Upravljanje energetskim stanjima uređaja

Arduino

Branimir Škrlec

Zagreb, lipanj 2017.
Sadržaj

Uvod .......................................................................................................................... 5

1. O uređaju Arduino ................................................................................................. 6
   1.1 O Arduino platformi .......................................................................................... 6
   1.2 Razvojne ploče ................................................................................................. 7
      1.2.1 Općenito o razvojnim pločama ................................................................. 7
      1.2.2 Mikrokontroler .......................................................................................... 8
      1.2.3 Digitalni pinovi ....................................................................................... 8
      1.2.4 Analogni pinovi ...................................................................................... 9
   1.3 Arduino Uno R3 ............................................................................................... 10
   1.4 Programiranje Arduino uređaja ..................................................................... 11

2. RTC modul ............................................................................................................ 12
   2.1 DS3231 RTC .................................................................................................. 12
   2.2 Shema spajanja DS3231 i Arduino Uno R3 .............................................. 13
      2.2.1 Prvo pokretanje DS3231 ....................................................................... 14
      2.2.2 Alarmi ...................................................................................................... 15

3. Energetska stanja uređaja Arduino .................................................................... 17
   3.1 Stanje Idle ........................................................................................................ 17
   3.2 Stanje ADC Noise Reduction ........................................................................ 18
   3.3 Stanje Power-down ....................................................................................... 18
   3.4 Stanje Power-save ......................................................................................... 19
   3.5 Stanje Standby .............................................................................................. 19
   3.6 Ostale mogućnosti ......................................................................................... 19

4. Implementacija na uređaju Arduino .................................................................. 21
   4.1 Korištene knjižnice i pomoćne funkcije ..................................................... 21
4.2 Usporedba stanja mirovanja................................................................. 25
5. Zaključak ....................................................................................... 30
Literatura ......................................................................................... 31
Skraćenice ....................................................................................... 34
Uvod

U današnje vrijeme tehnologija je dostupna svakome. Svakog dana u svijetu nastaje sve više mobilnih uređaja, računala, satova i mnogih drugih uređaja. Svi oni za svoj rad koriste neki oblik energije, bilo električnu energiju ili neki drugi oblik energije. Zbog velikog broja takvih uređaja raste potreba za smanjenjem potrošnje energije uređaja koji se ne nalaze u aktivnom stanju npr. zaslon mobilnog uređaja ne mora biti uključen ukoliko trenutno nitko ne koristi taj mobilni uređaj i slično.

Potrošnja električne energije je danas jedno od najvažnijih pitanja. Svijet zahtjeva sve više i više električne energije pa se tako predviđa skoro dvostruki porast potrošnje do 2035. godine.

Postoje brojni načini na koje je moguće smanjiti potrošnju energije, no vjerojatno najefikasniji i najkorišteniji je isključivanje funkcionalnosti koje nam nisu potrebne. Na taj način mogu se ostvariti značajne uštede energije. Jedan takav slučaj će biti obrađen i u ovom radu na primjeru uređaja Arduino.

Uređaj Arduino je mikrokontroler koji je u zadnjih nekoliko godina pronašao svoju primjenu u raznim područjima ljudskog života, ponajviše u Internetu Stvari (engl. Internet of Things). Baš zbog svoje prilagodljivosti, proširljivosti i programirljivosti Arduino se počeo koristiti u razne svrhe, od pametnih kuća, daljinskih upravljača, GPS lokatora do mnogih drugih funkcija. Arduinov hardver je daleko slabiji od hardvera prosječnog osobnog računala, pa mu je za rad potrebna i manja količina energije. Kako je Arduino često korišten u projektima koji nemaju direktn pristup električnoj energiji gradske mreže, on može raditi i na bateriji. Zbog toga je važno posvetiti pažnju optimizaciji i minimalizaciji kod projektiranja sustava, kako bi se smanjila potrošnja električne energije kao i zahtjevi na hardver. Jedan od načina na koji nam sam Arduino omogućuje uštedu energije su njegova stanja mirovanja (engl. Sleep Modes) koja će biti detaljnije opisana i objašnjena u nastavku ovog rada.
1. O uređaju Arduino

1.1 O Arduino platformi

Arduino je platforma otvorenog koda (engl. open-source) namijenjena za modeliranje i izgradnju elektroničkih uređaja, koja obuhvaća hardver, koji sačinjavaju same razvojne ploče s mikrokontrolerima (prikazano na slici Slika 1) i dodaci koji se nazivaju štitovi (engl. shields) koji proširuju skup mogućnosti razvojnih ploča, i softver koji uključuje programsko okruženje pod nazivom Arduino IDE te razne programske knjižnice koje olakšavaju pisanje programskog koda u jeziku koji je vrlo sličan programskim jezicima C i C++.

Projekt Arduino je nastao 2003. godine pod nazivom Wiring kojeg je u sklopu svog diplomskog rada razvio kolumbijski student Hernando Barragán, pod vodstvom Massima Banzi-a i Caseya Reas-a. Wiring je zamišljen kao program za studente IDII-a (Interaction Design Institute Ivrea) u Italiji čiji je cilj bio napraviti jeftin i jednostavan alat za razvoj raznih uređaja koji vrše neki oblik interakcije sa okolinom pomoću senzora (engl. sensors) i drugih pokretača (engl. actuators) kojeg bi na jednak način mogli koristiti i inženjeri i oni bez inženjerskih sposobnosti.


