SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 558

Izgradnja mikrojezgre za ugradbeni sustav

Marko Turk

Zagreb, lipanj 2013.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

U radu je opisan i izgrađen jednostavan ugradbeni sustav za upravljanje proizvoljnim procesima. Sustav je temeljen na ATmega8 mikrokontroleru koji je pripremljen da se može brzo i jednostavno ugraditi u željeno okruženje.

Namjena sustava je upravljanje jednostavnim procesima koji ne zahtjevaju veliku količinu računarske obrade[1]. Sustav ima malu potrošnju, može ga se brzo prilagoditi određenom procesu i pogodan je za brz razvoj sustava koji će na temelju ulaznih podataka samostalno odlučivati o akcijama koje je potrebno izvesti.

Za sustav je izgrađen jednostavan operacijski sustav koji pruža korisniku (programeru) sučelje za izvedbu željenog upravljanja. Podržani su vanjski prekidi na koje je moguće spojiti uređaje koji će komunicirati sa sustavom. Sustav ima podršku za alarme koji se izvode u zadanim vremenskim intervalima i pozivaju korisničke funkcije. Za interakciju sa sustavom postoji podrška za serijsku vezu pomoću koje se mogu zadavati naredbe sustavu i primati rezultati izvođenja.

Sustav se prilagođava svakom pojedinom procesu preko korisničkih programa. Korisnik ne mora brinuti o unutarnjem radu sustava. Koristeći sučelje sustava korisnik može brzo napisati program koji će upravljati željenim procesom.

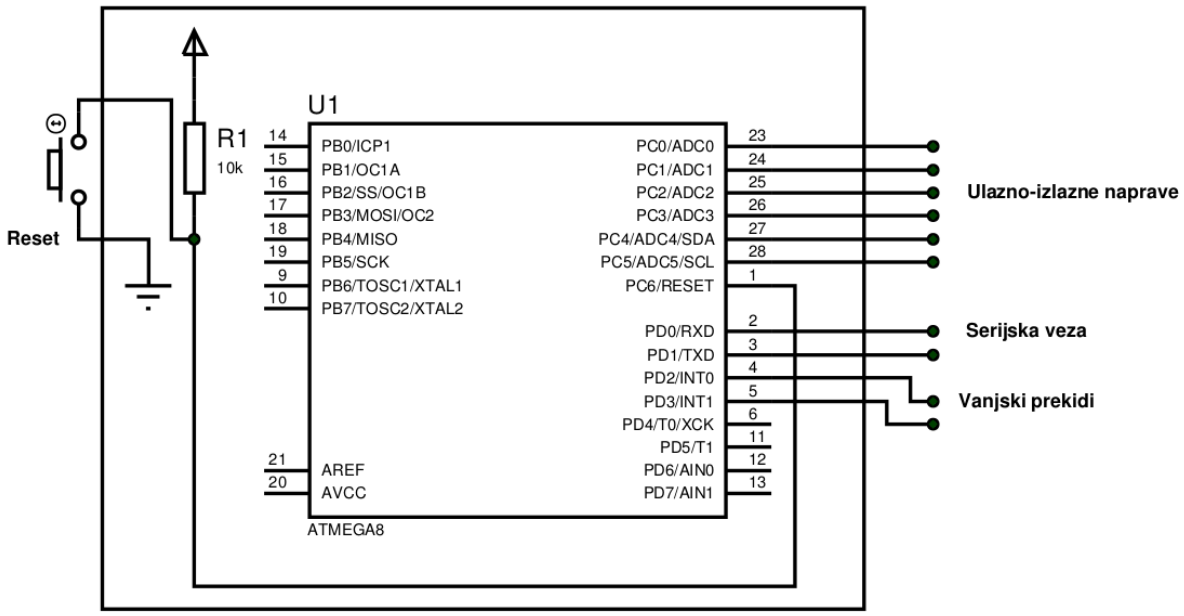
U 2. poglavlju opisan je način na koji je izvedeno sklopovlje sustava. Opisani su sklopovski dijelovi od kojih je sačinjen sustav i način na koji se spajaju vanjski uređaji. Razvojna okolina, način prevođenja izvornog koda sustava i korišteni alati opisani su u 3. poglavlju. Način izvedbe programske podrške opisan je u 4. poglavlju. Način izrade korisničkih programa opisan je u 6. poglavlju, a u 5. poglavlju navedeno je sučelje sustava. U 7. poglavlju nalaze se razni primjeri korisničkih programa.

