje studenata sa osnovnom strukturom mikrokontrolera, načinom funkcionisanja mikrokontrolerskih periferija i procesom razvoja ugrađenih aplikacija. Kroz niz poglavlja obrađuju se osnovni elementi mikrokontrolerskih sistema, pri čemu kroz svako poglavlje ovaj udžbenik obrađuje odgovarajuće osobenosti ATmega328p mikrokontrolera, najpristupačnijeg mikrokontrolera za studente, koji se koristi u Arduino UNO razvojnom sistemu. Jedan od glavnih razloga za njegovu popularnost jeste skup ugrađenih biblioteka, koje prikrivaju detalje implementacije niskog nivoa na ATmega328p

mikrokontroleru, što predstavlja veliku olakšicu za početnike. Većina literature koja obrađuje ATmega328p mikrokontroler bazirana je na korišćenju ovih ugrađenih biblioteka, te čitalac ne stiče znanja koja može primeniti na drugim mikrokontrolerskim platformama. Ovaj udžbenik studente upoznaje sa detaljnom strukturom ATmega328p periferija i načinom na koji ugrađene funkcije pristupaju njegovim periferijama. Na osnovu ovih znanja student će biti u stanju da pravilno koristi i modifikuje postojeće ili napiše sopstvene biblioteke za ovaj mikrokontroler ili za mikrokontrolere drugih serija.

Udžbenik u uvodnim poglavljima predstavlja ulogu mikrokontrolerskih sistema u ugrađenim sistemima i obrađuje arhitekturu ATmega328p mikrokontrolera. U sledećim poglavljima predstavljen je modularni razvojni sistem Arduino UNO, baziran na ATmega328p mikrokontroleru, i programsko razvojno okruženje Arduino IDE koje se koristi za pisanje mikrokontrolerskih aplikacija. Preostala poglavlja redom obrađuju sve periferne module koji su prisutni kod ATmega328p mikrokontrolera.

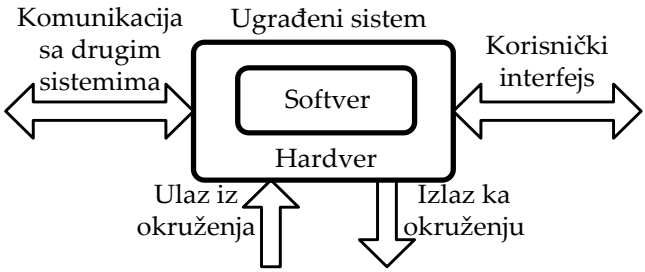
SADRŽAJ

POGLAVLJE 1: UVOD

Ugrađeni (eng. *embedded*) sistem predstavlja računarski sistem specifične namene ugrađen u određeni elektromehanički sistem u kojem on ostvaruje tačno definisanu funkciju. Danas skoro svaki uređaj „potrošačke" elektronike (eng. *consumer electronics*) u sebi sadrži ugrađeni sistem, međutim on je najčešće implementiran na takav način da je skriven od krajnjeg korisnika. Svrha ugradnje računara specifične namene u neki sistem jeste poboljšanje performansi sistema u smislu boljeg i efikasnijeg upravljanja implementacijom sofisticiranih upravljačkih algoritama. Jedan takav primer jesu savremene mašine za pranje veša, koje su u stanju da prilagode dužinu ciklusa pranja količini i stepenu zaprljanosti veša. Takođe ugrađeni sistemi pružaju veći broj funkcija dostupnih korisniku, kao što bi bilo upravljanje preko mobilnog telefona. Ugrađeni sistemi obezbeđuju veću pouzdanost jer su tolerantni na otkaze i pružaju mogućnost dijagnostike kvarova što može značajno skratiti vreme popravke.

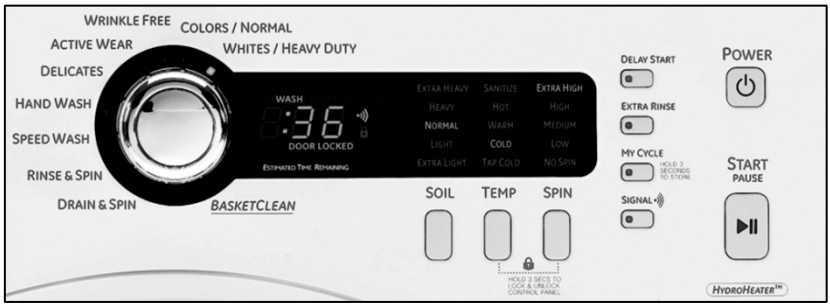
Osnovna funkcija ugrađenog sistema jeste kontrola procesa koji se odvija unutar nekog sistema, a ona se ostvaruje preko ulazno-izlaznih interfejsā (slika 1). Proces predstavlja niz progresivnih operacija koje slede jedna drugu na relativno utvrđen način kako bi se ostvario određeni ishod u sistemu u kojem se proces odvija. Ugrađeni sistemi najčešće moraju posedovati konkurentno i reaktivno ponašanje, odnosno moraju konkurentno izvršavati veći broj nezavisnih aktivnosti i odgovoriti na sekvence i kombinacije događaja. Takođe, u pojedinim slučajevima ugrađeni sistemi namenjeni za rad u realnom vremenu poseduju vremenske granice u kojima sistem mora reagovati. Ugrađeni sistemi

moraju raditi samostalno i biti u stanju da se izbore sa greškama koje se mogu javiti u toku rada, a da ne izazovu otkaz sistema (eng. *fault handling*).



Slika 1. Interfejsi ugrađenog sistema

Ugrađeni sistem mora posedovati i određeni korisnički interfejs, putem kojeg je moguća interakcija korisnika sa samim sistemom. Korišćenjem tastera ili prekidača korisnik može zadavati određene komande ugrađenom sistemu, dok se informacije o trenutnom statusu sistema korisniku prikazuju pomoću ekrana (eng. *display*) ili LED (eng. *light emitting diode*) svetlećih dioda. Vrlo je čest slučaj da se broj tastera ili prekidača koji se koristi za interakciju korisnika sa ugrađenim sistemom svede na najmanju prihvatljivu meru, što zbog tehno-ekonomskih ograničenja što zbog estetike samog uređaja (slika 2), te dato znači da pojedini tasteri mogu imati višestruke funkcije. Najefikasniji način interakcije sa korisnikom ostvaruje se preko ekrana osetljivog na dodir, ali je takvo rešenje najčešće neekonomično pa se umesto njega mogu koristiti sedmosegmentni indikatori u kombinaciji sa LED svetlećim diodama.

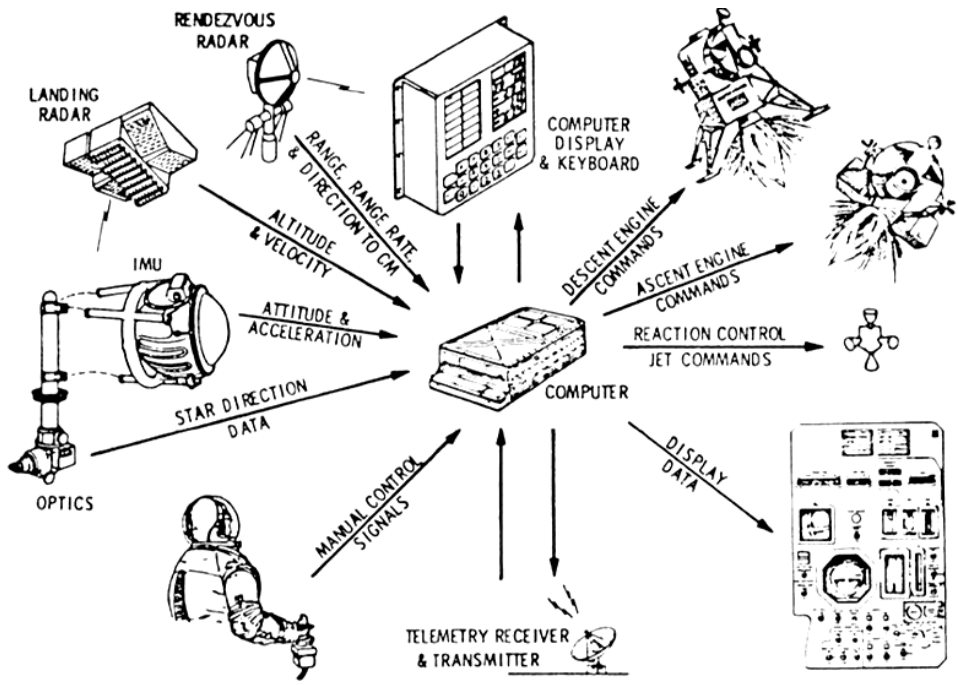


Slika 2. Primer korisničkog interfejsa ugrađenog sistema

Ugrađeni sistem ponekad mora komunicirati i sa drugim ugrađenim sistemima, jer je vrlo čest slučaj da sistem koji se koristi predstavlja samo

jedan deo nekog većeg sistema koji zahteva interakciju svih podsistema kako bi se ostvarila željena funkcija.

Prvi moderan prepoznatljiv ugrađeni sistem bio je AGC (eng. *Apollo Guidance Computer*), koji je razvijen za Apollo svemirske misije za letove na Mesec. Svaka Apollo misija do Meseca koristila je dva takva ugrađena sistema, jedan u komandnom modulu za let do Meseca i jedan u lunarnom modulu za sletanje na Mesec. AGC ugrađeni računar u lunarnom modulu kontrolisao je niz sistema prikazanih na slici 3. Razvoj ugrađenog sistema smatrao se najriskantnijim delom Apollo projekta, zbog velikih nepoznanica i tadašnjih tehnoloških ograničenja. Kako bi smanjili veličinu i težinu AGC-a za realizaciju upotrebljena su u to vreme nova monolitna integrisana kola (korišćena su dva troulazna NILI kola u jednom integrisanom kolu). Najizazovniji deo projekta bio je razvoj softvera koji je morao obezbediti niz funkcionalnosti od kojih je dobar deo morao raditi u realnom vremenu uz značajna ograničenja vezana za raspoloživi memorijski prostor od 72 kB koda.



Slika 3. Funkcije AGC računara u lunarnom modulu

Načini realizacije ugrađenih sistema

Ugrađeni sistemi mogu biti realizovani na različite načine u zavisnosti od ograničenja koja su postavljena u početnim fazama projektovanja. U tabeli 1. međusobno su upoređeni različiti načini realizacije ugrađenih sistema na osnovu niza tehno-ekonomskih ograničenja. Ugrađeni sistemi mogu biti realizovani na hardveru specijalne namene u vidu diskretne logike, ASIC (eng. *Application Specific Integrated Circuit*) integrisanih kola specifične namene ili u slučaju korišćenja rekonfigurabilnog hardvera, kao što su PLA (eng. *Programmabile Logic Array*), CPLD (eng. *Complex Programmable Logic Device*) и FPGA (eng. *Field Programmable Gate Array*). Hardver specijalne namene koristi se u slučaju kada je potrebna velika brzina odziva ugrađenog sistema, ali je ovakav način realizacije vrlo nefleksibilan i teško se može nadograđivati. Integrisana kola specifične namene daju najbolje karakteristike, ali iziskuju velike troškove projektovanja, te je jedino isplativo ako se ugrađeni sistemi proizvode u jako velikim serijama. Ako se ugrađeni sistemi izrađuju u manjim serijama ispativije je realizovati ih na rekonfigurabilnom hardveru.

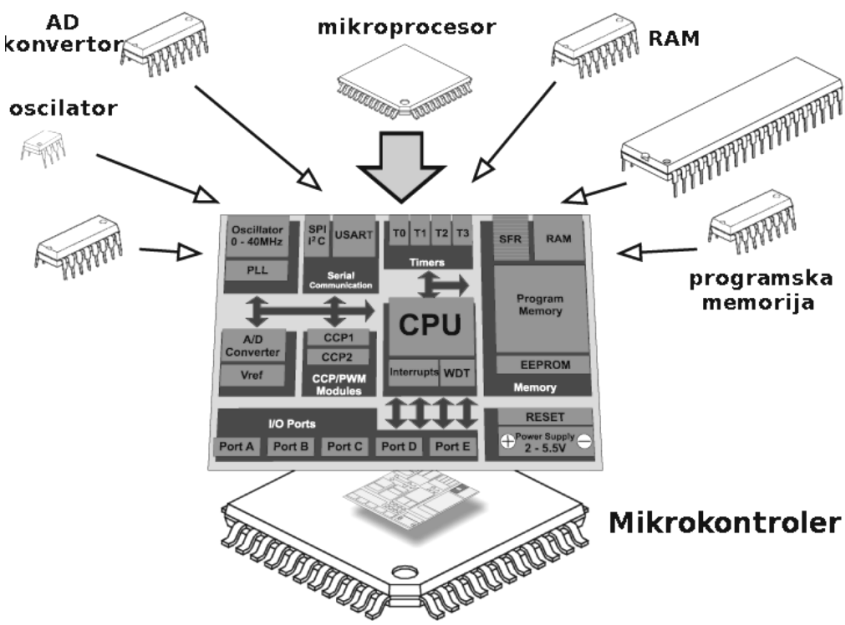
Tabela 1. Načini realizacije ugrađenih sistema

Implementacija	**Cena projek.**	**Cena izrade**	**Nadogradnja i prepravke**	**Dimen-zije**	**Potrošnja energije**	**Brzina sistema**
Diskretna logika	niska	srednja	teško	velike	velika	veoma brza
ASIC	vrlo visoka	veoma niska	teško	veoma male	mala	ekstremno brza
FPGA, CPLD	niska	srednja	lako	male	srednja/ visoka	veoma brza
Mikrokontroler	niska	niska/ srednja	lako	male	srednja	spora/ srednja
Računar opšte namene	srednja	srednja	lako	male/ srednja	srednja	srednja
Ugrađeni personalni računar	niska	visoka	lako	srednja	srednja/ visoka	brza

Osim realizacije ugrađenih sistema na specijalizovanom hardveru, oni se mogu realizovani pomoću softvera koji se izvršava na komercijalno dostupnom (generičkom) hardveru. Oni mogu biti realizovani na računarima opšte namene ili ugrađenim personalnim računarima ili na mikrokontrolerskim sistemima. Računari opšte namene i ugrađeni personalni računari bazirani su na mikroprocesorima koji se ne mogu samostalno koristiti u ugrađenim sistemima. Kod ovakvih računarskih sistema neophodno je spregnuti mikroprocesor sa memorijom i ulazno-izlaznim kolima kako bi se jedan ovakav sistem opšte namene mogao primeniti kao ugrađeni sistem. Ovakav pristup sa sobom nosi veće dimenzija, veću potrošnje energije i višu cenu ovakvih sistema.

Mikrokontrolerski sistemi

Mikrokontroler predstavlja kompletan računarski sistem realizovan u vidu monolitnog integrisanog kola, koji se sastoji od procesorskog jezgra ugrađenih memorija i niza perifernih interfejsa koji mu omogućavaju lakše povezivanje sa spoljašnjim okruženjem. Neki od najčešćih perifernih interfejsa mikrokontrolera jesu digitalni i analogni ulazi i izlazi I/O (eng. *Input/Output*), tajmeri i brojači, te serijski komunikacioni interfejsi (slika 4).



Slika 4. Struktura mikrokontrolera

Osnovna razlika između mikroprocesora i mikrokontrolera jeste u tome da mikroprocesor za svoj rad zahteva eksterne komponente kao što su operativna memorija, ulazno-izlazni podsistem kako bi se kreirao funkcionalan računarski sistem. Mikrokontroler u sebi sadrži sve ove komponente, tako da može samostalno kontrolisati određeni proces bez dodatnih komponenti.

Moderni mikroprocesori optimizovani su za velike brzine rada kako bi se postigle visoke performanse računarskih sistema opšte namene. Za razliku od njih, mikrokontroleri koji se koriste u sistemima specijalne namene optimizovani su u pravcu integracije većeg broja komponenti na čipu, masovne proizvodnje, niske cene i male potrošnje energije. Mikrokontroleri su namenjeni da rade u mnogo zahtevnijim okruženjima, u kojima je potrebno obezbediti upravljanje u realnom vremenu, kao i otpornost na varijacije napona napajanja, temperature i vlažnosti vazduha, vibracija i šumova. Oni se proizvode u znatno većem broju primeraka u odnosu na mikroprocesore (reda nekoliko desetina milijardi godišnje) i njihova oblast primene jesu elektronski uređaji koji nas okružuju, tzv. „potrošačka" elektronika, automobilska industrija, medicina, inteligentni senzori. Prednosti mikrokontrolerskih sistema u odnosu na računarske sisteme opšte namene jesu male dimenzije, niska cena i mala potrošnja električne energije.

Jedan od ograničavajućih faktora koji značajno utiče na proces projektovanja mikrokontrolerskih sistema jeste cena uređaja, koja utiče na to da projektant mora maksimalno iskoristi raspoložive resurse mikrokontrolera. Kompetetivno tržište ne usvaja proizvode koji ne pružaju adekvatnu vrednost za uloženi novac. Pored toga, ograničavajući faktori kod dizajna ovakvih sistema jesu veličina i težina, kao i mala potrošnja energije, a i uređaji moraju biti sposobni da rade u širokom opsegu radnih temperatura. Proces projektovanja mikrokontrolerskih sistema izazov je za inženjera jer zahteva znanja iz različitih oblasti računarstva:

- programiranja
- arhitekture i organizacije računarskih sistema
- digitalne elektronike
- osnova računarske tehnike.

Softver za mikrokontrolerske sisteme najčešće se piše u jezicima visokog nivoa, koji se prevodi u mašinski jezik korišćenjem programskog prevodilaca (eng. *compiler*). U slučaju da je potrebno da se realizuju vremensko kritični delovi programskog koda oni se mogu pisati u asembleru, koji predstavlja simbolički reprezent mašinskog jezika. Od brojnih jezika visokog nivoa programski jezik C postao je *de facto* standard za razvoj aplikacija za ugrađene sisteme. Za razliku od drugih jezika poput Jave i Pythona programski kod napisan u C-u pruža:

- preciznu kontrolu nad onim što procesor radi,
- efikasan kod koji se može izvršavati na slabijim procesorskim jezgrima,
- umeren zahtev za memorijom,
- predvidljivo ponašanje jer ne postoji operativni sistem ili Garbage Collector kao kod Jave.

Kod jednostavnijih ugrađenih aplikacija programski kod piše se za neposredno izvršavanje (eng. *bear metal*). Kako raste kompleksnost upravljačkih aplikacija, dolazi se do problema izvršavanja višestrukih nezavisnih zadataka na mikrokontrolerskom jezgru. U zavisnosti od konkretnog slučaja može se koristiti izvršavanje preko prekidnih rutina (eng. *foreground/background execution*), za neke slučajeve dovoljan je raspoređivač zadataka (eng. *scheduler*), a u nekim je neophodan RTOS (eng. *Real Time Operating System*) operativni sistem za rad u realnom vremenu.

POGLAVLJE 2: AVR ARHITEKTURA MIKROKONTROLERA

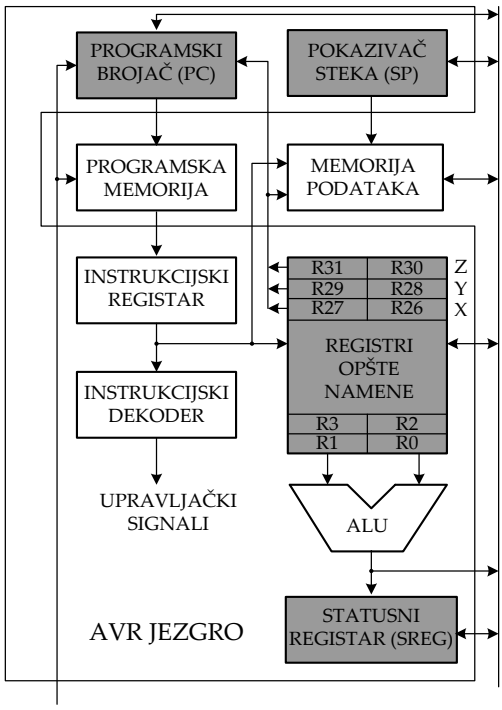
Procesorsko jezgro mikrokontrolera najčešće je zasnovano na RISC (eng. *Reduced Instruction Set Computer*) arhitekturi sa redukovanim setom instrukcija, pri čemu se širina magistrale podataka kreće od 8 do 32 bita, a radna frekvencija retko kada prelazi 100 MHz. RISC arhitekture predstavljaju posebnu familiju računarskih arhitektura, projektovanih sa ciljem ostvarivanja visoke energetske efikasnosti i niske cene. Posebnu grupu mikrokontrolera čine DSP (eng. *Digital Signal Processor*) procesori za obradu signala sa proširenim setom instrukcija koji su namenjeni za aplikacije koje zahtevaju brze matematičke operacije (obrada zvuka, slike, brza merenja i slično).

AVR arhitektura

AVR serija mikrokontrolera jedna je od najkorišćenijih mikrokontrolerskih familija za realizaciju nekomercijalnih ugrađenih aplikacija. Njihovoj popularnosti najviše je doprinelo uključivanje u niz Arduino otvorenih razvojnih sistema koji se najčešće koriste u edukativne svrhe za razvoj jednostavnih ugrađenih sistema. AVR arhitektura osmišljena je 1992. godine od strane AVR arhitekturu osmislili su 1992. godine Alf-Egil Bogen i Vegard Voland, studenti sa Norveškog tehnološkog instituta (eng. *Norwegian Institute of Technology*). U saradnji sa kompanijom Atmel 1996. godine razvijen je prvi mikrokontroler sa oznakom AT90S8515 koji je bio jedan od prvih mikrokontrolera koji je koristio FLASH memoriju za skladištenje programa, za razliku od ostalih

proizvođača mikrokontrolera koji su u to vreme koristili programabilne ROM memorije (jednokratni programabilni ROM, EPROM ili EEPROM).

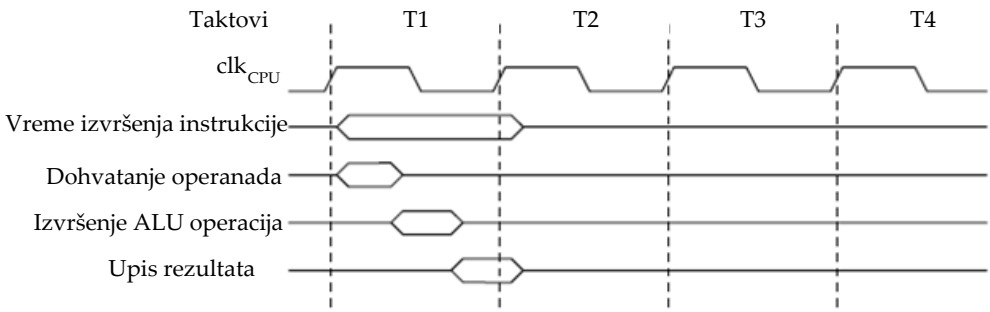
AVR arhitektura zasnovana je na modifikovanoj Harvard arhitekturi, kod koje se program i podaci čuvaju u fizički odvojenim memorijskim adresnim prostorima, pri čemu postoji mogućnost čitanja podataka iz programske memorije korišćenjem posebnih instrukcija. AVR arhitektura prvobitno je definisana kao 8-bitna arhitektura, da bi Atmel 2006. godine predstavio AVR-32 bitnu arhitekturu koja nije doživela značajan komercijalni uspeh. U okviru AVR 8-bitne arhitekture prvobitno su razvijeni mikrokontroleri ATtiny familije, zatim su sledili mikrokontroleri ATmega familije i AVR DX i AVR mikrokontroleri specifične namene. Instrukcijska arhitektura mikroprocesora definiše skup programskih dostupnih registara, prikazanih na slici 5, koji se mogu modifikovati skupom od 131 instrukcije. Ova arhitektura definiše 32 registra opšte namene i tri registra specijalne namene: statusni registar SREG (eng. *Status Register*), pokazivač steka SP (eng. *Stack Pointer*) i programski brojač PC (eng. *Program Counter*).



Slika 5. Struktura AVR procesorskog jezgra

AVR koristi LOAD/STORE arhitekturu kod koje aritmetičko-logička jedinica ALU (eng. *Arithmetic Logic Unit*) može izvršavati operacije samo nad podacima (operandima) koji se nalaze u registrima opšte namene. Instrukcije mogu adresirati najviše dva operanda koji se nalaze u registrima opšte namene, pri čemu se rezultat izvršenja instrukcije smešta na adresu prvog operanda. Takođe, umesto drugog operanda može se koristiti i neposredna vrednost (konstantna) iz tekuće instrukcije. Registre opšte namene čine 32 preslikana registra R0 do R31 sa vremenom pristupa od jednog takta, što omogućava izvršavanje operacija aritmetičke logičke jedinice u jednom ciklusu.

LOAD/STORE arhitekture ne dozvoljavaju aritmetičko-logičkim instrukcijama da direktno pristupaju memoriji, pa je neophodno pre izvršenja aritmetičko-logičkih instrukcija korišćenjem LOAD instrukcije preneti podatke iz SRAM memorije u registre opšte namene, a po završetku izvršenja instrukcije rezultat koji se smesti u registar opšte namene instrukcijom STORE smešta se nazad u memoriju, kao što je prikazano na talasnom dijagramu na slici 6.

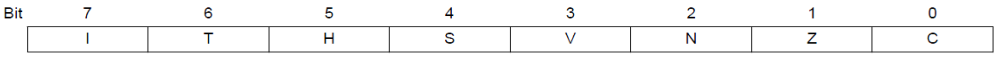


Slika 6. Faze izvršenja aritmetičko logičkih instrukcija

Mikrokontrolersko jezgro podržava osmobitne aritmetičke operacije sabiranja, oduzimanja, inkrementiranja, dekrementiranja, negacije i množenja. Podržane su i logičke operacije I, ILI, EXILI, operacije setovanja i resetovanja pojedinačnih bitova u registru ili setovanja i resetovanja celokupnog registra.

Statusni registar

Statusni registar sadrži statusne bitove koji daju informacije o statusu rezultata nakon poslednje izvršene aritmetičke instrukcije. Ove informacije mogu se koristiti za promenu toka programa kako bi se izvršile instrukcije uslovnog skoka. Statusni registar sadrži sledeće bitove prikazane slikom 7:



Slika 7. Struktura statusnog registra

- C-bit (eng. *Carry*) setuje se pri pojavi prenosa/pozajmice na poziciji najveće težine kod aritmetičkih operacija.
- Z-bit (eng. *Zero*) setuje se kada je rezultat aritmetičko-logičke operacije jednak nuli.
- N-bit (eng. *Negative*) setuje se kada je rezultat aritmetičko-logičke operacije sa označenim brojevima negativan.
- V-bit (eng. *Overflow*) setuje se pri pojavi prekoračenja kod aritmetičkih operacija sa označenim brojevima, odnosno kada rezultat operacije izađe van dozvoljenog opsega za predstavljanje osmobitnih označenih brojeva od –128 do +127. Prekoračenje se detektuje kada se sabiranjem dva pozitivna broja dobije negativan rezultat, odnosno sabiranjem dva broja dobije pozitivan rezultat.
- S-bit (eng. *Sign*) setuje se kao rezultat ekskluzivne ILI operacije između N i V bita kako bi se korigovala vrednost znaka u slučaju pojave prekoračenja. U slučaju pojave prekoračenja vrednost N bita biće invertovana kako bi se pokazao ispravan znak rezultata.
- H-bit (eng. *Half carry*) setuje se pri pojavi poluprenosa sa pozicije nižeg 4-bitnog polubajta (eng. *nibble*) na poziciju višeg polubajta. Bit poluprenosa pogodan je za BCD (eng. *Binary Coded Decimal*) operacije kod kojih se decimalne cifre predstavljaju pomoću 4-bita.
- T-bit (eng. *Bit Copy Storage*) koristi se za privremeno smeštanje bitova kao rezultat izvršenja podrutina.
- I-bit (eng. *Global Interrupt Enable*) bit omogućava globalne prekide.

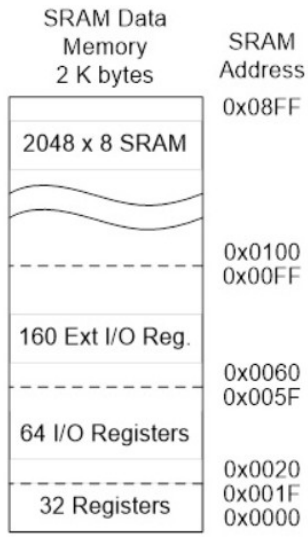
Vrednosti pojedinačnih statusnih bitova mogu se setovati/resetovati korišćenjem asemblerskih instrukcija prikazanih u tabeli 2.

Tabela 2. Instrukcije za setovanje i resetovanje statusnih bitova

Statusni bit	Instrukcija setovanja	Instrukcija resetovanja
C	sec	clc
Z	sez	clz
N	sen	cln
V	sev	clv
S	ses	cls
H	seh	clh
T	set	clt
I	sei	cli

Memorijski podsistem

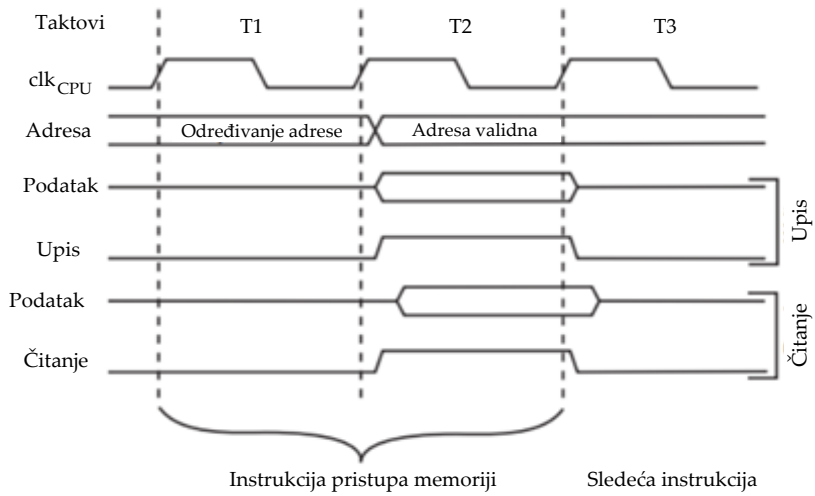
Memorijski podsistem sastoji se od memorije za podatke i programske memorije u kojoj se skladište instrukcije. Memorija za podatke podeljena je u nekoliko memorijski oblasti prikazanih na slici 8. Prvu oblast čine registri opšte namene R0 do R31, kojima se osim direktnih registarskih adresa može pristupati preko dodeljenih memorijskih adresa 0x0000 do 0x001F.



Slika 8. Struktura AVR procesorskog jezgra

Sledeću oblast čine 64 ulazno-izlazna registra, kojima se tipično pristupa preko IN i OUT instrukcija sa adresama od 0x00 do 0x3F. Ova je oblast memorijski preslikana u opseg memorijskih adresa od 0x0020 do 0x003F. Pored ovog skupa postoji i 160 proširenih I/O registara kojima se

može pristupiti isključivo preko dodeljenih adresa u opsegu od 0x0060 do 0x00FF. Nakon ove oblasti sledi oblast SRAM memorije kapaciteta 2048 reči koje se nalaze na adresama od 0x0100 do 0x08FF. Memorijskim lokacijama može se pristupiti korišćenjem direktnog adresiranja pri čemu je adresa podataka u memoriji navedena u samoj instrukciji. Korišćenjem X, Y i Z registara, koji obuhvataju registre R26 do R31, na raspolaganju je nekoliko tipova indirektnog adresiranja kao što su: indirektno adresiranje, indirektno adresiranje sa pomerajem, indirektno adresiranje sa predekrementiranjem i indirektno adresiranje sa postikrementiranjem. Instrukcije za pristup memoriji zahtevaju dva taktna ciklusa, u prvom taktnom ciklusu određuje se adresa na osnovu korišćenog tipa adresiranja a u drugom taktnom ciklusu pristupa se željenoj memorijskoj lokaciji prema talasnom dijagramu prikazanom na slici 9.



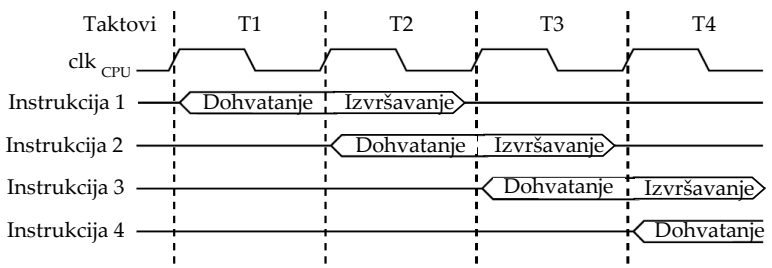
Slika 9. Ciklus pristupa memorijskoj lokaciji

Instrukcijska memorija sastoji se od 16k instrukcija koje mogu biti dužine 16 ili 32 bita, pa je ukupna veličina programske FLASH memorije 32KB. Pošto je AVR zasnovan na modifikovanoj Harvard arhitekturi, na raspolaganju su LPM (eng. *Load Program Memory*) instrukcija za čitanje i SPM (eng. *Store Program Memory*) instrukcija za upis podataka u FLASH memoriju. Zahvaljujući ovim instrukcijama moguća je izmena programa u radu kod mikrokontrolera.

Programski brojač

Programski brojač jeste registar širine 14 bita koji ukazuje na adresu sledeće instrukcije u programskoj FLASH memoriji i u stanju je da adresira program veličine od maksimalno 16K instrukcija. Po završetku tekuće instrukcije programski brojač uvećava se za jedan kako bi pokazao na adresu sledeće instrukcije PC+1. Izuzetak su instrukcije uslovnog i bezuslovnog skoka kod kojih se vrši skok na adresu PC+k+1 koja se određuje na osnovu pomeraja ka specificiranom u adresnom delu instrukcije. Instrukcije uslovnog skoka proveravaju uslov na osnovu vrednosti statusnih bitova prethodne aritmetičko-logičke operacije i, ukoliko je uslov ispunjen, izvršiće se skok, a u suprotnom izvršiće se sledeća instrukcija.

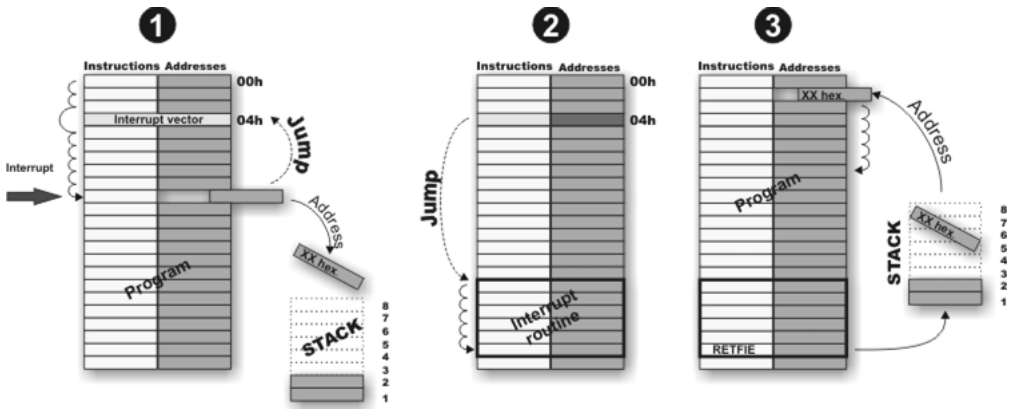
AVR instrukcije realizuju se kroz dve faze, dohvatanje i izvršavanje. U fazi dohvatanja programski brojač pokazuje na adresu instrukcije koja se prenosi iz programske memorije u instrukcijski registar. Zatim se u fazi izvršavanja sadržaj instrukcije dekodira instrukcijskim dekoderom koji generiše upravljačke signale koji omogućavaju izvršenje konkretne instrukcije. AVR procesorsko jezgro koristi dvostepenu protočnu obradu instrukcija prikazanu na slici 10, koja omogućava da se, dok se jedna instrukcija izvršava, sledeća instrukcija unapred dohvata iz programske memorije. Ovaj koncept omogućava izvršavanje instrukcija u svakom ciklusu takta što omogućava visoke performanse od 20 MIPS (eng. *Million Instructions Per Second*) pri maksimalnom radnom taktu od 20 MHz. Poređenja radi, Intel 8051 CISC jezgru potrebno je najmanje 12 taktnih ciklusa da izvrši jednu instrukciju pa bi njegove performanse pri istom radnom taktu bile svega 1.67 MIPS.



Slika 10. Protočna obrada instrukcija

Pokazivač steka

Stek predstavlja bezadresnu memoriju koja je organizovana po LIFO (eng. *Last In First Out*) principu, kod koje je poslednji upisan podatak prvi prilikom čitanja. Ovakva organizacija stek memorije pogodna je za realizaciju poziva podrutina (slika 11). Podrutina predstavlja poseban deo programskog koda, koja se poziva iz glavnog programa ili druge podrutine, pri čemu se trenutni tok programa privremeno zaustavlja. Prilikom poziva podrutine na vrhu steka pamti se vrednost programskog brojača na kojoj je trenutni tok programa zaustavljen i koja se naziva povratna adresa. Svaka podrutina završava se instrukcijom za povratak, čiji je zadatak da sa vrha steka preuzme prethodno sačuvanu povratnu adresu i postavi je u registar programskog brojača. Na taj način nastaviće se izvršavanje programskog koda od mesta na kojem je tok programa prekinut pozivom podrutine.



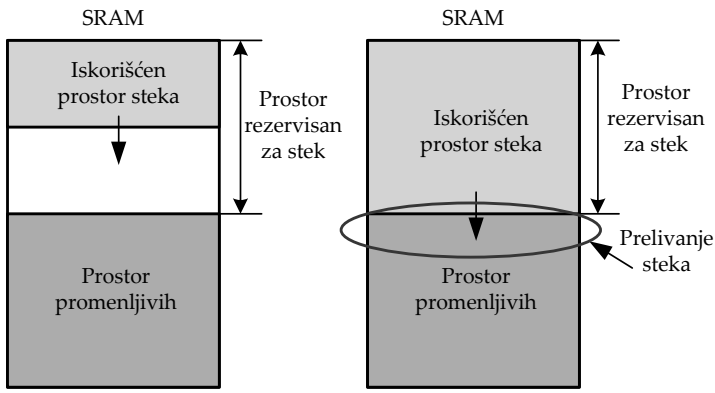
Slika 11. Upotreba steka pri pozivu i povratku iz podrutine

Stek se može implementirati kao nezavisna memorija, ali se najčešće implementira u delu memorije za podatke. S obzirom da je memorija za podatke adresibilna, neophodno je korišćenje specijalnog registra koji se naziva pokazivač steka SP (eng. *Stack Pointer*), koji će ukazivati na trenutnu poziciju vrha steka u memoriji za podatke. Pokazivač steka kod ATmega328p mikrokontrolera širine je 11 bitova kako bi bio u stanju da adresira celokupni prostor memorije za podatke veličine 2048 B. S obzirom da je mikrokontroler 8-bitni, pokazivač steka sastoji se od nižeg SPL i višeg SPH registra, pri čemu se u višem registru koriste samo niža tri bita od osam dostupnih. Kako bi se stek odvojio od korisničkih promenljivih, on je

implementiran na samom kraju programske memorije i on raste od viših ka nižim adresama. Jedna od prvih instrukcija u korisničkom programu postavlja vrednost pokazivača steka na poslednju adresu u memoriji za podatke 0x08FF, od koje u regularnom radu stek raste ka nižim adresama.

Instrukcije za direktnu manipulaciju sa stekom jesu PUSH i POP. Instrukcija PUSH smešta osmobitni podatak na vrh steka pri čemu dolazi do automatskog dekrementovanja pokazivača steka kako bi pokazao na sledeću slobodnu lokaciju u steku. POP instrukcija uzima podatak sa vrha steka i inkrementuje pokazivač steka. Pored ove dve instrukcije različite instrukcije poziva potprograma (direktni, indirektni i relativni poziv potprograma) automatski smeštaju povratnu adresu, odnosno sadržaj programskog brojača na stek. S obzirom da je veličina programskog brojača 14 bitova, on zauzima dve memorijske lokacije na steku, pa se sa ovim instrukcijama sadržaj pokazivača steka umanjuje za dve memorijske lokacije. Analogno ovim instrukcijama instrukcije povratka iz potprograma uzimaju poslednje dve vrednosti sa steka i smeštaju ih u programski brojač i inkrementiraju vrednost programskog brojača za dva.