1.2 Razvojne ploče

1.2.1 Općenito o razvojnim pločama

Hardverski dio platforme Arduino predstavlja razvojna ploča. Razvojna ploča osim mikrokontrolera sadrži i druge sklopopove kao što su sklop za programiranje mikrokontrolera, sklop za napajanje, razne priključke i pinove i mnoge druge. Nekoliko Arduino razvojnih ploča je prikazano na slici (Slika 1).

Slika 1 Različite vrste Arduino razvojnih ploča (preuzeto sa [13])

Sklopopove od kojih se sastoji razvojna ploča možemo vidjeti na slici Slika 2 od kojih ćemo najvažnije kratko opisati u nastavku.

Slika 2 Sklopopi Arduino razvojne ploče (preuzeto sa [11])
1.2.2 Mikrokontroler

Mikrokontroleri (engl. MCU-Microcontroller Unit) jesu središnja komponenta svake Arduino razvojne ploče. Mikrokontroler je maleno računalo u obliku integriranog kruga (engl. Socket on Chip) koji posjeduju vlastitu memoriju, procesor, ulaze i izlaze. Ta računala su znatno slabija od klasičnih osobnih računala. Dok se kod klasičnih računala takt procesora kreće u redovima veličine nekoliko GHz, takt procesora mikrokontrolera se kreće od nekoliko MHz do nekoliko desetaka MHz. Količina memorije mikrokontrolera je također mnogo manja od količine memorije klasičnog računala te iznosi od nekoliko kB do nekoliko desetaka kB. Iz gore navedenih karakteristika proizlazi da će i potrošnja energije mikrokontrolera biti manja od one klasičnog računala što je jedan od razloga njihove široke upotrebe. Mikrokontrolere danas možemo pronaći gotovo posvuda oko nas, najčešće kao dio ugradbenih računalnih sustava, primjerice u automobilima, alarmnim sustavima, pametnim kućama, bijeloj tehnici, sustavima za navodnjavanje i slično. Mikrokontroler ATmega8 možemo vidjeti na slici (Slika 3).

![Slika 3 Mikrokontroler ATmega8 (preuzeto sa [12])]

1.2.3 Digitalni pinovi

Digitalni pinovi Arduino razvojne ploče se mogu koristiti i kao ulazni i kao izlazni. Kod digitalnih ulaza, program provjerava da li na određenom pinu postoji napon. Svaki pin
sadrži interni pull-up otpornik kojeg je moguće uključiti koristeći funkciju `digitalWrite()` kad je pin definiran kao ulazni. Maksimalna struja koja može prolaziti kroz pin iznosi 40 mA. Svaki Arduino ima 13 digitalnih pinova.

Pinovi 0 (RX) i 1 (TX) služe za primanje i odašiljanje TTL podataka.

Pinovi 2 i 3 su pinovi koji služe za detekciju vanjskih prekida, odnosno mogu se konfigurirati kako bi obradili prekid kad registriraju nisku ili visoku razinu napona ili pak promjenu u razini napona. Za obradu prekida moramo prekidu pridružiti prekidni potprogram pomoću funkcije `attachInterrupt()`.

Pinovi 4,5,6,9,10,11 omogućuju 8-bitni PWM (engl. Pulse Width Modulation – tehnika za dobivanje analognih rezultata pomoću digitalnih sredstava) izlaz pomoću funkcije `analogWrite()`.

Pinovi 10,11,12,13 podržavaju SPI (engl. Serial Peripheral Interface) komunikaciju koja, iako podržana hardverski, još nije uključena u Arduino jezik.

1.2.4 Analogni pinovi

Arduino razvojna ploča sadrži 5 analognih pinova. Svaki pin podržava 10-bitnu analogno-digitalnu pretvorbu koristeći funkciju `analogRead()`. Raspon dobivenih digitalnih vrijednosti je od 0-1023. Većina analognih pinova se također može koristiti kao digitalni pinovi.

Pinovi 4 (engl. SDA-Serial Data Line) i 5 (engl. SCL-Serial Clock Line) omogućuju I2C (engl. Inter-Integrated Circuit) komunikaciju koristeći Wire knjižnicu.

Arduino sadrži i dva pina za napajanje, jednog od 3.3 V i drugog od 5 V.
1.3 Arduino Uno R3

U sklopu ovog završnog rada, za ispitivanje energetskih stanja uređaja Arduino, korišten je Arduino Uno (Slika 4). Uno je najpoznatija i najraširenija Arduino ploča jer predstavlja idealan izbor kako za početnike, tako i za naprednije korisnike. Uno na talijanskom znači jedan i odabrano je kako bi označilo izdavanje verzije 1.0 Arduino IDE softvera.

Arduino Uno je razvojna ploča temeljena na ATmega328P mikrokontroleru. ATmega328P je 8-bitni RISC mikrokontroler sa 32 kB programske flash memorije, 2kB SRAM memorije i procesorom takta 16 MHz. Uno posjeduje 14 digitalnih ulazno/izlaznih pinova, od čega 6 može biti korišteno kao PWM izlaz, 6 analognih ulaznih pinova, 16 MHz integrirani kristal, USB priključak, utičnicu za napajanje, I2C priključke (SCL i SDA), ICSP priključak i gumb za resetiranje.