[1] Mikrokontroler ima malu procesorsku snagu i nije pogodan za izvođenje operacija koje koriste brojeve s pomičnim zarezom

2. Sklopovlje sustava

Na slici 2.1 prikazano je sklopovlje od kojeg je sustav sačinjen. Sustav je nazvan RTCS (*Real-Time Control System*).

Za pokretanje sustava opisanog u ovom radu potrebno je spojiti komponente prema prikazanoj shemi i podesiti ih prema uputama u nastavku.



Slika 2.1: RTCS sklopovlje

2.1. Mikrokontroler

Glavna komponenta sustava je Atmelov AVR mikrokontroler ATmega8, 8 bitni RISC[1] mikrokontroler s modificiranom Harvard arhitekturom. Instrukcije se spremaju u Flash memoriju, a RAM se koristi za podatke [3].

[1] *Reduced instruction set computer*, dizajn procesora kod kojeg su instrukcije jednostavne i obavljaju samo osnovne funkcije

Harvard arhitektura ima fizički odvojenu radnu memoriju (za podatke i stog) od programske memorije i različit adresni prostor za pojedine memorije. Pristup programskoj memoriji moguć je samo kod dohvaćanja instrukcija. U modificiranoj Harvard arhitekturi može se čitati i pisati iz programske memorije korištenjem posebnih instrukcija procesora [1].

Programski jezik C nije dizajniran za Harvard arhitekturu pa su prevoditelji (engl. *compiler*) posebno prilagođeni za upravljanje odvojenim adresnim prostorom [1]. Način na koji se upravlja adresnim prostorima u ovom sustavu opisan je u 4. poglavlju.

ATmega8 ima 8 KB Flash programske memorije za instrukcije. Flash memorija podržava *In-System* programiranje što znači da se mikrokontroler može programirati direktno u odredišnom sustavu bez potrebe vađenja i spajanja na programator. Uz EEPROM (*Electrically Erasable Programmable Read-Only Memory*) od 512 okteta koji se može koristiti za spremanje postavka sustava, mikrokontroler ima i ugrađen SRAM (*Static random-access memory*) veličine 1 KB koji se koristi za podatke [3]. Sadržaj RAM memorije izgubi se kod svakog pokretanja pa je inicijalizirane podatke koje sustav koristi potrebno spremiti u Flash programsku memoriju i kod pokretanja sustava prekopirati u RAM. Način na koji je u ovom sustavu izvedena inicijalizacija podataka opisan je u 4.3.

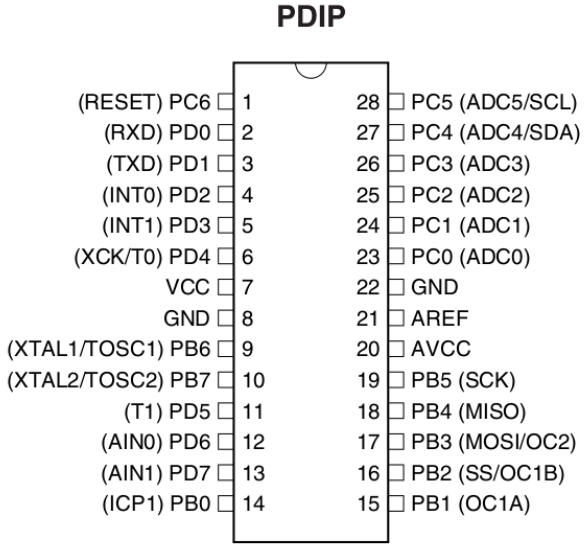
Mikrokontroler je dostupan u tri tipa pakiranja: PDIP (*Plastic Dual in-line package*), TQFP (*Thin Quad Flat Package*) i MLF (*Micro leadframe*) [3]. U ovom sustavu korišteno je PDIP pakiranje[2]. Konfiguracija konektora u PDIP pakiranju prikazana je na slici 2.2.