Tokom poziva podrutine na stek se smeštaju povratne adrese, argumenti i lokalne promenljive koje se potom uklanjaju prilikom povratka iz potprograma. Stek će tokom regularnog izvršavanja programa rasti i sažimati se u skladu sa brojem pozvanih podrutina. Prilikom ugnežđenih poziva podrutinā može doći do prelivanja steka (eng. *stack overflow*) u memorijsku oblast koja se koristi za skladištenje podatka (slika 12) i ovakva situacija dovodi do nepredviđenog rad računarskog sistema.

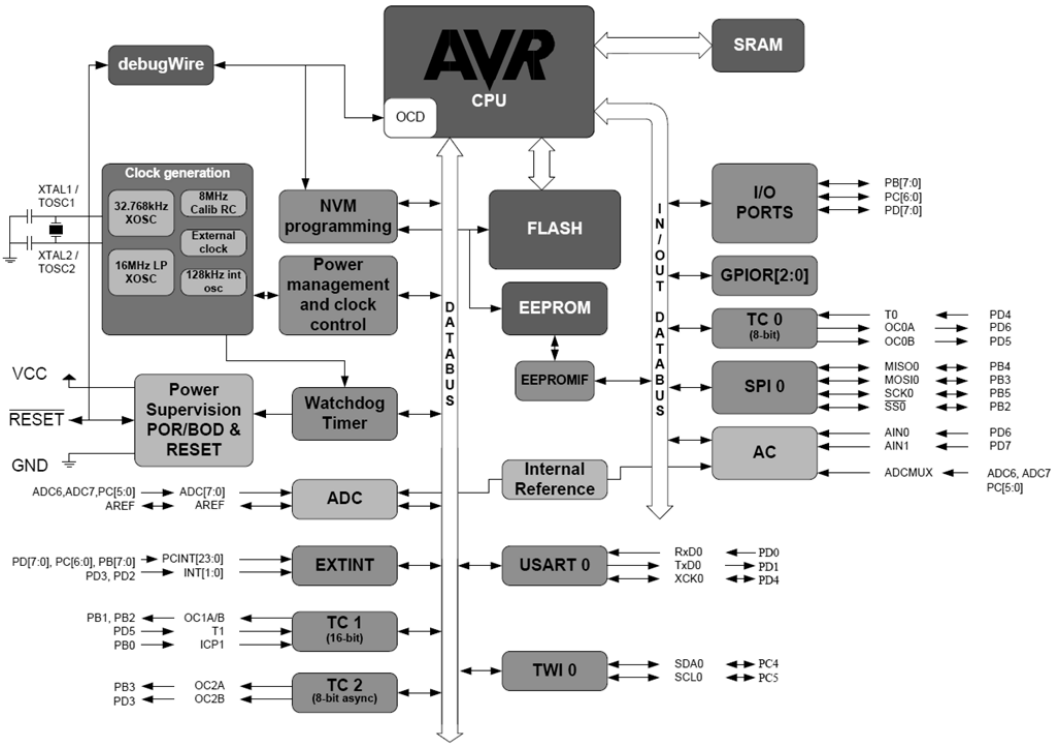


Slika 12. Prelivanje steka

Kako bi se ova situacija predupredila, neki mikrokontroleri poseduju hardverski ograničenu veličinu steka koja, kada se premaši, generiše se signal za resetovanje mikrokontrolera. Pravilo dobre prakse jeste da se izbegava korišćenje duboko ugnežđenih poziva, a u nekim slučajevima obavezno je izbegavati rekurzivne pozive podrutina koje lako mogu dovesti do prelivanja steka. Tokom faze testiranja sistema najpre se celokupna oblast koju može koristiti stek popuni nekom unapred poznatom vrednošću (šablonom), te se tokom rada sistema prati maksimalna veličina steka na osnovu broja preostalih neprepisanih simbola.

Podsistem ulazno-izlaznih periferija

Mikrokontrolersko jezgro sa programskom memorijom i memorijom za podatke čini unutrašnji računarski sistem koji nije u stanju da vrši interakciju sa svojom okolinom. Zahvaljujući perifernim interfejsima, prikazanim na slici 13, ATmega procesorsko jezgro u stanju je da vrši interakciju sa svojim okruženjem preko ulazno-izlaznih pinova.

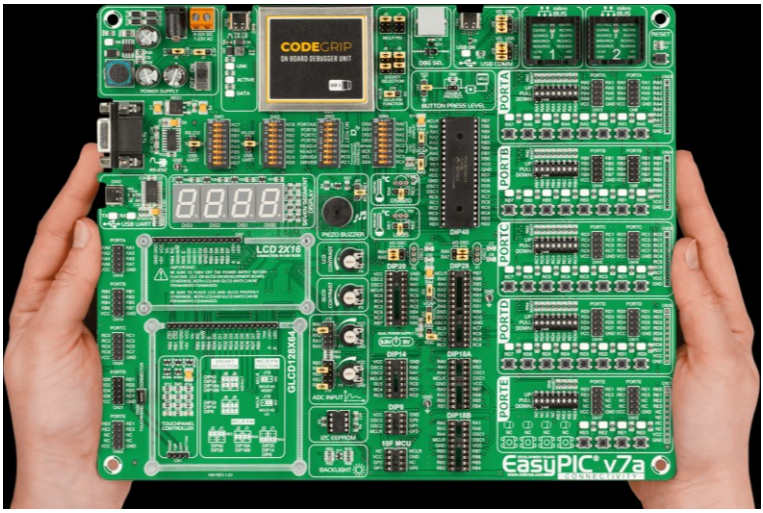


Slika 13. Struktura mikrokontrolera sa periferijama

Periferije komuniciraju sa procesorom preko svojih registara koji su memorijski preslikani u memoriju za podatke i dostupni na memorijskim adresama od 0x0020 do 0x00FF. Na raspolaganju je 224 registra koji su podeljeni u dve oblasti, bazičnu oblast od 64 I/O registra i proširenu oblast od 160 I/O registra. Procesor ATmega328p mikrokontrolera pristupa bazičnim I/O registrima ulazno-izlaznih periferija preko 8-bitne ulazno-izlazne magistrale, korišćenjem IN/OUT instrukcija za čije je izvršenje potreban jedan taktni ciklus. Preostalim I/O registrima može se pristupiti samo preko magistrale podataka korišćenjem LOAD i STORE instrukcija čije izvršenje zahteva dva taktna ciklusa.

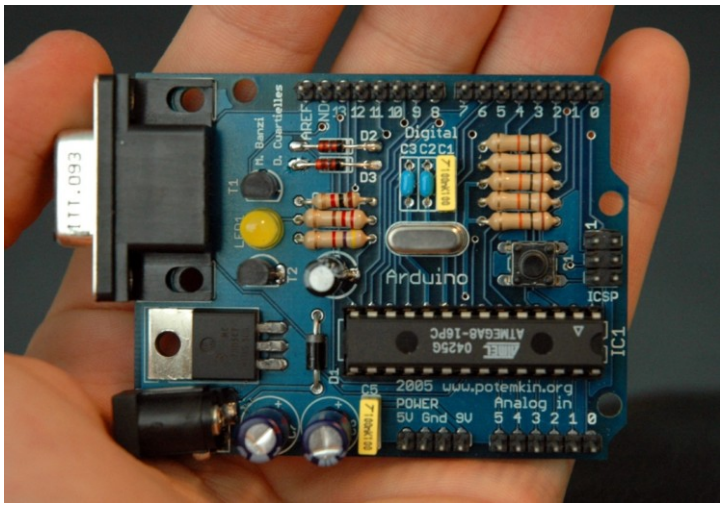
POGLAVLJE 3: ARDUINO UNO RAZVOJNI SISTEM

Mikrokontroler predstavlja kompletan računarski sistem realizovan na jednom integrisanom kolu, ali ipak za svoj rad zahteva i neke eksterne komponente, kao što su sistem za napajanje, izvor takta, interfejs za programiranje i ulazno-izlazne interfejse ka sistemu u koji se mikrokontroler ugrađuje. Sve ove komponente zajedno sa mikrokontrolerom smeštaju se na PCB (eng. *Printed Circuit Board*) štampane ploče, a date ploče poseduju jako dobre elektro-mehaničke karakteristike. Međutim, proces projektovanja štampanih ploča može biti izazovan za početnike, kojima su na raspolaganju gotove štampane ploče sa mikrokontrolerom, koji se nazivaju razvojni sistemi (eng. *Development Board*). Razvojni sistemi mogu sadržati veliki broj perifernih uređaja i interfejsa (slika 14), koji mogu znatno olakšati razvoj ugrađenih aplikacija. Kada se na razvojnom sistemu realizuje željena aplikacija, neophodno je da se projektuje štampana ploča sa komponentama koje su korišćene na razvojnom sistemu. Ugradnja same razvojne ploče u sistem jako je nepraktična, što zbog njene visoke cene, kao i zbog velikih dimenzija usled jako velikog broja komponenti koje su prisutne na ploči razvojnog sistema. U poslednje vreme kompleksni razvojni sistemi u značajnoj su meri potisnuti od strane modularnih razvojnih sistema, gde osnovni modul razvojnog sistema čini mikrokontrolerska štampana ploča sa minimalnim brojem komponenti.



Slika 14. Mikrokontrolerski razvojni sistem

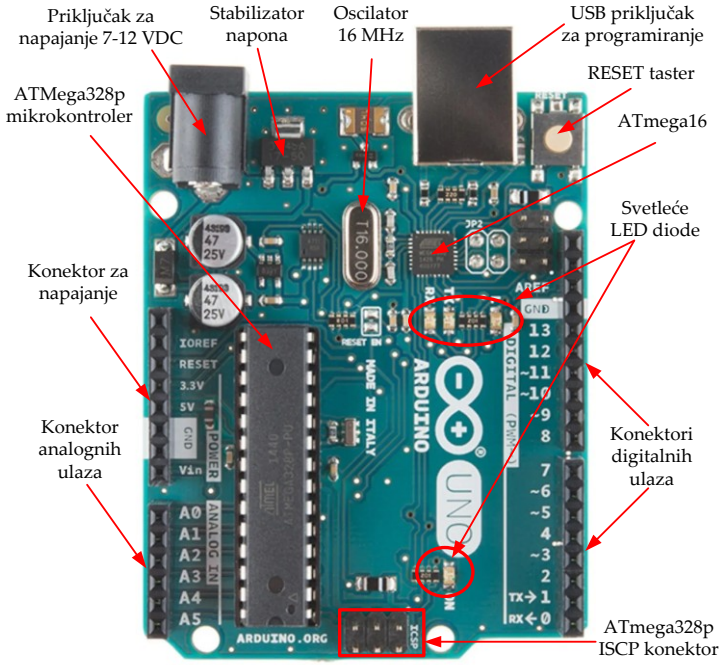
Jedan od prvih modularnih razvojnih sistema pod nazivom Arduino razvijen je sa ciljem da se početnicima omogući jednostavan i ekonomičan razvoj mikrokontrolerskih aplikacija. Arduino razvojni sistem, prikazan na slici 15, dizajniran je tako da koristi najjednostavnije moguće komponente koje se lako mogu nabaviti na tržištu i koristio je RS232 interfejs za programiranje ili komunikaciju sa računarom.



Slika 15. Jedan od prvih modularnih Arduino razvojnih sistema 2005. godine

Nakon prvih Arduino razvojnih sistema, 2010. godine na tržištu se pojavio Arduino UNO, otvoreni mikrokontrolerski sistem, zasnovan na ATmega328p mikrokontroleru koji je postigao globalni uspeh. Arduino

UNO, prikazan na slici 16, predstavlja razvojni sistem malih dimenzija, sa 14 digitalnih ulaza/izlaza i 6 analognih ulaza koji se takođe mogu koristiti i kao digitalni ulazi/izlazi opšte namene. Svi ulazni/izlazni pinovi povezani su na niz konektora smeštenih na obodu razvojne ploče, preko kojih je moguće povezati osnovnu razvojnu ploču sa nizom modula za proširenje koji se na engleskom jeziku nazivaju štitovi (engl. Shield).



Slika 16. Komponente Arduino UNO razvojnog sistema

Arduino UNO razvojni sistem umesto RS232 interfejsa koristi USB interfejs putem kojeg se obezbeđuje i napajanje razvojnog sistema od 5 V pri maksimalnoj struji od 500 mA. Postoji i mogućnost dovođenja jednosmernog napon u opsegu 7÷12 V preko okruglog konektora sa centralnim pinom od 2.1 mm. Ovaj napon stabilizuje se preko linearnog regulatora LM1117, koji obezbeđuje maksimalnu struju od 1 A. Ulazni jednosmerni napon putem posebnog pina na konektoru V_{in} može se koristiti za napajanje delova razvojnog sistema koji zahtevaju viši napon od 5 V, kao što su releji i elektromotori.

Arduino UNO poseduje eksterni kristalni oscilator od 16 MHz koji se koristi kao primarni izvor taktnog signala. Preostale komponente na

razvojnom sistemu jesu taster za resetovanje i niz svetlosnih dioda, koje se koriste za indikaciju prisustva napajanja od 5 V i status serijske komunikacije ka računaru. Na dnu ploče nalazi se šestopinski ICSP (eng. *In Circuit Serial Programming*) konektor za direktno programiranje ATmega328p mikrokontrolera korišćenjem eksternog programatora. Ovaj metod programiranja koristi se u današnje vreme jer je veoma neuobičajeno programirati integrisana kola pre nego što se zaleme na štampanu ploču, naročito ako su u pitanju SMD (eng. *Surface Mounted Device*) komponente. Eksterni programatori predstavljaju uređaje koji su nezaobilazni alat koji koriste inženjeri za razvoj ugrađenog softvera. Oni najčešće osim mogućnosti programiranja poseduju mogućnost hardverskog debagovanja. Ovakvi alati najčešće su vrlo skupi i nedostupni početnicima.

Većina današnjih mikrokontrolera poseduje mogućnost učitavanja koda pomoću pokretačkog programa (eng. *bootloader*). Pokretački program poseban je deo koda koji po restartu pruža mogućnost da se napisani program u binarnom obliku može prebaciti sa računara u FLASH memoriju mikrokontrolera. U slučaju Arduino UNO razvojnog sistema, kada se mikrokontroler resetuje, pokretački program konfiguriše serijski port mikrokontrolera za prenos podataka sa računara. Kada se na računaru pokrene proces programiranja, specijalni signal na Arduino razvojnom sistemu izaziva resetovanje mikrokontrolera što primorava pokretač da prima podatke sa računara i smešta ih u FLASH memoriju. Kada se završi prenos podataka u FLASH memoriju, pokretački program skače na početnu lokaciju aplikacije i počinje da izvršava instrukcije ugrađenog programa. U slučaju reseta pokretački program neće videti specijalni signal na serijskom portu i skočiće na lokaciju aplikacije u FLASH memoriji. Na ovaj vrlo jednostavan način izvrši se programiranje mikrokontrolera bez potrebe za eksternim programatorom. S obzirom da ATmega328p mikrokontroler ne poseduje USB interfejs, na Arduino UNO ploči nalazi se još jedan dodatni mikrokontroler ATmega16, čija je uloga da pretvara signale između USB interfejsa računara u UART interfejsa ATmega328p mikrokontrolera.

POGLAVLJE 4: ARDUINO IDE PROGRAMSKO OKRUŽENJE

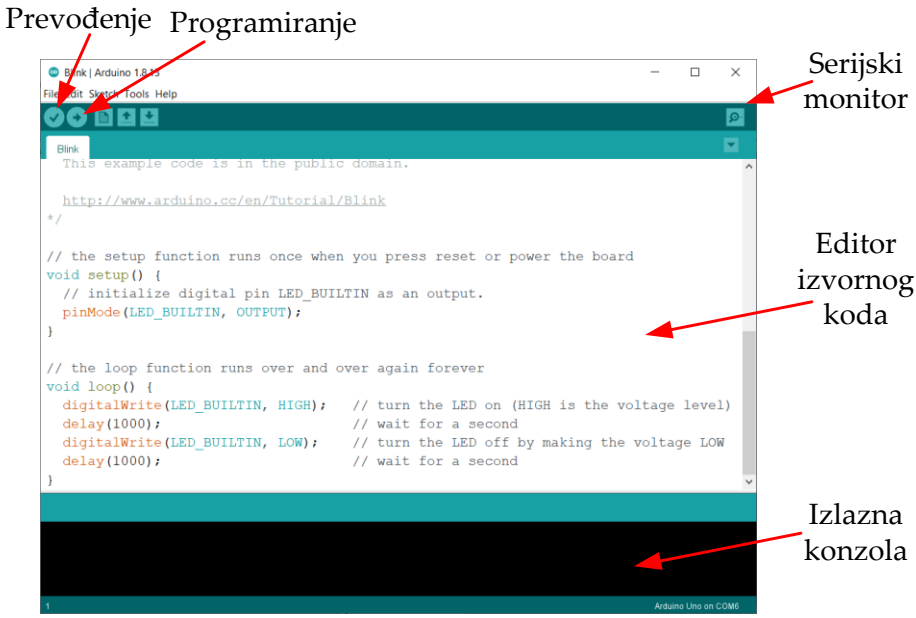
Jedan od najznačajanih razloga za globalni uspeh Arduino ugrađene razvojne platforme jeste koncept da ova platforma bude što je moguće otvorenija. Osim električnih šema razvojnog sistema koje su dostupne svima, uspehu platforme doprinelo je i Arduino IDE (eng. *Integrated Development Environment*) programsko okruženje za razvoj ugrađenih aplikacija. Ovo okruženje omogućava pisanje i prevođenje koda visokog nivoa za niz Arduino razvojnih sistema, kao i programiranje istih sa personalnog računara. Ovo programsko okruženje zasnovano je na Javi zahvaljujući čemu se programi za ugrađeni hardver mogu pisati i pod Windows, MacOSx ili Linux operativnim sistemima. Arduino IDE programsko okruženje, osim Arduino UNO razvojnih sistema, podržava desetine Arduino razvojnih sistema koji su zasnovani na drugim mikrokontrolerskim platformama što je i jedan od razloga popularnosti.

Program u Arduino IDE okruženju napisan je u vidu beleške (eng. *scratch*) koja se pamti kao jedna datoteka sa .ino ekstenzijom. Predprocesiranjem beleške od strane Arduino IDE razvojnog okruženja dodaje se datoteka zaglavlja za izabrani razvojni sistem koja uključuje sve definicije potrebne za odabranu arhitekturu mikrokontrolera. U sledećem koraku generiše se C++ datoteka koja se prevodi korišćenjem *avr-gcc* prevodioca. Kao rezultat prevođenja dobija se mašinski kod u Intel HEX formatu koji se korišćenjem programatora ili pokretačkog programa na mikrokontroleru upisuje u programsku memoriju mikrokontrolera, odakle se izvršava od strane procesorskog jezgra (slika 17).

Slika 17. Postupak programiranja iz razvojnog okruženja

Arduino IDE okruženje za razvoj mikrokontrolerskih aplikacija, prikazano na slici 18, sastoji se od:

- editora izvornog koda programa (C/C++),
- prevodioca (kompajlera), koji prevodi izvorni C/C++ kod programa u mašinski kod,
- programatora za prebacivanje mašinskog koda u programsku FLASH memoriju mikrokontrolera,
- serijskog monitora za komunikaciju sa Arduinom posredstvom virtuelnog serijskog porta.



Slika 18. Izgled Arduino IDE razvojnog okruženja

Jedan od glavnih razloga za popularnost Arduino platforme kod početnika jesu biblioteke ugrađenih funkcija koje prikrivaju detalje niskog nivoa neophodne za implementaciju koda na ciljanom mikrokontroleru. S obzirom da ovo predstavlja veliku olakšicu za početnika, ona ipak ne pomaže da se steknu znanja koja bi mogla da se primene na bilo koji drugi mikrokontroler. Jedan od ciljeva ovog udžbenika jeste da čitaoca upozna sa načinom kako ugrađene funkcije direktno utiču na ATmega328p mikrokontroler kako bi stekao opšta znanja koja će moći da primeni i na mikrokontrolerima drugih serija.

Stil pisanja programa za ugrađene aplikacije

Stil pisanja programa predstavlja skup smernica koje se koriste prilikom pisanja izvornog koda za računarski program kako bi se omogućilo drugim programerima da budu u stanju da čitaju i razumeju izvorni kod. U slučaju ugrađenih aplikacija prvih nekoliko redova u izvornoj su datoteci komentari koji opisuju datoteku, njenu svrhu, autora, verziju itd. Takođe, na početku je potrebno specificirati vezu ugrađenog koda sa spoljašnjim hardverom, odnosno specifikacija veza.

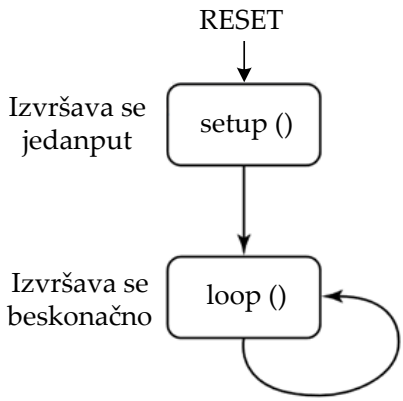
Zatim slede #include naredbe za uključivanje datoteka zaglavlja koje govore kompajleru gde da potraži kod koji će se koristiti, koji nije uključen u trenutnu datoteku. Preporučeno je da se prototipovi funkcija definišu u posebnoj datoteci zaglavlja kako bi se izbegle greške zbog redosleda kompajliranja. Prototip funkcije jeste deklaracija funkcije koja izostavlja telo funkcije, ali samo navodi tip povratne vrednosti, ime funkcije i tipove i nazive argumenata. U toku prevođenja, kompajler uključuje neophodne datoteke zaglavlja koje se skupa kompajliraju sa glavnom izvornom datotekom.

Pravilo je dobre prakse da se na sam početak programa postave fragmenti koda koji se tokom razvoja aplikacije mogu menjati. Fragmenti koda najčešće predstavljaju numeričke parametre, mada mogu predstavljati i stvarne delove koda, definišu se pomoću #define direktive i nazivaju se makronaredbe. Svaka makronaredba sastoji se od imena koje je

preporučeno da bude napisano velikim slovima koje se pri prevođenju zamenjuju definisanim fragmentom koda.

Globalne promenljive predstavljaju tip promenljive koje su dostupne svim delovima programksog koda i one se deklarišu na samom početku programa svojim imenom i tipom. Globalne promenljive predstavljaju memorijske lokacije preko kojih nezavisni delovi koda mogu razmenjivati informacije, npr. između glavnog programa i prekidne rutine. Međutim, poseban oprez mora biti posvećen upotrebi globalnih promenljivih jer one mogu biti promenjene od bilo kog dela koda u bilo kom trenutku što može dovesti do problema trke (eng. *race condition*).

Nakon uvodnog dela sledi izvršni deo programa koji se sastoji od C/C++ koda koji čine dve osnovne funkcije setup () i loop () čiji je redosled izvršavanja prikazan na slici 19. Setup funkcija koristi se od strane programera za inicijalno podešavanje mikrokontrolera i ona se izvršava samo jednom, neposredno posle restarta mikrokontrolera. Nakon završetka setup funkcije započinje beskonačno izvršavanje tela loop funkcije u kojoj se nalazi glavni program.



Slika 19. Princip izvršavanja Arduino IDE programa

Nakon setup i loop funkcija slede definicije ostalih funkcija koje se pozivaju iz glavnog programa. Preporuka je da se za svaku funkciju u komentaru ispred nje jasno opiše šta ona radi, koji su joj argumenti i šta je povratna vrednost funkcije.

Prilikom podešavanja periferija mikrokontrolera vrednost pripadajućih registara najčešće se postavlja u heksadecimalnom brojnom

sistemu. Ukoliko je potrebno postaviti vrednosti pojedinačnih bitova u registru, pri čemu ne bi smelo doći do promene ostalih, potrebno je koristiti odgovarajuće operacije za setovanje, resetovanje i proveru statusa bita putem bitmaski.

Otklanjanje grešaka u ugrađenim sistemima

Veoma bitan aspekt razvoja ugrađenih aplikacija jeste sposobnost testiranja ispravnog funkcionisanja ugrađenog sistema. Nerealno je očekivati da se prilikom izgradnje celokupnog hardvera u celosti može napisati upravljački softver sa kojim će ugrađeni sistem raditi kako se očekuje. Pravilo dobre prakse jeste inkrementalni razvoj hardvera i softvera pri čemu se nakon svakog koraka testira dodata funkcionalnost. Tokom razvoja hardvera i softvera mogu se pojaviti semantičke greške u radu sistema koji se nazivaju bug-ovi. Semantičke greške ne mogu se otkriti tokom kompajliranja jer za razliku od sintaksnih grešaka, ovaj tip grešaka u potpunosti zadovoljava sintaksu korišćenog jezika, tako da se one jedino mogu otkriti tokom testiranja sistema. Proces pokušaja rešavanja problema u softveru naziva se otklanjanje grešaka (eng. *debugging*). U slučaju regularnih računarskih aplikacija ovaj proces podrazumeva da programer u okviru programskog okruženja na raspolaganju ima poseban softver, nazvan debager, koji pruža sledeće mogućnosti u praćenju ponašanja aplikacije:

- zaustavljanje izvršavanja programa kada se dostigne određena linija koda koja se naziva tačka prekida (eng. *breakpoint*),
- kada se zaustavi izvršavanje programa, programer ima uvid u trenutni sadržaj registara centralnog procesora i sadržaja memorije (praćenje vrednosti promenljivih),
- izvrši jednu po jednu naredbu na visokom nivou ili instrukciju u asembleru (eng. *stepping*),
- nastaviti regularno izvršavanje programa.

Za razliku od regularnih računarskih aplikacija koje se programiraju i izvršavaju na istoj platformi ugrađene aplikacije programiraju se na personalnom računaru, a izvršavaju na ciljanoj ugrađenoj platformi, pa sam

softver u programskom okruženju nema mogućnosti da otkrije šta se dešava u mikrokontroleru bez posrednog uređaja koji se naziva hardverski debager (eng. *In Circuit Debugger*). Hardverski debager obično sadrži mikroprocesor koji je u stanju da komunicira sa ciljnim mikrokontrolerom putem posebnog interfejsa. Na taj način hardverski debager može kontrolisati izvršavanje programa ciljnog mikrokontrolera i prenositi informacije softverskom alatu za otklanjanje grešaka koji se nalazi na personalnom računaru. Zahvaljujući komunikaciji sa hardverskim debagerom, programeru je na raspolaganju skup funkcionalnosti za otklanjanje grešaka kao u slučaju klasičnih računarskih aplikacija.

Hardverski debager nezaobilazan je alat za profesionalni razvoj ugrađenih aplikacija, ali u nekim slučajevima on nije uvek dostupan za sve platforme. Pored toga oni mogu biti nedostupni za početnike, zbog njihove visoke cene koja može višestruko nadmašiti cenu mikrokontrolerskog sistema koji se razvija. Jedan od najčešće korišćenih interfejsā za debagovanje jeste JTAG interfejs, koji se koristi kod većine mikrokontrolera visokih performansi. Međutim kod mikrokontrolera slabijih performansi češće se koriste jednostavniji protokoli tipa Serial Wire Debug ili debugWire i slično.

Ukoliko hardverski debuger nije dostupan na raspolaganju su jednostavnije metode za otklanjanje grešaka, kao što je tehnika štampanja vrednosti izlaznih promenljivih na konzoli personalnog računara (eng. *trace code*). Korišćenjem ove tehnike programer dodaje linije koda koje na konzoli štampaju trenutni status programa i one se mogu iskoristiti kako bi se otkrilo mesto u programu na kojem dolazi do nepravilnog izvršavanja. U slučaju da serijski interfejs nije dostupan, najjednostavniji metod za praćenje izvršavanja koda kod ugrađenih sistema jeste korišćenje digitalnog izlaznog pina koji se u programu postavlja na stanje logičke nule ili jedinice i čije se stanje posmatra preko svetleće diode ili osciloskopa. Ovaj je način veoma koristan za otklanjanje grešaka u realnom vremenu, u kojem se može meriti kritično vreme ili se utvrditi da li dolazi do prekida.

ATmega328p poseduje debugWIRE bidirekcioni interfejs putem kojeg je moguće ostvariti kontrolu toka i izvršenja instrukcija i pristup registrima i memorijama mikrokontrolera. Ovaj interfejs ostvaruje komunikaciju sa

hardverskim debagerom preko dW linije, koja se deli sa linijom za eksterni RESET mikrokontrolera. Tokom procesa debagovanja preko debugWIRE interfejsa potrebno je obezbediti sledeće uslove za ispravan rad:

- RESET pin sme biti povezan sa Vcc naponom samo preko otpornika koji nije manji od 10kΩ (nije dozvoljeno povezivanje direktno sa Vcc naponom),
- potrebno je odspojiti kondenzatore i spoljašnje izvore reset signala koji su povezani sa linijom za eksterni reset.

Ovaj interfejs omogućava umetanje neograničenog broja tačaka prekida. Kada programer u razvojnom okruženju postavi tačku prekida, hardverski debager preko debugWIRE interfejsa umetnuće AVR BREAK instrukciju u programsku memoriju. Debagovanje za sada nije omogućeno za ATmega328p mikrokontrolere u Arduino IDE programskom okruženju.

POGLAVLJE 5: DIGITALNI ULAZNO-IZLAZNI PORTOVI OPŠTE NAMENE

Jedna od najčešće korišćenih periferija kod ugrađenih sistema jeste skup GPIO (eng. *General-Purpose Input/Output*) ulazno-izlaznih portova opšte namene. Ova komponenta prisutna je u svakom mikrokontroleru i ona obezbeđuje najopštiji vid interfejsa ka spoljašnjim uređajima u vidu pojedinačnih digitalnih linija. Korišćenjem programskih instrukcija za pristup GPIO pinu može se kontrolisati logička vrednost izlaznog pina ili čitati logička vrednost prisutna na ulaznom pinu. GPIO linije tipično su grupisane po portovima, pa pristup odgovarajućem pinu zahteva pristup pojedinačnom bitu u registru. Neke arhitekture podržavaju operacije setovanja i resetovanja bitova u registru, a u slučaju da one nisu podržane neophodno je koristiti bitmaske. Svaki GPIO pin podrazumevano je kontrolisan kao I/O opšte namene i alternativno se može dodeliti jednoj od perifernih funkcija mikrokontrolera.

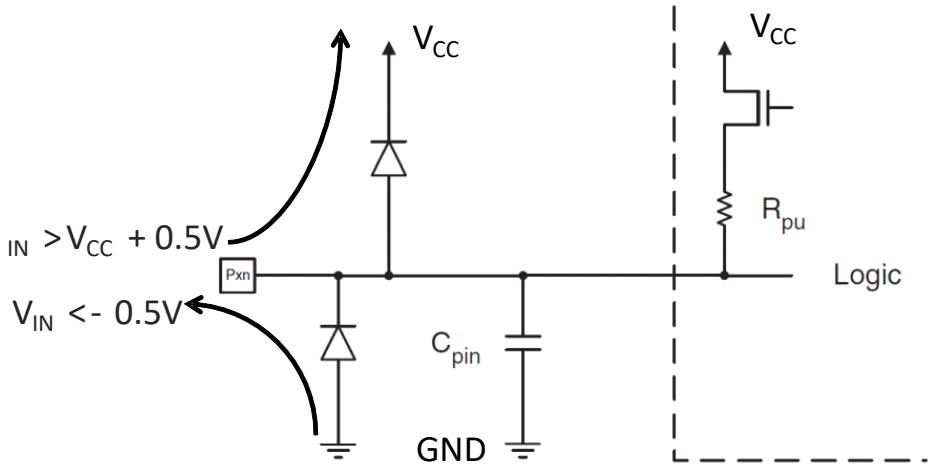
ATmega328P mikrokontroler poseduje 23 GPIO digitalna pina opšte namene koji su grupisani u tri 3 GPIO porta: 8-bitni port B, 7-bitni Port C i 8-bitni port D. U slučaju Arduino UNO razvojne platforme, koja je bazirana na ATmega328P mikrokontroleru, GPIO pinovi povezani su sa priključcima razvojnog sistema koji su prikazani u tabeli 3.

Tabela 3. Raspored ATmega328p pinova na Arduino UNO razvojnom sistemu

GPIO pin	Arduino UNO port
PB0	8
PB1	9
PB2	10
PB3	11
PB4	12
PB5	13
PB6	Eksterni oscilator
PB7	
PC0	A0
PC1	A1
PC2	A2
PC3	A3
PC4	A4
PC5	A5
PC6	Eksterni reset
PD0	0
PD1	1
PD2	2
PD3	3
PD4	4
PD5	5
PD6	6
PD7	7

Pojedini GPIO pinovi zauzeti su od strane Arduino UNO platforme i ne mogu se koristiti kao GPIO opšte namene. U pitanju su GPIO pinovi PB6 i PB7 preko kojih je povezan eksterni kvarcni oscilator od 16 MHz i GPIO pin PC6 preko kojeg je povezan taster za eksterni reset mikrokontrolera.

Svaki GPIO pin poseduje zaštitne (eng. *clamping*) diode čija je uloga da zaštite pin od ekstremnih pozitivnih i negativnih napona (slika 20). Zaštitne diode nemaju uticaja na funkcionalnost GPIO pina kada je na njemu prisutan napon u propisanim granicama od 0 V do Vcc napona napajanja.



Slika 20. Funkcija zaštitinih dioda GPIO pina

U slučaju da se na pinu pojavi visoki pozitivan napon gornja zaštitna dioda biće direktno polarisana i ograničavaće napon ulaznog pina na vrednost Vcc+0.5 V. U slučaju da se na pinu pojavi negativni napon donja zaštitna dioda biće direktno polarisana i ograničiće napon ulaznog pina na vrednost od –0.5 V. Maksimalna vrednost struje koju mogu ograničavati zaštitne diode iznosi 1 mA (eng. *max injection current*), pa u slučaju pojave većih struja kroz zaštitne diode može doći do trajnog oštećenja pina. U slučaju da na ulazni pin mogu doći naponi van propisanih granica potrebno je na red sa GPIO pinom dodati otpornik koji će ograničiti maksimalnu struju na nižu vrednost od maksimalne dozvoljene struje zaštitnih dioda koja iznosi 1 mA.

Svaki GPIO ulazno-izlazni pin može biti nezavisno konfigurisan kao:

- digitalni ulaz sa opcionim pull-up otpornikom čija je uloga da obezbedi stanje logičke jedinice na pinu, ukoliko je pin neaktivan,
- digitalni izlaz sa maksimalnom izlaznom strujom od 20 mA u izvornom (eng. *source*) i ponornom (eng. *sink*) režimu.

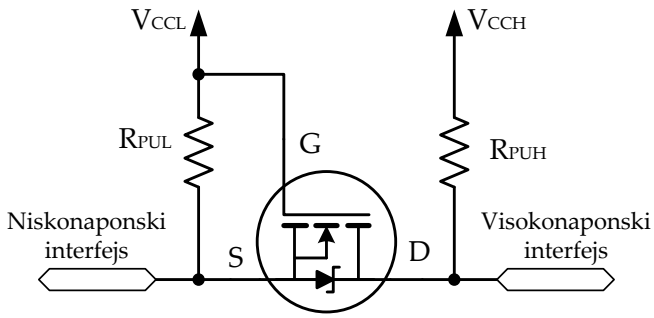
Digitalni ulaz koristi se za očitavanje logičke vrednosti koja je prisutna na ulaznom pinu. Vrednosti napona na ulaznom pinu, koje su manje od maksimalnog napona logičke nule V_{IL}, biće pročitane kao logička nula, dok će vrednosti veće od minimalnog napona logičke jedinice V_{IH} biti protumačene kao logička jedinica. U slučaju da se na ulaznom pinu pojavi

vrednost napona, koja je između maksimalnog napona logičke nule i minimalnog napona logičke jedinice, logička vrednost neće moći sa sigurnošću biti protumačena ni kao logička nula ni kao logička jedinica. Stoga je preporučeno da se na ulazni pin dovode naponi čije su vrednosti za ATmega328p mikrokontroler definisani u tabeli 4.

Tabela 4. Naponski nivoi logičke nule i jedinice digitalnog ulaza

Parametar	**Oznaka**	**Uslov**	**Min**	**Max**
Ulazni napon logičke nule	V_{IL}	V_{CC}=2.7÷5.5 V	–0.5 V	$0.3V_{CC}$
Ulazni napon logičke jedinice	V_{IH}	V_{CC}=2.7÷5.5 V	$0.6V_{CC}$	V_{CC}+0.5 V

Prilikom povezivanja digitalnog ulaznog pina sa spoljašnjim kolima potrebno je osigurati da se na digitalni ulaz dovode propisani napon logičke nule i logičke jedinice. Ukoliko naponski nivoi spoljašnjeg kola nisu kompatibilni, neophodno je koristiti translatore nivoa kako bi naponski nivo bio ispravno protumačen od strane mikrokontrolera. U slučaju da je naponski nivo na ulazu viši od propisanog neophodno je koristiti naponski razdelnik, a u slučaju da je niži koristi se translator nivoa sa tranzistorom prikazan na slici 21.



Slika 21. Bidirekcioni translator nivoa

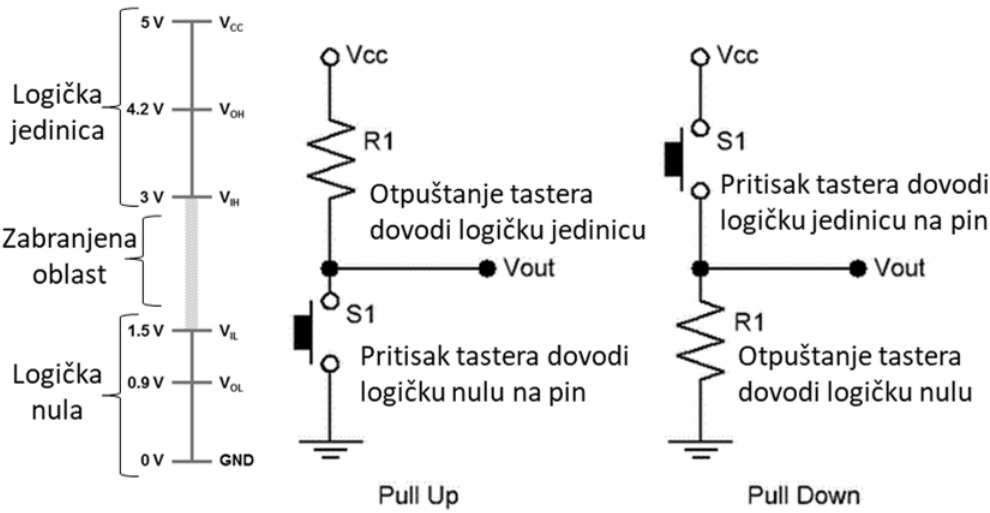
Bidirekcioni translator nivoa poseduje dva otpornika koja odvlače niskonaponsku i visokonaponsku stranu translatora na odgovarajuće naponske nivoe logičke jedinice. Kada niskonaponska strana povuče liniju na logičku nulu, otvara se MOSFET koji povlači i visokonaponsku stranu na stanje logičke nule. Takođe, kada visokonaponska strana povuče liniju

na logičku nulu, preko ugrađene diode u telu MOSFET tranzistora i niskonaponska strana biće povučena na nulu što će za posledicu imati otvaranje MOSFET tranzistora.

U slučaju da je na digitalni ulaz potrebno povezati taster neophodno je osigurati da su oba naponska nivoa jasno definisana i kada je taster pritisnut i kada je otpušten. U tu svrhu neophodno je koristiti pull-up ili pull-down otpornike koji ulaz odvode u odgovarajuće stanje po otpuštanju tastera prema tabeli 5 i slici 22. U slučaju da se ovi otpornici ne koriste po otpuštanju tastera ulaz bi zadržao logičku vrednost kao i kada je taster pritisnut, pri čemu bi zbog male kapacitivnosti ulaznog kola bilo kakve smetnje lako mogle da utiču na logičku vrednosti ulaza. Stoga, preporuka je da se digitalni ulazni pinovi nikada ne ostavljaju nepovezani.