Uno je sam po sebi vrlo jednostavan za korištenje. Može se napajati pomoću USB porta ili putem DC priključka naponom od 6 V pa do 20 V, uz preporučeni napon između 7 V i 12 V.

*Slika 4 Arduino Uno R3 (preuzeto sa [14])*
1.4 Programiranje Arduino uređaja

Za programiranje Arduino uređaja potrebno je USB kabelom ploču povezati sa računalom te koristiti neko integrirano razvojno okruženje za pisanje, prevodjenje i prebacivanje koda na sam Arduino. Najpoznatije razvojno okruženje za Arduino programere je Arduino IDE.

Arduino IDE je prenosiva (engl. cross-platform) aplikacija pisana u programskom jeziku Java, koja podržava programske jezike C i C++. Samo sučelje je iznimno jednostavno, kao što je vidljivo na slici ispod (Slika 5).

Okruženje se sastoji od uređivača koda i nekoliko izbornika. Kod je moguće prenijeti na Arduino pomoću jednog klika. Pravila za pisanje koda su identična kao i za bilo koji drugi programski jezik. Okruženje omogućuje i instalaciju dodatnih programskih knjižnica putem ugrađenog pretraživača. U samom okruženju se nalaze brojni primjeri ispravno napisanih kodova koje je potrebno samo prenijeti na Arduino. Većina knjižnica također sadrži i primjere koji se mogu prenijeti na uređaj Arduino. Svaki Arduino program, odnosno skica, se sastoji od dvije glavne funkcije koje moraju biti definirane a to su:

- **setup()** – funkcija koja se izvršava jednom, pri pokretanju Arduina, u njoj se najčešće inicijaliziraju i definiraju razni parametri
- **loop()** – funkcija koja se neprestano poziva dok je god uređaj Arduino uključen, prvi poziv slijedi nakon izvršavanja funkcije **setup()**
2. RTC modul

RTC (engl. Real Time Clock) je elektronički uređaj koji vrlo precizno prati vrijeme. Prilikom izrade ovog završnog rada koristit ćemo DS3231 tvrtke Maxim Integrated.

2.1 DS3231 RTC

DS3231 (Slika 6) je jeftin a istovremeno vrlo precizan I²C RTC sa ugrađenim TCXO (engl. Temperature Compensated Crystal Oscillator) što mu daje veću stabilnost prilikom promjene temperature u rasponu od -40 °C do +85 °C, koju s običnim oscillatorom ne bi mogli dobiti. Sam RTC sadrži i utor za bateriju što mu omogućuje rad i kad je glavni dovod energije prekinut.

DS3231 prati sekunde, minute, sate, dan u tjednu, datum: dan u mjesecu, mjesec i godinu. Kada jednom postavimo vrijeme možemo pratiti kalendar sve do 2100. godine. Sat
radi u 24 i 12 satnom formatu, a ima ugrađena i dva programabilna alarma. Modul i kontroler povezani su preko dvosmjerne I²C komunikacije. Shemu dizajn DS3231 moguće je vidjeti na slici (Slika 7).

2.2 Shema spajanja DS3231 i Arduino Uno R3

DS3231 ima 6 pinova. Pinovi VCC i GND, koji služe za napajanje modula, se povezuju sa Arduinovim pinovima 5 V (ili 3.3 V) i GND. Pin SDA se povezuje sa pinom A4 na uređaju Arduino Uno, dok se pin SCL povezuje sa pinom A5. Pinovi SDA i SCL se koriste za I²C komunikaciju. Moguć je i alternativan način spajanja jer Arduino Uno R3 ima posebne SDA i SCL pinove namijenjene za I²C komunikaciju.Opisani način spajanja vidljiv je na slici (Slika 8).
2.2.1 Prvo pokretanje DS3231

Prilikom prvog spajanja RTC-a i uređaja Arduino potrebno je podesiti datum i vrijeme. Kod koji je korišten za podešavanje datuma i vremena prikazan je na slici (Isječak 1).

```cpp
#include <DS3231_Simple.h>

DS3231_Simple Clock;

void setup() {
    Serial.begin(9600);
    Clock.begin();
}

void loop() {
    DateTime MyTimestamp; // Create a variable to hold the data
    MyTimestamp.Day = 3;
    MyTimestamp.Month = 10;
    MyTimestamp.Year = 20;
    MyTimestamp.Hour = 14;
    MyTimestamp.Minute = 17;
    MyTimestamp.Second = 33;
    Clock.write(MyTimestamp); // Then write it to the clock
    // we will read it back for example...
    Serial.print("The time has been set to: ");
    Clock.printTo(Serial);
    Serial.println();
    Serial.println("End Of Program (RESET to run again)");
    while (1);
}
```

Isječak 1 Postavljanje vremena i datuma
U primjeru je korištena knjižnica DS3231_Simple koju je napisao James Sleeman i koja nam pruža osnovne funkcionalnosti DS3231 kao što su postavljanje i čitanje vremena, postavljanje alarma i bilježenje (engl. logging) podataka.