2.1.1. Način spajanja i konfiguriranja mikrokontrolera

Konektori mikrokontrolera iskorišteni su na sljedeći način:

- Konektori 23–28 (PC0–PC5) iskorišteni su za spajanje s ulazno-izlaznim uređajima. Mogu pojedinačno biti podešeni za ulaz ili za izlaz.
- Konektor 1 (RESET) spojen je na tipku za resetiranje sustava. Pritiskom na tipku, sustav se vrati u početno stanje i kreće s ponovnim izvođenjem. RESET je aktivan u logičkoj nuli.

[2]Izabran je PDIP jer je pogodan za korištenje na razvojnoj pločici i može ga se iskoristiti u odredišnom sustavu bez izrade tiskane pločice. Operacijski sustav koji je u ovom radu izgrađen radi sa svim vrstama pakiranja u kojima ATmega8 dolazi i korisnik može po želi odabrati pakiranje koje mu najbolje odgovara. Konfiguracije konektora za ostala podnožja opisane su u [3]



Slika 2.2: Raspored konektora na PDIP podnožju

– RXD (konektor 2) i TXD (konektor 3) služe za spajanje sustava na serijsku vezu.

– Konektori 4 i 5 (INT0 i INT1) koriste se za prihvaćanje vanjskih prekida. Podešeni su da generiraju prekid na padajući brid. Interno su spojeni na *pull-up* otpornik.

Mikrokontroler je potrebno spojiti na izvor napajanja od $5V$ preko konektora 8 (GND, niska razina napajanja) i konektora 7 (VCC, visoka razina napajanja). Preporuča se da se spoji napajanje od minimalno $15mA$.

ATmega8 ima ugrađen 8 bitni brojač vremena koji može generirati prekide na zadane intervale. Takav brojač u ovom je sustavu iskorišten za implementaciju alarma. Brojač vremena podešen je da generira prekid svakih $16ms$ čime je definirana maksimalna razlučivost alarma.

2.1.2. Spajanje vanjskog kristalnog oscilatora

Mikrokontroler podržava frekvenciju takta do $16MHz$. Moguće je koristiti interni oscilator ili podesiti sustav da koristi vanjski RC ili kristalni oscilator [3]. Izvor takta određuje se preko *Flash Fuse* bitova koji se mogu podešavati koristeći programator opisan u 3.3. Moguće vrijednosti *Fuse* bitova prikazane su u tablici 2.1.

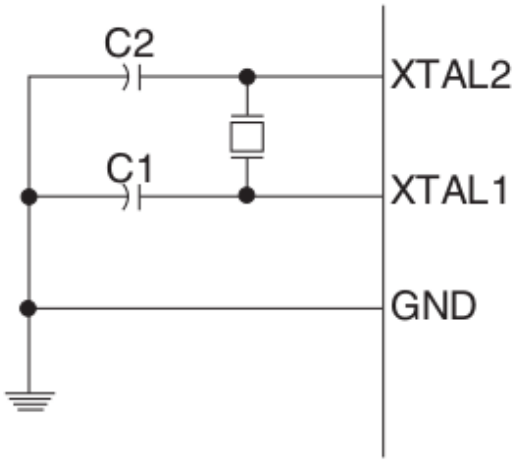
U ovom sustavu korišten je takt od $4MHz$ i potrebno je pažljivo podesiti mikrokontroler jer implementacija alarma direktno ovisi o zadanoj frekvenciji.

Vanjski kristalni oscilator od $4MHz$ spaja se na mikrokontroler preko konektora

Tablica 2.1: Odabir izvora takta procesora

Izvor takta	**CKSEL3..0**
Vanjski kristalni ili keramički oscilator	1111–1010
Vanjski kristal male frekvencije	1001
Vanjski RC oscilator	1000–0101
Kalibrirani interni RC oscilator	0100–0001
Vanjski izvor takta	0000

XTAL2 i XTAL1 prema shemi na slici 2.3. Za kondenzatore C1 i C2 preporučju se vrijednosti od $12 - 22pF$.



Slika 2.3: Način spajanja vanjskog kristalnog oscilatora

2.2. Spajanje vanjskih uređaja na sustav

Na sustav je moguće spojiti serijski terminal, ulazno-izlazne uređaje i uređaje koji generiraju prekid.

2.2.1. Spajanje serijskog terminala

Serijskim terminalom mogu se zadavati naredbe sustavu i primati rezultati izvođenja. Osim terminala, mogu se spojiti i različiti drugi uređaji s kojima će se preko serije komunicirati.