Tabela 5. Logički nivoi pull-up i pull-down digitalnog ulaza

Ulazna konfiguracija	**Taster pritisnut**	**Taster otpušten**
Pull-up	0	1
Pull-down	1	0



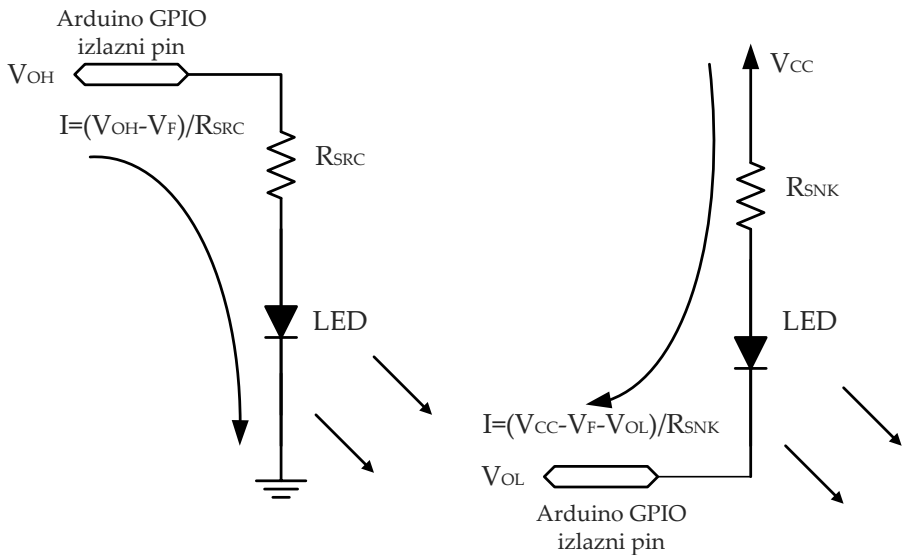
Slika 22. Pull-up i pull-down konfiguracija ulaznog GPIO pina

Kako bi se redukovao broj eksternih komponenti mikrokontroleri poseduju ugrađene otpornike koji mogu ostvarivati pull-up i/ili pull-down funkciju. U slučaju ATmega328p mikrokontrolera ulazni pinovi mogu se konfigurisati sa pull-up otpornikom čija je vrednost 20 ÷ 50 kΩ.

U slučaju da je GPIO port kontrolisan kao izlaz on može raditi u jednom od sledeća dva režima:

- izvorni režim (source) u kojem pri naponu logičke jedinice struja izlazi iz izalznog pina,
- ponorni režim (sink) u kojem pri naponu logičke jedinice struja ulazi u izlazni pin.

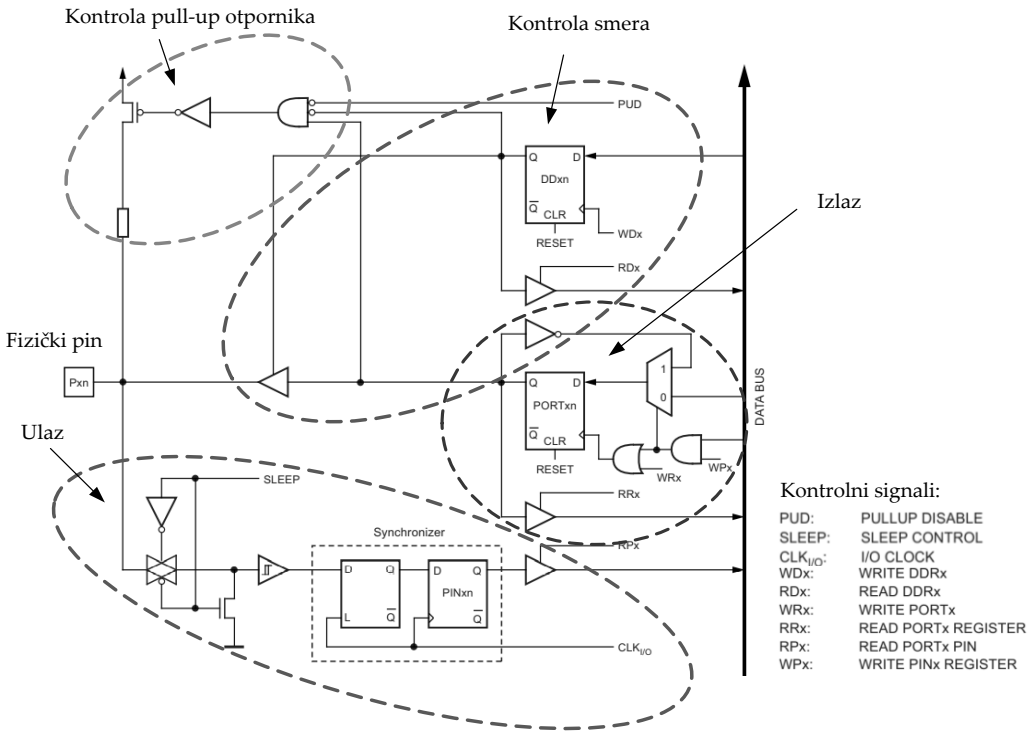
Na slici 23. prikazani su načini povezivanja svetlećih dioda na GPIO izlaz u izvornom i ponornom režimu. Prilikom povezivanja spoljašnjih kola potrebno je voditi računa da ne bi trebalo da maksimalna struja izlaznog pina pređe vrednost od 20 mA za pojedinačni pin, odnosno 100 mA za celokupan port.



Slika 23. Izvorni i ponorni režim izlaznog GPIO pina

Opšta struktura GPIO pina, prikazana na slici 24, sastoji se iz sledećih celina:

- fizički pin koji je povezan sa nožicom mikrokontrolera,
- kontrola smera pina koja se koristi kako bi se odabralo da li je pin ulazni ili izlazni,
- ulazni deo koji omogućava čitanje logičkog nivoa na pinu,
- izlazni deo koji omogučava podešavanje logičkog nivoa na pinu,
- pull-up otpornik koji obezbeđuje stanje logičke jedinice na pinu ukoliko je pin neaktivan.



Slika 24. Unutrašnja struktura GPIO pina Pxn

Svaki GPIO pin n porta x koristi po jedan bit u sledeća tri registra kojima se pristupa preko odgovarajuće I/O adrese registra:

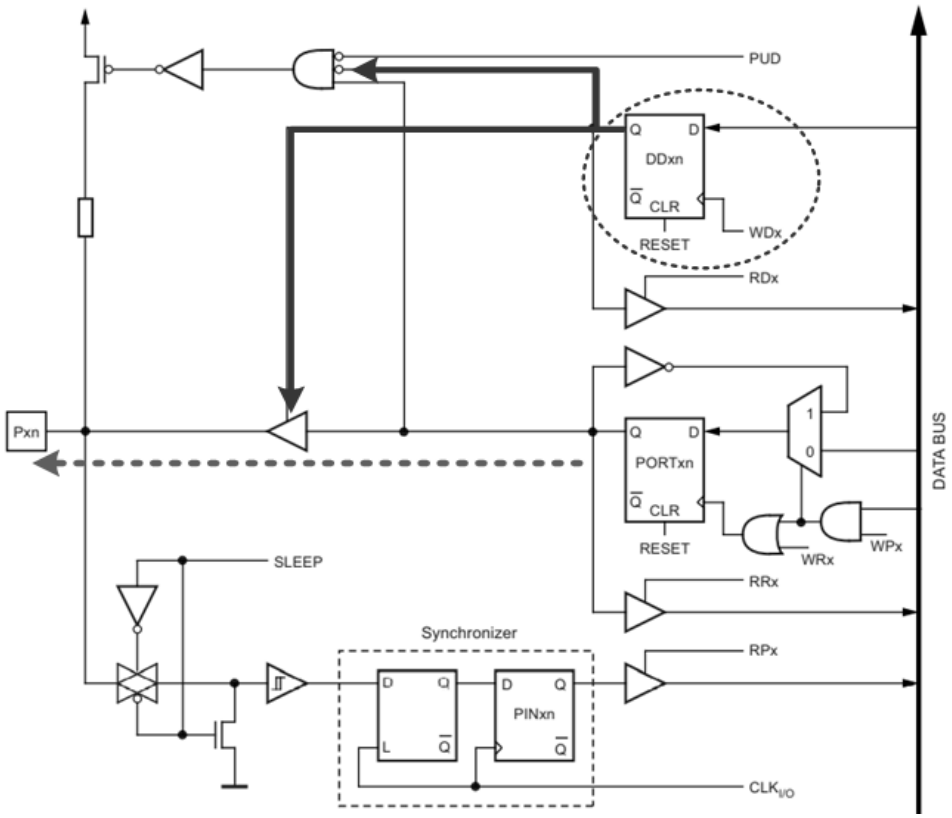
- DDxn bit u DDRx registru kontroliše smer pina,
- PORTxn bit u PORTx registru kontroliše izlaznu vrednost pina,
- PINxn bit u PINx registru sadrži ulaznu vrednost pina.

Prilikom operacija čitanja i upisa sadržaja u registrima može se pristupiti samo celokupnom osmobitnom sadržaju, pa u slučaju potrebe da se setuje, resetuje, invertuje ili proveri vrednost pojedinačnog bita u registru neophodno je koristiti odgovarajuće operacije sa bitskim maskama. Kod ovakvih operacija sadržaj registra najpre se pročita i nad njim se izvrši odgovarajuća logička operacija sa bitmaskom kao drugim operandom, nakon čega se dobijeni rezultat smešta u registar. Na ovaj se način čitaju i upisuju svih osam bitova, ali, zahvaljujući bitmasci, menja se samo bit od interesa. Kontrola smera GPIO pina obavlja se preko odgovarajućeg bita u registru DDRx. U slučaju da je DDxn bit jednak jedinici pin Pxn biće konfigurisan kao izlazni pin, a u slučaju da je jednak nuli pin Pxn biće konfigurisan kao ulazni pin prema tabeli 6.

Tabela 6. Kontrola smera digitalnog pina

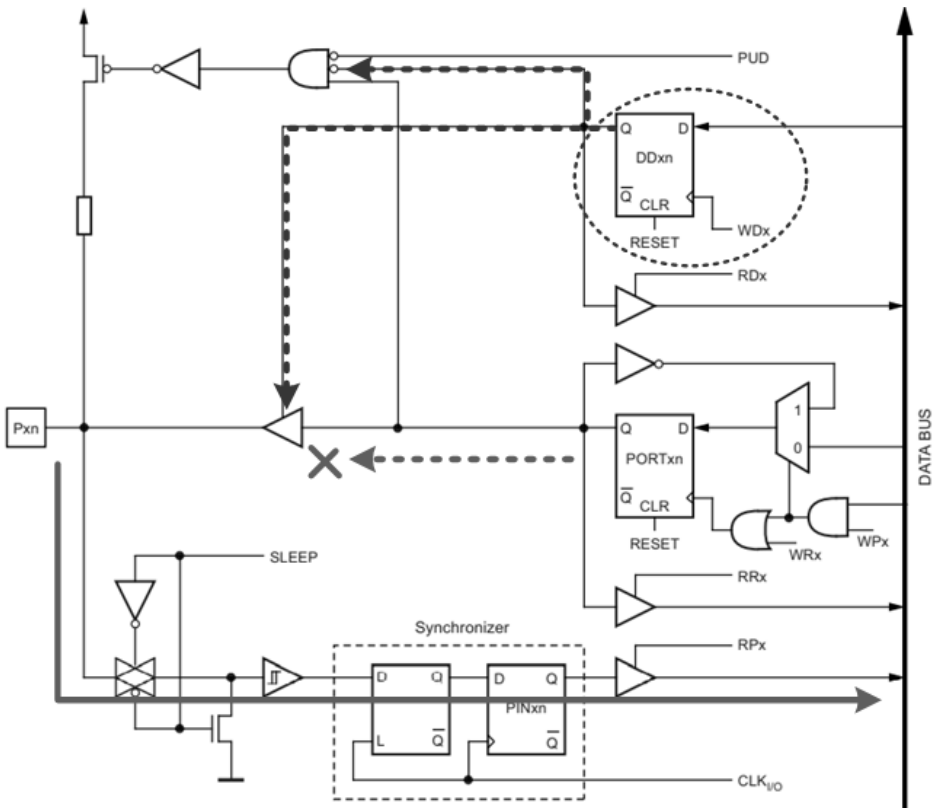
DDxn	PORTxn	Smer	Pull-up	Stanje fizičkog pina
0	0	Ulaz	Ne	Stanje visoke impedanse (Z)
0	1	Ulaz	Da	Pin na visokom nivou, dok se eksterno ne povuče na niski nivo
1	0	Izlaz	Ne	Izlaz na niskom nivou (sink)
1	1	Izlaz	Ne	Izlaz na visokom nivou (source)

Kontrola smera pina ostvaruje se preko DDxn flip-flopa (na slici 25) koji kada je setovan preko trostatičkog bafera dozvoljava pristup izlaznom flip-flopu PORTxn fizičkom pinu Pxn. U tom slučaju, kada je DDxn bit setovan, vrednost napona na fizičkom pinu biće kontrolisana od strane PORTxn flip-flopa. U slučaju da je PORTxn=1 pin će biti u stanju logičke jedinice ($V_{xn} > V_{OH}$), a ako je PORTxn=0 pin će biti u stanju logičke nule ($V_{xn} < V_{OL}$). U slučaju da je pin podešen kao izlazni Pull-up otpornik je isključen. Upisom logičke jedinice u bit PINxn registra invertuje se (eng. *toggle*) vrednost PORTxn flip flopa.



Slika 25. Konfigurisanje izlaznog Pxn pina

Ulazni deo stalno je povezan sa fizičkim pinom, pa je čitanje stanja fizičkog pina preko bita PINxn registra nezavisno od smera pina definisanog bitom DDxn. U slučaju da je DDxn flip-flop resetovan (slika 26) on će preko trostatičkog bafera onemogućiti pristup fizičkom pinu Pxn izlaznom PORTxn flip-flopu, pa će logička vrednost na fizičkom pinu biti zavisna samo od spoljašnjeg kola koje je povezano na fizički pin. Setovanjem PORTxn flip-flopa uključuje se pull-up otpornik, koji se može onemogućiti preko PUD bita (eng. *Pull-Up Disable*). Logika za čitanje stanja fizičkog pina, koja se nalazi ispred PINxn registra, koristi se kako bi se promena logičkog nivoa na fizičkom pinu sinhronizovala sa internim taktom mikrokontrolera. Sinhronizacija unosi kašnjenje koje se kreće od ½ do 1½ periode sistemskog takta u zavisnosti od trenutka promene.



Slika 26. Konfigurisanje ulaznog Pxn pina

Arduino omogućava jednostavnu kontrolu GPIO digitalnih pinova korišćenjem ugrađene wiring_digital biblioteke pomoću sledeće tri funkcije:

- *pinMode(pin, mode)* – promena smera pina,
- *digitalWrite(pin, val)* – upis izlazne vrednosti pina,
- *digitalRead(pin)* – čitanje ulazne vrednosti pina.

Kako bi se korisnicima olakšao razvoj aplikacija, Arduino IDE dozvoljava korišćenje makronaredbi za pristup digitalnim pinovima: *0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, (BUILT_IN), A0, A1, A2, A3, A4, A5* koje su definisane u *pins_arduino.h* datoteci zaglavlja.

Funkcija za definisanje smera *pinMode(pin, mode)* definiše pin kao ulazni ili izlazni. Argument mode može biti postavljen kao:

- OUTPUT,
- INPUT,
- INPUT_PULLUP.

Izvorni kod ove funkcije prikazan je u sledećem listingu:

```
void pinMode(uint8_t pin, uint8_t mode){
      uint8_t bit = digitalPinToBitMask(pin);
      uint8_t port = digitalPinToPort(pin);
      volatile uint8_t *reg, *out;

      if (port == NOT_A_PIN) return;

      reg = portModeRegister(port);
      out = portOutputRegister(port);

      if (mode == INPUT) {
            uint8_t oldSREG = SREG;
 cli();
            *reg &= ~bit;
            *out &= ~bit;
            SREG = oldSREG;
      } else if (mode == INPUT_PULLUP) {
            uint8_t oldSREG = SREG;
 cli();
            *reg &= ~bit;
            *out |= bit;
            SREG = oldSREG;
      } else {
            uint8_t oldSREG = SREG;
 cli();
            *reg |= bit;
            SREG = oldSREG;
      }
}
```

Pozivom funkcije *pinMode* najpre se za prosleđeni argument *pin* utvrđuje kom portu pripada i priprema se odgovarajuća bitska maska za pristup ciljanom bitu. Na osnovu utvrđenog porta određuju se odgovarajući DDx i PORTx registri. U slučaju da je potrebno postaviti pin kao ulazni (mode INPUT) sadržaj DDx i PORTx resetuje se korišćenjem bitske maske. U slučaju da je potrebno postaviti pin kao ulazni sa aktiviranim pull-up otpornikom (mode INPUT_PULLUP) resetuje se sadržaj DDx, a sadržaj registra PORTx setuje se korišćenjem bitske maske. U svim ostalim slučajevima (kada je mode OUTPUT ili drugo), sadržaj DDx se setuje kako bi se pin postavio kao izlazni. Može se primetiti da se pre manipulacije sa registrima pamti kontekst SREG statusnog registra i onemogućavaju se prekidi korišćenjem cli() funkcije kako bi se sprečila pojava prekida prilikom podešavanja smera pina. Nakon završetka podešavanja smera pina zapamćeni kontekst kopira se u SREG statusni registar.

Funkcija za čitanje vrednosti pina *digitalRead* vraća vrednost logičke jedinice (HIGH) ili logičke nule (LOW) u zavisnosti od napona na digitalnom pinu. Pozivom funkcije digitalRead prosleđeni argument pin utvrđuje kojem portu on pripada i priprema se odgovarajuća bitska maska za čitanje ciljanog bita. Čitanjem željenog bita preko bitske maske iz PINx registra u slučaju logičke jedinice vratiće se vrednost HIGH a u suprotnom LOW.

```
int digitalRead(uint8_t pin){
      uint8_t timer = digitalPinToTimer(pin);
      uint8_t bit = digitalPinToBitMask(pin);
      uint8_t port = digitalPinToPort(pin);

      if (port == NOT_A_PIN) return LOW;
      if (timer != NOT_ON_TIMER) turnOffPWM(timer);
      if (*portInputRegister(port) & bit) return HIGH;
      return LOW;
      }
```

Funkcija za upis izlazne vrednosti *digitalWrite(pin, value)* dodeljuje vrednost (HIGH ili LOW) registru PORT. Ako je port definisan kao izlazni (OUTPUT), napon na pinu biće 5 V za HIGH, odnosno 0 V za LOW. Ako je port konfigurisan kao ulazni, funkcija *digitalWrite(pin, value)* uključuje

(HIGH) ili isključuje (LOW) interni pull-up otpornik. Funkcija za upis vrednosti na pin *digitalWrite* najpre za prosleđeni argument pin utvrđuje kom portu pripada i priprema se odgovarajuća bitska maska za pristup ciljanom bitu. Zatim se čita vrednost PORTx registra i u zavisnosti od vrednosti (eng. *value*) koju treba upisati koristi se bitska maska za setovanje ili resetovanje željenog bita. I kod ove funkcije pamti se kontekst SREG statusnog registra i onemogućavaju prekidi korišćenjem cli() funkcije kako bi se sprečila pojava prekida prilikom promene izlazne vrednosti pina.

```
void digitalWrite(uint8_t pin, uint8_t val){
      uint8_t timer = digitalPinToTimer(pin);
      uint8_t bit = digitalPinToBitMask(pin);
      uint8_t port = digitalPinToPort(pin);
      volatile uint8_t *out;

      if (port == NOT_A_PIN) return;
      if (timer != NOT_ON_TIMER) turnOffPWM(timer);
      out = portOutputRegister(port);
      uint8_t oldSREG = SREG;
      cli();
      if (val == LOW) {
            *out &= ~bit;
      } else {
            *out |= bit;
      }
      SREG = oldSREG;
}
```

Može se uočiti da ugrađene biblioteke funkcija u značajnoj meri olakšavaju pisanje programa, ali sa stanovišta performansi one troše mnogo više procesorskog vremena i u slučaju potrebe za pisanjem efikasnijeg koda preporuka je da se vrši direktno podešavanje registara za kontrolu smera porta.

Primer korišćenja digitalnog pina za uključenje/isključenje ugrađene svetleće diode na GPIO pinu 13:

```
void setup() {
      pinMode(13, OUTPUT); // setuje pin 13 kao izlaz
      digitalWrite(13, LOW); // resetuje vrednost pina 13
}
void loop() {
      digitalWrite(13, HIGH); // setuje vrednost pina 13
      delay(1000); // čekanje od jedne sekunde
      digitalWrite(13, LOW); // resetuje vrednost pina 13
      delay(1000); // čekanje od jedne sekunde
      }
```

Vreme izvršavanja naredbi mikrokontrolera izuzetno je kratko. Amega328p mikrokontroler izvršava 16 miliona instrukcija u sekundi, što je jako brzo za čovekovo opažanje. Kako bi se izvršavanje naredbi prilagodilo čovekovom opažanju, potrebno je generisati određeno kašnjenje između sukcesivnih programskih naredbi. Za to se koristi funkcija softverksog kašnjenja *delay(ms)* koja pauzira izvršenje programa za dati parametar vremena u milisekundama. Ova funkcija predstavlja petlju u kojoj procesor ne radi ništa korisno već samo „troši" vreme izvršavajući niz praznih *yield()* funkcija u telu petlje.

```
void delay(unsigned long ms)
{
      uint32_t start = micros();
      while (ms > 0) {
            yield();
            while ( ms > 0 && (micros() - start) >= 1000) {
                  ms--;
                  start += 1000;
            }
      }
      }
```

POGLAVLJE 6: MEHANIZAM PREKIDA

Zamislimo mikrokontrolersku aplikaciju kod koje dolazi do promene boje trobojne svetleće diode (RGB LED diode) kada korisnik pritisne taster. U ovom primeru trobojna svetleća dioda povezana je tako da svaku komponentu boje, crvenu, zelenu i plavu, kontroliše poseban GPIO pin. Kombinacijom ovih izlaza moguće je generisati osam različitih boja prikazanih u sledećoj tabeli 7.

Tabela 7. Generisanje nijansi boje pomoću trobojne RGB svetleće diode

Boja	**Crvena**	**Zelena**	**Plava**
Bez boje	LOW	LOW	LOW
Plava	LOW	LOW	HIGH
Zelena	LOW	HIGH	LOW
Cijan	LOW	HIGH	HIGH
Crvena	HIGH	LOW	LOW
Magenta	HIGH	LOW	HIGH
Žuta	HIGH	HIGH	LOW
Bela	HIGH	HIGH	HIGH

Najjednostavnije način detekcije pritiska tastera jeste korišćenjem tehnike prozivanja (eng. *Polling*), koja se oslanja na proces uzastopnog čitanja vrednosti ulaznog pina korišćenjem funkcije *digitalRead*. Upoređivanjem trenutne vrednosti sa prethodno pročitanom vrednošću utvrđuje se da li je došlo do eksternog događaja, tj. pri promeni logičkog stanja sa ’1’ na ’0’ ili ’0’ u ’1’, kao što je prikazano u sledećem listingu koda.

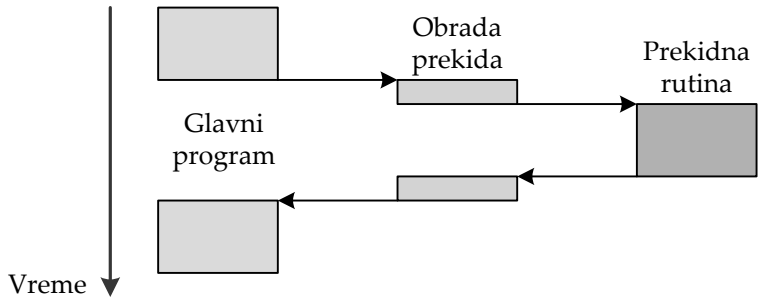
```
#define BUTTON 2
#define RED 5
#define GREEN 6
#define BLUE 7
```

```
int state=0,button_state;
void setup() {
      pinMode(BUTTON,INPUT_PULLUP);
      pinMode(RED, OUTPUT);
      pinMode(GREEN, OUTPUT);
      pinMode(BLUE, OUTPUT);
}
void loop() {
 if (digitalRead(BUTTON)==LOW && button_state == HIGH){
      state++;
      button_state=LOW;
      }
 if (state>=8)
      state=0;
 switch (state){
 case 0: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,LOW);
      }break;
 case 1: {
      digitalWrite(RED,LOW);
       digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,HIGH);
      }break;
 case 2: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,LOW);
      }break;
 case 3: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,HIGH);
      }break;
 case 4: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,LOW);
      }break;
 case 5: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,HIGH);
      }break;
 case 6: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,LOW);
      }break;
 case 7: {
      digitalWrite(RED,HIGH);
```

```
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,HIGH);
      }break;
 default: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,LOW);
      }break;
 }
 button_state=digitalRead(BUTTON);
 delay(300);
}
```

Nedostatak ove tehnike jeste spor odziv jer je neophodno eksplicitno proveriti da li je taster pritisnut. Kako bi se dobilo brže vreme odziva potrebno je češće proveravati status, što za posledicu ima trošenje procesorskog vremena. Vreme odziva zavisi od vremena potrebnog za celokupno procesiranje. Primenom ove tehnike teško je realizovati sistem sa većim brojem aktivnosti koje se mogu brzo odazvati.

Alternativni način za detekciju eksternih događaja jeste upotrebom prekida. Izvršavanje korisničkog programa diktirano je od strane programskog brojača (eng. *Program Counter*) koji pokazuje na adresu naredne instrukcije. Prekidi su eksterni asinhroni događaji koji nisu povezani sa kodom koji procesor trenutno izvršava. Kada se pojavi zahtev za prekid, specijalan hardver u mikrokontroleru provera da li su ispunjeni uslovi za obradu prekida i, ako jesu, završava se izvršenjem tekuće instrukcije i trenutni kontekst procesora pamti se na steku. Pri tom se pokreće rutina za servisiranje prekida ISR (eng. *Interrupt Service Routine*) gde procesor izvršava instrukcije prekidne rutine. Na kraju prekidne rutine nalazi se instrukcija za povratak, koja restaurira kontekst programa sa steka kako bi se nastavilo sa izvršenjem prekinutog dela programa (slika 27).



Slika 27. Postupak obrade prekida

Ova tehnika neuporedivo je brža i efikasnija od tehnike prozivanja, jer hardver obrađuje prekid neposredno po njegovoj pojavi, a kod se izvršava samo kada je to potrebno. Prekidna tehnika omogućava dobro skaliranje, jer vreme odziva ne zavisi od kompleksnosti programa, programski moduli koda mogu se razvijati nezavisno i moguće je definisanje prioriteta prekida.

Kod jednostavnih procesora može se izgubiti na vremenu reagovanja na prekid dok se utvrdi koja je periferija izazvala prekid – softverski se proveravaju svi izvori prekida u prekidnoj rutini. Vektorska tabela prekida sadrži početne adrese prekidnih rutina za svaki tip prekida koje procesor automatski poziva, čime se znatno skraćuje vreme reakcije na prekid.

ATmega328 mikrokontroler sadrži tabelu vektora prekida sa 26 različitih izvora prekida koji su locirani na početku adresnog prostora FLASH memorije. Svaki izvor prekida u tabeli 8. definiše JMP instrukciju apsolutnog skoka na adresu odgovarajuće prekidne rutine. Vektori prekida predstavljaju instrukcije apsolutnog skoka koje zauzimaju prostor od dve regularne instrukcije. Svaki izvor prekida poseduje prioritet koji je definisan njegovom adresom u tabeli, prekidi koji se nalaze na nižim adresama imaju viši prioritet, RESET ima najviši prioritet a zatim prekid INT0 (eng. *External Interrupt Request 0*). Prekid višeg prioriteta u stanju je da prekine izvršavanje prekidne rutine nižeg prioriteta.

Kada nastupi prekid, mikrokontroler završava tekuću instrukciju i smešta vrednost programskog brojača na stek. Zatim resetuje I-bit u SREG registru kako bi sprečio pokretanje obrade novih prekida dok traje obrada tekućeg prekida. Vreme reakcije na prekid za svaki tip prekida minimum je 4 taktna ciklusa i nakon četvrtog taktnog ciklusa započinje se sa izvršavanjem prve instrukcije prekidne rutine. Prekidna rutina izvršava se do njenog kraja kada se pozivom instrukcije *reti* vrši povratak iz prekidne rutine koji traje četiri taktna ciklusa, tokom kojih se dvobajtna vrednost programskog brojača uzima sa steka i setuje se I bit u SREG. Povratkom iz prekidne rutine uvek se izvršava jedna instrukcija pre nego što se pristupi servisiranju dodatnih prekida koji su na čekanju.

Ulaskom u prekidnu rutinu,programski se može setovati I bit kako bi omogućilo servisiranje ugnežđenih prekida. Svi prekidi koji imaju veći

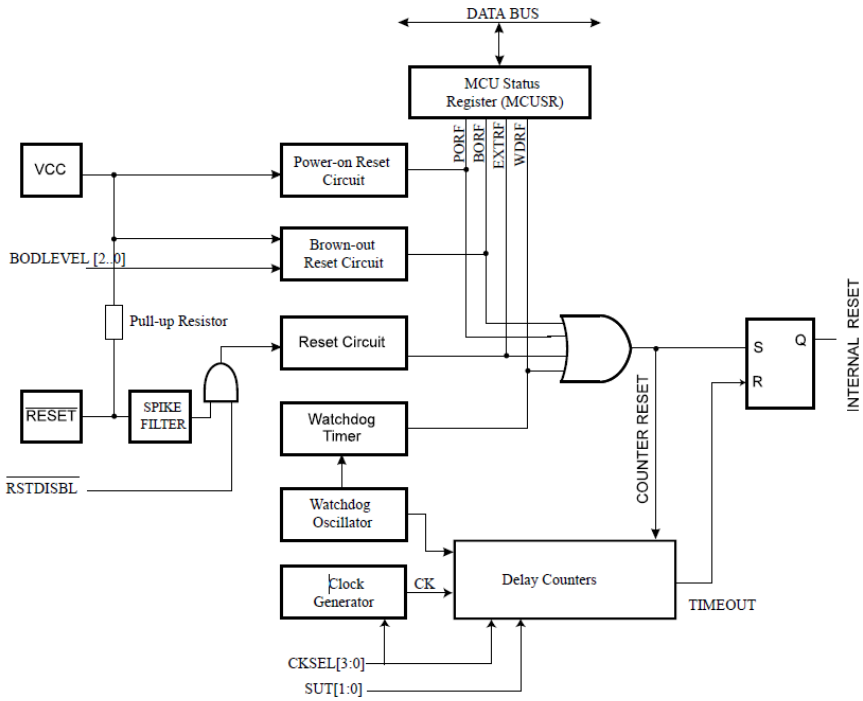
prioritet od tekućeg prekinuće ga, a servisiranje prekida nastaviće se povratkom u prekinutu servisnu rutinu. Setovanje I bita obavlja se preko SEI instrukcije, čiji su ekvivalentni funkcije *sei()* i *interrupts()* u Arduino IDE programskom okruženju. Resetovanje I bita obavlja se preko CEI instrukcije, čiji su ekvivalentni funkcije *cei()* i *noInterrupts ()* u Arduino IDE programskom okruženju.

Tabela 8. Vektorisana tabela prekida ATmega328p mikrokontrolera

R.br	Adresa	Instrukcija	Komentar
1	0x0000	jmp RESET	Reset
2	0x0002	jmp EXT_INT0	Eksterni prekid 0
3	0x0004	jmp EXT_INT1	Eksterni prekid 1
4	0x0006	jmp PCINT0	Promena na portu 0
5	0x0008	jmp PCINT1	Promena na portu 1
6	0x000A	jmp PCINT2	Promena na portu 2
7	0x000C	jmp WDT	Sigurnosni brojač
8	0x000E	jmp TIM2_COMPA	Tajmer 2 poređenje A
9	0x0010	jmp TIM2_COMPB	Tajmer 2 poređenje B
10	0x0012	jmp TIM2_OVF	Tajmer 2 prekoračenje
11	0x0014	jmp TIM1_CAPT	Tajmer 1 beleženje
12	0x0016	jmp TIM1_COMPA	Tajmer 1 poređenje A
13	0x0018	jmp TIM1_COMPB	Tajmer 1 poređenje B
14	0x001A	jmp TIM1_OVF	Tajmer 1 prekoračenje
15	0x001C	jmp TIM0_COMPA	Tajmer 0 poređenje A
16	0x001E	jmp TIM0_COMPB	Tajmer 0 poređenje B
17	0x0020	jmp TIM0_OVF	Tajmer 0 prekoračenje
18	0x0022	jmp SPI_STC	Kraj SPI prenosa
19	0x0024	jmp USART_RXC	Završetak USART prijema
20	0x0026	jmp USART_UDRE	USART bafer prazan
21	0x0028	jmp USART_TXC	Završetak USART slanja
22	0x002A	jmp ADC	Završetak A/D konverzije
23	0x002C	jmp EE_RDY	Završetak EEPROM upisa
24	0x002E	jmp ANA_COMP	Promena na komparatoru
25	0x0030	jmp TWI	TWI prekid
26	0x0032	jmp SPM_RDY	Završetak SPM upisa

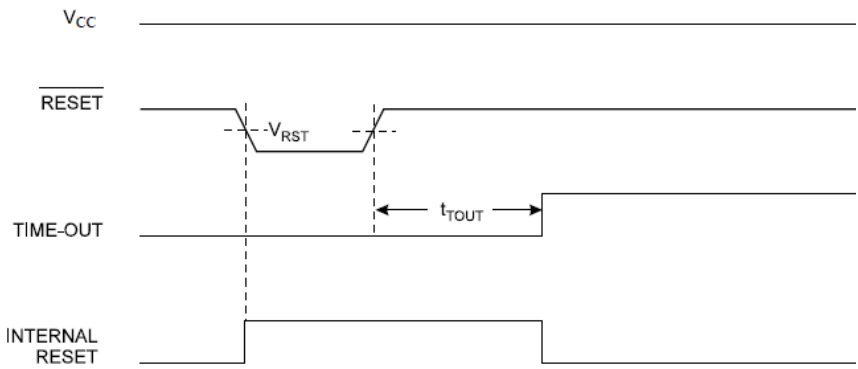
Reset mikrokontrolera

Izvor prekida sa najvećim prioritetom jeste Reset koji predstavlja događaj ponovnog pokretanja izvršenja korisničke aplikacije. Tokom resetovanja svi ulazno-izlazni registri postavljaju se na podrazumevane inicijalne vrednosti i program započinje izvršavanje sa adrese 0x0000 koja je specificirana RESET vektorom. Na toj adresi nalazi se instrukcija bezuslovnog skoka u prekidnu rutinu za servisiranje reseta. Reset događaj nastaje setovanjem internog reset signala koji se u mikrokontroleru može generisati iz različitih izvora prikazanih na slici 28.



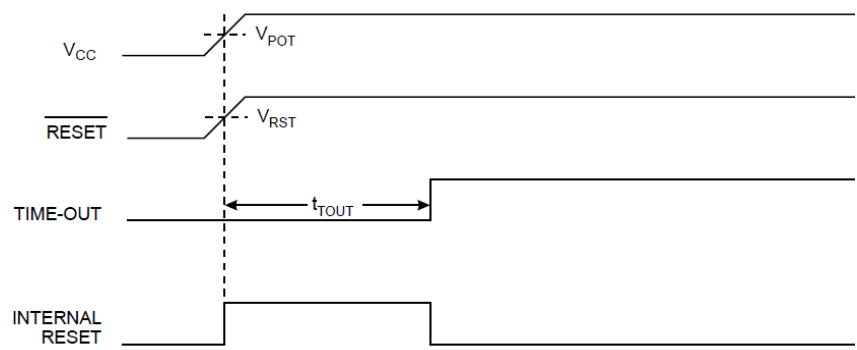
Slika 28. Kolo za reset mikrokontrolera

- Eksterni reset nastaje kada se PC6 pin dovede na stanje logičke nule (napon na pinu manji od napona V_{RST}=0.2Vcc) u trajanju od bar 2.5 μs (slika 29). Arduino UNO razvojni sistemu poseduje RESET tastera za ekterni reset. Kako bi se sprečilo slučajno resetovajne, ovaj se ulaz drži na stanju logičke jedinice korišćenjem eksternog pull-up otpornika od 10kΩ. Korišćenjem bita RSTDISBL može se isključiti funkcija eksternog reseta na PC6 pinu i on se tada može koristiti kao regularni GPIO pin.



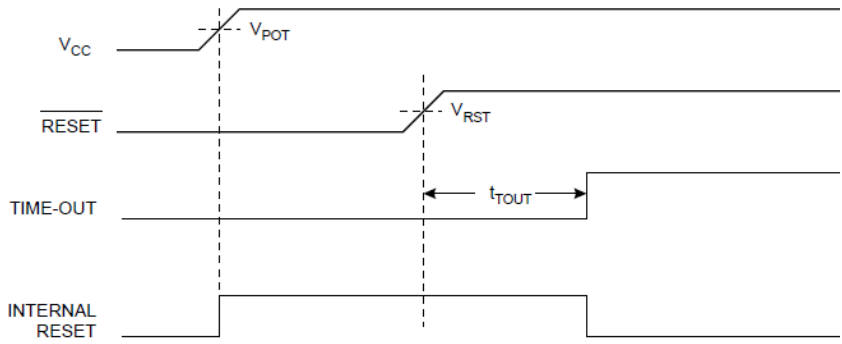
Slika 29. Eksterni reset mikrokontrolera

- Reset po uključenju (eng. *Power-on Reset*) služi kako bi se sprečio početak izvršavanja programa neposredno po uključenju mikrokontrolera. Po uspostavljanju napona napajanja Vcc, mikrokontroler se drži u stanju reseta dok napon napajanja ne pređe vrednost VPOT (eng. *Power-On reset Threshold*) i generator takta ne počne da generiše stabilni signal takta (slika 30). Kada se uspostavi ispravan rad oscilatora, interni brojač odbrojava potreban broj impulsa i nakon isteka specificiranog intervala mikrokontroler se otpušta iz stanja reseta i započinje izvršavanje prve instrukcije programa. Ovim se sprečava da mikrokontroler započne sa izvršavanjem programa dok se nalazi u nestabilnom stanju po uključenju koje može izazvati nepredviđeno ponašanje.



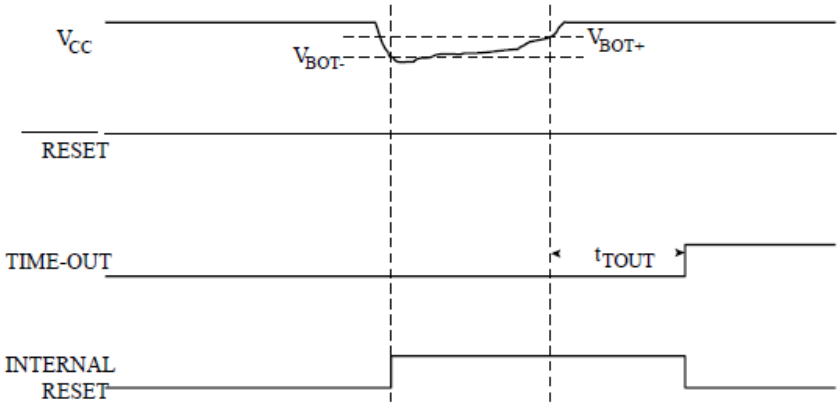
Slika 30. Reset mikrokontrolera po uključenju

Ukoliko postoji potreba reset po uključenju mikrokontrolera može se dodatno produžiti korišćenjem eksternog signala za reset koji dolazi iz spoljašnjeg kola (slika 31).



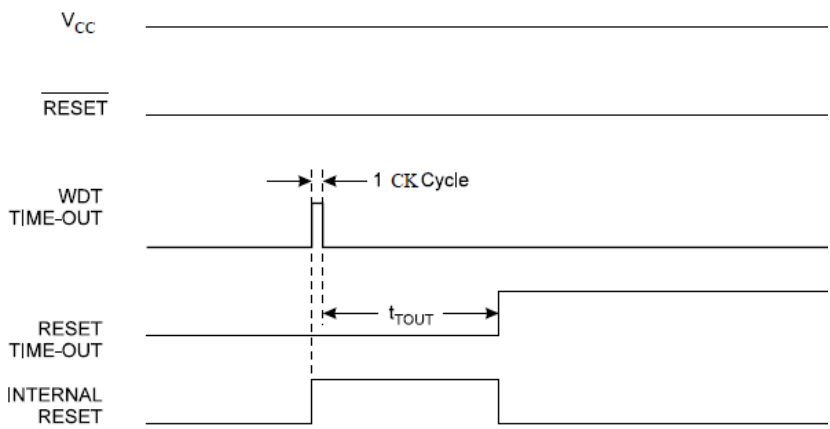
Slika 31. Produženi reset mikrokontrolera po uključenju

- Reset po gubitku napajanja (eng. *Brown-out Reset*) nastaje kada napon napajanja mikrokontrolera tokom rada opadne ispod neke bezbedne granice što može izazvati gubitak podataka zapamćenih u registrima i memoriji. Kako bi se sprečilo da takav događaj izazove nepredviđeno ponašanje mikrokontrolera, koristi se BOD kolo koje tokom rada mikrokontrolera nadzire napon napajanja Vcc i poredi ga sa VBOT naponom (eng. *Brown-Out reset Threshold*) čija se vrednost može podesiti pomoću BODLEVEL bitova. Kako bi se osigurao ispravan rad, napon resetovanja VBOT poseduje dva naponska praga, $V_{BOT+} = V_{BOT} + V_{HYST}/2$ i $V_{BOT-} = V_{BOT} - V_{HYST}/2$. Kada napon napajanja opadne ispod V_{BOT-} nivoa u trajanju od t_{BOD}, dolazi do momentalnog reseta mikrokontrolera, a kada napon napajanja premaši vrednost V_{BOT+}, mikrokontroler će izaći iz reseta nakon što interni brojač odbroji potreban broj impulsa (slika 32).