2.2.2 Alarmi

DS3231 nam omogućuje postavljanje 2 alarma, sa tom razlikom što je razlučivost prvog alarma u sekundama a drugog u minutama. Postoji nekoliko vrsta alarma ovisno o korištenoj programskoj knjižnici. Za DS3231_Simple moguće je podesiti alarm:

- svake sekunde (ALARM_EVERY_SECOND)
- kad trenutna vrijednost sekundi odgovora traženoj vrijednosti
  (ALARM_MATCH_SECOND)
- kad trenutna vrijednost u minutama i sekundama odgovara traženim
  (ALARM_MATCH_SECOND_MINUTE)
- kad trenutno vrijeme odgovara traženom vremenu
  (ALARM_MATCH_SECOND_MINUTE_HOUR)
- kad trenutno vrijeme i datum odgovaraju traženom
  (ALARM_MATCH_SECOND_MINUTE_HOUR_DATE)
- kad trenutno vrijeme i dan u tjednu odgovaraju traženom
  (ALARM_MATCH_SECOND_MINUTE_HOUR_DOW)

Isječak 2 prikazuje kod kojim postavljamo alarm.
```c
#include <DS3231_Simple.h>
DS3231_Simple Clock;
void setup() {
    Serial.begin(9600);
    Serial.println();
    Clock.begin();
    Clock.disableAlarms(); // First we will disable any existing alarms
    DateTime MyTimestamp = Clock.read(); // Get an initialized timestamp
    MyTimestamp.Second = 30; // We want the alarm at this second
    // And now add the alarm to happen when the second matches
    Clock.setAlarm(MyTimestamp, DS3231_Simple::ALARM_MATCH_SECOND);
    Clock.setAlarm(DS3231_Simple::ALARM_EVERY_MINUTE);
}
void loop() {
    uint8_t AlarmsFired = Clock.checkAlarms(); // To check the alarms we just ask the clock
    // check if either alarm is fired by using a "bitwise and"
    if (AlarmsFired & 1) {
        Clock.printTo(Serial); Serial.println("First alarm has fired!");
    }
    if (AlarmsFired & 2) {
        Clock.printTo(Serial); Serial.println("Second alarm has fired!");
    }
}
```

*Isječak 2 Postavljanje alarma*
3. Energetska stanja uređaja Arduino

Uređaju Arduino je za rad potrebna određena količina električne energije. Kako je Arduino u današnje vrijeme primarno namijenjen kao uređaj za projekte koji kao izvor energije koriste baterije raznih kapaciteta, jedan od najvećih problema je optimizacija potrošnje energije za što dulji rad.


Drugi način na koji je moguće postići značajne uštede energije jesu stanja mirovanja (engl. *sleep modes*). Za vrijeme kad Arduino ne obavlja nikakav koristan posao moguće je isključiti određene sklopove kako bi uređaj trošio manje energije. Ovisno o tome kakve uštede želimo i kakve funkcije treba obavljati naš uređaj moguće je koristiti jedan od nekoliko stanja mirovanja.

Kako bi omogućili mikrokontroleru korištenje stanja mirovanja potrebno je postaviti bit SE (*Sleep Enable*) u MCUCR (*MCU Control Register*) te u kodu pozvati funkciju za stavljanje uređaja u mirovanje. Ta funkcija postavlja bitove SM2, SM1 i SM0 u MCUCR ovisno o odabranom stanju mirovanja (Beard, 2017).

Stanja mirovanja koja su objašnjena i istražena prilikom stvaranja ovog rada su:

- Idle
- ADC Noise Reduction
- Power-save
- Standby
- Power-down

3.1 Stanje Idle

Kako bi mikrokontroler ušao u *idle* stanje mirovanja potrebno je bitove SM2, SM1 i SM0 postaviti na 000. *Idle* stanje zaustavlja procesor ali i dalje omogućuje korištenje SPI-
a, USART-a, analognog komparatora, ADC-a, I²C –a, timera/brojača, Watchdog-a i sustava prekida. Ovo stanje mirovanja u osnovi zaustavlja clkCPU i clkFLASH.

Mikrokontroler je u ovom stanju mirovanja moguće probuditi koristeći vanjske prekide, kao i unutrašnje prekide kao što su Timer Overflow i USART Transmit Complete. Dodatno, ukoliko nam nije potrebno buđenje izazvano od analognog komparatora, istog je moguće isključiti postavljajući bit ACD u ACSR-u (Analog Comparator Control and Status Register).

### 3.2 Stanje ADC Noise Reduction

Bitove SM2, SM1 i SM0 potrebno je postaviti na 001 kako bi mikrokontroler ušao u stanje ADC Noise Reduction prilikom poziva funkcije za mirovanje. U ovom stanju mirovanja MCU zaustavlja procesor dopuštajući ADC, vanjske prekide, I²C, timer/brojač2 i Watchdog. U osnovi, ovo stanje mirovanja zaustavlja clkI/O, clkCPU i clkFLASH dopuštajući ostalim satovima da rade.

U ovom stanju se poboljšava okolina buke za ADC, omogućujući mjerenja s većom razlučivosti. Ako je ADC omogućen, konverzija počinje automatski kada mikrokontroler prijede u ovo stanje rada. Osim ADC Conversion Complete prekida, isključivo vanjski reset, Watchdog reset, Brown-out reset, I²C Address Match prekid, prekid izazvan timerom/brojačem2, SPM prekid i vanjski prekid na pinovima D2 ili D3 mogu probuditi mikrokontroler.