Serijski terminal spaja se na sustav preko RXD i TXD konektora. RXD liniju uređaja koji spajamo na ovaj sustav potrebno je spojiti na TXD konektor, a TXD liniju na RXD konektor. Terminal koji se spaja potrebno je namjestiti na brzinu od 9600 bps (*bits per second*), 8 podatkovnih bitova, 1 završni bit (engl. *stop bit*) i bez bitova za provjeru pariteta.

2.2.2. Spajanje ulazno-izlaznih uređaja

Ulazno-izlazni uređaji spajaju se na konektore PC0–PC5. Svaki konektor se pojedinačno može podesiti za ulaz ili za izlaz. U sustavu se može čitati stanje svakog konektora i kontrolirati izlaz — postavljati ga na visoku ili nisku razinu.

Maksimalna struja kroz pojedinačne konektore ne smije prelaziti $20mA$. Detaljnije električne karakteristike mogu se naći u [3].

2.2.3. Spajanje uređaja koji generiraju prekid

Prekidi se spajaju na konektore INT0 i INT1 mikrokontrolera. Prekid se generira kada se detektira padajući brid. Interno su konektori za prekide spojeni na *pull-up* otpornik, a uređaji koji žele generirati prekid moraju na kratko spojiti konektor na nisku razinu napajanja. Nakon što se generira jedan prekid, daljnji prekidi na istoj liniji neće se ponavljati dok se ponovno ne detektira prijelaz iz visoke u nisku razinu.

Svaki prekid poziva zadanu korisničku funkciju gdje korisnik može obraditi prekid i obaviti potrebne akcije. Moguće je izvođenje više korisničkih funkcija za isti prekid.

3. Razvojna okolina i alati

Prije korištenja sustava potrebno je prevesti izvorni kod i u simulatoru testirati željenu funkcionalnost. Nakon uspješne simulacije potrebno je izgraditi sklopovlje opisano u 2. poglavlju i programirati mikrokontroler. Svi potrebni koraci detaljnije su opisani u nastavku.

Za rad sa sustavom, prevođenje sustava, simuliranje i programiranje mikrokontrolera preporuča se korištenje neke Linux distribucije. Svi koraci opisani u ovom radu pretpostavljaju da korisnik ima već instaliranu Linux distribuciju.

3.1. Prevođenje izvornog koda

Sustav je programiran u C programskom jeziku, a neki dijelovi u kojima je potrebno direktno komunicirati s mikrokontrolerom napisani su u asembleru. Za prevođenje izvornog koda korišten je AVR-GCC prevoditelj [12]. Upute za instalaciju mogu se pronaći u [13].

Izvorni kod sustava nalazi se u `git` [17] repozitoriju koji se može dohvatiti iz [18]. Nakon što se skine cijeli repozitorij potrebno je otvoriti granu `oszur_kernel2` u kojoj je izvorni kod sustava opisanog u ovom radu.

Sustav se prevodi koristeći `make` program. Način prevođenja i neke postavke sustava nalaze se u datoteci `Makefile`. Za prevođenje sustava potrebno se pozicionirati u direktorij s izvornim kodom i izdati naredbu:

```
$ make
```

nakon čega će se stvoriti izvršna verzija sustava u direktoriju `build`. Datoteka `rtcs.hex` je Intelov HEX format koji se koristi za programiranje mikrokontrolera, a `rtcs.elf` je ELF izvršna datoteka koja se može koristiti u nekim simulatorima. Detaljnije o simulatorima i programiranju mikrokontrolera opisano je u 3.2 i 3.3.

Korisnički programi nalaze se u `programs` direktoriju. Za svaki novi program potrebno je stvoriti novi poddirektorij i prije prevođenja koda dodati novi program u

`Makefile` datoteku u varijablu `DIRS_P`. Korisnički program bit će preveden zajedno sa sustavom.

Izbor programa koji će se pokrenuti radi se u datoteci `kernel/startup.c`. Nakon linije `/* start desired program(s) */` potrebno je dodati poziv željenog programa. Način izrade korisničkih programa opisan je u 6. poglavlju.

3.2. Simulatori za razvoj i testiranje

Za ispitivanje rada sustava mogu se koristiti simulatori. Opisano je nekoliko simulatora, neki podržavaju stvaranje cijelog okruženja s vanjskim uređajima, a neki samo simuliraju AVR mikrokontroler.