Slika 32. Reset mikrokontrolera po gubitku napajanja

- Reset od strane sigurnosnog brojača (eng. *Watchdog System Reset*) koristi se kako bi se omogućilo resetovanje mikrokontrolera u slučaju nepravilnog izvršavanja korisničke aplikacije. Ovaj tip reseta zasnovan je na sigurnosnom brojaču koji je taktovan iz nezavisnog izvora i koji konstantno odbrojava impulse. U toku normalnog rada potrebno je periodično vršiti resetovanje sigurnosnog brojača korišćenjem namenske instrukcije Watchdog Timer Reset (WDR), kako bi se sprečilo da on dođe do maksimalne vrednosti i izazove reset mikrokontrolera. U slučaju nepravilnog izvršavanja izazvanog nekom softverskom greškom neće doći do resetovanja sigurnosnog brojača, on će doći do maksimalne vrednosti i resetovati mikrokontroler (slika 33). Zahvaljujući sigurnosnom brojaču mikrokontroler će nakon reseta moći ponovo nastaviti da kontroliše ugrađeni sistem.



Slika 33. Reset mikrokontrolera od strane sigurnosnog brojača

Eksterni prekidi

Eksterni prekidi generisani su od strane pojedinačnih pinova INT0 i INT1 ili promenom stanja pina PCINTx na portovima B, C i D. INT0 i INT1 eksterni prekidi jesu prekidi sa najvećim prioritetom posle reseta i u slučaju ATmega328p mikrokontrolera oni se nalaze na pinovima PD2 i PD3 porta D, odnosno pinovima 2 i 3 Arduino UNO razvojnog sistema. Korišćenjem EICRA (eng. *External Interrupt Control Register*) registra može se nezavisno za svaki od ova dva prekida podesiti uslov pod kojim dolazi do prekida.

U slučaju INT0 eksternog prekida podešavaju se bitovi ISC01i ISC00 prema tabeli 9, dok se za INT1 eksterni prekid podešavaju se bitovi ISC11i ISC10.

Tabela 9. Uslov za generisanje INTx eksternog prekida

ISCx1	ISCx0	Uslov za generisanje prekida
0	0	Nizak nivo na INTx generiše zahtev za prekid
0	1	Promena nivoa na INTx generiše zahtev za prekid
1	0	Silazna ivica na INTx generiše zahtev za prekid
1	1	Uzlazna ivica na INTx generiše zahtev za prekid

Prekidna rutina eksternih prekida definiše se kao posebna funkcija koja se ne poziva nigde u programu, već se pri prevođenju povezuje sa vektorskom tabelom prekida upotrebom funkcije *attachInterrupt(interrupt, ISR, mode)*. Parametri ove funkcije jesu:

- *interrupt* predstavlja redni broj izvora eksternog prekida. Preporuka je da se za prosleđivanje ove vrednosti koristi povratni rezultat funkcije *digitalPinToInterrupt(pin)*, koja će za pin 2 vratiti vrednost 0 (INT0) a za pin 3 vratiti vrednost 1 (INT1),
- *ISR* predstavlja pokazivač na naziv funkcije koja predstavlja prekidnu rutinu za obradu eksternog prekida,
- *mod* predstavlja uslov pod kojim dolazi do prekida i on može imati neku od navedenih vrednosti:
    - **LOW** – pokreće ISR kada je pin na stanju logičke nule,
    - **CHANGE** – pokreće ISR kada je došlo do promene logičkog stanja pina,
    - **FALLING** – pokreće ISR na silaznu ivicu na pinu,
    - **RISING** – pokreće ISR na uzlaznu ivicu na pinu.

Ukoliko je potrebno u toku rada može se i odjaviti prekidna rutina korišćenjem funkcije *detachInterrupt(interrupt)*.

U sledećem listingu koda prikazan je način korišćenja eksternih prekida na pinu 2 za promenu boje trobojne svetleće diode.

```
#define BUTTON 2
#define RED 5
#define GREEN 6
#define BLUE 7
int state=0;
```

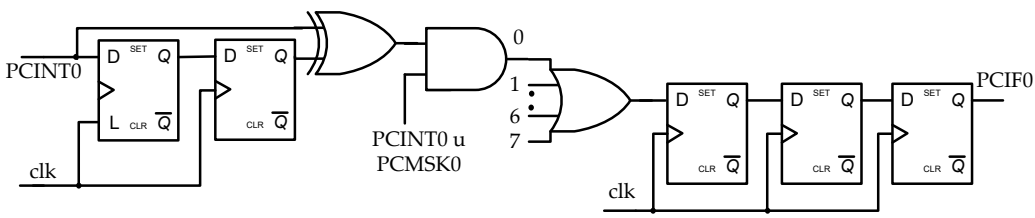
```
void change_state() {
 state++;
 if (state>=8)
    state=0;
}

void setup() {
 pinMode(BUTTON,INPUT_PULLUP);
 pinMode(RED, OUTPUT);
 pinMode(GREEN, OUTPUT);
 pinMode(BLUE, OUTPUT);
 attachInterrupt(digitalPinToInterrupt(BUTTON), change_state,
FALLING);
   }

void loop() {
 switch (state){
 case 0: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,LOW);
      }break;
 case 1: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,HIGH);
      }break;
 case 2: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,LOW);
      }break;
 case 3: {
      digitalWrite(RED,LOW);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,HIGH);
      }break;
 case 4: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,LOW);
      }break;
 case 5: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,LOW);
      digitalWrite(BLUE,HIGH);
      }break;
 case 6: {
      digitalWrite(RED,HIGH);
      digitalWrite(GREEN,HIGH);
      digitalWrite(BLUE,LOW);
```

```
        }break;
  case 7: {
        digitalWrite(RED,HIGH);
        digitalWrite(GREEN,HIGH);
        digitalWrite(BLUE,HIGH);
        }break;
  default: {
        digitalWrite(RED,LOW);
        digitalWrite(GREEN,LOW);
        digitalWrite(BLUE,LOW);
        }break;
    }
}
```

Bilo koja promena na nekom od pinova porta može biti detektovana korišćenjem kola za detekciju ivice koje se sastoji od XOR kola koje detektuje promenu trenutnog u odnosu na prethodno stanje pina. Ulazni leč na slici 34. pamti trenutnu vrednost pina, koja se prosleđuje sledećem flip-flopu koji tu vrednost pamti kao prethodnu. XOR kolo poredi prethodnu i trenutnu vrednost i, ako se one razlikuju (postoji promena), generisaće se logička jedinica. Prekid se može maskirati (omogućiti) za svaki pojedinačni pin korišćenjem odgovarajućeg PCINT bita u PCMSKx registru. Maskirani prekidi osam pinova dovode se na logičko ILI kolo nakon kojih se sinhronišu sa signalom takta kako bi generisali zbirni zahtev za prekid PCIF. Ovaj vid prekida grupisan je po portovima tako da PCINT0 ima najviši prioritet a PCINT2 ima najniži prioritet. PCINT0 će generisati prekid ako se promeni bilo koji od PCINT[7:0] pinova, koji se nalaze na portu PB [7:0]. PCINT1 će generisati prekid ako se promeni bilo koji od PCINT[14:8] pinova, koji se nalaze na portu PC [6:0]. PCINT2 će generisati prekid ako se promeni bilo koji od PCINT[23:16] pinova, koji se nalaze na portu PD [7:0].



Slika 34. Detekcija prekida na promenu stanja pina

Prekid će signalizirati da je došlo do promene na celom portu a programer mora proveriti koji je tačno pin odgovoran za promenu stanja. Prekidi na promenu stanja porta nisu podržani ugrađenim funkcijama Arduino IDE okruženja i oni se mogu koristiti ili direktnim pristupom registrima PCICR, PCIFR, PCMSK2, PCMSK1, PCMSK0 ili korišćenjem dodatnih biblioteka.

POGLAVLJE 7: BINARNI BROJAČKI MODULI

Binarni brojački moduli, predstavljaju jednu od bitnijih periferija mikrokontrolera, koje se koristi za precizno merenje vremena ili brojanje događaja. Mikrokontroler može posedovati nekoliko ovih brojača koji mogu biti širine 8 ili 16 bita.

Softversko brojanje i merenje proteklog vremena

Merenje vremena i generisanje vremenske zadrške moguće je ostvariti softverski, korišćenjem funkcija za čekanje (eng. *delay*). Funkcija za čekanje realizovana je pomoću petlje u kojoj procesor provodi neko zahtevano vreme izvršavajući iteracije petlje čije je vreme izvršavanja unapred poznato (najčešće se u telu petlje izvršavaju NOP instrukcije). Broj iteracija petlje direktno je srazmeran proteklom vremenu, kako je prikazano u sledećem listingu koda.

```
void delay_ms(unsigned long t)
unsigned long i:
while (i<X*t) //X iteracije petlje u 1ms
      i++;
return;
```

Brojanje događaja takođe se može realizovati softverski, pri čemu bi program morao stalno proveravati stanje ulaza kako bi prebrojao svaku njegovu promenu kao što je prikazano u sledećem listingu koda.

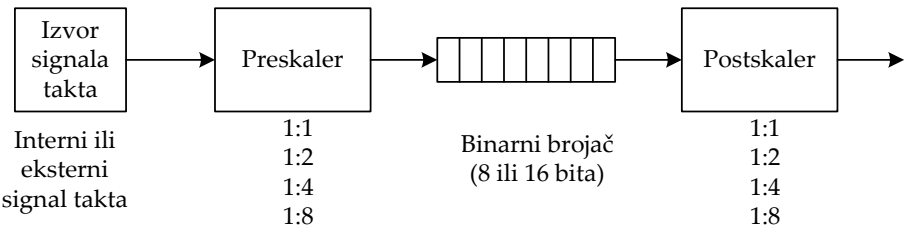
```
unsigned int count;
      loop() {
      while (!Button); // čekanje na pritisak tastera
            count++;
            }
```

Softverska implementacija onemogućava realizaciju aplikacija koje izvršavaju višestruke nezavisne zadatke (multitasking) koji su česti kod mikrokontrolerskih sistema. Stoga, softversko merenje vremena nepraktično je u slučaju potrebe za konkurentnim izvršavanjem više zadataka.

Binarni brojački moduli

Navedene probleme vezane za merenje vremena i brojanje događaja mnogo se efikasnije mogu rešavati korišćenjem ugrađene periferije koja sadrži binaran brojač. Ovaj modul mikrokontrolera zasnovan je na podesivom binarnom brojaču, koji može raditi kao tajmer ili brojač. U slučaju da radi u režimu tajmera ovaj modul koristi sistemski takt kao izvor impulsa za binarni brojač. S obzirom da je frekvencija sistemskog takta konstantna, tajmerski režim rada može se koristiti za precizno merenje vremena. U slučaju rada u brojačkom režimu izvor impulsa za binarni brojač jesu asinhroni događaji koji se dovode preko pina mikrokontrolera i u ovom slučaju modul se koristi za precizno brojanje ovih impulsa.

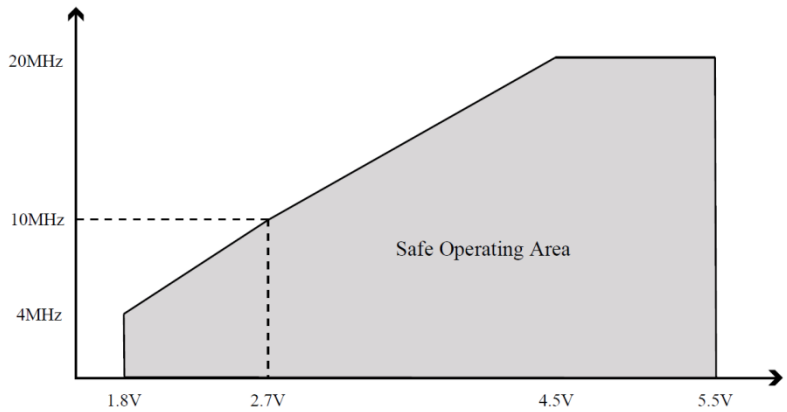
Opšta struktura binarnog brojača prikazana na slici 35, sastoji se od izvora taktnih impulsa, preskalera i postkalera i binarnog brojača. Preskaler i postkaler digitalni su delitelji frekvencije i njihova je uloga da smanje frekvenciju ulaznog signala za podesivi faktor skaliranja.



Slika 35. Opšta struktura binarnog brojača

Tajmerski modul koristi se za precizno merenje vremena, pri čemu preciznost merenja najviše zavisi od preciznosti generisanih impulsa od strane oscilatora mikrokontrolera. Oscilator predstavlja električno kolo koje proizvodi taktni signal određene frekvencije. Oscilatori mogu biti izrađeni na bazi mehaničkih rezonatora ili električnih fazno pomerenih oscilatora na bazi RC kola.

Maksimalna radna frekvencija Atmega328p mikrokontrolera iznosi 20 MHz pri naponu napajanja većem od 4.5 V. U slučaju da je napon napajanja niži, prema dijagramu sa slike 36, radna frekvencija opada tako da je pri naponu napajanja od 2.7 V maksimalna radna frekvencija 10 MHz. Arduino UNO razvojni sistem koristi oscilator od 16 MHz tako da u slučaju napajanja putem baterije ne bi trebalo da napon napajanja bude niži od 3.8 V.

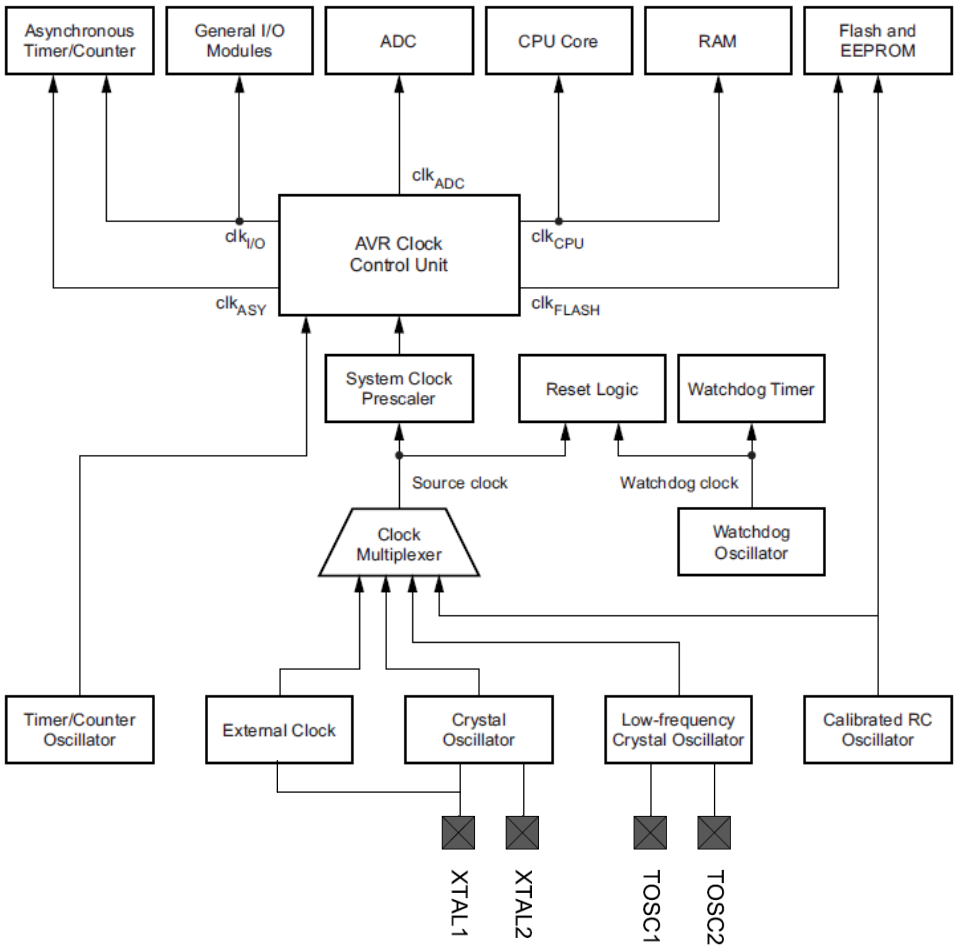


Slika 36. Maksimalna radna frekvencija u zavisnosti od napona napajanja

RC oscilatori predstavljaju oscilatore koje generišu taktni signal čija je frekvencija proporcionalna vrednostima korišćene otpornosti i kapacitivnosti u oscilatoru. Ovi su oscilatori vrlo kompaktni i poseduju jako kratko vreme oscilovanja, pa se često integrišu u sam mikrokontroler kao interni RC oscilator takta. Ovaj tip oscilatora ne zahteva eksterne komponente, što je jako pogodno za uređaje malih dimenzija. Međutim, tačnost generisanja frekvencije veoma je zavisna od napona napajanja i temperature i može varirati od 5 do 50 % nominalne frekvencije oscilatora. Ovakvi tipovi oscilatora mogu se realizovati i kao LR i LC konfiguracije ali su one nepraktične zbog velikih dimenzija koje zahteva prigušnica.

Atmega328p mikrokontroler za sistemski takt jezgra mikrokontrolera može koristiti takt iz internog kalibrisanog RC oscilatora frekvencije 8 MHz ili iz eksternih oscilatora koji se povezuju na pinove RB6 i RB7. Kao eksterni oscilatori mogu se koristiti niskofrekventni kristalni oscilatori, visokofrekventni kristalni oscilatori, ili se može koristiti takt iz eksternog oscilatora koji se dovodi preko pina RB6. Izvor takta se određuje preko

multipleksera na slici 37, koji je kontrolisan od strane CKSEL (eng. *Clock Prescaler Register*) bitova. Sistemski takt može se usporiti korišćenjem podesivog sistemskog preskalera koji se kontroliše preko CLKPR (eng. *Clock Prescaler Register*) registra u koracima od 1:1 do 1:256.



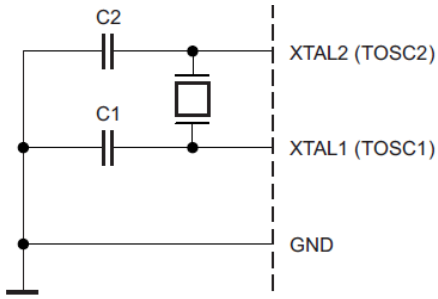
Slika 37. Izvori i distribucija sistemskog takta

Atmega328p poseduje interni kalibrisani RC oscilator frekvencije 8 MHz i interni oscilator niske potrošnje radne frekvencije 128 kHz. Tačnost (stabilnost) frekvencije oscilatora najčešće se izražava u relativnim jedinicama ppm (eng. *parts per million*) koje predstavljaju odnos odstupanja frekvencije u Hz i nominalne frekvencije oscilatora izražene u MHz. Interni kalibrisan RC oscilator generiše taktni signal frekvencije 8 MHz sa tačnošću od ±10% (±100000 ppm) pri naponu napajanja od 3 V i temperaturi od 25 °C.

Ovaj je oscilator kalibrisan prilikom procesa proizvodnje, ali tačnost oscilatora može se menjati usled uticaja temperature i napona napajanja, kao i usled starenja komponenti. U većini slučajeva potrebno je da korisnik izvrši kalibraciju kojom se može povećati tačnost izlazne frekvencije do ±1 % (±10000 ppm). Kalibracija internog oscilatora može se izvesti upisom kalibracione vrednosti u OSCCAL (eng. *Oscillator Calibration Register*) registar. U ovom registru upisana je inicijalna kalibraciona vrednost određena u procesu proizvodnje mikrokontrolera. Proces kalibracije podrazumeva dovođenje signala iz nekog referentnog izvora (može se koristiti signal generator ili napon iz distributivne mreže) na ulaz mikrokontrolera pri čemu se korišćenjem posebno napisane aplikacije meri vreme trajanja ovih impulsa korišćenjem internog RC oscilatora. Pošto je vreme trajanja impulsa iz referentnog izvora poznato, pojaviće se razlika u merenju vremena korišćenjem internog RC oscilatora, pa je potrebno izračunati vrednost za kalibraciju takta i upsati u OSCCAL registar. Vrednost u ovom registru prikazana je u formatu gde je bit 7 predviđen za vrednost znaka (0 za negativnu, a 1 za pozitivnu korekciju frekvencije), dok binarna vrednost preostalih bitova predstavlja korekcionu vrednost. Korekciju je preporučeno uraditi iterativno, pri čemu se vrednost OSCCAL inkrementira/dekrementira u koracima od 1 (koje odgovaraju promeni frekvencije od 50kHz), pa se vrši ponovno merenje. Ukoliko se interno kalibirsani RC oscilator sa tačnošću izlazne frekvencije ±1 % (±10000 ppm) koristi kao izvor takta, greška u merenju vremena na nivou jednog časa iznosi ±36 sekundi, a na nivou jednog dana ±14.4 minuta što je jako neprecizo za bilo kakvu prezizniju upotrebu. Drugi interni oscilator male snage generiše taktni signal frekvencije 128 kHz pri naponu napajanja od 3 V i temperaturi 25 °C i ovaj interni izvor takta dosta je neprecizniji i nema mogućnost kalibracije.

Drugi tip oscilatora koji se koristi kao izvor takta kod mikrokontrolera jesu mehanički rezonatori koji rade na principu piezoelektričnog efekta. Oni se izrađuju od keramike ili kristala kvarca, tako što se preciznim zasecanjem kristala definiše njegova rezonantna učestanost. Kada se kvarcni kristal pobudi električnim signalom počeće da vibrira na rezonantnoj učestanosti koja zavisi od njegovih dimenzija. Ako se kvarcni

kristal postavi u kolo oscilatora sa povratnom spregom, oscilator će početi da generiše taktni signal sa rezonantnom učestanošću oscilovanja koja je jednaka rezonantnoj učestanosti kristala kvarca. Kvarcni oscilatori poseduju znatno bolju tačnost generisane frekvencije sa greškom koja je reda ±0.01 % (±100 ppm). Faktori koji utiču na tačnost kvarcnog oscilatora jesu temperatura, mehanički stres i starenje. Ukoliko bi se kao izvor takta koristio kvarcni oscilator sa tačnošću izlazne frekvencije od ±100 ppm, greška u merenju vremena na nivou jednog dana iznosila bi ±8.5 sekundi što je dosta preciznije u odnosu na RC oscilatore. Mikrokontroler ATmega328p poseduje ugrađeno kolo oscilatora, samo je neophodno povezati kvarcni kristal na pinove RB6 (XTAL1) i RB7(XTAL2) zajedno sa dva kondenzatora od 22 nF, kao na slici 38.

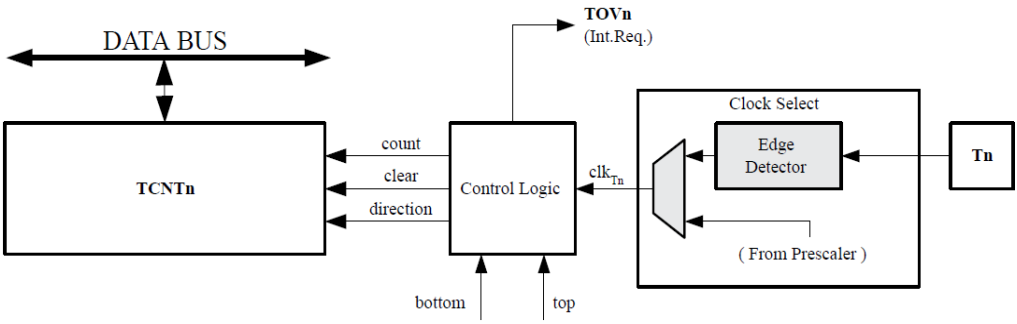


Slika 38. Povezivanje kristala kvarca sa ugrađenim kolom oscilatora

U slučaju da je u aplikaciji potrebno koristiti preciznije oscilatore za merenje vremena, može se koristiti temperaturno kompenzovani digitalni oscilator koji poseduje tačnost od ±10 ppm. Ovaj tip oscilatora meri temperaturu kvarcnog oscilatora, na osnovu koje vrši kompenzaciju izlazne frekvencije. Još precizniji tip oscilatora jeste temperaturno stabilisani kvarcni oscilator, koji održava kvarcni kristal na tačno definisanoj temperaturi korišćenjem grejača i na taj način obezbeđuje tačnost generisane frekvencije ±0.1 ppm. Nedostatak ove vrste oscilatora jeste to što mu je potrebno određeno vreme da se kristal zagreje na željenu temperaturu. Takođe, ovakvi oscilatori većih su dimenzija i troše više energije za svoj rad u odnosu na nekompenzovane kvarcne oscilatore. Preciznije izvedbe oscilatora podrazumevaju GPS sinhronizovani temperaturno stabilisani oscilator, koji koristi signal sa GPS satelita na kojima se nalaze precizni atomski časovnici i njime koriguje takt kvarcnog

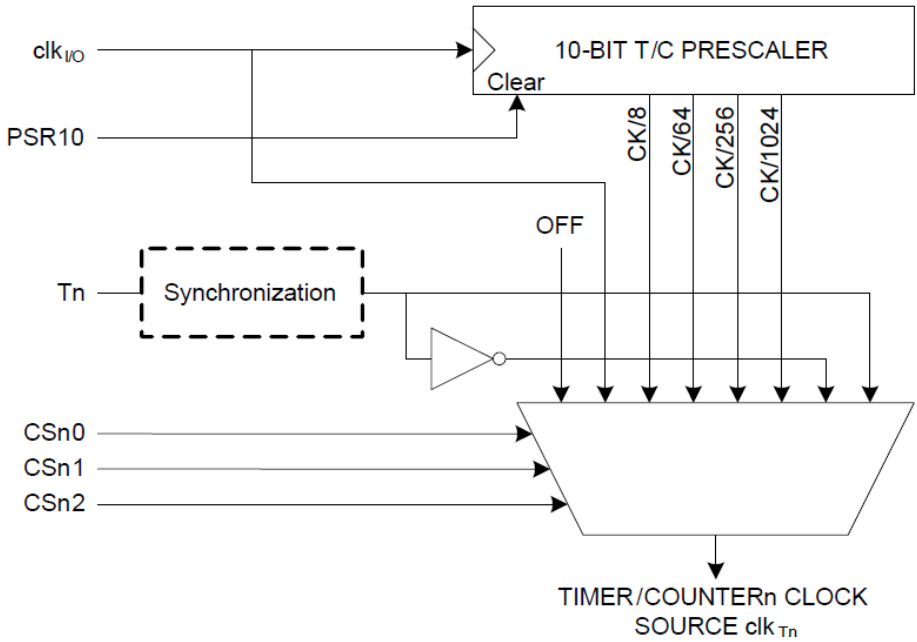
oscilatora i postiže tačnost generisanja frekvencije ±0.01 ppm. Nedostatak GPS disciplinovanog oscilatora jeste to što zahteva prijem GPS signala preko antene i ne može raditi u zatvorenim prostorima. Najpreciznji oscilatori jesu atomski časovnici na bazi cezijuma i rubidijuma koji poseduju tačnost generisanja frekvencije od ±0.0001 ppm, ali su oni vrlo kompleksni i skupi. Jedan od načina kojima se može meriti vreme baziran je na merenju frekvencije od 50 Hz na kojoj radi distributivna mreže. Tokom dana se u centru u Švajcarskoj prati frekvencija distributivne mreže koja mora imati 4320000 perioda (50 Hz × 60 s/min × 60 min/h × 24 h/danu) i ta se vrednost koriguje svakog dana u 8 časova ujutru tako što se povećava ili smanjuje frekvencija celokupne distributivne mreže povećanjem ili smanjenjem aktivne snage mreže.

Atmega328p mikrokontroler poseduje dva 8-bitna TC0 i TC2 i jedan 16-bitni TC1 brojač. Opšta struktura binarnog brojačkog modula prikazana na slici 39. sastoji se od kola za izbor taktnih impulsa (eng. *Clock Select*), kontrolne logike (eng. *Control Logic*) i binarnog brojača.



Slika 39. Opšta struktura binarnog brojačkog modula

Kolo za izbor taktnih impulsa, prikazano na slici 40, sastoji se od multipleksera koji kontrolišu bitovi CSn[2:0], preko koga se bira da li impulsi dolaze od sistemskog takta mikrokontrolera $clk_{I/O}$ ili sa eksternog pina Tn preko kola za detekciju ivice. Kolo za detekciju ivice ima zadatak da izvrši sinhronizaciju eksternog signala takta sa sistemskim taktom, a zatim preko detektora ivice prosleđuje impuls kada se detektuje odgovarajuća uzlazna ili silazna ivica taktnog signala. Sistemski takt mikrokontrolera može se dodatno usporiti preko preskalera na frekvencije 1:8, 1:64, 1:256 i 1:1024 pomoću CSn[2:0] bitova u tabeli 10.



Slika 40. Kolo za izbor takta

Tabela 10. Izbor takta binarnog brojača

CSn [2:0] bitovi	Takt brojača	Izvor takta
000	-	Brojač zaustavljen
001	$clk_{I/O}$	Bez preskalera
010	$clk_{I/O}/8$	Preskaler 1:8
011	$clk_{I/O}/64$	Preskaler 1:64
100	$clk_{I/O}/256$	Preskaler 1:256
101	$clk_{I/O}/1024$	Preskaler 1:1024
110	clk_{Tn}	Silazna ivica takta na Tn pinu
111	clk_{Tn}	Uzlazna ivica takta na Tn pinu

Kontrolna logika brojača pored ulaznih taktnih impulsa koristi i BOTTOM i TOP ulaze. BOTTOM ulaz aktivira se kada vrednost brojača dostigne minimalnu vrednost, tj. nulu (0x00 za 8-bitni brojač ili 0x0000 za 16-bitne brojač). TOP ulaz aktivira se kada vrednost brojača dostigne maksimalnu vrednost koja može biti jedna od sledeće dve vrednosti:

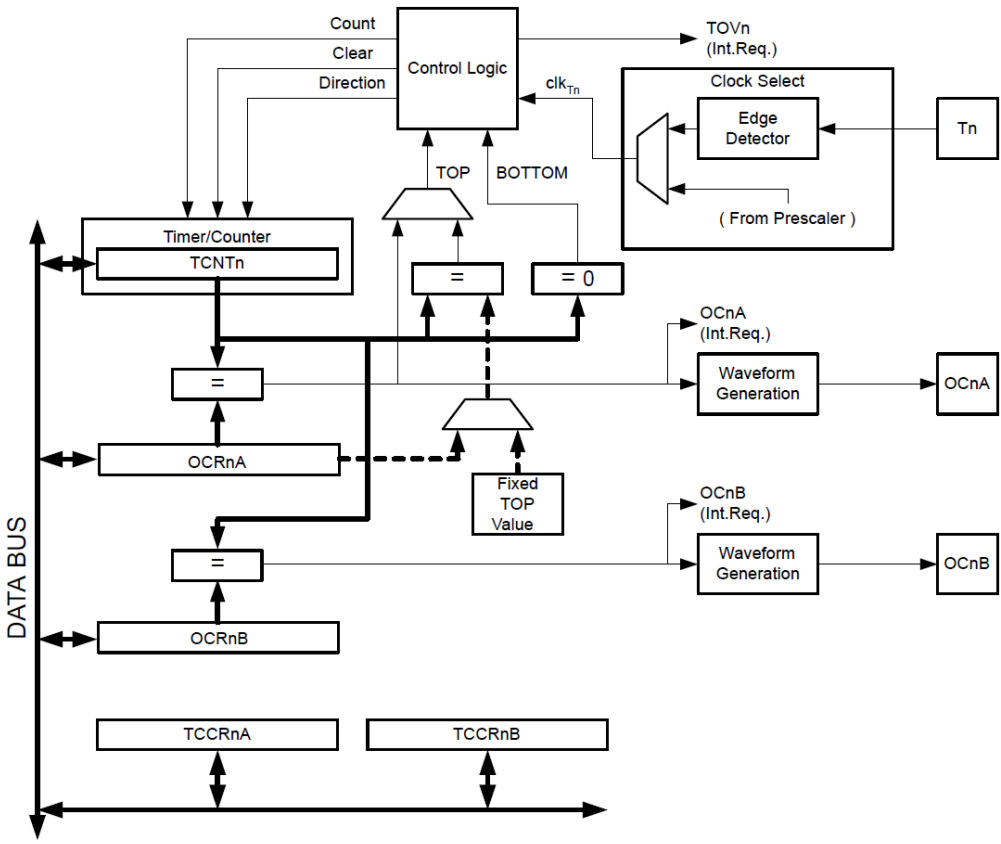
- MAX se aktivira kada brojač dođe do maksimalne vrednosti brojača (0xFF ili 255 za 8-bitni brojač ili 0xFFFF ili 65535 za 16-bitne brojač),
- TOP se aktivira kada brojač dođe do vrednosti uskladištene u OCRnA registru.

Na osnovu vrednosti kontrolnih ulaza clkTn, top i bottom i postavljenog režima rada kontrolna logika generisaće sledeće izlazne signale:

- count – inkrementira ili dekrementira vrednost TCNTn u zavisnosti od smera signala direction,
- direction – izbor između inkrementiranja i dekrementiranja,
- clear – resetuje vrednost TCNTn na nulu.

TC0 binarni brojački modul

TC0 predstavlja osmobitni binarni brojački modul ATmega328p mikrokontrolera koji je u stanju da broji u opsegu od nule do maksimalne vrednosti TCNT0 brojača od 255. Struktura TC0 modula prikazana je na slici 41. i od eksternih signala ovaj modul poseduje ulaz T0 na pinu PD4 i izlaze OC0A i OC0B na pinovima PD5 i pinu PD6.



Slika 41. Struktura TC0 binarnog brojača

TC0 generiše tri tipa prekida, prvi tip TOV0 (eng. *TC0 Ovefrlow*) nastaje kada dođe do prekoračenja gornje vrednosti brojača, a druga dva nastaju kao rezultat poređenja trenutne vrednosti brojača sa sadržajem registara OCR0A i OCR0B. Arduino IDE razvojno okruženje koristi TC0 binarni brojač za ugrađene funkcije za merenje proteklog vremena *micros()* i *millis()*. Ove dve funkcije svoj rad baziraju na podešavanju TC0 modula koje obezbeđuje tačno merenje proteklog vremena. U slučaju da se ove funkcije koriste u aplikaciji nije preporučeno menjati podešavanja ovog brojačkog modula. TC0 modul radi u režimu tajmera pri čemu je sistemski takt Arduino UNO okruženja od 16 MHz usporen preko preskalera od 1:64 na 250 kHz podešavanjem CS0[2:0] bitova u TCCR0B registru. U tom će se slučaju vrednost brojača inkrementirati svake 4 μs, što i predstavlja rezoluciju merenja vremena sa funkcijom *micros()*. Funkcija koristi prekidnu rutinu koja se pokreće prilikom prekoračenja maksimalne vrednosti tajmera, odnosno na svakih 256 impulsa, tj. 1024 μs.

```
unsigned long micros() {
      unsigned long m;
      uint8_t oldSREG = SREG, t;
      cli();
      m = timer0_overflow_count;
      t = TCNT0;
      SREG = oldSREG;
      return ((m<<8)+t)*(64/clockCyclesPerMicrosecond());
}
```

U prekidnoj rutini inkrementira se 32-bitna globalna promenljiva *timer0_overflow_count* koja predstavlja broj prekoračenja brojača. Prilikom poziva funkcije *micros()* preuzimaju se vrednosti broja prekoračenja brojača *m* i trenutne vrednosti brojača *t* i korišćenjem izraza *(m<<8)+t* određuje se ukupan broj impulsa koje je brojač izbrojao od pokretanja programa. Pošto se sistemski takt od 16MHz usporava 64 puta, potrebno je dobijeni broj impulsa pomnožiti sa 4 μs i funkcija će kao povratnu vrednost vratiti broj proteklih mikrosekundi od početka izvršavanja korisničke aplikacije. Pošto je povratna vrednost neoznačeni 32-bitni broj, on će prikazivati ispravno vreme sve dok ne dođe do prekoračenja nakon 2^{32} μs, što odgovara vremenskom intervalu od 4295 sekundi (2^{32} 10^{-6}s). Ukoliko korisnička aplikacija upotrebljava funkciju *micros()*, neophodno je proveriti ponašanje koda kada nastupi prekoračenje, nakon 71.5 minuta kontinuiranog rada.

Funkcija *millis()* takođe koristi prekidnu rutinu koja se pokreće prilikom prekoračenja maksimalne vrednosti tajmera, na svakih 1024 μs u kojoj se inkrementira 32-bitna globalna promenljiva *timer0_millis*.

```
unsigned long millis(){
      unsigned long m;
      uint8_t oldSREG = SREG;
      cli();
      m = timer0_millis;
      SREG = oldSREG;
      return m;
      }
```

Prekidna rutina poziva se na 1024 μs, pa je potrebno izvršiti korekciju merenja vremena, odnosno nakon 42 iteracije potrebno je dodati još 1 ms, kako bi se kompenzovalo akumuliranje greške od 24 μs (42x24 μs = 1.008 ms). Za kompenzaciju vremena u prekidnoj rutini koristi se globalna promenljiva *timer0_fract* koja se svakom iteracijom inkrementira za FRACT_INC=3 dok ne pređe vrednost od FRACT_MAX=125, kada se na ukupan broj milisekundi dodaje još jedna milisekunda, kao što je prikazano u tabeli 11.

Tabela 11. Kompenzacija greške merenja vremena kod ugrađene funkcije millis ()

Iteracija	Proteklo vreme	timer0_fract	timer0_millis	Greška
1	1.024 ms	3	1	- 0.024 ms
2	2.048 ms	6	2	- 0.048 ms
3	3.072 ms	9	3	- 0.072 ms
....	....	....	....	....
41	41.984 ms	123	41 + 1	-0.984 ms
42	43.008 ms	1	43	- 0.008 ms
43	44.032 ms	4	44	- 0.032 ms
....	....	....	....	....
83	84.992 ms	124	84 + 1	-0.992 ms
84	86.016 ms	2	86	- 0.016 ms
85	87.040 ms	2	87	- 0.040 ms
....	....	....	....	....
125	128 ms	125	128	0 ms

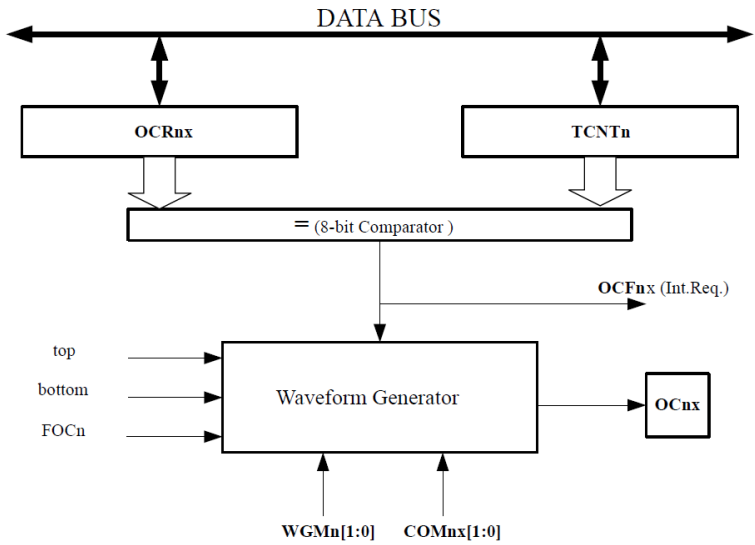
```
ISR(TIMER0_OVF_vect){
      unsigned long m = timer0_millis;
      unsigned char f = timer0_fract;

      m += MILLIS_INC;
      f += FRACT_INC;
      if (f >= FRACT_MAX) {
            f -= FRACT_MAX;
            m += 1;
      }

      timer0_fract = f;
      timer0_millis = m;
      timer0_overflow_count++;
   }
```

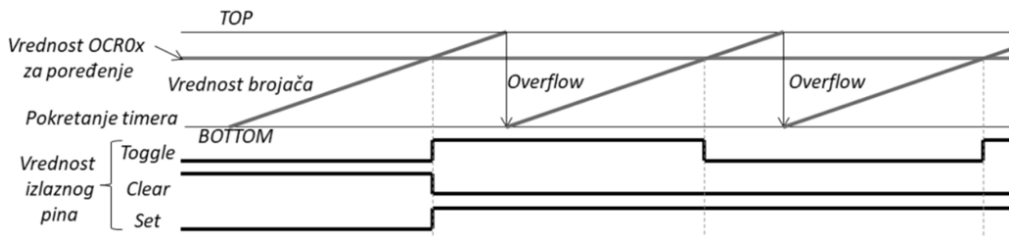
Povratna vrednost funkcije *millis()* jeste neoznačeni 32-bitni broj, on će prikazivati ispravno vreme sve dok ne dođe do prekoračenja nakon 2^{32} ms, što odgovara vremenskom intervalu od 49.7 dana (2^{32} 10^{-3}s/86400s/danu). Ukoliko korisnička aplikacija upotrebljava funkciju *millis()*, neophodno je proveriti ponašanje koda kada nastupi prekoračenje, nakon 50 dana kontinuiranog rada.