### 3.3 Stanje Power-down

Bitove u registrima SM2, SM1 i SM0 je potrebno postaviti na 010 kako bi pozivom funkcije za mirovanje mikrokontroler prešao u stanje power-down. U ovom stanju, vanjski oscillator se zaustavlja, dok vanjski prekidi, I²C i Watchdog nastavljaju s radom, ukoliko su omogućeni. Samo vanjski reset, Watchdog reset, Brown-out reset, I²C Address Match prekid te vanjski prekid na pinovima D2 ili D3 mogu probuditi mikrokontroler. Ovo stanje mirovanja u osnovi zaustavlja sve generirane satove, dopuštajući rad samo asinkronih modula.
*Power-down* je stanje mirovanja u kojem se ostvaruju najveće uštede energije. Potrebno je imati na umu da se za buđenje mikrokontrolera prekidom koristi niska razina te ju je potrebno zadržati takvom neko vrijeme dok se mikrokontroler ne probudi.

Kod buđenja iz *power-down* stanja postoji kašnjenje od trenutka kad su zadovoljeni kriteriji za buđenje do trenutka kad je mikrokontroler ponovno potpuno uporabljiv. To omogućuje da se sat ponovno pokrene i postane stabilan nakon zaustavljanja.

### 3.4 Stanje Power-save

Ukoliko su bitovi SM2, SM1 i SM0 postavljeni na 011, pozivom funkcije mikrokontroler će prijeći u *power-save* stanje. Ovo stanje je gotovo identično *power-down* stanju uz jednu iznimku.

Ukoliko je timer/brojač2 praćen asinkrono, tj. postavljen je bit AS2 u ASSR, timer/brojač2 će raditi za vrijeme mirovanja. Uređaj je moguće probuditi uz Timer Overflow ili Output Compare ako su postavljeni odgovarajući bitovi u TIMSK-u (*Timer Interrupt Mask Register*) i ako je postavljen bit Global Interrupt Enable u SREG-u (*Status Register*).

Ovo stanje mirovanja u osnovi zaustavlja sve satove osim osim clkASY, dopuštajući rad samo asinkronih modula uključujući timer/brojač2 ukoliko je praćen asinkrono.

### 3.5 Stanje Standby

Za ulazak u ovo stanje je prije poziva funkcije potrebno postaviti bitove SM2, SM1 i SM0 na 110. Ovo stanje mirovanja je identično power-down stanju uz iznimku što oscilator ostaje uključen.

Iz *standby* stanja će mikrokontroleru biti potrebno 6 ciklusa da se probudi što je nešto brže u odnosu na power-down stanje.

### 3.6 Ostale mogućnosti

Osim gore navedenih načina za smanjenje potrošnje energije, dodatne uštede moguće je postići i na druge načine poput promjenom radne frekvencije procesora, što je nešto složeniji posao od ostalih, ili upravljanjem registrom za smanjenje snage - PRR-om (*Power Reduction Register*) što je značajno jednostavnije. PRR je 8 bitni registar pomoću kojeg
možemo pojedinačno odabrati sklope koji su nam potrebni za ostvarenje ciljane funkcionalnosti. Bitovi u PRR-u su redom:

- Bit 7 – PRTWI (Power Reduction Two Wire Interface)
- Bit 6 – PRTIM2 (Power Reduction Timer/Counter2)
- Bit 5 – PRTIM0 (Power Reduction Timer/Counter0)
- Bit 4 – rezerviran
- Bit 3 – PRTIM1 (Power Reduction Timer/Counter1)
- Bit 2 – PRSPI (Power Reduction Serial Peripheral Interface)
- Bit 1 – PRUSART0 (Power Reduction Universal Synchronous/Asynchronous Receiver/Transmitter)
- Bit 0 – PRADC (Power Reduction Analog to Digital Converter)

Za postavljanje pojedinih bitova u PRR postoje i definirane funkcije kako bi nam dodatno olakšale posao, kao što su `power_adc_disable()`, `power_spi_disable()`, `power_USART0_disable()`, `power_TIMER0_disable()`, `power_TIMER1_disable()`, `power_TWI_disable()`...

Potrebno je upamtiti da se PRR primjenjuje samo u aktivnom i idle stanju, jer su u ostalim stanjima ovi moduli ionako već isključeni, kao što je navedeno u [5].

Još jedan dodatan način uštede je isključivanje sklopa za utvrđivanje naglih promjene vrijednosti struje (engl. BOD - Brown-out Detection) kojim se postižu vrlo velike uštede energije, no istovremeno se narušava sigurnost našeg uređaja (Gammon, 2012).
4. Implementacija na uređaju Arduino

U ovom poglavlju će biti prikazani jednostavni primjeri buđenja uređaja Arduino iz stanja mirovanja u određenim vremenskim trenutcima koristeći vanjski sat DS3231, odnosno njegov alarm, te konkretne vrijednosti struje koju Arduino koristi za svoj rad. Kao referentna točka, kako bi mogli bolje odrediti uštedu energije, poslužit će nam količina energije koju Arduino koristi kad ne radi ništa (Isječak 3). Sva mjerenja su obavljena na otporniku od 5 Ω pomoću oscilokopa/voltmetra i računala kao što je prikazano na slici (Slika 9).