Svrha pokretanja sustava u simulatorima je brza provjera funkcionalnosti bez potrebe korištenja pravog sklopovlja. U simulatorima je lakše naći eventualne greške u sustavu jer je moguće u svakom trenutku zaustaviti simulaciju i analizirati trenutno stanje mikrokontrolera. Moguće je testirati rad sustava sa različitim vanjskim uređajima i tek kada je korisnik zadovoljan funkcionalnošću kreće se u testiranje na pravom sklopovlju.

3.2.1. *Simavr* simulator

Simavr je AVR simulator otvorenog koda za Linux koji koristi AVR-GCC. Potrebno je skinuti izvorni kod simulatora, prevesti ga i instalirati na sustav [11]. Moguće je direktno simulirati Intelov HEX format.

Naredbom

```
$ simavr -m atmega8 -f 4000000 -ff rtcs.hex
```

pokreće se simulacija mikrokontrolera i vidljiv je ispis podataka sa serije. Zastavica `-m` određuje tip mikrokontrolera koji se simulira. Sa zastavicom `-f` zadaje se frekvencija takta (u Hz), a zastavicom `-ff` zadajemo putanju do izvršne datoteke sustava (Intel HEX format).

Simulator ima mogućnost slanja prekida, postavljanje i čitanje sa konektora mikrokontrolera, spajanje serije i ostalih vanjskih uređaja. Za takve naprednije mogućnosti potrebno je napisati vlastiti program koji će preko sučelja *Simavr* programa upravljati mikrokontrolerom. Detaljna dokumentacija simulatora još ne postoji, potrebno je čitati izvorni kod [11].

Primjer jedne simulacije nalazi se u izvornom kodu sustava [18], grana `oszur_kernel2`, direktorij `simulation/simavr_rtcs/`. Potrebno je izvršnu datoteku sustava, `rtcs.elf`, staviti u navedeni direktorij i pokrenuti naredbu `make` nakon čega se stvori izvršna datoteka za pokretanje simulacije. U primjeru je napravljen prekidač koji je spojen na INT0 liniju mikrokontrolera. Pritiskom na tipku `'r'` aktivira se prekidač, pokreće se prekid i vidljiv je ispis mikrokontrolera preko serije.

Simulator je u fazi razvoja i neke mogućnosti mikrokontrolera još nisu implementirane. Ne postoji podrška za alarme i slanje podataka mikrokontroleru preko serije nije moguće ako se koristi prekid za primanje znakova kao u ovom sustavu.

3.2.2. *SimulAVR* simulator

SimulAVR je simulator otvorenog koda za Atmelove ATtiny i ATmega mikrokontrolere. Izvorni kod i način instalacije može se naći u [15].

SimulAVR se pokreće s naredbom:

```
$ simulavr -F 4000000 -d atmega8
               -f build/rtcs.elf -W 0x2c,-
```

Zastavica `-W` u naredbi označava da simulator čita podatke iz zadanog registra (izlazni registar za seriju na adresi `0x2c`) i zapisuje podatke na standardni izlaz. Na taj je način izvedena serijska veza s mikrokontrolerom. Neki ostali parametri simulatora navedeni su u tablici 3.1.

Simulator se može spojiti sa GDB [14] *debuggerom* koristeći naredbu

```
$ simulavr -F 4000000 -d atmega8 \
    -f build/rtcs.elf -W 0x2c,- -g -p 5050
```

koja otvara vrata 5050 i čeka da se korisnik spoji sa GDB programom. GDB se pokreće naredbom

```
$ avr-gdb
```

Nakon pokretanja *debuggera* potrebno ga je spojiti na simulator i učitati izvršnu datoteku sustava izdajući sljedeće dvije naredbe GDB programu

```
target remote tcp:localhost:5050
file /putanja/do/rtcs.elf
```

Upute za *debuggiranje* korištenjm GDB programa mogu se naći u [7].

Simulator podržava i upravljanje konektorima mikrokontrolera. Više informacija može se pronaći u [15].