TC0 poseduje dve nezavisne jedinice za poređenje (eng. *Output Compare Units*) koje porede trenutnu vrednost brojačkog registra sa sadržajem registara OCR0A i OCR0B (slika 42). Kada vrednost brojača postane jednaka vrednosti jednog od ova dva registra, komparator će generisati signal podudaranja (eng. *Compare Match*).



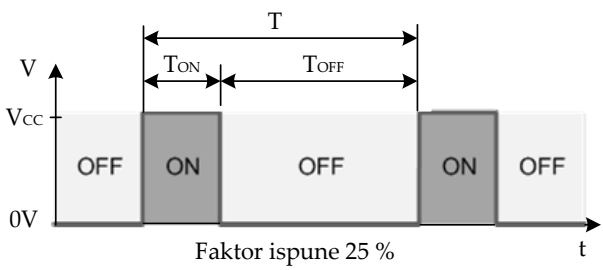
Slika 42. Struktura kola za poređenje

Signal podudaranja setovaće OCF0x (x je A ili B) bit u TIFR0 (eng. *TC0 Interrupt Flag Register*) i ukoliko je omogućen ovaj tip prekida automatski će se pokrenuti odgovarajuća prekidna rutina pri čemu će se automatski resetovati odgovarajući OCF0x bit. Signal podudaranja OCF0x predstavlja ulaz za talasni generator (eng. *Waveform Generator*) koji je spregnut sa OC0x izlaznim pinom. Talasni generator u slučaju poređenja može setovati, resetovati ili invertovati (eng. *toogle*) vrednost izlaznog pina OC0x, što može biti korisno za generisanje preciznih vremenskih impulsa (slika 43). Talasni generator koristi i dodatne signale *top* i *bottom* kako bi u zavisnosti od režima rada uticao na vrednost izlaznog pina OC0x.

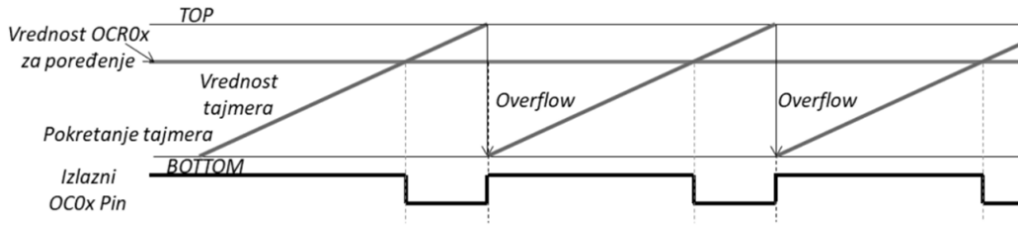


Slika 43. Funkcija talasnog generatora u normalnom režimu rada

Izlazni pin tajmera OC0A nalazi se na PD5 pinu a OC0B na pinu PD6. Arduino IDE ugrađena funkcija *analog_write(pin,value)* koristi talasni generator u režimu za brzo generisanje PWM (eng. *Pulse-Width Modulation*) impulsno širinski modulisanih signala setovanjem WGM0[2:0] bitova u TCCR0A registru. Impulsno širinska modulacija predstavlja digitalni fiksnu frekvenciju i sa promenljivim odnosom intervala trajanja logičke nule (eng. *off time*) i logiče jedinice (eng. *on time*), prikazan na slici 44, koji se naziva faktor ispune (eng. *duty cycle*). U ovom režimu izlazni pin OC0x setuje se na jedinicu kada se aktivira *bottom* signal, a resetuje kada se generiše signal podudaranja sa odgovarajućim OCR0x registrom (slika 45).



Slika 44. Talasni dijagram PWM impulsno širinske modulacije



Slika 45. Funkcija talasnog generatora u brzom PWM režimu rada

Ugrađena funkcija *analog_write(pin,value)* u sledećem listingu koda upisuje vrednost (eng. *value*) u opsegu od 0 do 255 u OCR0x registar, od koje će zavisiti odnos vremenskih intervala u kojima signal ima vrednost logičke nule i vrednost logičke jedinice, odnosno faktor ispunjenosti PWM signala. S obzirom da je TC0 podešen da generiše TOV0 prekide na svakih 1024 μs frekvencija PWM signala biće konstantna i iznosiće 976Hz (1/1024 μs).

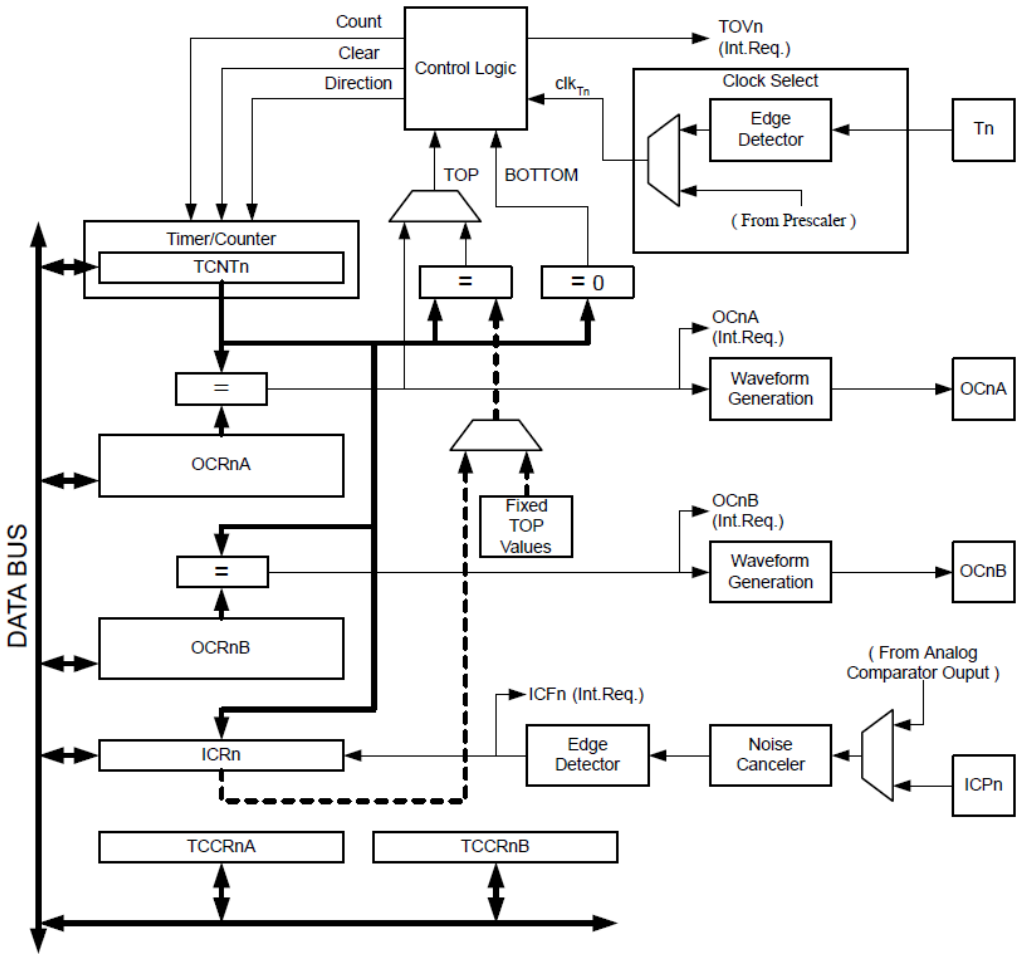
```
void analogWrite(uint8_t pin, int val)
{
      pinMode(pin, OUTPUT);
      if (val == 0)
            digitalWrite(pin, LOW);
      else if (val == 255)
            digitalWrite(pin, HIGH);
      else
      {
            switch(digitalPinToTimer(pin))
            {
                  case TIMER0A:
                        sbi(TCCR0A, COM0A1);
                        OCR0A = val; // set pwm duty
                        break;
                  #endif

                  #if defined(TCCR0A) && defined(COM0B1)
                  case TIMER0B:
                        sbi(TCCR0A, COM0B1);
                        OCR0B = val; // set pwm duty
                        break;
                  #endif
```

Impulsno širinska modulacija jedan je od najčešće korišćenih načina za kontrolu kola za upravljanje energetskim pretvaračima (eng. *power driver*) koja se koriste za upravljanje LED izvorima svetlosti, motorima i prekidačkim napajanjima.

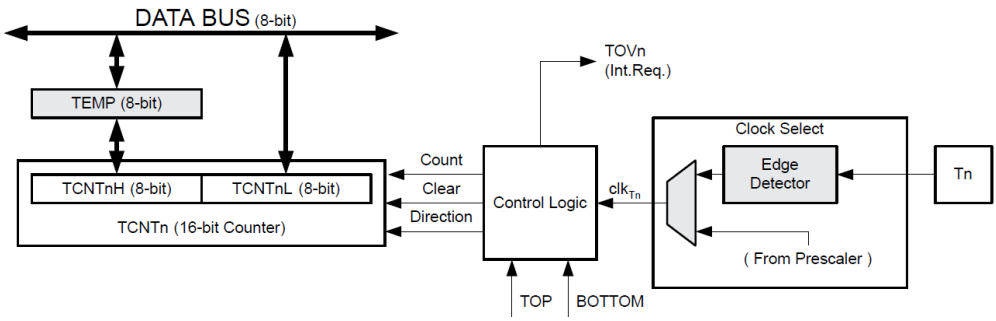
TC1 binarni brojački modul

TC1 modul predstavlja 16-bitni binarni brojač, prikazan na slici 46, koji poseduje slične funkcionalnosti kao i 8-bitni TC0 brojač. Jedine razlike ogledaju se u 16-bitnim registrima koji omogućavaju znatno duže intervale brojanja (prekoračenje brojača nastupa pri vrednosti od 65536). Dodatno, TC1 brojač poseduje jedinicu za beleženje događaja (eng. *Capture*), koja omogućava pamćenje tačnog trenutka kada se neki događaj desio. TC1 modul poseduje ulaz T1 na pinu PD4, ulaz ICP1 na pinu PB0 i izlaze OC1A i OC1B na pinovima PB1 i pinu PB2.



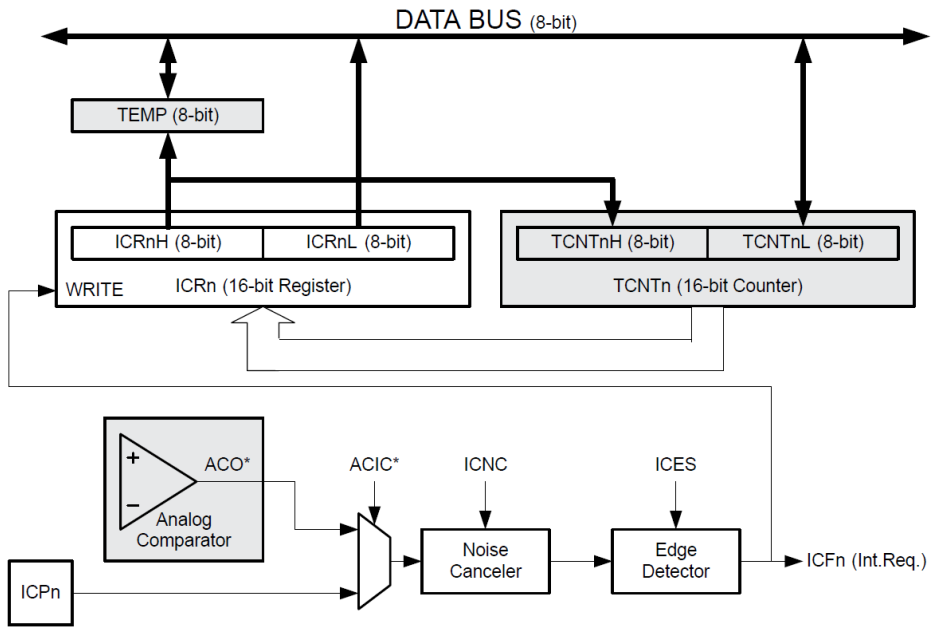
Slika 46. Struktura 16-bitnog brojača

Pristup 16-bitnim registrima TCNT1, OCR1A/B i ICR1 iziskuje dve operacije čitanja ili upisa, pošto se ovim registrima pristupa preko 8-bitne magistrale podataka. U tu svrhu TC1 tajmer poseduje 8-bitni TEMP registar za privremeno skladištenje višeg bajta prilikom 16-bitnog pristupa koji se deli između svih 16-bitnih registara (slika 47). Pristup nižem bajtu automatski pokreće 16-bitnu operaciju čitanja ili upisa. Kada se pročita niži bajt 16-bitnog registra, u istom taktnom ciklusu dolazi i do prenosa višeg bajta u TEMP, iz koga se vrednost mora pročitati naknadno. Pre početka operacije 16-bitnog upisa neophodno je prvo upisati vrednost predviđenu za upis na viši bajt u TEMP registar. Prilikom upisa nižeg bajta dolazi i do kopiranja sadržaja TEMP registra u istom ciklusu takta čime je obezbeđeno konzistentno ponašanje prilikom upisa i čitanja 16-bitnih podataka.



Slika 47. Operacija 16-bitnog upisa/čitanja

Jedinica za beleženje događaja koristi se kako bi se zapamtilo vreme (eng. *time-stamp*) kada je se neki događaj dogodio. Eksterni događaj može biti doveden sa eksternog ICP1 pina ili iz analognog komparatora (slika 48). Izvor eksternog događaja odabira se preko multipleksera koga kontroliše ACIC (eng. *Analog Comparator Input Capture*) bit. Signal eksternog događaja prolazi kroz kolo za otklanjanje uticaja šuma koje filtrira signal, ako je ICNC (eng. *Input Capture Noise Canceler*) bit setovan, što uvodi kašnjenje od četiri taktna ciklusa. Zatim se pomoću bita ICES (eng. *Input Capture Edge Select*) odabira tip ivice koja će izazvati beleženje događaja (0 za silaznu i 1 za uzlaznu ivicu). Kada se pojavi odgovarajuća ivica, biće generisan zahtev za prekid i doći će do automatskog upisa trenutnog sadržaja brojačkog registra u registar ICR (eng. *Input Capture Register*). Na taj način može se zabeležiti tačan trenutak kada se dogodio neki eksterni događaj, što može biti korisno prilikom merenja frekvencije signala i slično.



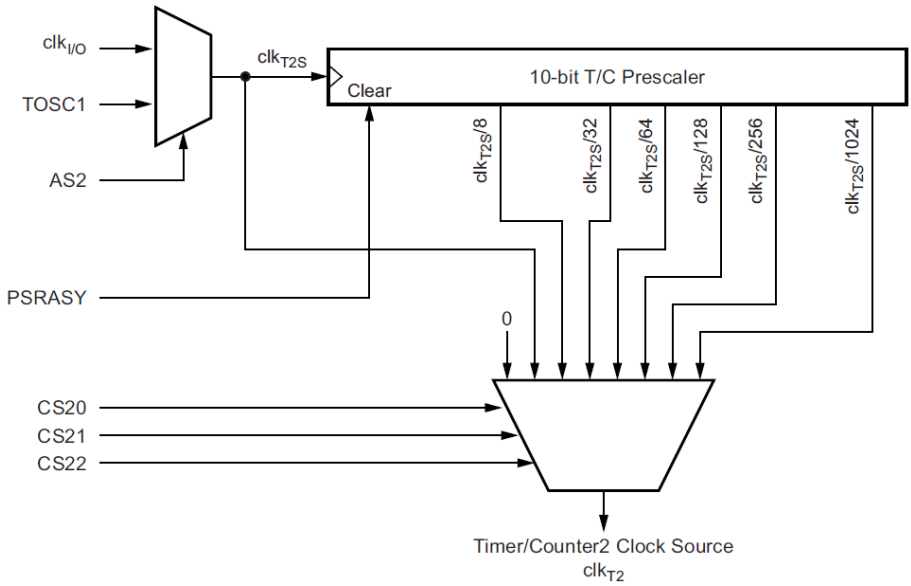
Slika 48. Struktura bloka za beleženje događaja

TC1 tajmer kod Arduino IDE okruženja podešen je na vrednost preskalera 1:64, tako da pri sistemskom taktu od 16 MHz on uvećava svoju vrednost na svakih 4 μs. Režim rada TC1 brojača podešen je na PWM 8-bitni fazno korektni režim. U ovom režimu rada brojač broji unapred do vrednosti 0x00FF, nakon čega kreće da broji unazad ka vrednosti 0x0000. Zahtev za TOV1 prekid generiše se kada tajmer dođe do TOP vrednosti odnosno na svakih 2048 μs.

TC2 binarni brojački modul

TC2 modul jeste još jedan 8-bitni binarni brojač slične strukture kao prethodno opisani TC0 brojač. Jedna od razlika u odnosu na TC0 brojač jeste ta što se TC2 brojač može taktovati od strane ugrađenog niskofrekventnog ocilatora, koji koristi kvarcni kristal radne frekvencije 32.768 kHz, povezan na pinove TOSC1 i TOSC2. Ovaj izvor takta koristi se kod RTC (eng. *Real Time Clock*) časovnika realnog vremena koji se koriste za precizno merenje vremena sa funkcijom kalendara koji prati datume i dane u nedelji u kalendarskoj godini. Rad TC2 tajmera sa kvarcnim kristalom od 32.768 kHz omogućava vrlo malu potrošnju struje od 0.75 μA što omogućava da mikrokontroler radi bez mrežnog napajanja preko baterije.

Izbor izvora takta TC2 brojača između sistemskog i niskofrekventnog takta (slika 49) ostvaruje se preko multipleksera kontrolisanog od strane AS2 bita iz ASSR (eng. *Asynchronous Status Register*). Takt izabranog oscilatora TC2 modula podešava se pomoću CS2 bitova u TCCR2B registru prema tabeli 12.



Slika 49. Struktura preskalera TC2 binarnog brojača

Tabela 12. Izbor takta TC2 binarnog brojača

CS2 [2:0] bitovi	Takt brojača	Izvor takta
000	-	Brojač TC2 zaustavljen
001	clk_{T2S}	Bez preskalera
010	$clk_{T2S}/8$	Preskaler 1:8
011	$clk_{T2S}/32$	Preskaler 1:32
100	$clk_{T2S}/64$	Preskaler 1:64
101	$clk_{T2S}/128$	Preskaler 1:128
110	$clk_{T2S}/256$	Preskaler 1:256
111	$clk_{T2S}/1024$	Preskaler 1:1024

Podešavanjem preskalera na vrednost od 1:128 moguće je usporiti takt iz niskofrekventnog oscilatora na 256 Hz tako da će TC2 brojač svake sekunde generisati prekid usled prekoračenja. Ovakvo podešavanje TC2 brojača može se iskoristiti od strane korisničke aplikacije koja će meriti vreme uvećavajući promenljivu za sekunde nakon svakog prekida. Realno

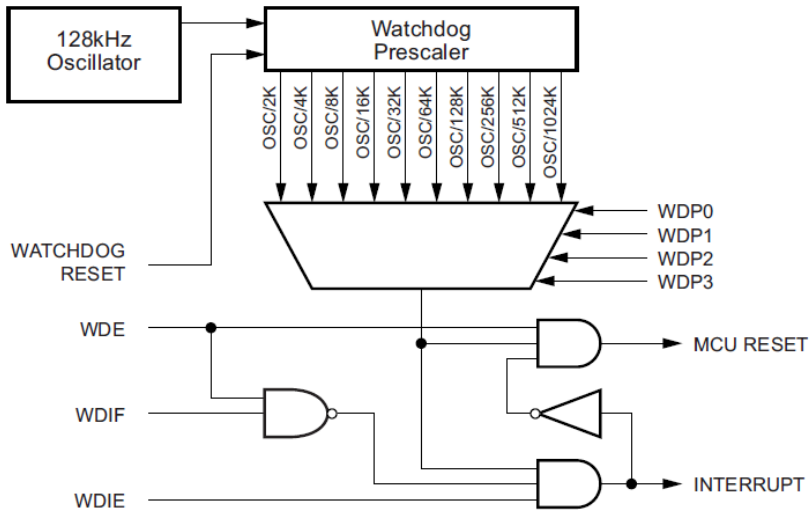
vreme može se predstavljati u različitim formatima, a jedan od najčešće korišćenih jeste UNIX format u kojem se vreme predstavlja u sekundama pomoću 32-bitne neoznačene promenljive, koja meri vreme od referentnog datuma koji je nastupio u četvrtak u ponoć, 1. januara 1970. godine. Na osnovu akumuliranog vremena u sekundama proračunima se može doći do tačnog datuma i dana u nedelji.

TC2 modul poseduje izlaze OC2A i OC2B na pinovima PB3 i pinu PD3. U Arduino UNO razvojnom sistemu TC2 tajmer taktuje se oscilatorom od 16 MHz i koristi se od strane *Tone* biblioteke za generisanje zvuka pomoću zvučnika ili zujalice povezane na OC2A ili OC2B izlaze.

Sigurnosni tajmer

Ugrađeni sistemi projektuju se da rade samostalno, bez potrebe za čovekovom intervencijom. Međutim, kao i svi računarski sistemi i oni su podložni greškama koje mogu izazvati zastoj i blokiranje daljeg izvršavanja upravljačkog programa. Kao posledica pojave ovih grešaka može doći do neregularnog rada upravljačke aplikacije, što može dovesti do otkaza celokupnog sistema i ozbiljnih posledica po sam sistem i njegovu neposrednu okolinu. U slučaju zastoja obično nije moguće čekati da čovek izvrši restetovanje sistema jer je brzina čovekovog reagovanja isuviše spora da bi ispunila uslove rada sistema. Sigurnosni (eng. *watchdog*) tajmer namenjen je da zaštiti mikrokontrolerski sistem od neregularnog rada ugrađenog sistema. Princip rada sigurnosnog tajmera zasniva se na redovnom primanju signala od ugrađenog sistema kojim se potvrđuje da je sistem operativan (eng. *heartbeat*). Sigurnosni tajmer odbrojava vreme od poslednjeg primljenog signala i, ukoliko sledeći signal zakasni, pokrenuće se procedura oporavka i resetovanja ugrađenog sistema.

Sigurnosni brojač u ATmega328p mikrokontroleru, prikazan na slici 50, sastoji se iz delitelja frekvencije, multipleksera i nezavisnog ugrađenog oscilatora radne učestanosti od 128 kHz. Željeni broj impulsa brojača, odnosno vreme reagovanja brojača, odabira se preko multipleksera korišćenjem WDP[3:0] bitova u WDTCSR (eng. *Watchdog Timer Control Register*) registru prema tabeli 13.



Slika 50. Struktura sigurnosnog brojača

Tabela 13. Podešavanje preskalera sigurnosnog brojača

WDP[3:0] bitovi	Broj taktova	Vreme
0000	2048 (2k)	16 ms
0001	4096 (4k)	32 ms
0010	8192 (8k)	64 ms
0011	16384 (16k)	128 ms
0100	32768 (32k)	256 ms
0101	65536 (64k)	512 ms
0110	131072 (128k)	1.024 s
0111	262144 (256k)	2.048 s
1000	524288 (512k)	4.096 s
1001	1048576 (1024k)	8.192 s
ostalo	rezervisano	-

Primer upotrebe sigurnosnog brojača prikazan je na primeru svetleće diode koja trepti sa intervalom koji se tokom izvršavanja programa povećava. Prilikom svake iteracije loop() funkcije poziva se instrukcija za resetovanje sadržaja sigurnosnog brojača.

```
#include<avr/wdt.h>
int t = 10; //vreme zadrške 10ms
void setup() {
    pinMode(13, OUTPUT);
    wdt_enable(WDTO_1S); //Interval sigurnosnog brojača 1s
}
```

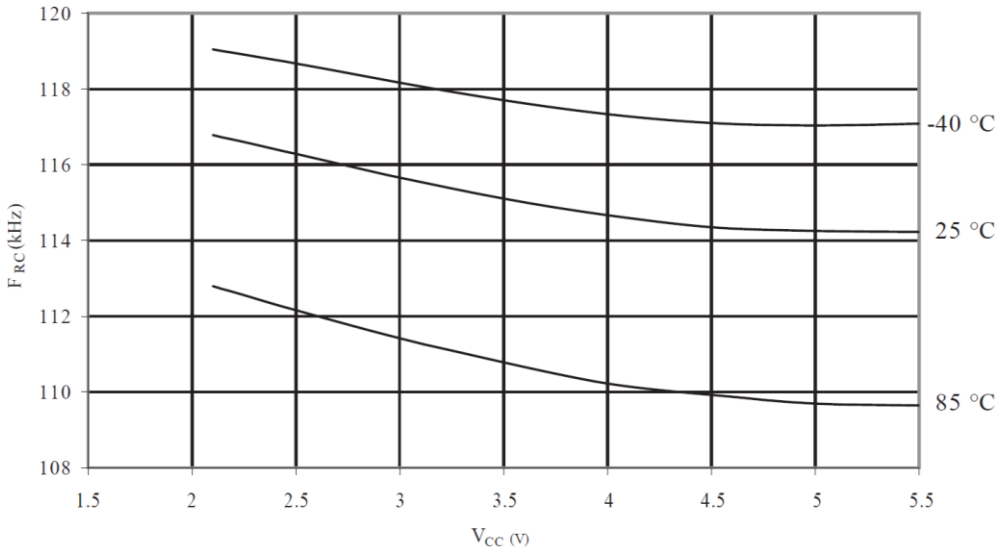
```
void loop() {
    digitalWrite(13, HIGH);
    delay(t);
    digitalWrite(13, LOW);
    delay(t);
    wdt_reset(); // Reset sigurnosnog brojača
    t=t+10; // Povećanje zadrške za 10ms
 }
```

Kada vreme izvršavanja petlje bude izvan granice intervala sigurnosnog brojača, koji je podešen na interval od 1s, sigurnaosni brojač generisaće signal i doći će do resetovanja mikrokontrolera i ponovnog pokretanja programa. Kada sigurnosni brojač odbroji podešeni broj taktova, generiše se signal koji može izazvati zahtev za prekid i/ili reset mikrokontrolera u zavisnosti od režima rada u WDTCSR registru kao što je prikazano tabelom 14.

Tabela 14. Podešavanje režima rada sigurnosnog brojača

WDTON	WDE	WDIE	Režim rada	Akcija
1	0	0	Zaustavljen	bez akcije
1	0	1	Prekidni režim	WDT prekid
1	1	0	Reset režim	Reset
1	1	1	Prekidni i reset režim	WDT prekid praćen resetom

Ukoliko je odabran prekidni režim, sigurnosni brojač generisaće prekid kada odbroji željeni broj ciklusa. Ovaj tip prekida može se koristiti kao opšti sistemski tajmer koji ograničava maksimalno vreme dozvoljeno za određene operacije, generišući prekid kada operacija traje duže od očekivanog. Sistemski tajmer najčešće se koristi od strane operativnog sistema za rad u realnom vremenu. Ovaj tip prekida može se koristiti i za buđenje mikrokontrolera iz režima niske potrošnje. Potrebno je napomenuti da je frekvencija internog oscilatora jako zavisna od temperature i napona napajanja, kao što je prikazano na slici 51, stoga nije preporučena njegova upotreba za precizno merenje vremena.



Slika 51. Uticaj temperature i napona napajanja na frekvenciju internog oscilatora sigurnosnog brojača

U slučaju da je odabran režim resetovanja sistema sigurnosni brojač generisaće sistemski reset kada odbroji željeni broj ciklusa, a da pri tome nije resetovan korišćenjem specijalne WDR (eng. *WathcDog Reset*) instrukcije. Ovaj režim obično se koristi za sprečavanje blokiranja upravljačkog programa u slučaju pojave hardverskih ili softverskih grešaka.

Režim prekida i resetovanja sistema kombinuje prethodna dva režima rada tako što kada sigurnosni brojač odbroji željeni broj ciklusa prvo generiše zahtev za prekid i u prekidnoj rutini čuva kritične parametre pre resetovanja sistema, nakon čega prelazi u režim resetovanja sistema. Na ovaj način moguće je definisati tačku oporavka (eng. *restore point*), koja bi bila zapamćena kada se završi određeni deo programa. U slučaju da nastupi reset sistema program bi mogao nastaviti izvršavanja od zapamćene tačke oporavka kao da se pre toga ništa nije dogodilo. Ova karakteristika pokazala je svoju vrednost tokom problema pri sletanju Apola 11 (eng. Apollo 11) na površinu Meseca, prilikom kog je AGC računar pretrpeo brojna resetovanja zbog preopterećenja, ali je nastavio da kontroliše lunarni modul u fazi sletanja bez ikakvih problema.

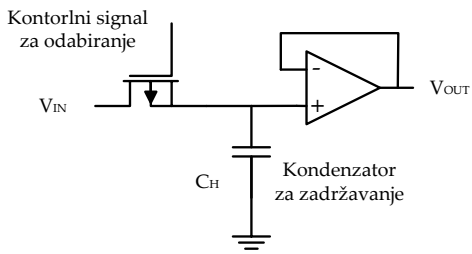
POGLAVLJE 8: ANALOGNI MODULI

Pored velikog broja digitalnih ulazno-izlaznih pinova opšte namene mikrokontroleri poseduju i analogne module. Oni omogućavaju ne samo da se detektuje da li je na ulaz dovedena logička nula ili jedinica već i da se očita numerička vrednost koja je proporcionalna ulaznom naponu zahvaljujući integrisanom analogno/digitalnom konvertoru. Takođe, pojedini tipovi mikrokontrolera poseduju ugrađene i digitalno/analogne konvertore, koji su u stanju da na izlazu generišu analogni napon. Analogni signali kontinualni su i u vremenu i po vrednosti i oni kao takvi nisu pogodni da se koriste u računarski sistemima koji su po prirodi diskretni u vremenu i po vrednosti. Zbog toga se kao interfejs između analognih signala i računara koriste A/D i D/A konvertori koji su u stanju da pretvaraju jedan oblik signala u drugi.

A/D konverzija

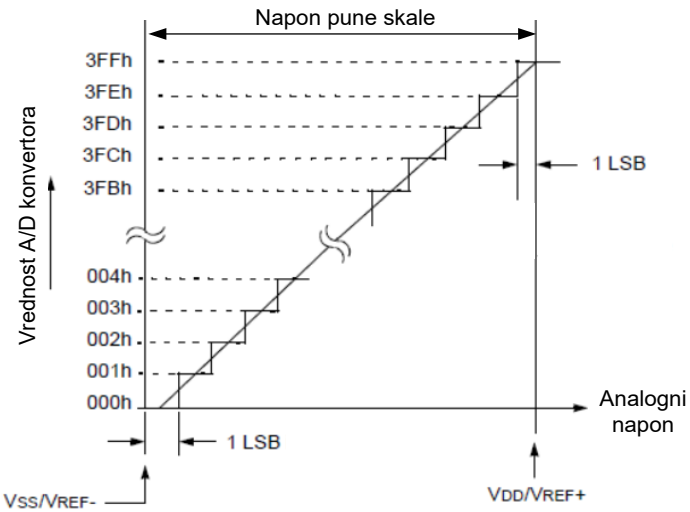
Postupak pretvaranja analognog signala u digitalni započinje uzorkovanjem (eng. *sampling*) analognog signala u pravilnim vremenskim intervalima. Pošto analogni signal ima definisanu vrednost u svakom vremenskom intervalu, odmereni signal imaće definisane vrednosti samo u trenucima odmeravanja. Ako je brzina promene signala između dva odmerka dovoljno mala, odmereni analogni signal biće verno predstavljen ovim odmercima. Nikvistova teorema odabiranja definiše da se bilo koji analogni signal može verno rekonstruisati ako je učestanost odmeravanja bar dva puta veća od najviše učestanosti u spektru analognog signala za koji se vrši odabiranje. Analogni signal odabira se pomoću kola za odmeravanje i zadržavanje (eng. *Sample and Hold*), čija je funkcija da

preuzme trenutnu vrednost ulaznog analognog napona V_{IN} i zapamti je tokom procesa A/D konverzije. Tipična struktura S/H kola, prikazana na slici 52, sastoji se od MOSFET-a, kondenzatora i jediničnog neinvertujućeg pojačavača. U fazi uzorkovanja, kontrolni signal uključuje MOSFET koji omogućava da se kondenzator napuni na vrednost ulaznog analognog napona V_{IN}. Nakon isteka potrebnog vremena isključuje se MOSFET tranzistor i kondenzator zadržava uzorkovanu vrednost V_{IN}. Uloga jediničnog pojačivača jeste preslikavanje napona V_{IN} na njegov izlaz pri čemu se sprečava pražnjenje kondenzatora tokom procesa A/D konverzije.



Slika 52. Struktura kola za odmeravanje i zadržavanje

Odmereni signal diskretan je u vremenu, ali ne i po vrednosti, pa je potrebno sprovesti postupak njegove diskretizacije po vrednosti. Ovim se postupkom analogni signal, koji poseduje beskonačan skup vrednosti, transformiše u diskretan signal sa ograničenim brojem vrednosti, prema slici 53. Rezolucija A/D konvertora predstavlja broj bitova koji se koriste za predstavljanje diskretizovane binarne vrednosti signala.



Slika 53. Digitalna konverzija analognog napona

Diskretizacijom signala nepovratno se gubi deo informacije o analognom signalu i vrednost te greške jednaka je ±½ V_{LSB} napona kvantizacije. Vrednost greške kvantizacije biće manja što je rezolucija A/D konverzije veća. A/D konvertor vrši konverziju diskretizovanog analognog ulaznog signala V_{IN} u N-to bitnu vrednost D u odnosu na opseg A/D konvertora koji je definisan V_{REF+} pozitivnim i V_{REF-} negativnim referentnim naponom. Pozitivni i negativni referentni naponi mogu se zadati u opsegu napajanja A/D konvertora od 0 V do V_{dd} i definišu opseg pune skale A/D konvertora (eng. *Full Scale Range*) u kojem se ulazni napon V_{IN} sledećim jednačinama pretvara u digitalnu vrednost u opsegu [0÷2^N-1]. Po završetku A/D konverzije vrednost ulaznog napona biće konvertovana u N-bitnu vrednost D:

$$\frac{D}{2^N - 1} = \frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}}$$

$$D = \frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}} (2^N - 1)$$

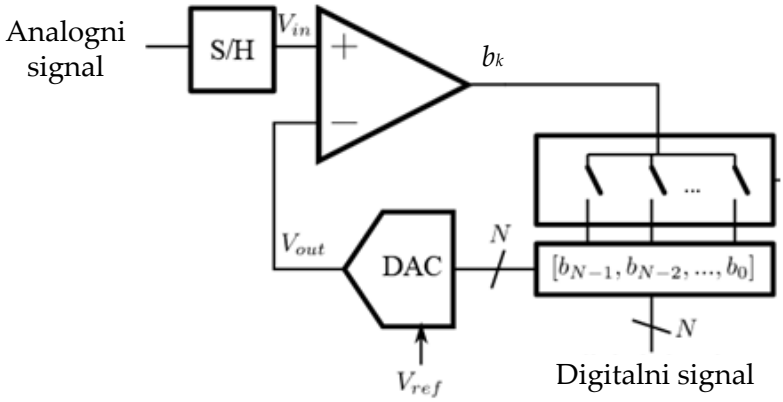
Kako bi se odredio ulazni napon V_{IN} potrebno je u korisničkoj aplikaciji izvršiti sledeći proračun:

$$V_{IN} = \frac{V_{REF+} - V_{REF-}}{2^N - 1} * D + V_{REF-}$$

A/D konvertori vrše digitalizaciju signala sa određenom greškom koja je zavisna od rezolucije A/D konvertora, te greška kvantizacije V_{LSB} iznosi:

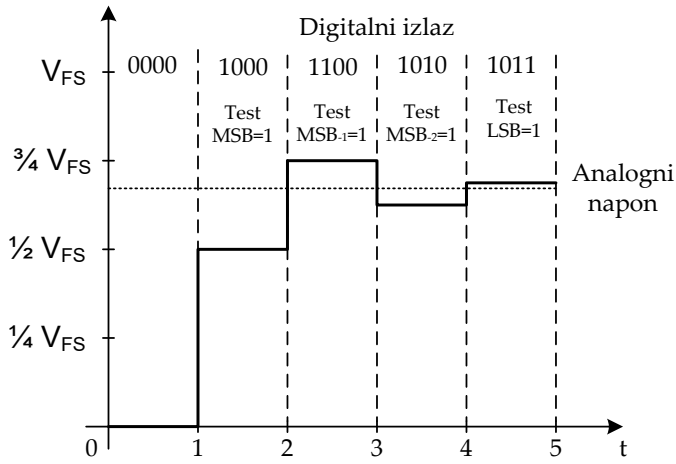
$$V_{LSB} = \frac{V_{REF+} - V_{REF-}}{2^N - 1}$$

Postoji više tipova A/D konvertora koji se razlikuju prema rezoluciji i vremenu potrebnom da se izvrši proces A/D konverzije. Jedan od najčešće korišćenih tipova u mikrokontrolerima jeste A/D konvertor sa sukcesivnim aproksimacijama koji poseduju jednostavnu konstrukciju i dobre performanse za upotrebu u većem broju mikrokontrolerskih aplikacija. Ovaj konvertor, čija je stuktura prikazana na slici 54, sadrži kolo za uzorkovanje i zadržavanje i analogni komparator kao i SAR (eng. *Succesive Approximation Register*) registar sa sukcesivnim aproksimacijama i digitalno/analogni konvertor.



Slika 54. Struktura A/D konvertora sa sukcesivnim aproksimacijama

Postupak započinje sa uzorkovanjem vrednosti ulaznog napona, koja se mora odrzavati na konstantnoj vrednosti tokom celog procesa A/D konverzije. Potom se setuje bit najveće težine u SAR registru pri čemu su svi ostali bitovi resetovani i na osnovu ove digitalne vrednosti D/A konvertor generiše izlazni napon koji je jednak polovini referentnog napona V_{REF} koji se poredi sa ulaznim naponom V_{IN}. U slučaju da je ulazni napon veći komparator će generisati logičku jedinicu na osnovu koje će SAR registar zadržati postavljeni bit na stanju logičke jedinice. U suprotnom slučaju, ako je ulazni napon manji, SAR registar resetovaće postavljeni bit na stanje logičke nule. Nakon završene iteracije preći će se na postavljanje sledećeg najznačajnijeg bita i postupak će se ponoviti za sve bitove, nakon čega će u SAR registru ostati binarna vrednost koja odgovara vrednosti ulaznog napona V_{IN} kao što je prikazano na slici 55.