Slika 9 Obavljanje mjerenja

```
void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to run repeatedly:
}
```

Isječak 3 Najjednostavniji mogući program

Prilikom vrćenja gore navedenog koda, Arduino prolazi struja jakosti 0,6 mA.

4.1 Korištene knjižnice i pomoćne funkcije

Za ostvarivanje ovog rada korištene su AVR programske knjižnice koje nam pružaju podskup standardnih C knjižnica za ATMELOve mikrokontrolere, Wire programska
knjižnica koja omogućava I²C komunikaciju koju je napisao Nicholas Zambetti te već ranije spomenutu DS3231_Simple programsku knjižnicu autora James-a Sleeman-a (Boxall, 2014; Nedelkovski, 2016; Genovese, 2016).


```c
void sleepNow() {
  digitalWrite(wakePin, HIGH);
  pinMode(wakePin, INPUT_PULLUP);
  /*
   * There are 5 different sleep modes:
   * SLEEP_MODE_IDLE
   * SLEEP_MODE_ADC
   * SLEEP_MODE_PWR_SAVE
   * SLEEP_MODE_STANDBY
   * SLEEP_MODE_PWR_DOWN
   */
  set_sleep_mode(SLEEP_MODE_IDLE);
  noInterrupts();
  sleep_enable();
  // turn off brown-out enable in software
  //MCUCR = bit (BODS) | bit (BODSE); // turn on brown-out enable select
  //MCUCR = bit (BODS);  // this must be done within 4 clock cycles of above
  attachInterrupt(digitalPinToInterrupt(wakePin), wakeUp, LOW);
  interrupts();
  sleep_mode(); //here the device is actually put to sleep
}
```

Isječak 4 Funkcija mirovanja

Isječak 5 nam prikazuje dio koda koji se izvršava kad se pojavi prekid. Prvo što je potrebno napraviti u prekidnoj rutini je obrisati bit SE u kontrolnom registru mikrokontrolera, što se
ovdje obavlja naredbom `sleep_disable()`. Ono što je još potrebno napraviti u prekidnoj rutini jest odvojiti prekidnu rutinu od digitalnog pina 2 pomoću funkcije `detachInterrupt(pin)`. Razlog zbog kojeg je to potrebno jest taj što će, nakon što RTC generira signal niske razine u funkciji alarma, signal na pinu ostati u niskoj razini, a kako smo definirali da nam se prekidna rutina izvršava kad je na digitalnom pinu 2 signal niske razine te će se prekidna rutina stalno pozivati i izvršavati. Ostale linije u ovom kodu služe samo za dodatnu kontrolu.

```c
void wakeUp() {
    sleep_disable();
    detachInterrupt(digitalPinToInterrupt(wakePin));
    Serial.println("External interrupt detected!");
    clockInterrupt = true;
}
```

Isječak 5 Prekidna rutina

Isječak 6 nam prikazuje funkciju `setup()`. Funkcija `setup()` se pokreće samo jednom i to prilikom pokretanja uređaja Arduino. U danim primjerima funkcija `setup()` će imati ulogu inicijalizacije alarma. Prvo što je potrebno napraviti kako bismo postavili alarm je onemogućiti alarme koji su trenutno uključeni kako se neki od njih ne bi aktivirao prije no što postavimo željeno vrijeme alarma, ili ukoliko mi koristimo samo jedan alarm, a trenutno su aktivna oba. To se ostvaruje pomoću funkcije `disableAlarms()`. Ova funkcija ustvari ne onemogućuje alarome već ih samo postavlja na neki nedohvatljiv datum (moguće je spriječiti da alaromi generiraju signal niske razine na SQW pinu, no oni i dalje ostaju zapisani u registru te se stalno uspoređuju sa trenutnim vremenom). Iduća naredba u ovom kodu je naredba `read()` koja saznaje trenutno vrijeme RTC-a. Sam alarm postavljamo naredbom `setAlarm(vrijeme, tip alarma)` koja postavlja kontrolne bitove i vrijeme alarma u RTC-u.

```c
void setup() {
    Serial.begin(9600);
    Serial.println();
    Clock.begin();

    //digitalWrite(wakePin, HIGH);
    //pinMode(wakePin, INPUT_PULLUP);

    Clock.disableAlarms();

    DateTime myTime = Clock.read();

    myTime.Second = 0; //time in seconds
    myTime.Minute = 57; //time in minutes
    myTime.Hour = 16;  //time in hours

    Clock.setAlarm(myTime, DS3231_Simple::ALARM_MATCH_SECOND);
}
```