Tablica 3.1: Zastavice *SimulAVR* simulatora

Zastavica	Opis
`-F`	Odabir frekvencije takta (u Hz)
`-d`	Vrsta mikrokontrolera
`-f`	Putanja do izvršne datoteke sustava
`-W`	Čitanje podataka sa registra mikrokontrolera
`-u`	Otvara vanjsko sučelje za kontrolu konektora
`-g`	Čeka na spajanje GDB programa
`-p`	Određuje vrata na koja se potrebno spojiti GDB programom

3.2.3. *Proteus 8* okruženje

Proteus 8 [5] je za razliku od ostalih opisanih simulatora vlasnički softver.

Podržava simulaciju mikrokontrolera i spajanje raznih vanjskih uređaja. Može kontrolirati konektore, slati prekide i moguće je spojiti serijski terminal na mikrokontroler. Osim simulacije mikrokontrolera ovim je softverom moguće izgraditi cijelo okruženje u kojem će sustav raditi i na taj način testirati sve komponente sustava.

Primjer *Proteus* projekta za ovaj sustav može se pronaći u [18], grana `oszur_kernel2`, u direktoriju `simulation/proteus_rtcs`.

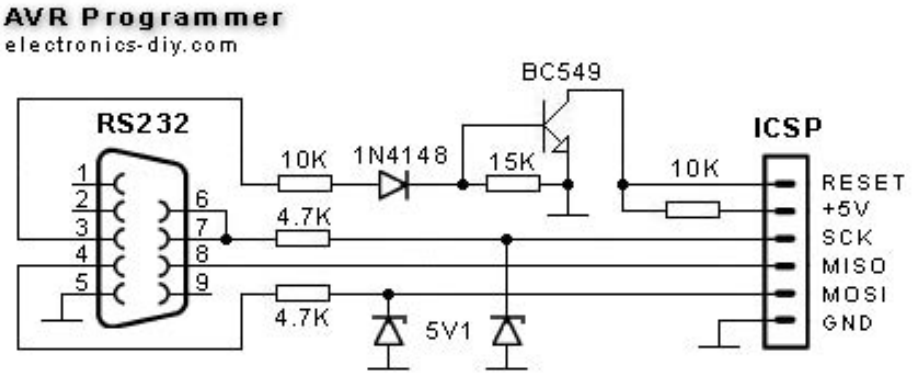
3.3. Izvedba programatora

Za programiranje mikrokontrolera može se iskoristiti jednostavan programator opisan u [6]. Spaja se na serijsku vezu računala (RS232 priključak) i podržava *In-System* programiranje. Programator je moguće vrlo jeftino i brzo izgraditi, sastoji se od nekoliko elektroničkih komponenata koje je moguće direktno spojiti bez potrebe za tiskanom pločicom.

Nedostatak programatora je korištenje serijskog priključka računala jer se takav priključak ne nalazi na novijim računalima. Korištenje konvertora sa USB na RS232 većinom isto nije moguće. Programator koristi tehniku direktnog postavljanja bitova (engl. *Big banging*) koju ne podržava većina konvertera i programiranje je jako sporo i podložno greškama kod programiranja.

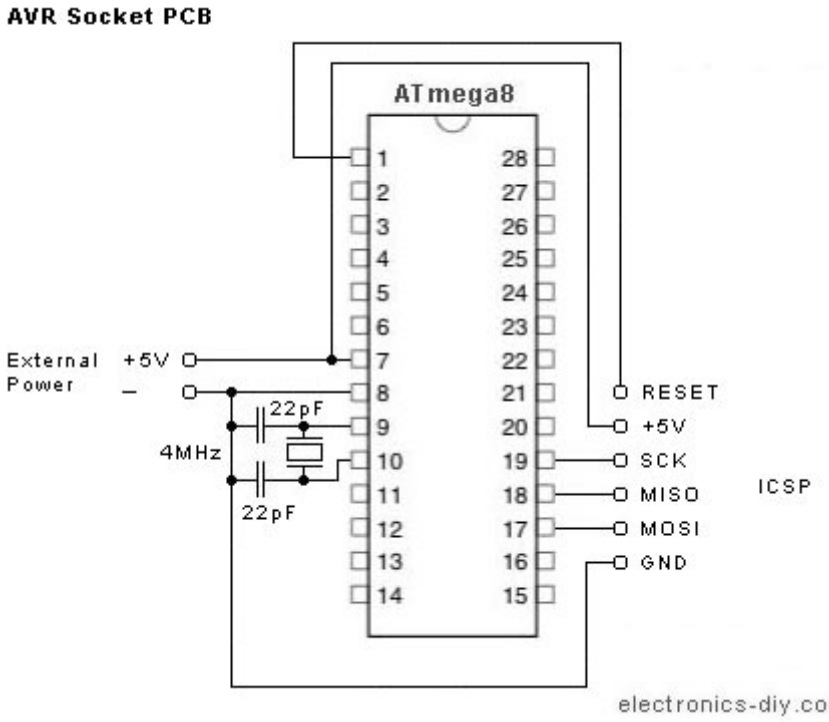
Za programiranje sustava potrebno je izgraditi programator prema shemi na slici 3.1.