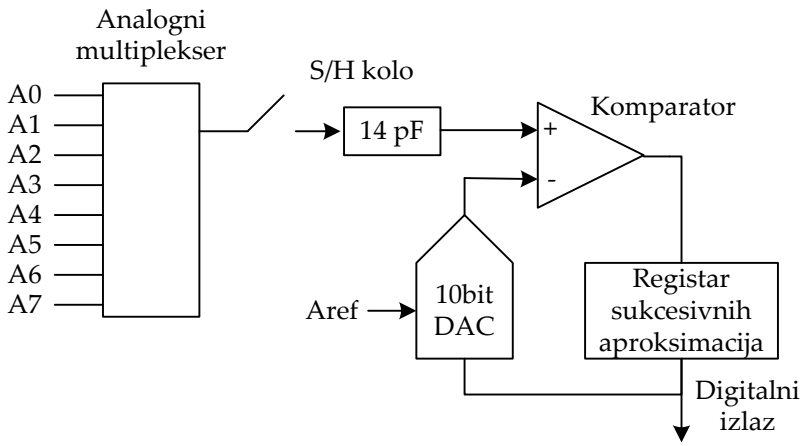
Slika 55. Postupak A/D konverzije sa sukcesivnim aproksimacijama

A/D konvertorski modul

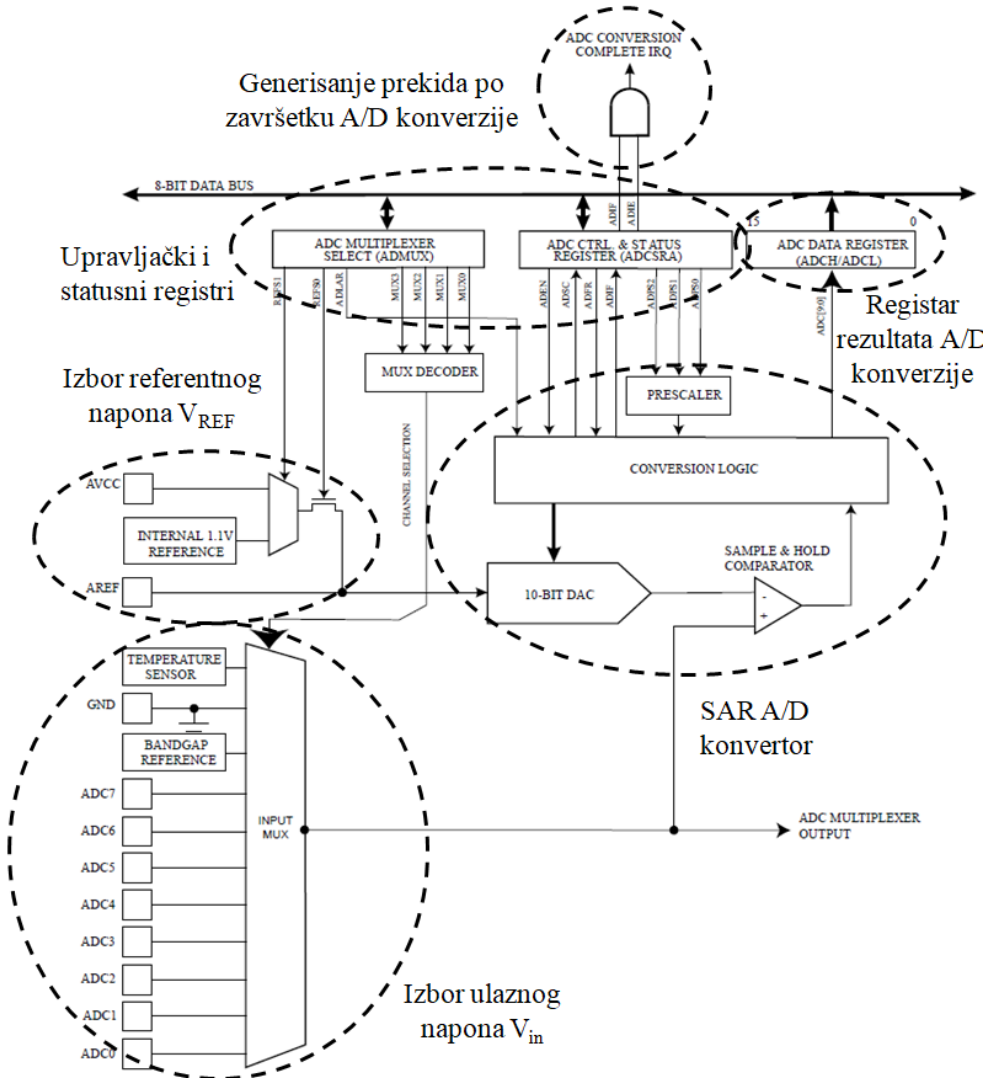
Atmega328p mikrokontroler poseduje A/D konvertor sa sukcesivnim aproskimacijama rezolucije 10-bita. Tipično trajanje A/D konverzije jeste od 13÷260 μs u zavisnosti od rezolucije i radnog takta A/D konvertora. Maksimalna brzina odabiranja jeste 76.9 kSPS (eng. – *kilo samples per second*), odnosno 15 kSPS pri maksimalnoj rezoluciji. A/D konvertor povezan je preko analognog multipleksera sa ulaznim pinovima porta A pri čemu kao donji refrentni napon V_{REF-} uvek koristi napon od 0 V (GND) a kao gornji refrentni napon V_{REF+} može koristiti:

- napon napajanja A/D konvertora AVcc,
- interni izvor referentnog napona od 1.1 V,
- eksterni izvor referentnog napona koji se povezuje preko pina AREF.

Blok šema A/D konvertora ATmega328p mikrokontrolera prikazana je na slici 56. i sastoji se od analognog multipleksera, kola za odabiranje i zadržavanje, komparatra, D/A konvertora i SAR A/D konvertora. Detaljna struktura A/D konvertora ATmega328p mikrokontrolera, koja se može videti na slici 57, uključuje i izvor referentnog napona, skup kontrolnih registara i kolo za generisanje prekida po završetku A/D konverzije.



Slika 56. Blok šema A/D konvertora ATmega328p mikrokontrolera



Slika 57. Struktura A/D konvertora ATmega328p mikrokontrolera

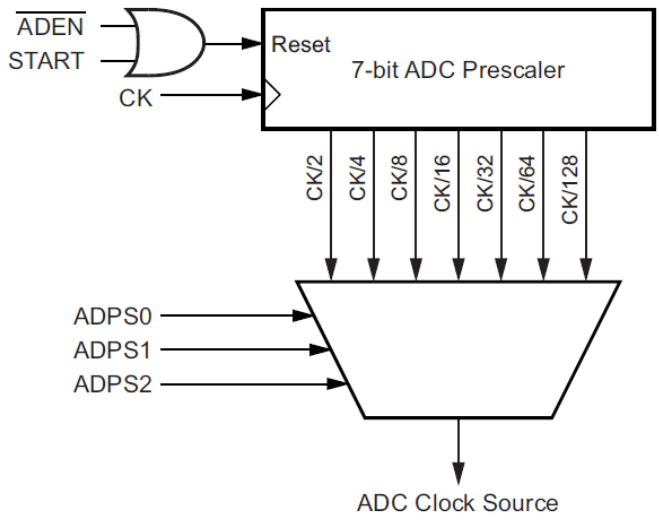
Uobičajeno je da se A/D konvertor deli između više ulaznih pinova, pa se ovi ulazi dovode na ulaz A/D konvertora preko analognog multipleksera. A/D konvertor kod 328p mikrokontrolera u jednom trenutku može izvršiti samo jednu A/D konverziju nad nekim ulazom. Za A/D konverziju na drugom ulazu potrebno je promeniti kanal multipleksera i pokrenuti proces A/D konverzije. U slučaju da je zahtevana promena kanala dok je proces A/D konverzije u toku A/D konvertor će završiti tekuću konverziju pre nego što izvrši promenu kanala. Selekcioni ulaz multipleksera kontroliše se preko MUX[3..0] bitova u ADMUX (eng. *ADC Multiplexer Selection Register*) registru prema tabeli 15.

Tabela 15. Podešavanje multipleksera A/D konvertora

MUX[3:0] bitovi	Ulaz multipleksera	Arduino UNO pin
0000	ADC0	A0
0001	ADC1	A1
0010	ADC2	A2
0011	ADC3	A3
0100	ADC4	A4
0101	ADC5	A5
0110	ADC6	-
0111	ADC7	-
1000	Temperaturni senzor	-
1001	rezervisano	-
.....	rezervisano	-
1101	rezervisano	-
1110	1.1 V referentni napon	-
1111	0 V (GND)	-

Na ulaze multipleksera direktno se dovodi napon sa ulaznih pinova A0, A1, A2, A3, A4, A5 Arduino UNO razvojnog sistema pri čemu ulazni portovi ADC6 i ADC7 nisu dostupni na razvojnom sistemu. Interni temperaturni senzor povezan je na osmi kanal analognog multipleksera i omogućava merenje interne temperature mikrokontrolera. Izvor internog referentnog napona od 1.1 V povezan je na četrnaesti kanal analognog multipleksera i može se koristiti za merenje AVcc napona napajanja A/D konvertora. Napon 0 V (GND) povezan je na poslednji kanal i koristi se za utvrđivanje ofseta A/D konverotra. Ofset predstavlja odstupanje očitavanja A/D konvertora kada se na njegov ulaz dovede vrednost ulaznog napona koja je jednaka nuli.

S obzirom da vreme trajanja A/D konverzije zavisi od rezolucije, u slučaju da se koristi maksimalna rezolucija od 10 bita proces A/D konverzije završava se za 13 taktnih ciklusa. U slučaju prve A/D konverzije vreme konverzije jeste 25 taktnih ciklusa pošto je potrebno određeno vreme za uspostavljanje stabilnog rada analognih kola po prvom uključenju. Izvor takta A/D konvertora jeste sistemski takt mikrokontrolera koji se usporava preko preskalera čija je stuktura prikazana na slici 58.



Slika 58. Struktura preskalera A/D konvertora ATmega328p mikrokontrolera

Maksimalna radna frekvencija A/D konvertora pri maksimalnoj rezoluciji iznosi 200 kHz. Pri maksimalnoj radnoj učestanosti brzina odmeravanja u režimu maksimalne rezolucije iznosi oko 15000 odmeraka u sekundi (15 kSPS). Pošto je sistemski takt često mnogo brži, neophodno je korišćenje posebnog preskalera kako bi se sistemski takt dovoljno usporio, korišćenjem bitova ADPS [2:0] u ADCSRA registru (eng. *ADC Control and Status Register A*) prema tabeli 16.

Tabela 16. Podešavanje preskalera A/D konvertora

ADPS [2:0] bitovi	**Učestanost preskalera**
000	Fs/2
001	Fs/4
010	Fs/8
011	Fs/16
100	Fs/32
101	Fs/64
110	Fs/128

Minimalna radna frekvencija A/D konvertora iznosi 50 kHz, što zahteva minimalni sistemski takt mikrokontrolera od 100 kHz u slučaju korišćenja A/D konvertora (minimalno podešavanje preskalera Fs/2). S obzirom da Arduino UNO koristi sistemski takt od 16 MHz neophodno je korišćenje maksimalne vrednosti preskalera od 128, kako bi se obezbedio radni takt

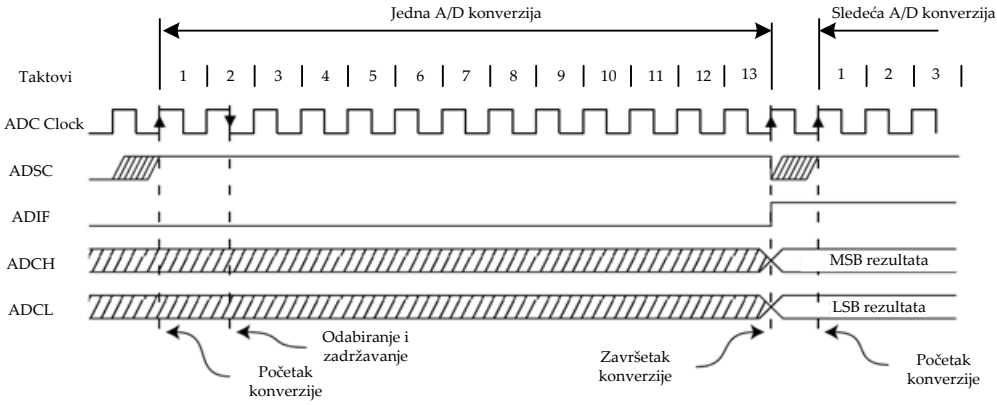
A/D konvertora od 125 KHz. Pri ovom radnom jedna A/D konverzija sa maksimalnom rezolucijom u proseku će trajati oko 100 μs, pa će A/D konvertor moći maksimalno da izvršiti oko 10000 konverzija sekundi.

A/D konvertor omogućava podešavanje pozitivnog referentnog napona, pri čemu je negativni referentni napon uvek vezan za 0 V. Na raspolaganju je nekoliko izvora referentnog napona koji se selektuju preko REFS[1:0] bitova u ADMUX registru prema tabeli 17. Pozitivni referentni napon može biti napon napajanja AVcc, eksterni referentni napon doveden preko Aref pina ili interni izvor referentnog napona od 1.1 V.

Tabela 17. Podešavanje referentnog napona A/D konvertora

REFS[1:0]	**Vref+ referentni napon**
00	AREF pin, interna referenca isključena
01	AVcc pin sa eksternim kondenzatorom na AREF pinu
10	rezervisano
11	Interna referenca od 1.1 V sa eksternim kondenzatorom na AREF pinu

Preduslov za početak A/D konverzije jeste da je A/D konvertor uključen setovanjem ADEN (eng. *ADC Enable*) bita u ADCSRA registru. Proces A/D konverzije, prikazan na talsnom dijagramu na slici 59, započinje setovanjem ADSC (eng. *ADC Start Conversion*) bita u ADCSRA registru i on će ostati setovan tokom procesa konverzije. Završetkom A/D konverzije koja traje 13 taktnih impulsa u slučaju maksimalne rezolucije, ovaj bit se automatski resetuje i rezultat A/D konverzije se upisuje u ADCH i ADCL registre. Podrazumevano je da je 10-bitni rezultat desno poravnat (8 bitova rezultata A/D konverzije nalazi se u ADCL registru a preostala dva bita u ADCH registru). Rezultat može biti i levo poravnat setovanjem ADLAR (eng. *ADC Left Adjust Result*) bita u ADMUX registru. S obzirom da čitanje desetobitne vrednosti zahteva čitanje dva registra, čitanjem ADCL registra privremeno se zabranjuje A/D konvertoru upis rezultata dok se ne pročita i sadržaj ADCH registra kako bi se sprečilo nevalidno ponašanje.



Slika 59. Talasni dijagram pojedinačne A/D konverzije

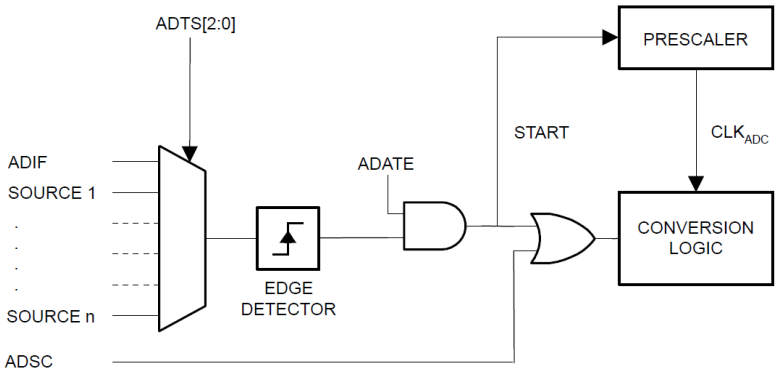
Arduino IDE okruženje poseduje ugrađenu funkciju *analogRead(pin)*, koja izvršava pojedinačnu A/D konverziju na odabranom pinu i kao rezultat vraća celobrojnu vrednost. Funkcija najpre korišćenjem bitmaski podešava naponsku referencu i željeni kanal i pokreće proces A/D konverzije setovanjem ADSC bita. Ova je funkcija blokirajuća jer čeka završetak A/D konverzije u *while* petlji proveravajući status ADSC bita. Po završetku A/D konverzije bit se resetuje i korišćenjem ugrađene ADC makronaredbe najpre se čita sadržaj ADCL registra koji se sabira sa vrednošću ADCH registra koja je pomerena za 8 binarnih mesta u levo.

```
int analogRead(uint8_t pin){
      ....
      ADMUX = (analog_reference << 6) | (pin & 0x07);
      .....
      // start the conversion
      sbi(ADCSRA, ADSC);
      // ADSC is cleared when the conversion finishes
      while (bit_is_set(ADCSRA, ADSC));
      // ADC macro takes care of reading ADC register.
      // avr-gcc implements the proper reading order:
      // ADCL is read first.
      return ADC;
   }
```

Poziv *analogRead(pin)* funkcije pokreće A/D konverziju na odabranom pinu setovanjem ADSC bita, pri čemu se u *while* petlji čeka kraj A/D konverzije koji se signalizira automatskim resetom ADSC bita. Alternativno, može se omogućiti prekid A/D konvertora preko ADIE (eng. *ADC Interrupt Enable*) bita koji će po završetku A/D konverzije setovati

ADIF bit (eng. *ADC Interrupt Flag*) i pozvati odgovarajuću ADC prekidnu rutinu. Na taj način, procesor može izvršavati druge delove korisničke aplikacije umesto da čeka na završetak A/D konverzije kao u slučaju *analogRead(pin)* funkcije.

Proces A/D konverzije može se pokretati automatski od strane različitih periferija prema šemi na slici 60, pod uslovom da je ADATE bit (eng. *ADC Auto Trigger Enable*) setovan. Izbor periferije koja će automatski pokrenuti proces A/D konverzije odabira se pomoću ADTS[2:0] (eng. *ADC Trigger Select*) bitova prema tabeli 18.



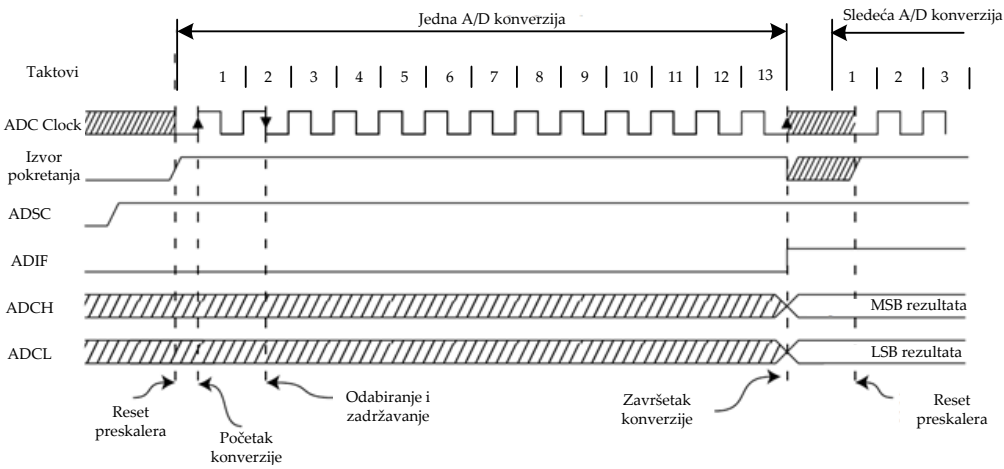
Slika 60. Kolo za automatsko pokretanje A/D konverzije

Tabela 18. Podešavanje izvora za pokretanje A/D konverzije

ADTS[2:0]	**Izvor pokretanja A/D konverzije**
000	ADIF bit prekid
001	Analogni komparator
010	Eksterni zahtev za prekid INT0
011	Timer/Counter0 Compare Match A
100	Timer/Counter0 Overflow
101	Timer/Counter1 Compare Match B
110	Timer/Counter1 Overflow
111	Timer/Counter1 Capture Event

Kada se detektuje pozitivna ivica signala iz odabrane periferije, dolazi do automatskog pokretanja A/D konverzije. Kako bi se pokrenula nova A/D konverzija, neophodno je da se detektuje pozitivna ivice signala za automatski start. Ukoliko je signal konstantan nakon završetka tekuće konverzije nova A/D konverzija neće biti pokrenuta. Po završetku A/D konverzije doći će do setovanja ADIF bita. Kako bi A/D konvertor mogao

automatski da pokrene novu A/D konverziju korišćenjem ADIF bita neophodno je njegovo resetovanje. On može biti resetovan ručno u slučaju da se ne koriste prekidi ili automatski ulaskom u ADC prekidnu rutinu. Ostali izvori prekida vezani su za tajmerske module koji se mogu podesiti da generišu zahteve za A/D konverziju u preciznim i pravilnim vremenskim intervalima. Talasni dijagram automatki pokrenute A/D konverzije prikazan je na slici 61.



Slika 61. Talasni dijagram automatski pokrenute A/D konverzije

Integrisani temperaturni senzor

ATmega328p mikrokontroler poseduje integrisani temperaturni senzor sa osetljivošću od 1 mV/°C koji je povezan na osmi ulaz analognog multipleksera A/D konvertora. Tipične vrednosti napona očitanog sa senzora pri različitim temperaturama prikazane su u tabeli 19.

Tabela 19. Vrednosti izlaznog napona integisanog temperaturnog senzora

Temperatura senzora	Napon temperaturnog senzora
-45 °C	242 mV
+25 °C	314 mV
+85 °C	380 mV

Navedene vrednosti napona integrisanog temperaturnog senzora mogu znatno varirati usled varijacija proizvodnog procesa, tako da je neophodno izvršiti kalibraciju senzora kako bi se poboljšala tačnost merenja. S obzirom da nije moguće pročitati vrednost internog senzora temperature

korišćenjem *analogRead* funkcije, sledeći listing koda prikazuje postupak čitanja temperature preko registara.

```
ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
ADCSRA |= _BV(ADEN); // Ukljucenje ADCa
delay(20);
ADCSRA |= _BV(ADSC); // Početak AD konverzije
while (bit_is_set(ADCSRA,ADSC));
unsigned int tADC = ADCW; //Vrednost temperature tADC
float t=(((ADCW*1100.0)/1023.0)-242.0)*130.0/138.0-45.0;
```

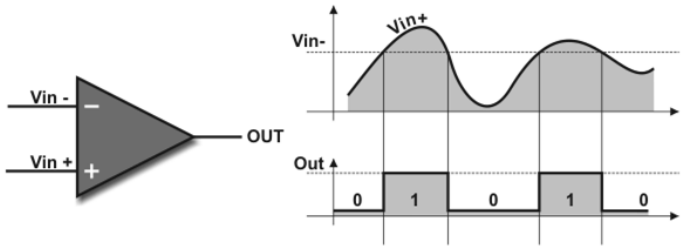
Interna naponska referenca

ATmega328p mikrokontroler poseduje internu naponsku referencu koja je povezana na četrnaesti ulaz analognog multipleksera. Ova referenca generiše poznati napon od 1.1 V, čijom se A/D konverzijom dobija digitalna vrednost koja zavisi od opsega A/D konvertora. U slučaju da se mikrokontroler napaja neregulisanim naponom iz baterije kod koje napon vremenom opada, korišćenjem interne naponske reference moguće je izmeriti napon baterije ako se on proglasi za pozitivni referentni napon. Na taj način moguće je pratiti pražnjenje baterije i korisniku signalizirati kada je vreme da se baterija zameni ili dopuni. Sledeći listing koda prikazuje kako se može izmeriti napon napajanja Vcc.

```
ADMUX = (_BV(REFS0) | _BV(MUX3)| _BV(MUX2)| _BV(MUX1));
ADCSRA |= _BV(ADEN); // Ukljucenje ADCa
delay(20);
ADCSRA |= _BV(ADSC); // Početak AD konverzije
while (bit_is_set(ADCSRA,ADSC));//Cakaj kraj AD konverzije
unsigned int Vbg = ADCW; //Vrednost bandgap reference Vb
float Vcc=(1023.0*1.1)/float(Vbg); //Napon napajanja Vcc
```

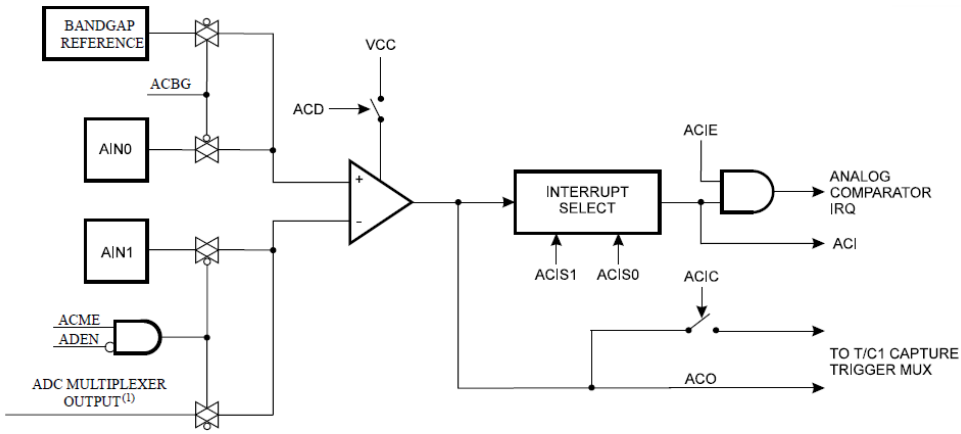
Analogni komparator

Analogni komparator jeste komponenta koja poredi dva analogna napona na svojim ulazima i generiše digitalni izlaz. Usled velikog pojačanja mala pozitivna ili negativna razlika napona između njegovih ulaza biće pojačana do tog nivoa da će izlaz komparatora otići u zasićenje. Kada je analogni napon na V_{IN-} ulazu veći od napona na V_{IN+} ulazu izlaz komparatora na niskom je nivou. U suprotnom slučaju, izlaz komparatora na visokom je nivou kao što je prikazano na slici 62.



Slika 62. Princip rada analognog komparatora

ATmega328p mikrokontroler poseduje jedan analogni komparator čija je stuktura prikazana na slici 63. Njegov neinvertujući ulaz povezan je sa pinom AN0 (PD6) ili internom naponskom referencom koja se selektuje setovanjem ACBG bita. Invertujući ulaz povezan je sa pinom AN1 (PD7) i analognim multiplekserom koji se selektuje pomoću ACME i ADEN bitova (neophodno je isključiti A/D konvertor kako bi se mogao koristiti njegov analogni multiplekser kao izvor signala za invertujući ulaz komparatora). Izlaz komparatora može pokrenuti Timer/Counter1 Input Capture function ili generisati ANALOG COMP prekid u zavisnosti od postavljenog uslova za generisanje prekida preko ACIS[1:0] bitova prema tabeli 20.



Slika 63. Struktura analognog komparatora ATmega328p mikrokontrolera

Tabela 20. Podešavanje tipa prekida analognog komparatora

ACIS[1:0]	**Tip prekida komparatora**
00	Prekid na pomenu izlaza komaparatora
01	Rezervisano
10	Prekid na silaznu ivicu izlaza komaparatora
11	Prekid na uzlaznu ivicu izlaza komaparatora

POGLAVLJE 9: SERIJSKI KOMUNIKACIONI MODULI

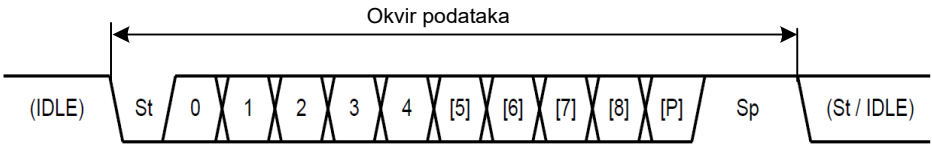
Mikrokontroler može razmenjivati podatke sa perifernim uređajima paralelno ili serijski. Paralelni prenos podrazumeva istovremeno slanje podatka preko n linija, dok serijski prenos podrazumeva da se isti podatak šalje jednom linijom ali kroz n sukcesivnih taktova. Paralelni prenos značajno je brži u odnosu na serijski, pa se on prvenstveno koristi za razmenu podataka unutar mikrokontrolera. Međutim, paralelni prenos jako je nepraktičan za komunikaciju sa periferijama van mikrokontrolera, prvenstveno što zahteva jako veliki broj pinova. Danas u većini mikrokontrolerskih sistema koristi se serijski prenos, koji omogućava znatno efikasniju komunikaciju, čak i na većim udaljenostima.

Serijski prenos koristi pomerački registar kojim se podatak koji dolazi iz mikrokontrolera u paralelnom formatu transformiše u niz bitova koji se šalju jedan za drugim serijskom linijom veze. U prijemniku podaci se sa serijske linije upisuju u pomerački registar, iz kojeg ih mikrokontroler može pročitati u paralelnom formatu kada se završi prijem podataka. Kako bi prijemnik i predajnik mogli međusobno da razmenjuju podatke, potrebno je da poštuju određeni skup unapred definisanih pravila koje definiše komunikacioni protokol. Takođe, predajnik i prijemnik moraju biti međusobno sinhronizovani kako bi ispravno razmenjivali podatke. Sinhronizacija između prijemnika i predajnika može se ostvariti korišćenjem posebne linije za prenos taktnog signala, pa se u tom slučaju govori o sinhronim serijskim protokolima. U slučaju sinhronih serijskih protokola, prijemnik i predajnik permanentno su sinhronizovani signalom koji generiše jedan od njih (najčešće master uređaj).

Asinhrona serijska komunikacija

Asinhroni serijski protokoli ne koriste posebnu liniju za takt već ostvaruju sinhronizaciju preko linije za podatke. Ova sinhronizacija nije permanentna, već se obavlja samo na početku transmisije. Usled razlika u radnim taktovima lokalnih oscilatora u prijemniku i predajniku, posle određenog vremena dolazi do gubitka sinhronizacije, pa se postupak sinhronizacije mora ponoviti.

Jedan od najčešće korišćenih periferija za serijsku komunikaciju jeste asinhroni serijski primopredajnik UART (eng. *Universal Asynchronous Receiver Transmiter*). Ovaj protokol koristi posebne linije za slanje i prijem podataka i ne koristi liniju za takt. Sinhronizacija se obavlja na početku poruke, prikazane na slici 64, korišćenjem start bita, tako što predajnik promeni logički nivo na liniji sa logičke jedinice koja označava mirovanje (eng. *idle*) na logičku nulu. Ovu tranziciju na liniji koristi prijemnik kako bi sinhronizovao svoj generator takta.

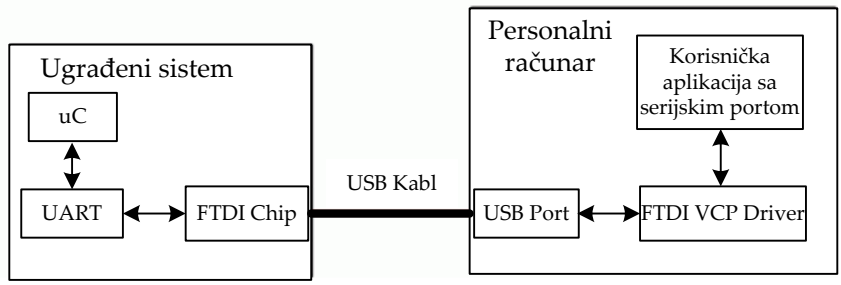


Slika 64. Struktura UART poruke

Prijemnik i predajnik koriste istu bitsku brzinu (eng. *baud rate*) koja definiše vreme trajanja jednog bita na liniji. Zatim predajnik šalje određeni broj bitova podataka koji se kreće od 5 do 8, pri čemu se prvo šalje bit najmanje težine. Nakon prenosa poslednjeg bita predajnik može opciono poslati bit parnosti (eng. *parity bit*) koji definiše broj jedinica u poruci i ona može biti parna (eng. *even*) ili neparna (eng. *odd*). Bit parnosti koristi se kako bi se detektovale jednostruke bitske greške u podatku, jer će usled promene jednog bita doći i do promene parnosti broja jedinica. Treba napomenuti da bit parnosti ne može detektovati dvostruke greške jer se kod njih menjaju dva bita pa ne dolazi do promene parnosti, ali ovaj tip grešaka mnogo je ređi u odnosu na jednostruke greške. Poruka se završava sa jednim ili dva stop bita koja odvode liniju na stanje logičke jedinice. Kada se završi prenos jedne poruke, može se započeti sa prenosom sledeće

poruke ili linija može ostati na neaktivnom stanju ukoliko je prenos završen. Neophodno je da i prijemnik i predajnik koriste isti format komunikacije kako bi uspešno razmenjivali poruke. Format razmene poruka definiše se brojem bitova podataka, tipom parnosti i brojem stop bitova, pri čemu je start bit implicitno podrazumevan. Primer formata UART poruke 8E1 definiše da se prenosi 8 bitova podataka, sa parnim bitom parnosti i jednim stop bitom. Bitska brzina prenosa zadaje se u bitovima po sekundi, pa bitska brzina od 115200 bps (eng. *bits per second*) predstavlja razmenu 115200 bitova u jednoj sekundi, uključujući i start, bit parnosti i stop bitove, pri čemu je vreme trajanja jednog bita 8.7 μs.

UART modul za komunikaciju koristi TTL (eng. *Transistor Transistor Logic*) naponske nivoe logičke nule i jedinice, međutim sam format poruke kompatibilan je sa RS232 i RS485 interfejsima koji koriste drugačije naponske nivoe, pa je u tu svrhu neophodno koristiti odgovarajuća integrisana kola koja služe za translaciju nivoa. RS232 interfejs ranije se koristio kod personalnih računara za serijsku komunikaciju na razdaljinama do 15 m, ali ga je u poslednje vreme u personalnim računarima u potpunosti potisnuo USB interfejs. Ovaj interfejs koristi integrisana kola koja, kada se povežu na računar, kreiraju virtuelni serijski port preko kojeg se ostvaruje komunikacija sa Arduino UNO razvojnim sistemom kao što je prikazano na slici 65.

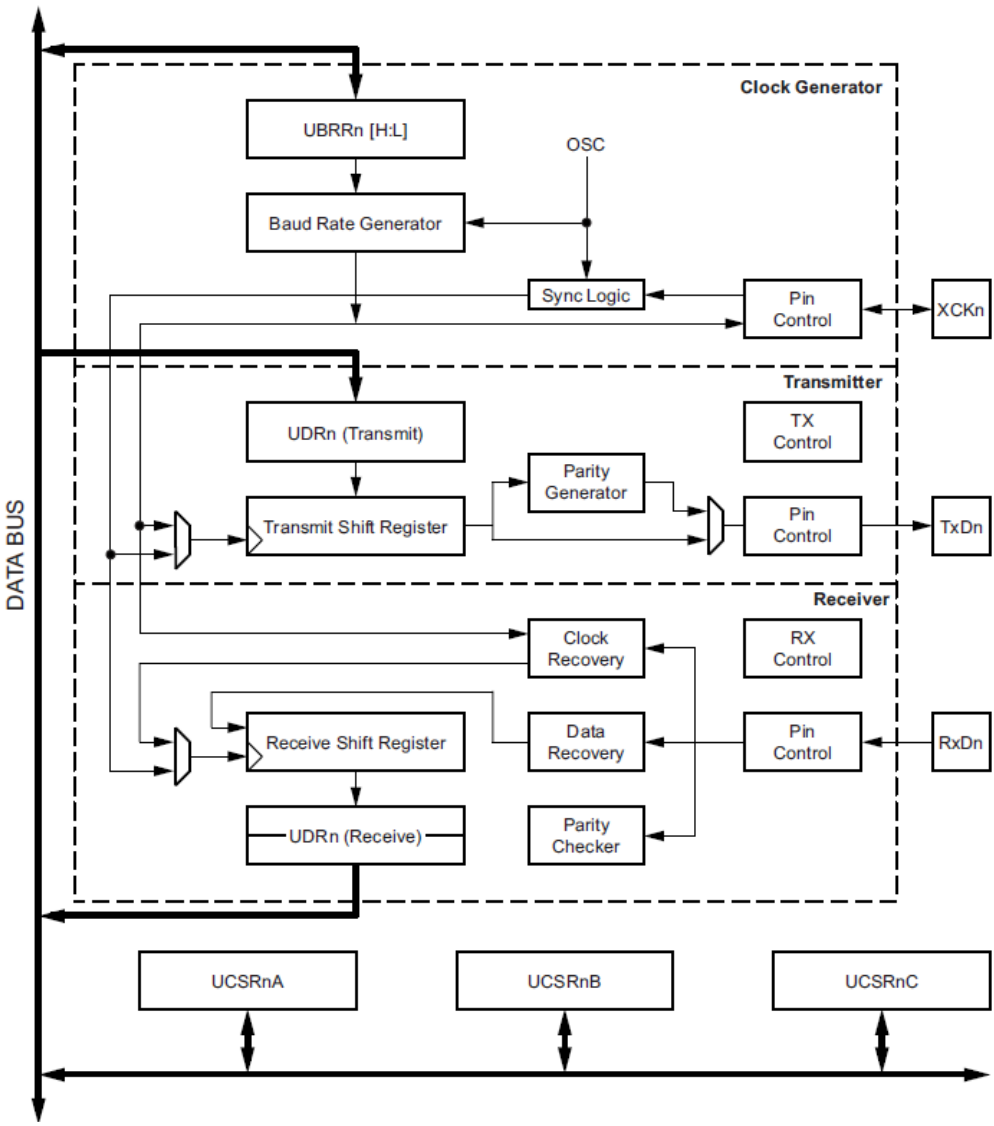


Slika 65. UART komunikacija sa računarom preko USB porta

RS485 interfejs najčešće se koristi u industriji za komunikaciju sa različitim tipovima uređaja kao što su senzori, aktuatori, frekventni regulatori i omogućava komunikaciju na razdaljinama do 1200 m.

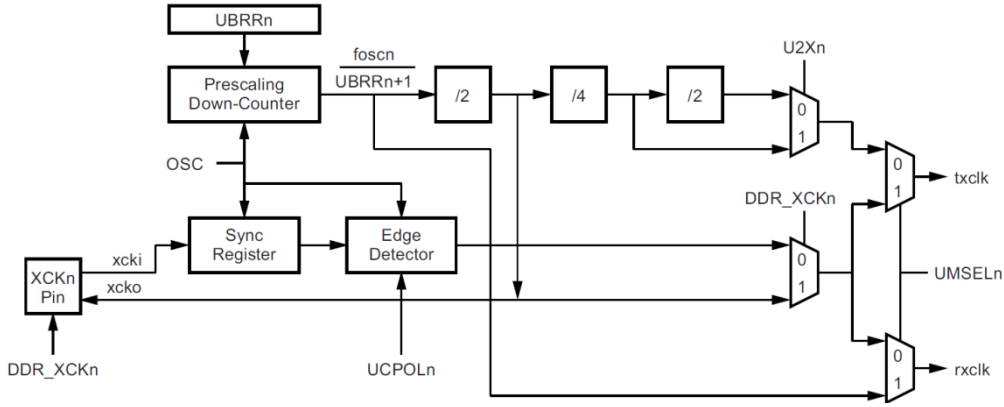
Asinhrono/sinhroni USART modul

ATmega328p mikrokontroler poseduje USART (eng. *Universal Synchronous Asynchronous Receiver Transmitter)* periferiju koja podržava asinhroni i sinhroni prenos. Ova jedinca, čija je stuktura prikazana na slici 66, sastoji se od generatora bitske brzine (eng. *baud rate generator*), predajnika, prijemnika i skupa kontrolnih registara. U slučaju da se USART modul koristi u sinhronom režimu rada generator takta sinhronizovaće svoj signal takta korišćenjem XCK linije, koja se nalazi na pinu PD4.



Slika 66. Struktura USART modula

Generator takta, prikazan na slici 67, koristi sistemski takt oscilatora koji preko preskalera usporava do željene bitske brzine. Usporavanje sistemskog takta vrši se preko 16-bitnog preskalera sa brojačem unazad koji generiše impuls takta svaki put kada odbroji do nule. Ciklus odbrojavanja započinje upisom vrednosti iz UBRR (eng. *USART Baud Rate Register*) registra u brojač koji odbrojava do nule, generiše impuls takta i ponovo upisuje početnu vrednost za sledeći ciklus.



Slika 67. Struktura generatora bitske brzine

Bitska brzina zavisna je i od niza kontrolnih bitova koji definišu režim rada USART modula i predstavljena je tabelom 21.

Tabela 21. Podešavanje bitske brzine USART modula

USART režim rada	**Bitska brzina**
Asinhroni režim (U2Xn = 0)	$BAUD = \frac{f_{osc}}{16(UBRR+1)}$
Asinhroni brzi režim (U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRR+1)}$
Sinhroni master režim	$BAUD = \frac{f_{osc}}{2(UBRR+1)}$

Usled celobrojne vrednosti koja se upisuje u UBRR registar generator će generisati takt koji se malo razlikuje od željene bitske brzine. Tipične bitske brzine koje se mogu ostvariti korišćenjem sistemskog takta od 16 MHz kod Arduino UNO razvojnog sistema prikazane su u tabeli 22. kao i očekivana greška generatora bitske brzine. Može se uočiti da su najveće greške generatora u slučaju bitskih brzina 115200 bps i 230400 bps koje iznose nekoliko procenata.