Isječak 6 Funkcija setup()
Isječak 7 prikazuje funkciju loop() koja se prvi put izvodi nakon funkcije setup() a zatim se ponavlja. U funkciji loop() izvodimo sve radnje za koje želimo da se ponavljaju. U ovom primjeru u funkciji loop() pozivamo pomoćnu metodu sleepNow() te provjeravamo koji se alarm aktivirao pomoću funkcije checkAlarms() za bolju kontrolu izvršenja programa. Dio koda sa donje slike koji je zakomentiran služi za ostvarivanje dodatnih ušteda energije korištenjem registra za uštedu energije (power_spi_disable(), power_timer0_disable(), power_timer1_disable(), power_timer2_disable(), power_twi_disable()...) i isključivanjem analogno-digitalnog pretvornika (power_adc_disable()). Nakon što se uređaj vrati iz stanja mirovanja u normalan način rada potrebno je ponovno omogućiti sve onemogućene sklopove, što je ovdje ostvareno pomoću funkcije power_all_enable(), i obnoviti statusni registar analogno-digitalnog pretvornika.

```c
void loop() {
    noInterrupts();

    byte old_ADCSRA = ADCSRA;//saving content of register
    // disable ADC
    ADCSRA = 0;
    power_adc_disable();
    //disable all possible interrupt sources so it doesn't wake up before alarm rings
    power_spi_disable();
power_timer0_disable();
power_timer1_disable();
power_timer2_disable();
power_twi_disable();

    UCSR0B &= ~bit(RXEN0); // disable receiver
    UCSR0B &= ~bit(TXEN0); // disable transmitter

    interrupts();
    sleepNow();
    //enable all circuitry
    power_all_enable();

    ADCSRA = old_ADCSRA;//restore register content

    UCSR0B |= bit(RXEN0); // enable receiver
    UCSR0B |= bit(TXEN0); // enable transmitter

    uint8_t AlarmsFired = Clock.checkAlarms();
    if (AlarmsFired & 1) {
        Clock.printTo(Serial); Serial.println(": First alarm has fired!");
    }
    if (clockInterrupt) {
        //code when interrupt happens
        Serial.println();
        Serial.println("We passed through interrupt routine!");
        clockInterrupt = false;
    }
}
```

Isječak 7 Funkcija loop()
4.2 Usporedba stanja mirovanja

Na idućih nekoliko slika prikazani su rezultati mjerenja napona na otporniku od 5Ω spojenom u seriju sa Arduinom. Kako je otpor od 5 Ω relativno malen u usporedbi sa unutarnjim otporima u Arduino razvojnoj ploči, na njemu se dešava i relativno malen pad napona. No iako je pad napona malen, i dalje je moguće vidjeti kako koje stanje mirovanja utječe na potrošnju energije.

Slika 10 prikazuje pad napona na otporu prilikom vrćenja praznog programa. Vidljivo je da i u slučaju kad nismo u stanju mirovanja pad napona nije velik (oko 3 mV) jer je otpor otpornika malen prema otporu cijelog kruga.

Slika 11 prikazuje pad napona na otporu kad se Arduino nalazi u idle stanju mirovanja. U ovom stanju nije došlo do velikog smanjenja potrošnje, pad napona je nešto manji nego u prethodnom slučaju, no i dalje je oko 3 mV. Kao što je i ranije navedeno, idle stanje nam ne pruža neke iznimne uštede te sigurno nije najbolja opcija ukoliko želimo smanjiti potrošnju.

Slika 12 prikazuje pad napona na otporniku od 5 Ω u ADC noise reduction stanju mirovanja. U ovom stanju je pad napona već nešto značajniji u odnosu na idle stanje te ovdje iznosi oko 2 mV. Vidljivo je da je u ovom stanju moguće ostvariti veće uštede nego u prethodnom primjeru, no i dalje postoje bolje opcije.

Na idućoj slici (Slika 13) prikazan je pad napona kad se Arduino nalazi u stanju standby. Pad napona je ovdje još manji te iznosi nešto manje od 2mV (oko 1,8 mV). U odnosu na pad napona u početnoj situaciji, kad vrtimo prazan program, ovdje se on smanjio za oko 1,2 mV, odnosno on iznosi 60% početne vrijednosti. Ovo stanje je već sasvim dobra opcija za uštedu energije, no postoje još bolje.
Slika 10 Potrošnja praznog programa

Slika 11 Idle stanje
Slika 12 ADC Noise Reduction stanje

Slika 13 Standby stanje
Slika 14 se odnosi na pad napona na otporniku kad se Arduino nalazi u power-save stanju. Na ovoj slici vrijednost napona na otporniku iznosi približno 1,8 mV što je gotovo identično standby stanju (razlika zasigurno postoji ali je nije moguće detektirati na ovako malenim vrijednostima i s relativno velikom razlučivošću).

Slika 15 prikazuje pad napona na otporniku kad se Arduino nalazi u power-down stanju. Ovo je stanje u kojem se postižu najznačajnije uštede energije jer je i najmanje sklopova aktivno. U našem primjeru pad napona na otpor iznosi oko 1 mV što iznosi 33,3% početne vrijednosti. Postigli smo smanjenja napona na otporniku za 2 mV, te iako se ono u ovom primjeru čini relativno maleno, u gotovo svim realnim situacijama uštede bi bile iznimno velike.
Tablica 1 prikazuje iznos struje koja prolazi otpornikom u svim stanjima mirovanja. Pošto je otpor u svim slučajevima isti, jakosti struja se odnose jednako kao i naponi.