Tabela 22. Greška bitske brzine USART modula

Bitska brzina [bps]	f_{osc}=16 MHz			
	U2X=0		U2Xn=1	
	UBRN	Greška	UBRN	Greška
2400	416	- 0.1%	832	0.0 %
4800	207	0.2 %	416	- 0.1%
9600	103	0.2 %	207	0.2 %
14.4k	68	0.6 %	138	- 0.1 %
19.2k	51	0.2 %	103	0.2 %
28.8k	34	-0.8 %	68	0.6 %
38.4k	25	0.2 %	51	0.2 %
57.6k	16	2.1 %	34	-0.8 %
76.8k	12	0.2 %	25	0.2 %
115.2k	8	-3.5 %	16	2.1 %
230.4k	3	8.5 %	8	-3.5 %
250k	3	0.0 %	7	0.0 %
0.5M	1	0.0 %	3	0.0 %
1M	0	0.0 %	1	0.0 %
Max	1 Mbps		2 Mbps	

Za transmisiju podataka USART modul koristi TXD pin koji se nalazi na pinu PD0. Pre početka transmisije neophodno je podesiti format komunikacije i generator bitske brzine korišćenjem ugrađene funkcije Arduino IDE programskog okruženja *begin(speed,config)*. Ova funkcija kao parametre očekuje bitsku brzinu *speed* koja se može zadati u skladu sa standardnim bitskim brzinama prikazanim u prethodnoj tabeli. Parametar *config* je opcioni i, ako se on izostavi, podrazumevani format komunikacije 8N1 sastoji se od osam bitova podataka, bez bita parnosti i sa jednim stop bitom. U slučaju potrebe za drugačijim formatom na raspolaganju su 24 različita formata komunikacije. Ova funkcija konfiguriše pinove PD0 i PD1 kao pinove za komunikaciju i u tom slučaju nije preporučeno da se koriste kao pinovi opšte namene. Pozivom funkcije *end()* onemogućava se serijska komunikacija i ovi se pinovi potom mogu koristiti kao pinovi opšte namene. Transmisija podataka započinje se upisom podatka u UDR registar i pod uslovom da je predajnik aktiviran preko Transmit Enable (TXEN) bita on će podatak poslati preki TXD pina predefinisanom bitskom brzinom i formatom. Kada se podatak preuzme iz prijemnog bafera

signalizirаće se da je prazan preko UDRE (eng. *USART Data Register Empty*) bita i tada se može upisati novi podatak u UDR registar. Po završetku transmisije pod uslovom da u transmisionom baferu nema novih podataka za slanje setovaće se bit TXC (eng. *Transmit Complete*). Oba ova bita u stanju su da generišu odgovarajće prekide, ukoliko su isti omogućeni. Ukoliko se koristi format sa bitom parnosti, isti će automatski biti izračunat i umetnut nakon poslednjeg bita podataka a pre prvog stop bita.

Arduino IDE koristi ugrađenu funkciju write() koja šalje jednu ili niz vrednosti preko TXD linije. Takođe na raspolaganju su i Serial.print(),Serial.println() funkcije koje prosleđeni parametar pretvaraju u niz ASCII karaktera i isti šalju preko TXD linije, pri čemu println() funkcija dodaje i neštampajuće karaktere /r/n za prelazak u novi red.

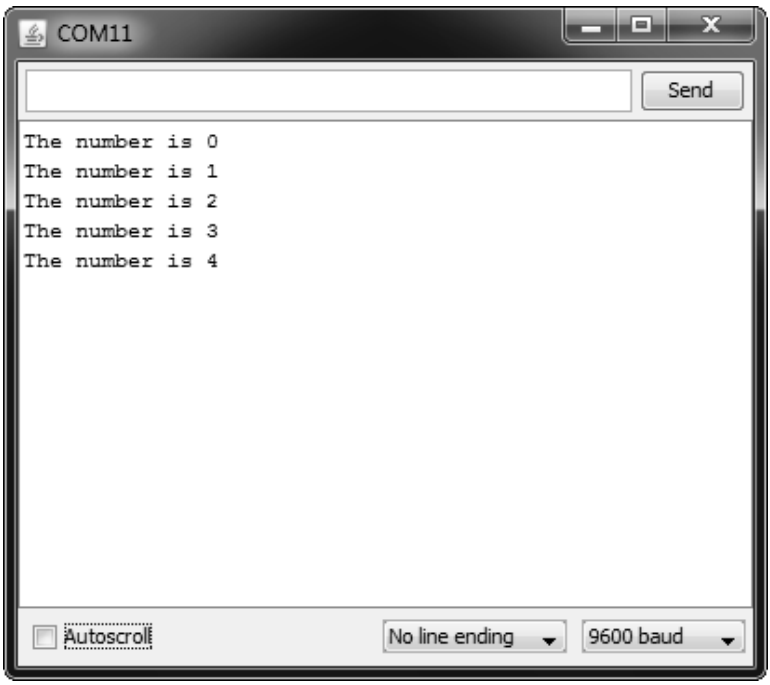
Prijem podataka započinje omogućavanjem prijemnika preko RXEN (eng. *Receive Enable*) bita. Kada prijemnik detektuje start bit, počeće da pomera bitove podataka u svoj pomerački registar i, kada detektuje prvi stop bit, premestiće primljeni podatak u prijemni bafer, odakle se može pročitati preko UDR registra. Prilikom prijema može doći do niza grešaka koje se signaliziraju preko odgovarajućih bitova. Ako je prijemni bafer popunjen i detektovan je start bit novog karaktera, postaviće se DOR (eng. *Data OverRun*) bit koji obaveštava da novi podatak ne može biti primljen. Kada se na očekivanoj poziciji ne detektuje stop bit, FE (eng. *Frame Error*) bit će se setovati kako bi ukazao na nevalidnu poruku. Ukoliko je omogućen bit parnosti, generator parnosti automatski će proveriti parnost primljene poruke i, ukoliko detektuje grešku, setovaće UPE (eng. *USART Parity Error*) bit na jedinicu.

Po završetku prijema setovaće se RXC (eng. *Receive Complete*) bit kako bi ukazao da u prijemnom baferu postoje nepročitani podaci. Ovaj bit generisaće zahtev za prekid u slučaju da su prekidi omogućeni. Arduino IDE ugrađena metoda *Serial.available()* proverava da li postoje nepročitani podaci u prijemnom baferu. U slučaju da podaci postoje, isti se mogu pročitati korišćenjem ugrađene metode *Serial.read()*, koja vraća primljeni podatak. Na raspolaganju su i metode koje omogućavaju čitanje niza bajtova *Serial.readBytesUntil(character, buffer, length)* u bafer određene

veličine sve dok se ne detektuje terminacioni karakter ili istekne predefinisano vreme čitanja podešeno pomoću metode *Serial.setTimeout()*. Metode *Serial.parseInt()* i *Serial.parseFloat()* omogućavaju prijem celobrojne i vrednosti u pokretnom zarezu dok ne istekne predefinisano vreme prijema.

Sledeći primer prikazuje korišćenje ugrađenih metoda klase Serial za serijski prenos bitskom brzinom 9600bps i štampanje vrednosti promenljive koja svakom iteracijom uvećava svoju vrednost na serijskom monitoru prikazanom na slici 68.

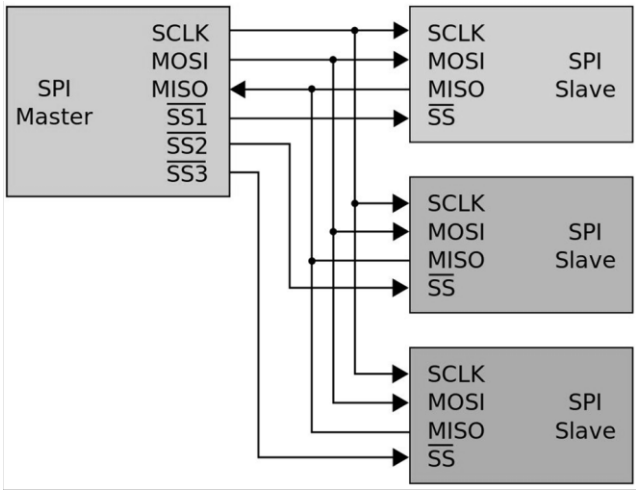
```
int number = 0;
void setup(){
 Serial.begin(9600); // Bitska brzina 9600 bps
}
void loop(){
 Serial.print("The number is ");
 Serial.println(number); // oštampaj broj
 delay(500); // zadrška od pola sekunde
 number++; // Uvećaj broj za jedan
 }
```



Slika 68. Primljene vrednosti na serijskom monitoru

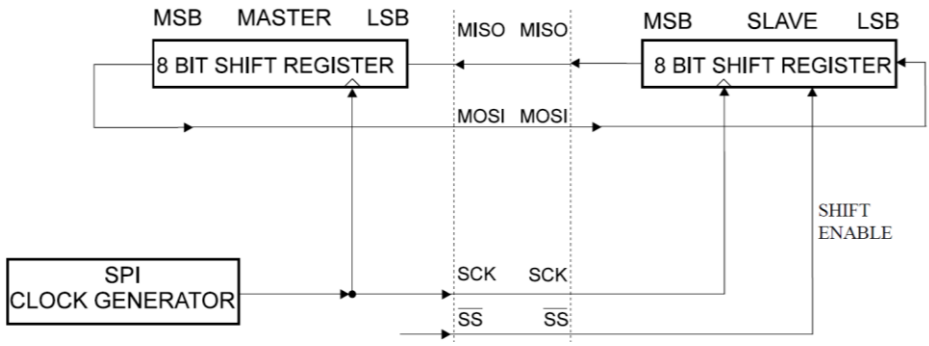
Serijska sinhrona komunikacija

Serijski periferni interfejs (SPI) predstavlja sinhroni komunikacioni interfejs, razvijen od strane Motorole sredinom 80-tih, koji je postao jedan od najčešće korišćenih protokola u ugrađenim sistemima. Ovaj interfejs omogućava prenos podataka velikim brzinama na kratkim udaljenostima (na nivou štampane ploče). SPI interfejs omogućava bidirekcioni (eng. *Full Duplex*) serijski prenos između glavnog uređaja (eng. *master*) i jednog perifernog uređeja ili više njih (eng. *slaves*). Na magistrali u jednom trenutku može biti aktivan samo master uređaj i jedan slejv uređaj, dok ostali slejv uređaji moraju biti pasivni. Master je sa svakim slejvom povezan preko posebne SS (eng. *Slave Select*) linije putem koje se vrši hardversko adresiranje kao što je prikazano na slici 69. Master inicira komunikaciju sa željenim slejvom postavljanjem njegove linije na stanje logičke nule, dok sve ostale SS linije ostaju na stanju logičke jedinice.



Slika 69. Princip hardverskog adresiranja na SPI magistrali

Kako je u pitanju sinhroni protokol, master je zadužen za generisanje signala takta SCK, koje se vodi do svih slejv uređaja. Postupak prenosa obavlja se između pomeračkih registara u masteru i slejvu preko MISO (eng. *Master Input Slave Output*) i MOSI (eng. *Master Output Slave Input*) linija koje formiraju topologiju prstena prikazanu na slici 70. SPI se naziva još i četvorožična veza zbog četiri signala koji učestvuju u komunikaciji (MISO, MOSI, SCK i SS).



Slika 70. Način funkcionisanja SPI komunikacije

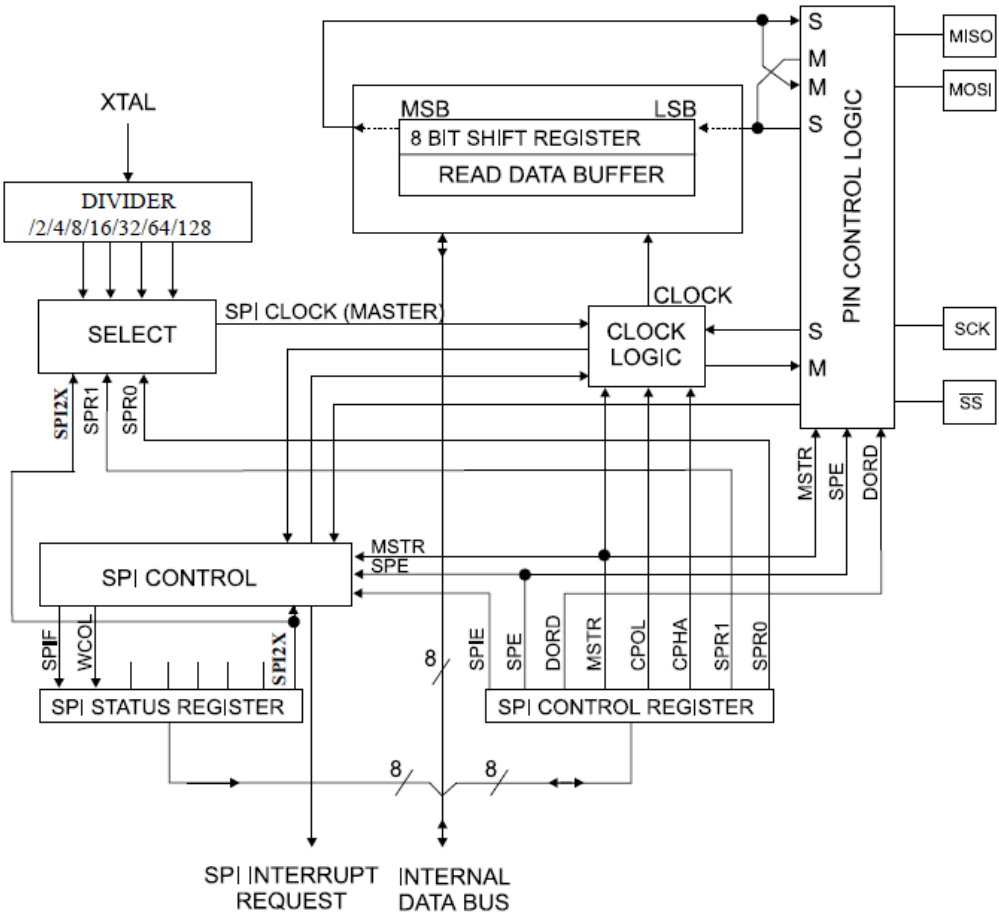
Tokom svakog SPI takta dolazi do ciklične razmene po jednog bita između mastera i slejva. Pomeranjem sadržaja registra ulevo master istiskuje svoj MSB bit na MOSI liniju, koji se u slejvu upisuje na poziciju LSB bita, dok slejv istiskuje svoj MSB bit na MISO liniji koji se u masteru upisuje na poziciju LSB bita. Prenos se izvodi za neku fiksnu veličinu reči, uobičajeno osam bitova. Nakon osam taktova sadržaji dva registra međusobno će se razmeniti. U slučaju da je potrebno nastaviti razmenu podataka u pomerački registar upisuje se nova vrednost i celokupan se proces nastavlja. Kada se prenos završi, master prestaje da generiše SCK signal takta i posavlja SS liniju slejva na logičku jedinicu.

SPI modul

SPI modul ATmega328p mikrokontrolera, preko pinovima prikazanim u tabeli 23, može raditi kao master ili slejv. Osnovni deo SPI modula (slika 71), čini pomerački SPDR (eng. *SPI Data Register*) registar, preko kojeg se odvija razmena podataka na SPI magistrali. Uloga SPI modula definiše se pomoću MSTR (eng. *Master/Slave Select*) bita u SPCR kontrolnom registru. U slučaju da je ovaj bit setovan na logičku jedinicu SPI modul imaće ulogu mastera, a u slučaju da je resetovan SPI modul imaće ulogu slejva.

Tabela 23. Raspored interfejsa SPI modula

SPI Interfejs	ATmega328p pin	Arduino UNO pin
SCK	PB5	13
MISO	PB4	12
MOSI	PB3	11
SS	PB2	10



Slika 71. Struktura SPI modula

Kada je SPI modul postavljen u ulogu mastera, neophodno je korišćenjem funkcije *digitalWrite* softverski aktivirati SS liniju odgovarajućeg slejv uređaja. Transmisija započinje upisom podatka u pomerački SPDR registar, čime se automatski pokreće generator takta koji će istisnuti bit po bit na magistralu nakon čega će prestati da generiše signal takta i setovati SPIF (eng. *SPI Interrupt Flag*) bit kako bi signalizirao kraj transmisije. U slučaju da su omogućeni prekidi preko SPIE (eng. *SPI Interrupt Enable*) bita u SPCR registru generisaće se SPI STC zahtev za prekid. Smer generisanja bitova zavisi od podešavanja DORD bita i u slučaju da je podešen na nulu najpre se istiskuje MSB, a u slučaju da je podešen na logičku jedinicu prvo će se istiskivati LSB bit podatka. Na kraju transmisije podatak koji je poslao slejv završiće u pomeračkom registru mastera iz kojeg se može isčitati. Frekvencija signala takta SPI magistrale

podešava se preko preskalera koji deli sistemski takt u odnosima koji se podešavaju bitovima SPI2X i SPR0 prema tabeli 24. Master može nastaviti sa upisom sledećeg bajta u SPDR registar ili će završiti transmisiju postavljanjem SS linije na stanje logičke jedinice.

Tabela 24. Podešavanje preskalera SPI modula

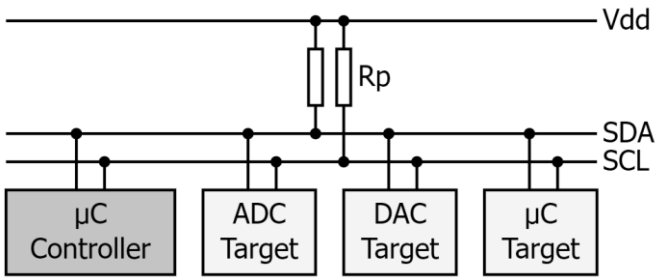
SPI2X	SPR0 [1:0] bitovi	Učestanost preskalera
0	00	Fs/4
0	01	Fs/16
0	10	Fs/64
0	11	Fs/128
1	00	Fs/2
1	01	Fs/8
1	10	Fs /32
1	11	Fs/64

U slučaju da je SPI modul podešen kao slejv on će biti neaktivan sve dok se SS linija ne povuče od strane mastera na nizak naponski nivo. Slejv može upisati podatak u svoj SPDR registar, ali do njegovog pomeranja neće doći sve dok se ne pojave signali takta na SCK liniji i kada je njegova SS linija aktivna. Po završetku istiskivanja podatka iz SPDR registra setovaće se SPIF (eng. *SPI Interrupt Flag*) bit kako bi signalizirao kraj transmisije i generisaće se SPI STC zahtev za prekid u slučaju da SPIE bit dozvoljava ovaj tip prekida.

I²C serijska komunikacija

SPI interfejs omogućava vrlo brz bidirekcioni prenos između mastera i jednog aktivnog slejv uređaja, brzinama do 10 Mbps. Nedostatak ovog tipa interfejsa jeste to što svaki od N uređaja koji dele zajedničku magistralu mora biti adresiran preko posebne SS linije, što ukupno zahteva 3+N pinova. U slučaju potrebe za većim brojem perifernih uređaja bilo bi neophodno zauzeti veliki broj pinova mikrokontrolera. Kao alternativa SPI interfejsu *Filips* (eng. *Philips*) 1982. godine razvio je I²C (eng. *Inter-Integrated Circuit*) interfejs koji koristi poludupleksni prenos sa većim brojem uređaja na magistrali preko samo dve linije. Za prenos podataka I²C interfejs koristi

SDA (eng. *Serial Data)* bidirekcionu liniju za prijem i slanje podataka između mastera i slejv uređaja. I²C interfejs dozvoljava povezivanje do 127 slejv uređaja koji mogu raditi na I²C magistrali, pri čemu svaki od slejv uređaja poseduje jedinstvenu 7-bitnu adresu preko koje je identifikovan od strane master uređaja. Način povezivanja uređaja na I²C prikazan je na slici 72. Za razliku od SPI interfejsa I²C ne zahteva posebne Slave Select linije. Kako je u pitanju serijski sinhroni interfejs, uređaji na magistrali koriste SCL liniju kojom se prenosi signal takta koji generiše master. U standardnom I²C režimu rada maksimalna frekvencija signala takta iznosi 100 kHz i ona je uslovljena maksimalnom kapacitivnošću magistrale od 400 pF na koju utiče broj priključenih slejv uređaja i dužina magistrale. U I²C režimu visoke brzine maksimalna učestanost signala takta iznosi 400 kHz, pri kapacitivnošću magistrale od 400 pF. I²C komunikacija namenjena je za komunikaciju sa periferijama na znatno manjim brzinama do 100 Kbps u standardnom režimu i 400 Kbps u I²C režimu visoke brzine. Dodatni režimi I²C protokola omogućavaju postizanje brzina i do 3.4 Mpbs.

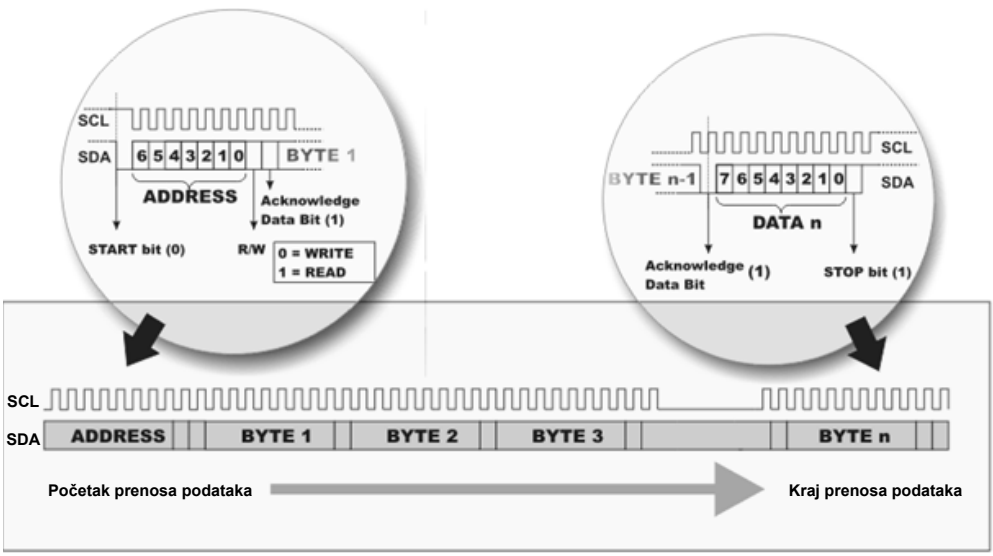


Slika 72. Topologija I²C magistrale

I²C interfejs sastoji se od dva bidirekciona pina SDA i SCL koji rade u režimu otvorenog kolektora ili otvorenog drejna. Konfiguracija otvorenog kolektora, odnosno otvorenog drejna, koristi se kada je potrebno povezati veći broj uređaja na zajedničku liniju magistrale. Ova konfiguracija dozvoljava uređaju da liniju samo može postaviti na stanje logičke nule, a u slučaju da su svi izlazi neaktivni linija se postavlja na stanje logičke jedinice pomoću pull-up otpornika. Tipičan napon I²C magistrale iznosi +5 V ili +3.3 V. Ukoliko uređaji na magistrali koriste različite napone napajanja, potrebno je obezbediti odgovarajuće translatore naponskih nivoa kako bi oni mogli međusobno komunicirati.

I²C interfejs zasnovan je na master/slejv tipu komunikacije, pri čemu na magistrali postoji samo jedan master i do 127 slejv uređaja. Master uređaj zadužen je za generisanje signala takta i on jedini može inicirati komunikaciju. Logička jedinca na I²C magistrali generiše se kada jedan od uređaja postavi svoj izlaz na niski naponski nivo, dok stanje logičke jedinice podrazumeva da su izlazi svih uređaja u stanju visoke impedanse.

Komunikacija na magistrali započinje tako što master povuče SDA liniju na stanje logičke nule čime signalizira zauzeće magistrale, kao što je prikazano na talasnom dijagramu na slici 73.



Slika 73. Format I²C komunikacije na magistrali

Nakon start bita master započinje da generiše takt na SCL liniji pri čemu se na SDA liniji prvo šalje 7-bitna adresa slejv uređaja kojem se master obraća (prvo se šalje MSB bit). Potrebno je obezbediti da na I²C magistrali postoji jednoznačno adresirani slejv uređaj i da ne postoje dva slejv uređaja sa istom adresom. Na raspolaganju su adrese u opsegu od 1 do 127, pri čemu se adresa 0 koristi za difuznu komunikaciju kada je poruka namenjena svim slejv uređajima na magistrali. Tokom transmisije adrese slejv uređaji upoređuju je sa svojom adresom i svi slejv uređaji čija se adresa ne poklapa sa adresom koju je poslao master odlaze u neaktivni režim sve dok se komunikacioni ciklus ne završi.

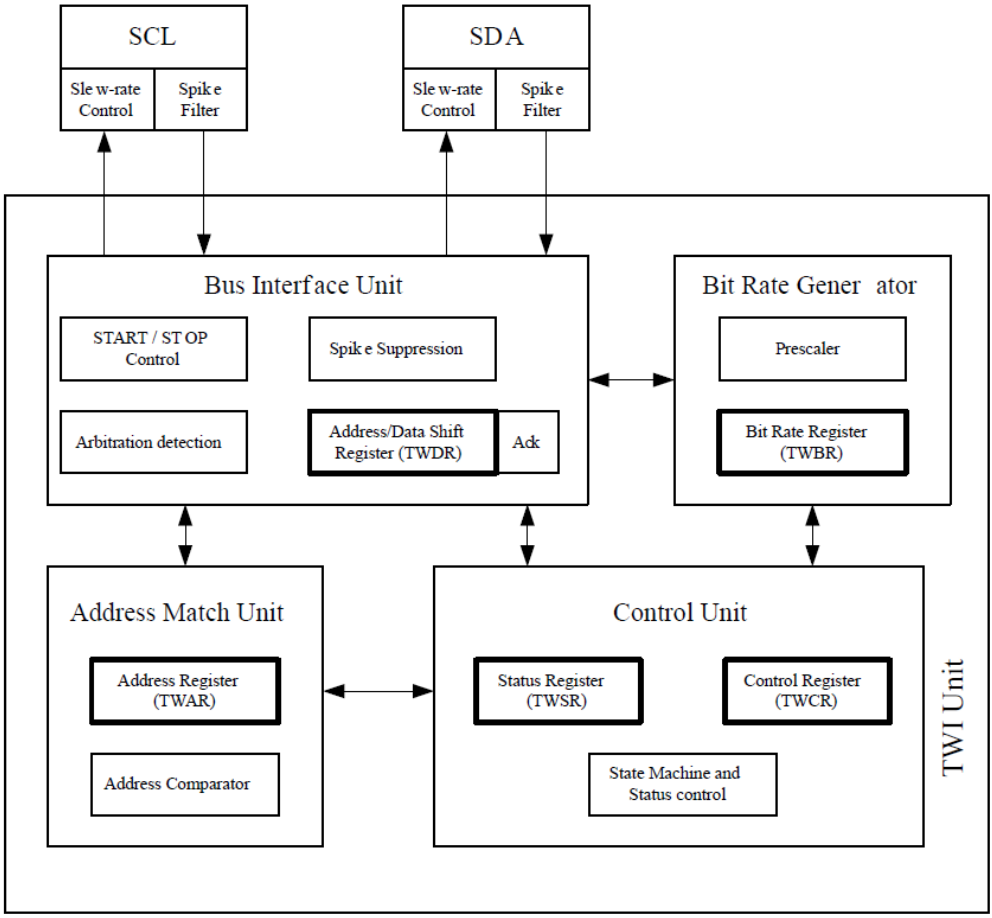
Nakon adrese šalje se R/W bit koji označava da li master želi da pročita ili upiše vrednost u slejv uređaj. U slučaju da je R/W bit nula master zahteva upis vrednosti u slejv, a u slučaju da je R/W jedinica,master zahteva čitanje vrednosti iz slejva. Sledeći ACK bit potvrde (eng. *Acknowledge*) koristi se od strane slejva kako bi se potvrdio da je prisutan na magistrali i može odgovoriti na zahtev mastera. Ukoliko je slejv spreman da odgovori na zahtev mastera, on će povući SDA liniju na stanje logičke nule tokom trajanja ACK bita. Nakon uspešne potvrde slejva master nastavlja da taktuje SCL liniju, nakon čega se razmenjuje niz bajtova između mastera i slejva. Svaki bajt sastoji se od 8 bitova podataka i jednog ACK bita koji se koristi kako bi prijemnik potvrdio uspešan prijem podataka. Transakcija se završava nakon poslednjeg prenetog bajta kada master generiše stop signal postavljanjem SDA linije na logički visoki nivo i prestankom generisanja takta na SCL liniji koja ostaje na visokom naponskom nivou. U slučaju da slejv nije prisutan na magistrali ili nije spreman da odgovori na zahtev mastera SDA linija ostaće na visokom naponskom nivou. Odsustvo potvrde master može iskoristiti da prekine aktivnost na magistrali slanjem stop bita preko SDA linije koji predstavlja povratak magistrale na visoki naponski nivo i prestaće sa generisanjem takta. Druga opcija jeste da master pokuša ponovnu transakciju generisanjem novog start bita (ponovljeni start odnosno restart).

Čest je slučaj da slejv uređaji sadrže podatke koji se nalaze u više registara. Pristup određenom registru u slejv uređaju obezbeđuje se korišćenjem pokazivačkog registra, tako što sledećem bajtu navodi adresu registra kojem želi pristupiti. Zatim master pokreće ponovnu sekvencu transmisije gde opet adresira slejva i upisuje ili čita sadržaj iz prethodno adresiranog registra.

TWI modul

TWI (eng. *Two Wire Interface*) modul ATMega328p mikrokontrolera kompatibilan je sa Philips I²C protokolom. SDA linija za podatke nalazi se na pinu PC4, a SCL linija za takt nalazi se na pinu PC5. Ovaj modul podržava standardni i brzi I²C režim sa maksimalnom frekvencijom

taktnog signala od 400 KHz i može raditi i u master i u slejv režimu. TWI modul poseduje 7-bitnu programabilnu adresu koja se može zadati u opsegu od 1 do 127. Takođe, ovaj modul prepoznaje difuznu adresu 0, koja se odnosi na sve slejv uređaje na magistrali. Prepoznavanje adrese na TWI magistrali generisaće TWI zahtev za prekid. Struktura TWI modula prikazana na slici 74, sastoji se od generatora bitske brzine, interfejsa sa TWI magistralom, komparatora adrese i kontrolne jedinice sa registrima.



Slika 74. Struktura TWI modula

Generator bistke brzine zadužen je za generisanje SCL signala takta kada TWI modul na magistrali ima ulogu mastera. Ovaj modul sastoji se od TWBR (eng. *TWI Bitrate Register*) registra i preskalera, pomoću kojih se podešava željena radna frekvencija prema formuli:

$$f_{SCL} = \frac{f_{osc}}{16 + 2 \cdot \text{TWBR} \cdot \text{Prescaler}}$$

Vrednost preskalera za SCL signal takta podešava se pomoću TWPS [1:0] bitova u TWSR registru prema tabeli 25.

Tabela 25. Podešavanje preskalera TWI modula

TWPS [1:0] bitovi	SCL preskaler
00	1
01	4
10	16
11	64

Arduino IDE programsko okruženje poseduje ugrađenu biblioteku Wire koja omogućava interakciju sa TWI modulom ATmega328p mikrokontrolera. Ova biblioteka podržava rad mikrokontrolera u ulozi mastera na I²C magistrali.

Ugrađena metoda *Wire.setClock(clockFrequency)* podešava frekvenciju SCL linije na željenu vrednost *clockFrequency* tako što postavlja vrednost TWBR registra prema formuli TWBR = ((F_CPU / frequency) – 16) / 2, pri čemu je vrednost preskalera postavljena na podrazumevanu vrednost preskalera od jedan. Tipična vrednost koja se koristi za podešavanje frekvencije SCL linije je 100 kHz.

Jedinica TWI interfejsa sa TWI magistralom povezana je sa SDA i SCL linijama preko kola za ograničavanje stope promene signala i jedinice za suzbijanje impulsa kraćih od 50ns. Ograničavanje stope promene signala koristi se kako bi se ograničila brzina promene napona na liniji magistrale da bi se ograničio uticaj visokofrekventnih komponenti koje mogu izazvati „zvonjenje" linije ili elektromagnetnu interferenciju. Ova jedinica sastoji se od pomeračkog TWDR registra čiji je zadatak istiskivanje i prijem podataka preko SDA linije, START/STOP kontrolera i kola za arbitraciju. START/STOP kontroler zadužen je za generisanje i detekciju START, RESTART i STOP bitova na magistrali. Ova je jedinica u stanju da detektuje START i STOP bitove na magistrali čak i kada je jezgro mikrokontrolera u režimu niske potrošnje i probudi ga kada je adresiran od strane mastera.

Jedinica za proveru adrese u stanju je da proveri da li se dodeljena adresa uređaja u TWAR (eng. *TWI Address Register*) registru poklapa sa primljenom adresom na TWI magistrali nakon START bita, pod uslovom da je automatska provera adrese omogućena.

POGLAVLJE 10: EEPROM I FLASH MEMORIJA

Tokom izvršavanja korisničke aplikacije podaci se čuvaju u RAM memoriji mikrokontrolera, dok se sam program nalazi u FLASH memoriji. Nestankom napajanja gube se svi sačuvani podaci u RAM memoriji. Ovi podaci mogu uključivati i parametre koje je zadao korisnik. Ponovno pokretanje upravljačke aplikacije zahtevalo bi ponovni unos parametara od strane korisnika. Kako bi se sprečio gubitak korisničkih podataka usled nestanka napajanja, mikrokontroleri raspolažu sa EEPROM memorijom (eng. *Electrically Erasable Programmable Read-Only Memory*). Ova memorija predstavlja programabilnu ROM memoriji čiji se sadržaj može brisati elektronskim putem. ATmega328p mikrokontroler sadrži EEPROM memoriju kapaciteta 1KB, koja se sastoji od 1024 adresibilne reči veličine 1B. Memorijskoj reči pristupa se korišćenjem 10-bitne adrese koja se zadaje preko registara EEARH i EEARL, dok se razmena podataka sa memorijskom reči vrši preko 8-bitnog EEDR registra. EEPROM memorija poseduje ograničeni broj upisa u memorijsku lokaciju od bar 100000 operacija upisa/brisanja. EEPROM memorija taktuje se iz internog RC oscilatora, jer je njena maksimalna radna učestanost 8.8 MHz.

EEPROM memorija

Operacija čitanja podatka iz EEPROMa započinje setovanjem EERE (eng. *EEPROM Read Enable*) bita u EECR (eng. *EEPROM Control Register*) registru. Podatak iz EEPROM-a čita se momentalno, pri čemu se procesor zaustavlja u trajanju od četiri taktna ciklusa. Pročitani podatak smešta se u

EEDR registar. U slučaju da nije bilo upisa EEPROM u memorijsku reč, iz reči se čita vrednost 255 (sve jedinice).

Arduino IDE poseduje ugrađenu EEPROM biblioteku sa funkcijom *read(address)* koja vraća pročitani bajt iz EEPROM memorijske lokacije sa adresom *address*. Pre operacije čitanja potrebno je proveriti da li je u toku operacija upisa u EEPROM, kako bi se sprečilo nevalidno čitanje. Takođe, na raspolaganju je i *get(address,type)* funkcija koja omogućava čitanje promenljive *type*, koja se nalazi na adresi *address* i zauzima određeni broj bajtova u EEPROM-u.

Operacija upisa u EEPROM zahteva znatno duže vreme u odnosu na operaciju čitanja koje traje nekoliko milisekundi. Promena vrednosti u EEPROM memoriji zahteva da se memorijska lokacija prvo obriše (eng. *erase cycle*) a zatim u nju upiše nova vrednost (eng. *write cycle*). Obe operacije pojedinačno se izvršavaju za vreme od 1.8 ms. dok atomična operacija brisanja i upisa traje ukupno 3.4 ms. Atomična operacija predstavlja sekvencu operacija koja se mora izvršiti kao celina bez prekidanja. Tip operacije odabira se preko EEPM[1:0] bitova (eng. *EEPROM Programming Mode*) prema tabeli 26.

Tabela 26. Tipovi EEPROM operacija

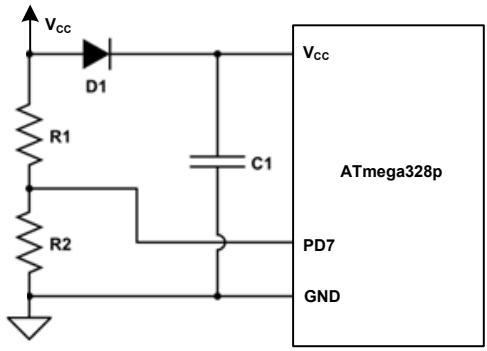
EEPM[1:0] bitovi	**Potrebno vreme**	**Tip operacije**
00	3.4 ms	Brisanje i upis
01	1.8 ms	Brisanje
10	1.8 ms	Upis
11	-	Rezervisano

Pre započinjanja upisa u EEPROM potrebno je postaviti adresu memorijske reči u EEARH i EEARL registre i podatak koji se upisuje u EEDR registar. Zatim je potrebno setovati EEMPE (eng. *EEPROM Master write enable*) bit i u roku od 4 taktna ciklusa setovati EEPE (eng. *EEPROM write enable*) bit, čime će započeti operacija upisa. Operacija upisa završiće se nakon nekoliko milisekundi i resetovaće se EEPE bit. Ukoliko su omogućeni prekidi pomoću EERIE bita, resetovanjem EEPE bita nakon završetka upisa generisaće se EE READY zahtev za prekid.

Arduino IDE, u okviru EEPROM biblioteke, poseduje ugrađenu funkciju *write(addr, val)* koja upisuje vrednost *val* u EEPROM memorijsku lokaciju sa adresom *address*. Pre operacije upisa potrebno je proveriti da li je u toku prethodna operacija upisa u EEPROM, pa će ova funkcija biti blokirana dok se ne završi prethodni upis. Nakon toga postavlja se tip EEPROM operacije upisa na atomični ciklus brisanja i upisa, upisuju se adresa reči i podatak *val* u odgovarajuće registre i pokreće kritična sekcija za upis. Kritična sekcija predstavlja deo programskog koda koja se mora izvršiti kao celina bez prekida, koji su onemogućeni tokom trajanja kritične sekcije. Potrebno je naglasiti da funkcija ne čeka da se upis završi, ali, ako se ponovi upis, on će biti blokiran dok se prethodna operacija upisa ne završi. Za veće tipove podataka na raspolaganju je funkcija *put(address,type)* koja omogućava upis promenljive *type* koja zauzima određeni broj bajtova na početnu adresu *address*. Kako bi se izbeglo upisivanje istovetne vrednosti na EEPROM na raspolaganju je funkcija *update(address, val)* koja prvo čita podatak sa adrese *address* i poredi ga da li je on isti sa vrednošću *val*. Ukoliko su vrednosti jednake upis se neće izvršiti, a ukoliko se razlikuje doći će do operacije upisa.

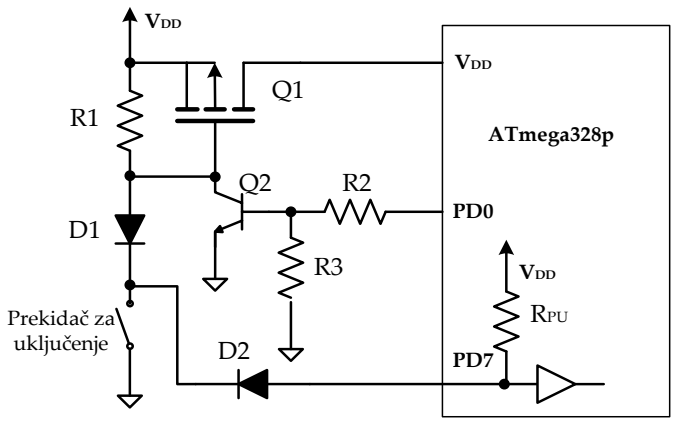
Potrebno je imati u vidu da jedna EEPROM lokacija dozvoljava u proseku 100000 operacija upisa. U slučaju čestih i periodičnih upisa doći će do vrlo brzog habanja korišćene EEPROM lokacije. U tom slučaju potrebno je osmisliti odgovarajući algoritam upisa koji će vrednost ciklično upisivati na veći broj EEPROM lokacija i tako sprečiti habanje EEPROM lokacija.

Preterano habanje EEPROM lokacija može se sprečiti upisom po gubitku napajanja, što se može ostvariti kolom za detekciju gubitka napajanja koje je prikazano na slici 75. Gubitak napajanja detektuje se naponskim razdelnikom koji može pokrenuti odgovarajuću prekidnu rutinu korišćenjem analognog komparatora. Zahvaljujući kondenzatoru i blokirajućoj diodi, mikrokontroler će u prekidnoj rutini imati dovoljno vremena da sačuva sve vrednosti bitnih promenljivih u EEPROM, pre nego što se aktivira BOR reset usled pražnjenja kondenzatora. Ponovnim uključenjem mikrokontrolera potrebno je u setup rutini restaurirati vrednosti promenljivih koje su sačuvane u EEPROM-u prilikom prethodnog isključenja.