**Tablica 1 Jakost struje kroz otpornik**

<table>
<thead>
<tr>
<th>Stanje</th>
<th>Jakost struje [mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0.6</td>
</tr>
<tr>
<td>ADC noise reduction</td>
<td>0.4</td>
</tr>
<tr>
<td>Standby</td>
<td>0.36</td>
</tr>
<tr>
<td>Power-save</td>
<td>0.36</td>
</tr>
<tr>
<td>Power-down</td>
<td>0.2</td>
</tr>
</tbody>
</table>
5. Zaključak

U ovom radu uspoređena su stanja mirovanja u kojima se uređaj Arduino može nalaziti. Svako stanje postiže uštede na način da isključuje određene sklopove koji se nalaze na razvojnoj ploči. Različiti sklopovi koriste različite količine energije te su u uštede različite. Odabir stanja mirovanja ovisit će o situaciji odnosno projektu za koji ćemo dizajnirati sustav. S obzirom na uštede postoji 5 stanja mirovanja: idle, ADC noise reduction, standby, power-save i power-down.

**Idle** stanje ne pruža neke osobito velike uštede jer isključuje tek nekoliko sklopova koji nisu izrazito važni za rad većine programa te ono samo po sebi nije vrlo često korišteno. **ADC noise reduction, standby i power-save** pružaju nešto značajnije uštede te je njihovu primjenu moguće naći u projektima kod kojih je potrebna brza reakcija nakon buđenja iz stanja mirovanja. Ova stanja isključuju veoma velik broj sklopova kako bi postigla tražene uštede. **Power-down** stanje pruža izrazito velike uštede energije jer isključuje najveći broj sklopova na razvojnoj ploči. Ovo stanje je korišteno u projektima koji bi trebali raditi i po nekoliko godina na baterijama. Malen problem kod ovog stanja je što ima 60 µs duže vrijeme buđenja što je potrebno imati na umu prilikom dizajniranja sustava.


Ukoliko standardno definirana stanja mirovanja nisu ono što nam je potrebno, odnosno nisu dovoljna, moguće je uštedu ostvariti skidanjem sklopova sa razvojne ploče. Bolje rečeno, moguće je sklopove povezati i bez razvojne ploče bez potrošnje energije na sklopove koji nisu potrebni.

Uređaj Arduino je korišten zbog jednostavnosti svog korištenja, vrlo dobre podrške i brojnih drugih mogućnosti, no on nije jedini uređaj koji ima mogućnost uštede energije te se u ovom radu opisani principi mogu primijeniti i na druge uređaje kako bi se postigla ušteda energije.
Literatura


Upravljanje energetskim stanjima uređaja Arduino

Sažetak

Potrošnja energije u današnje vrijeme raste nevjerojatnom brzinom. Jedan od razloga za to povećanje je i dostupnost visoko tehnoloških uređaja. Upravo je to izazvalo sve veća ulaganja u istraživanja u nove oblike energije i nove načine uštede energije. Kako bi se smanjila potrošnja energije potrebno je isključiti što više mogućnosti pojedinog uređaja u fazi kada se ne koristi. Kako bi se ovaj princip jasnije razumio, u sklopu ovog rada opisani su načini uštede energije uređaja Arduino isključivanjem pojedinih sklopova integriranih na razvojnu ploču., te ponovno uključivanjem istih po potrebi, odnosno u ovom radu ovisno o vanjskom satu. Uočljivo je da su uštede i u ovako jednostavnim slučajevima značajne te da bi gore navedeni princip mogao biti primijenjen i za potrošače većih razmjera.

Ključne riječi: Arduino, RTC, DS3231, stanja mirovanja

Managing power states of an Arduino device

Abstract

Power consumption in today's time grows at incredible speed. One of the reasons for the increase is the availability of high technology devices. It has caused ever greater investment in exploring new forms of energy and new ways of saving energy. In order to reduce power consumption, it is necessary to disable as many of the features of a particular device at a time when it is not used. In order to understand this principle more clearly, this work describes the methods for reducing the energy consumption of the Arduino device by turning off some of the integrated circuits on the development board, and by enabling them as needed, or as shown in this work depending on the external clock. It is noticeable that savings in such simple cases are significant so we can conclude that the above principle could be applied to larger consumers as well.

Keywords: Arduino, RTC, DS3231, sleep modes
### Skraćenice

<table>
<thead>
<tr>
<th>Abrevijaturni izraz</th>
<th>Engleski izraz</th>
<th>Srpski izraz</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC</td>
<td>Real Time Clock</td>
<td>sat sa stvarno-vremenskim svojstvima</td>
</tr>
<tr>
<td>SE</td>
<td>Sleep Enable</td>
<td>omogući stanje mirovanja</td>
</tr>
<tr>
<td>SREG</td>
<td>Status Register</td>
<td>statusni registar</td>
</tr>
<tr>
<td>TIMSK</td>
<td>Timer Interrupt Mask Register</td>
<td>registar za maskiranje prekida brojača</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
<td>Inter-integrirani krug</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
<td>mikrokontroler</td>
</tr>
<tr>
<td>MCUCR</td>
<td>Microcontroller Unit Control Register</td>
<td>statusni registar mikrokontrolera</td>
</tr>
<tr>
<td>ACSR</td>
<td>Analog Comparator Control and Status Register</td>
<td>kontrola analognog komparatora i statusni registar</td>
</tr>
<tr>
<td>ADC</td>
<td>Analog-Digital Conversion</td>
<td>analogno-digitalna pretvorba</td>
</tr>
</tbody>
</table>