Slika 75. Kolo za detekciju gubitka napajanja

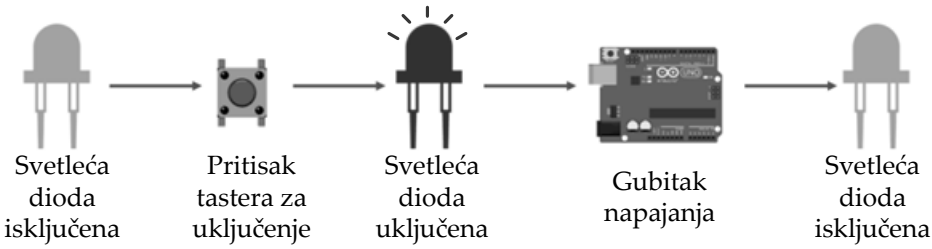
Drugi način za pamćenje bitnih parametara u EEPROM jeste pomoću kola za odloženo isključenje napajanja koje je prikazano na slici 76.



Slika 76. Kolo odloženog isključenje napajanja

ON/OFF prekidač uključuje MOSFET preko kojeg se napaja mikrokontroler, pri čemu se status prekidača prati preko posebnog ulaza sa pull-up otpornikom. Dok je prekidač uključen, izlaz PD0 drži uključen tranzistor koji paralelno sa prekidačem održava uključen MOSFET. Isključenje ON/OFF prekidača generiše prekidnu rutinu u kojoj se bitni podaci pamte na EEPROM, nakon čega se isključuje tranzistor čime se isključuje i mikrokontroler.

U slučaju korisničke aplikacije, kod koje se svetleća dioda uključuje i isključuje pritiskom tastera, pri nestanku napajanja na Arduino UNO razvojnom sistemu i ponovnom dolasku napajanja trenutni status svetleće diode biće izgubljen. To znači da korisnička aplikacija nije u stanju da zapamti podešavanja koje je zadao korisnik kao što se vidi na slici 77.



Slika 77. Gubitak statusa svetleće diode pri nestanku napajanja

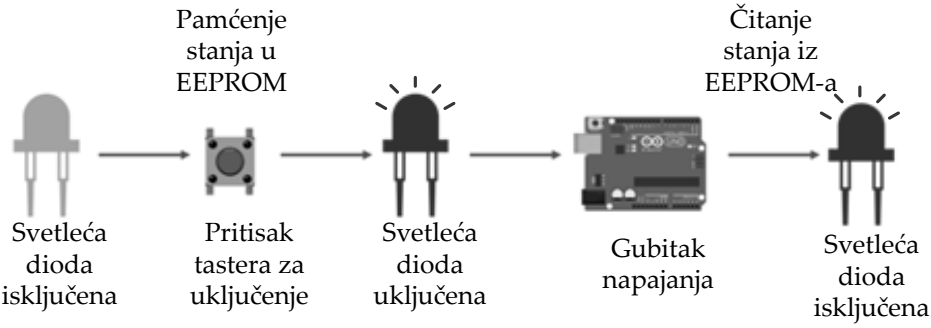
Zahvaljujući EEPROM memoriji pri svakoj promeni statusa svetleće diode ta se promena beleži na memorijsku lokaciju u EEPROMU na adresi 0, korišćenjem ugrađene funkcije *EEPROM.update(address, value)*. Na taj način svaka promena statusa biće zapamćena u EEPROM memoriji. Prilikom restarta mikrokontrolera u fazi inicijalizacije u *setup*() funkciji pročitaće se prethodno sačuvana promena u EEPROM memoriji pomoću ugrađene funkcije *EEPROM.read(address)* i vrednost će biti dodeljena trenutnoj vrednosti svetleće diode. Na taj način ugrađeni sistem uspeće da povrati prethodno stanje bez intervencije korisnika, što se vidi na slici 78.

```
#include <EEPROM.h>
const int buttonPin = 2; // taster
const int ledPin = 13; // LED dioda
int ledState; // promenljiva za stanje LED diode
int buttonState; // trenutno stanje tastera
int lastButtonState = LOW;// prethodno stanje tastera

void setup() {
 pinMode(buttonPin, INPUT);
 pinMode(ledPin, OUTPUT);
 digitalWrite(ledPin, ledState);
 checkLedState();
}

void loop() {
 int reading = digitalRead(buttonPin); // stanje tastera
 if(reading != buttonState) {
    buttonState = reading;
    if(buttonState == HIGH) {
      ledState = !ledState;
      }
 }
 digitalWrite(ledPin, ledState);
 EEPROM.update(0, ledState);
 lastButtonState = reading;
}
```

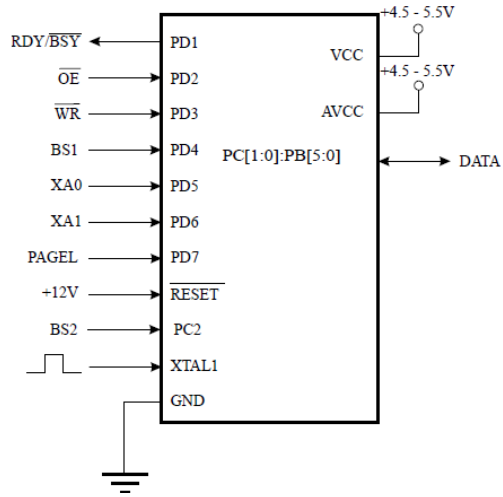
```
void checkLedState() {
 ledState = EEPROM.read(0);
 if(ledState == 1) {
    digitalWrite(ledPin, HIGH);
 }
 if(ledState == 0) {
   digitalWrite(ledPin, LOW);
 }
}
```



Slika 78. Čuvanje statusa svetleće diode i oporavak pri nestaku napajanja

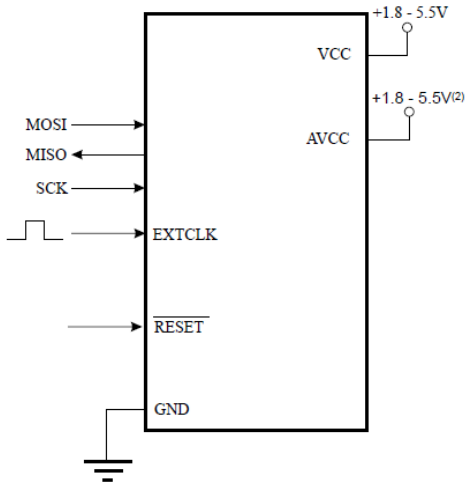
FLASH memorija

ATmega328p mikrokontroler sadrži 32KB reprogramabilne FLASH memorije za skladištenje programa koja ima izdržljivost od najmanje 10000 ciklusa pisanja/brisanja. S obzirom da su sve AVR instrukcije široke 16 ili 32 bita FLASH memorija sastoji se od 16k memorijskih reči širine 16 bita, tako da je programski brojač sa širinom adrese od 14 bita u stanju da adresira svih 16K memorijskih lokacija programa. Sadržaj FLASH memorije može se menjati eksternim programatorom u režimima paralelnog programiranja, serijskog programiranja ili samoprogramiranja preko pokretačkog programa. Paralelno programiranje FLASH memorije odvija se preko 8-bitnog interfejsa za podatke i korišćenjem niza upravljačkih linija koje se nalaze na portu D, prema slici 79. Ulazak u režim paralelnog programiranja zahteva dovođenje visokog napona od +12 V na reset pin, nakon čega se preko interfejsa za podatke šalju komande i podaci za upis, čitanje ili brisanje sadržaja FLASH i EEPROM memorije.



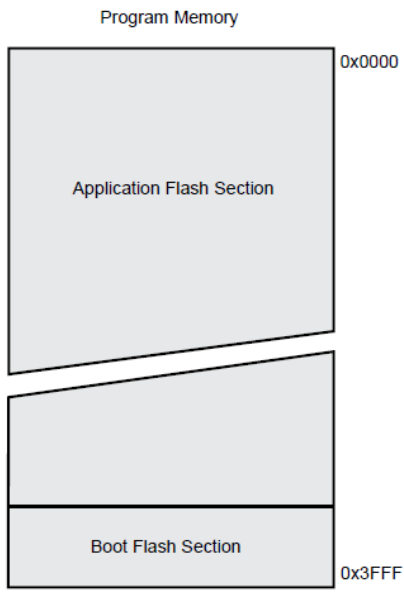
Slika 79. Paralelno programiranje FLASH memorije mikrokontrolera

Postupak serijskog programiranja FLASH memorije obavlja se preko SPI serijske magistrale, prikazane na slici 80. U slučaju Arduino UNO razvojnog sistema na razvojnom sistemu postoji 6-pinski ICSP (eng. *In Circuit Serial Programer*) konektor, preko kojeg je moguće programirati. Postupak programiranja započinje se povlačenjem reset linije na stanje logičke nule, nakon čega se preko SPI magistrale šalje naredba Programming Enable, koja omogućava režim serijskog programiranja. Korišćenjem režima serijskog programiranja moguće je čitanje, brisanje i upis sadržaja FLASH i EEPROM memorije, kao i čitanje i upis sistemskih bajtova (Lock, Fuse i Callibration).



Slika 80. Serijsko programiranje FLASH memorije mikrokontrolera

Radi bezbednosti softvera memorijski prostor FLASH programa podeljen je na dva odeljka: odeljak pokretačkog programa (eng. *Boot Loader*) i odeljak za aplikativni program u uređaju (slika 81). Pokretački program pruža mogućnost samoprogramiranja, odnosno čitanja i upisa programskog koda od strane samog mikrokontrolera. Kako bi pročitao podatke iz FLASH memorije mikrokontroler koristi LPM instrukcije, a za upis u programsku memoriju SPM instrukcije. Pokretački program nalazi se u posebnoj sekciji FLASH memorije i omogućava da se sadržaj celokupne FLASH memorije može čitati i menjati preko nekog od dostupnih interfejsa mikrokontrolera preko kojih se može prenosti programski kod. Na taj način pokretački program može modifikovati i samog sebe ili izbrisati iz FLASH memorije ukoliko dalje nije potreban.



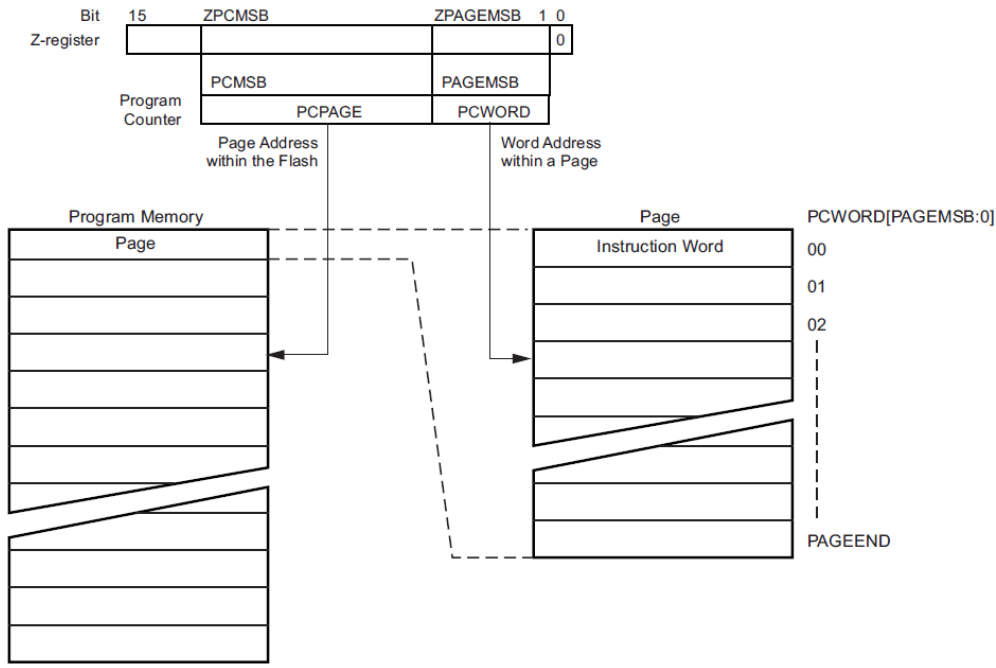
Slika 81. Organizacija FLASH memorije mikrokontrolera

Prava pristupa aplikativnom i pokretačkom programu u FLASH memoriji definišu se preko BLB0x i BLB1x bitova koji se postavljaju u procesu programiranja mikrokontrolera prema tabeli 27. Na taj način programer može odabrati odgovarajući nivo zaštite svoje ugrađene aplikacije zabranom iščitavanja i/ili promene programskog koda kako bi se ugrađeni sistem zaštitio od neovlašćenog pristupa. Podrazumevani režim podešavanja prava pristupa dozvoljava i upis i čitanje aplikativnog programa.

Tabela 27. Podešavanje prava pristupa aplikativnom programu

BLB02	**BLB01**	**Prava pristupa aplikativnom programu**
0	0	Zabranjen pristup SPM i LPM instrukcijama
0	1	Zabranjen pristup LPM instrukcijama
1	0	Zabranjen pristup SPM instrukcijama
1	1	Dozvoljen pristup SPM i LPM instrukcijama
BLB12	**BLB11**	**Prava pristupa pokretačkom programu**
0	0	Zabranjen pristup SPM i LPM instrukcijama
0	1	Zabranjen pristup LPM instrukcijama
1	0	Zabranjen pristup SPM instrukcijama
1	1	Dozvoljen pristup SPM i LPM instrukcijama

FLASH memorije zbog svoje organizacije ne dozvoljavaju upis na pojedinačnu memorijsku lokaciju, već je potrebno pristupiti grupi memorijskih reči koja se naziva stranica (eng. *page*). Za razliku od jednostavnog upisa u memorijsku lokaciju FLASH memorija zahteva da se stanica najpre obriše kako bi se u nju mogao upisati novi podatak. FLASH memorija ATmega328p mikrokontrolera podeljena je u 256 stranica koje se sastoje od 64 reči (širine 16 bita), čija je organizacija prikazana na slici 82.



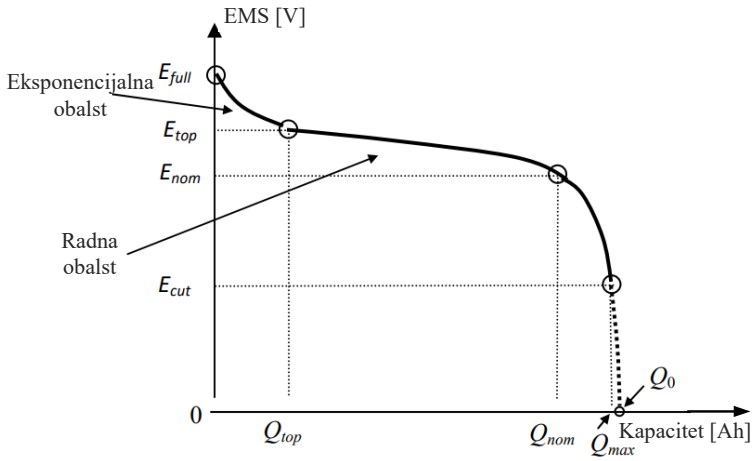
Slika 82. Stranična organizacija FLASH memorije mikrokontrolera

Prilikom upisa jedne reči potrebno je prvo pročitati prethodni sadržaj stranice u privremeni bafer, izbrisati stranicu, modifikovati sadržaj stranice u baferu i upisati novu stranicu. Za pristup memoriji koristi se Z pokazivač u koji se upisuje 14-bitna adresa reči u FLASH memoriju koja se deli na 8-bitnu adresu stanice PCPAGE i 6-bitnu adresu reči unutar stanice PCWORD. Pošto je FLASH memorija adresibilna na nivou bajta, LSB bit ne koristi se prilikom operacija upisa.

AVR arhitektura dozvoljava da se konstantama koje se čuvaju u programskoj memoriji može pristupati preko Z pokazivača korišćenjem instrukcije Load Program Memory(LPM). Na taj se način štedi memorija za podatke u slučaju da je npr. potrebno štampati konstantne strignove na serijskom izlazu ili LCD ekranu. Arduino IDE smeštaće stringove u programsku memoriju kada se konstantni string referencira sa *F(„Tekst stringa”)*.

POGLAVLJE 11: REŽIMI NISKE POTROŠNJE

Mikrokontrolerski sistemi kao i svi računari i elektronski uređaji za svoj rad zahtevaju izvor električne energije i mogu se napajati iz distributivne mreže ili iz baterija. Napajanje iz distributivne mreže poseduje niz nedostataka, koji ograničavaju dimenzije i težinu, ograničavaju mobilnost i mogu predstavljati potencijalnu opasnost za korisnika usled prisustva visokog napona u uređaju. Baterije predstavljaju elektrohemijski izvor energije koji prevazilazi mnoga ograničenja koja postoje u slučaju napajanja uređaja iz distributivne mreže. Baterije poseduju niz ograničenja kao što su nestabilan napon napajanja, ograničen kapacitet i samopražnjenje baterije. Napon baterije zavisi od tipa baterije i tokom životnog veka baterije se smanjuje prema dijagramu pražnjenja sa slike 83. Kapacitet baterija najčešće se izražava u jedinici Ah (amper-čas), koja prestavlja vreme tokom kojeg baterija može isporučiti nominalnu struju uređaju.



Slika 83. Dijagram pražnjenja baterije

U slučaju primarnih (nepunjivih) baterija standardni napon napajanja iznosi 1.5 V za cink-karbon baterije i 3 V za litijumske baterije. Napon baterije vremenom opada, pri čemu se mogu uočiti nekoliko karakterističnih oblasti rada. Na samom početku napon baterije u kratkom vremenu eksponencijalno opada dok ne dođe do vršnog kapaciteta Q_{top}, pri kojem je napon baterije E_{top}. Nakon toga baterija dolazi u nominalnu oblast u kojoj je pad napona konstantan tokom dužeg vremenskog perioda, dok napon baterije ne opadne do nominalnog napona E_{nom} do kojeg baterija mora obezbediti nominalni kapacitet Q_{nom}. Poslednja oblast nastupa kada baterija u potpunosti potroši nominalni kapacitet i napon baterije počinje znatno brže da opada. Kako bi se obezbedio pouzdan rad uređaja, čim napon opadne ispod minimalnog napona E_{cut} uređaj prestaće da radi i neophodno je izvršiti zamenu baterije. Sekundarne (punjive) baterije poseduju mogućnost dopunjavanja, pri čemu zahtevaju kontroler punjenja koji sprečava prepunjavanje baterija, koje može izazvati oštećenje baterije i uređaja. Nominalni napon punjivih baterija iznosi 1.2 V za nikl-metal hidrid (NiMh) baterije, 2 V za olovne baterije i 3.6 V za litijum-jonske baterije. Kako bi se napon baterije prilagodio naponu uređaja neophodno je povezati baterije na red ili koristiti DC/DC konvertore kako bi se napon baterije prilagodio naponu napajanja uređaja.

Ključna prednost mikrokontrolera u odnosu na klasične računare jeste niska potrošnja energije koja im omogućava dugotrajan rad sa jednom baterijom. Elektronski uređaji koji se napajaju direktno iz distributivne mreže obično ne zahtevaju korišćenje režima niske potrošnje, mada je to preporuka kako bi se povećala energetska efikasnost uređaja.

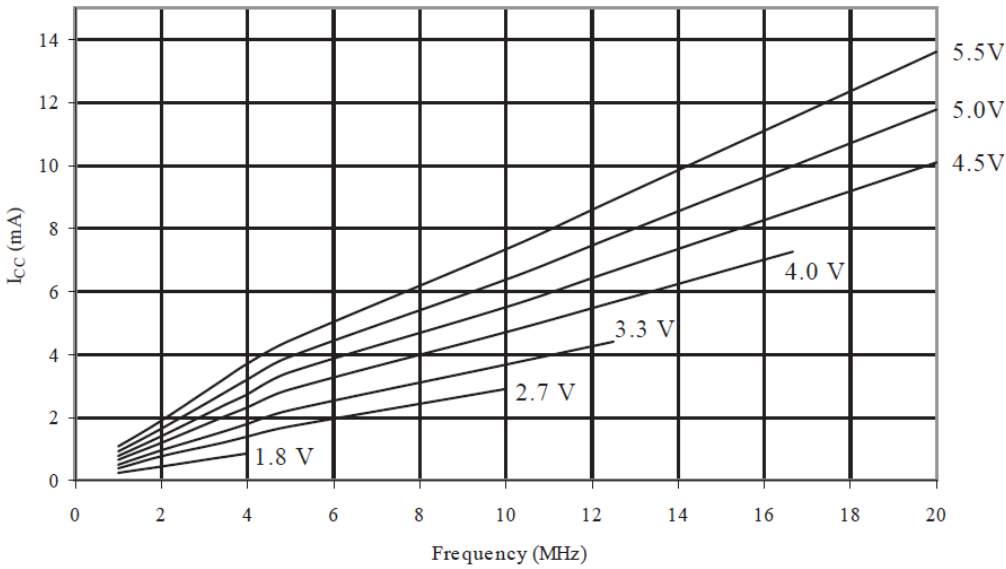
Potrošnja energije u integrisanim kolima može se podeliti na statičku i dinamičku potrošnju. Statička potrošnja potiče usled struja curenja (eng. *leakage current*) koje u integrisanom kolu teku između linija za napajanje i mase jer silicijum koji se nalazi između njih nije izolator već poluprovodnik. Statička potrošnja kod mikrokontrolera relativno je mala i zavisna je od napona napajanja. Ona se može dodatno smanjiti isključivanjem napajanja pojedinim periferijama mikrokontrolera koji se ne koriste. Kako bi se smanjila statička potrošnja potrebno je izvršiti analizu svih periferija koje se ne koriste u aplikaciji i njih isključiti korišćenjem odgovarajućih bitova.

Neke od periferija koje treba isključiti jesu analogni komparator, A/D konvertor, interna naponska referenca, BOR detektor, sigurnosni brojač i debugWire. Statička potrošnja ATmega328p mikrokontrolera sa svim isključenim periferijama ispod je 1 μA pri naponu napajanja od 3 V.

Dinamička potrošnja P_{dyn} javlja se prilikom uključivanja i isključivanja tranzistora, bazne gradivne komponente integisanih kola, data je sledećom formulom:

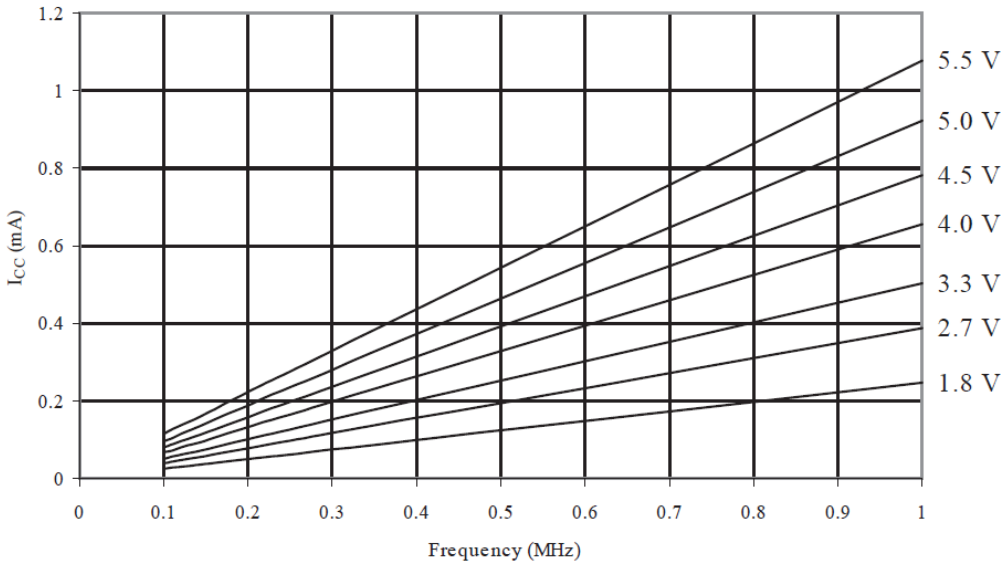
$$\mathrm{P_{dyn}} \sim N \cdot C \cdot U^2 \cdot f$$

gde je N broj tranzistora u kolu, C kapacitivnost, U napon napajanja i f frekvencija. Dinamička potrošnja direktno je proporcionalna radnoj frekvenciji i kvadratu napona napajanja i ona se može značajno redukovati smanjivanjem radne frekvencije i napona napajanja. Tipična potrošnja energije najčešće se izražava preko struje napajanja i ona je prikazana na slici 84. za različite napone napajanja i radne frekvencije oscilatora.



Slika 84. Struja napajanja u aktivnom režimu na visokim frekvencijama

Aktivna struja mikrokontrolera značajno opada sa naponom napajanja i radnom frekvencijom. Aktivna struja mikrokontrolera dosta je niža pri radnim taktovima ispod 1 MHz, tako da je u režimu rada sa internim RC oscilatorom radne frekvencije od 128 kHz aktivna struja oko 60 μA pri naponu napajana od 3 V (slika 85).



Slika 85. Struja napajanja u aktivnom režimu na niskim frekvencijama

U aktivnom režimu moguće je smanjiti potrošnju isključenjem takta pojedinim periferijama koje se ne koriste preko bitova u PRR (eng. *Power Reduction Register*) registru. Korišćenjem odgovarajućih bitova ovog registra može se pojedinačno isključiti takt za binarne brojače TC0, TC1 i TC2, USART, SPI i TWI serijske interfejse kao i za A/D konvertor, čime se mogu postići dodatne uštede energije.

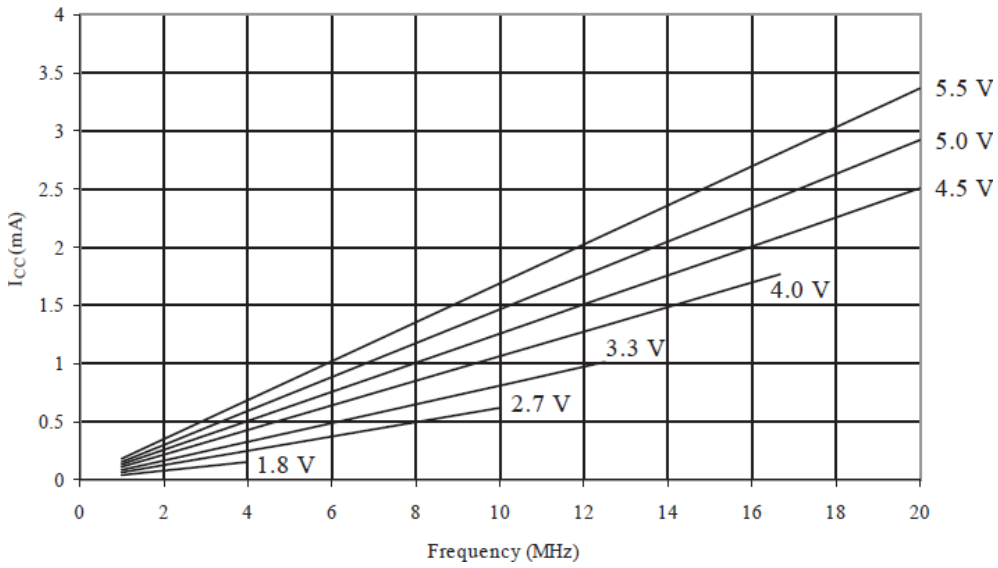
Cilj korišćenja režima niske potrošnje jeste smanjenje potrošnje energije uređaja kontrolisanjem njegovog ponašanja kako bi se produžio životni vek baterije. Režim niske potrošnje podešava se u SMCR (eng. *Sleep Mode Control Register*) registru pomoću SM[2:0] bitova, prema tabeli 28.

Tabela 28. Izbor režima niske potrošnje

SM[2:0]	**Režim niske potrošnje**
000	Neaktivan
001	Redukcija šuma A/D konvertora
010	Power down
011	Power save
100	Rezervisano
101	Rezervisano
110	Standby
111	Extended standby

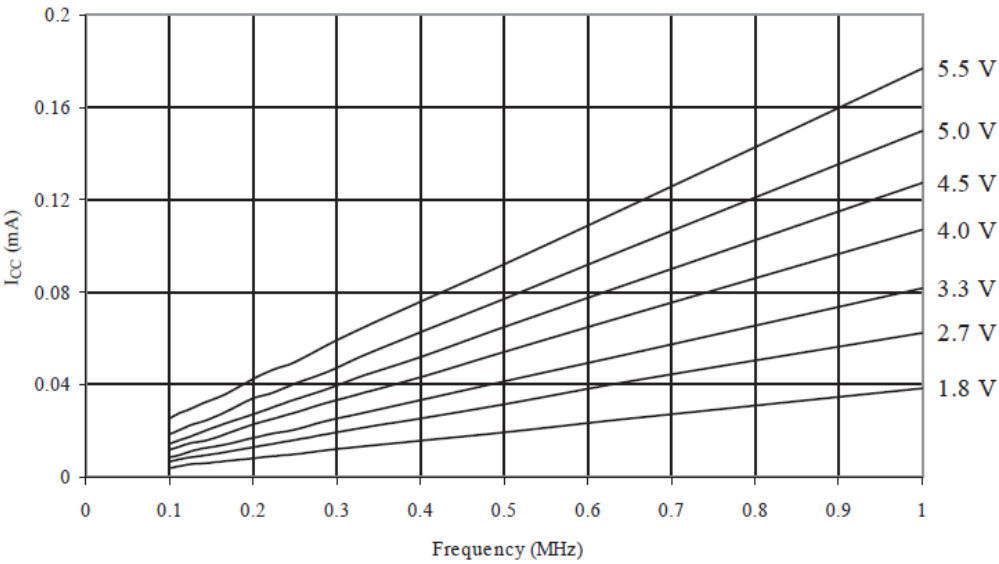
ATmega328p mikrokontroler omogućava korišćenje nekoliko režima niske potrošnje, u koje mikrokontroler ulazi pozivanjem SLEEP instrukcije pod uslovom da je setovan SE (eng. *Sleep Enable*) bit.

U neaktivnom (eng. *Idle*) režimu rada pozivom SLEEP instrukcije zaustavlja se procesorsko jezgro, pri čemu sve ostale periferije ostaju aktivne i taktovane od strane sistemskog oscilatora. Ovaj režim rada pogodan je kada je potrebno da procesorsko jezgro sačeka završetak neke akcije od strane periferije. Mikrokontrolersko jezgro biće probuđeno iz režima niske potrošnje bilo kojim događajem koji je u stanju da generiše zahtev za prekid. U tom slučaju mikrokontroleru je potrebno određeno vreme za pokretanje oscilatora i izvršavanje prekidne rutine, pre nego što se nastavi sa izvršavanjem instrukcije koja se nalazi neposredno posle SLEEP instrukcije. Prilikom ulaska i izlaska iz režima sna sadržaj registara i SRAM memorije ostaje nepromenjen. Potrošnja energije u neaktivnom režimu višestruko je niža nego u aktivnom, tako je da u režimu rada sa internim RC oscilatorom od 128 kHz neaktivna struja oko 15 μA pri naponu napajana od 3 V (slike 86 i 87).



Slika 86. Struja napajanja u neaktivnom režimu na visokim frekvencijama

Režim niske potrošnje sa redukcijom šuma A/D konvertora, pored procesorskog jezgra, isključuje i takt celokupnog I/O sistema kako bi smanjio nivo šuma koji može uticati na tačnost A/D konvertora. Ulaskom u ovaj režim niske potrošnje automatski se pokreće A/D konverzija.



Slika 87. Struja napajanja u neaktivnom režimu na niskim frekvencijama

Režim niske potrošnje pod nazivom duboki san (eng. *Power down*) jeste najefikasniji režim niske potrošnje u kojem se zaustavljaju svi oscilatori i jedini način da se mikrokontroler vrati u aktivni režim jeste eksterni prekid, detekcija TWI adrese ili istek intervala sigurnosnog brojača. U ovom je režimu potrošnja struje sa uključenim sigurnosnim brojačem 4.2 μA pri naponu napajanja od 3 V a svega 0.1 μA, ako je sigurnosni brojač isključen. Režim niske potrošnje sa uštedom energije (eng. *Power save*) sličan je režimu dubokog sna, pri čemu je jedina razlika ta što je u ovom režimu aktivan niskofrekventni oscilator TC2 modula. Poslednja dva režima niske potrošnje (eng. *Stanby*) i (eng. *Extended Standby*) zahtevaju postojanje eksternog oscilatora i po osobinama su slični režimima dubokog sna i uštede energije.

Režimi niske potrošnje mogu se koristiti preko *LowPower* biblioteke koja poseduje ugrađene funkcije za sve podržane režime rada. Mikrokontroler ulazi u režim dubokog sna pozivom funkcije *powerDown(SLEEP_1S, ADC_OFF, BOD_OFF)*. Ova funkcija kao parametre specificira podešavanje

sigurnosnog brojača kako bi se mikrokontroler probudio posle isteka njegovog perioda, kao i makronaredbe za isključenje A/D konvertora i BOD detektora.

Izvorni kod *powerDown* funkcije prikazan je u sledećem listingu:

```
void  LowPowerClass::powerDown(period_t period, adc_t adc,
bod_t bod){
      if (adc == ADC_OFF)      ADCSRA &= ~(1 << ADEN);

      if (period != SLEEP_FOREVER){
            wdt_enable(period);
            WDTCSR |= (1 << WDIE);
      }
      if (bod == BOD_OFF)
            lowPowerBodOff(SLEEP_MODE_PWR_DOWN);
      else
            lowPowerBodOn(SLEEP_MODE_PWR_DOWN);

   }
```

Pozivom ove metode najpre se proverava da li je potrebno isključiti A/D konvertor resetovanjem ADEN bita u ADCSRA registru. Potom se pristupa podešavanju perioda za sigurnosni brojač i njegovom aktiviranju, da bi se na kraju pozvala odgovarajuća funkcija u zavisnosti od željenog statusa BOD detektora koja odvodi mikrokontroler u režim niske potrošnje pozivom SLEEP instrukcije.

BIBLIOGRAFIJA

1. Jason Lang, *Projektovanje ugrađenih sistema*, ISBN: 978-86-80134-23-9, Agencija EHO, 2019.
2. David Russell, *Introduction to Embedded Systems Using ANSI C and the Arduino Development Environment*, ISSN: 1932-3166, Morgan & Claypool Publishers, 2010.
3. Muhammad Ali Mazidi, Sepehr Naimi, Sarmad Naimi, AVR Microcontroller and Embedded Systems: Using Assembly and C, *Pearson New International Edition*, ISBN: 978-0138003319, Pearson Custom Electronics Technology, 2011.
4. James M. Fiore, *Embedded Controllers Using C and Arduino*, ISBN13: 978-1796854879, 2021.
5. Tianhong Pan, Yi Zhu, *Designing Embedded Systems with Arduino A Fundamental Technology for Makers*, ISBN: 978-981-10-4417-5, DOI 10.1007/978-981-10-4418-2, Springer, 2018.
6. Smith Warwick, *C programiranje za Arduino*, ISBN: 9788680134086, Agencija Eho, 2017.
7. Smith Warwick, *Ultimate Arduino Uno Hardware Manual*, ISBN: 978-3-89576-434-9, Elektor, 2021
8. Frank O'Brien, *The Apollo Guidance Computer: Architecture and Operation*, ISBN: 978-1441908766, Springer Praxis Books, 2010.
9. Jack Ganssle, Tammy Noergaard, Fred Eady, Creed Huddleston, Lewin Edwards, David J. Katz, Rick Gentile, Ken Arnold, Kamal Hyder, and Bob Perrin, *Embedded Hardware: Know It All*, ISBN: 978-0-7506-8584-9, Elsevier, 2008.

10. Jean Labrosse, Jack Ganssle, Tammy Noergaard, Robert Oshana, Colin Walls, Keith Curtis, Jason Andrews, David J. Katz, Rick Gentile, Kamal Hyder, and Bob Perrin, *Embedded Software: Know It All*, ISBN: 978-0-7506-8583-2.

11. M. Verle, *PIC mikrokontroleri*, ISBN: 978-86-84417-14-7, Beograd: MikroElektronika, 2008.

12. Alexander Dean, *Embedded Systems Fundamentals with Arm Cortex-M based Microcontrollers: A Practical Approach*, ISBN: 978-1-911531-03-6ARM, Education Media UK, 2017.

13. *Arduino programsko okruženje*, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, <https://www.elektronika.ftn.uns.ac.rs/elektronika-e2/wp-content/uploads/sites/87/2018/03/Arduino-uputstvo.pdf>, 23. 1. 2023.

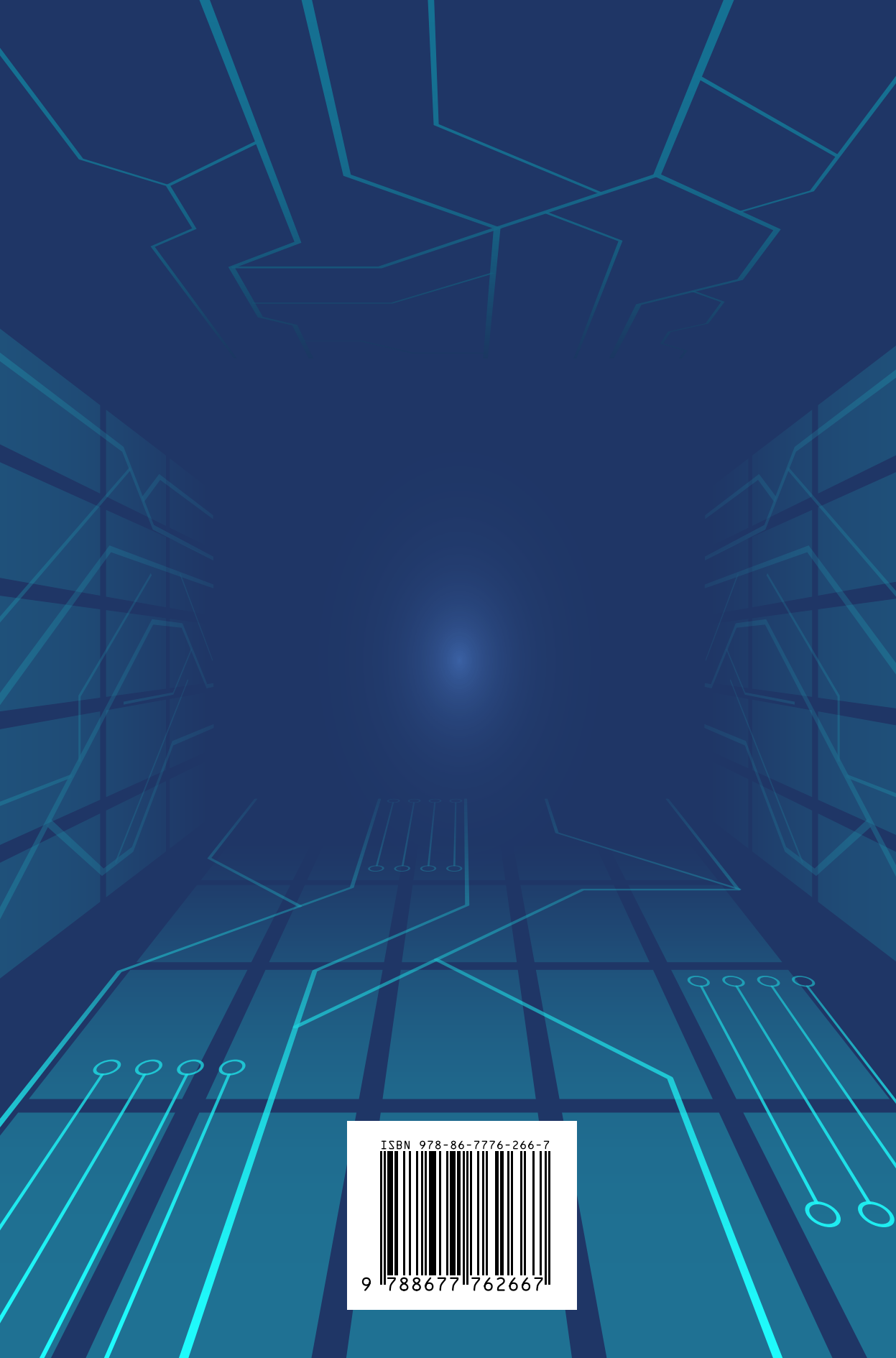
14. Atmel Corporation, *ATmega328P Datasheet: 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash*, 2015, <https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf>, 23. 1. 2023.

15. Microchip Technolgy, *ATmega328P Automotive Silicon Errata and Data Sheet Clarification*, 2022, <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/Errata/ATmega328P-Auto-SilConErrataClarif-DS80001060.pdf>, 23. 1. 2023.

16. *Arduino UNO R3 Product Reference Manual SKU*: A000066, Arduino 2023,<https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>, 23. 1. 2023.

17. *The Official Arduino AVR core – GitHub*, <https://github.com/arduino/ArduinoCore-avr>, 23. 1. 2023.

18. Brian W. Evans, *Arduino Programming Notebook*, 2007, <http://engineering.nyu.edu/gk12/ampscbri/pdf/ArduinoBooks/Arduino%20Programming%20Notebook.pdf>, 23. 1. 2023.


ISBN 978-86-7776-266-7
9 788677 762667