Programator se spaja na mikrokontroler prema shemi na slici 3.2. Za rad programatora potrebno je vanjsko napajanje od $5V$. Za programiranje se može koristiti softver



Slika 3.1: Shema programatora za mikrokontroler preuzata iz [6]

PonyProg [10] ili *AVRDUDE* [4]. U 3.4 opisan je način programiranja sa *AVRDUDE* programom.



Slika 3.2: Shema spajanja programatora na mikrokontroler preuzata iz [6]

3.4. Programiranje mikrokontrolera

Nakon prevođenja izvornog koda sustava potrebno je programirati mikrokontroler. *AVRDUDE* programom moguće je zapisivati Intelov HEX oblik izvršne datoteke sus-

tava na mikrokontroler, može se programirati EEPROM memorija i postavljati *Fuse* bitove.

Upute za instalaciju i korištenje programa nalaze se u [4].

3.4.1. Flash memorija

Programiranje Flash memorije radi se naredbom

```
$ avrdude -p m8 -c ponyser -P /dev/ttyS0 \
    -e -U flash:w:rtcs.hex
```

Zastavica -U označava memorijsku operaciju koja će se izvršiti, u ovom slučaju će se izvršna datoteka `rtcs.hex` zapisati u Flash memoriju mikrokontrolera. Ostale zastavice programa opisane su u tablici 3.2.

Tablica 3.2: Zastavice *AVRDUDE* softvera za programiranje

Zastavica	Opis
`-p`	Odabir vrste mikrokontrolera
`-c`	Vrsta programatora koji se koristi
`-U`	Odabir memorijske operacije
`-P`	Serijski port na koji je spojen programator
`-e`	Brisanje cijelog mikrokontrolera prije programiranja

3.4.2. Postavljanje *Fuse* bitova

Fuse bitovi mogu se postavljati sa sljedećom naredbom:

```
$ avrdude -p m8 -c ponyser -P /dev/ttyS0 \
    -e -U fuse:w:fuse.hex
```

U datoteku `fuse.hex` potrebno je zapisati jedan oktet podataka: željene postavke *Fuse* bitova. Detaljan opis značenja *Fuse* bitova nalazi se u [3].

4. Izvedba operacijskog sustava

Izvorni kod sustava sastoji se od slojeva koji su podijeljeni u tri direktorija: `arch` (sloj arhitekture), `kernel` (sloj jezgre) i `programs` (sloj korisničkih programa). U direktoriju `include` nalaze se datoteke s opisom funkcija koje povezuju sloj arhitekture i sloj jezgre. Svaki pojedini sloj detaljnije je opisan u 4.1.

Za prevođenje se koristi `Makefile` skripta u kojoj su opisani načini prevođenja C i asemblerskog izvornog koda. Način na koji se sustav povezuje u izvršnu datoteku opisan je u skripti `linker.ld`. Instrukcije se spreme na početnu adresu `0x0000` nakon čega slijede podaci. Podaci su pripremljeni da se prebace u memoriju na početnu lokaciju `0x800060` gdje se nalazi memorijski prostor RAM memorije. Prilikom programiranja sustava, instrukcije i podaci se nalaze u Flash memoriji, a tek prilikom pokretanja sustava se podaci kopiraju iz Flash memorije u RAM memoriju. Način na koji je izvedeno kopiranje podataka opisano je u 4.3.

4.1. Slojevi operacijskog sustava

4.1.1. Sloj arhitekture

U sloju arhitekture (engl. *Arch layer*) nalazi se kod koji komunicira direktno s mikrokontrolerom. Sloj arhitekture pruža ostalim slojevima pristup sklopovlju. Sastoji se od podsustava za prekide (`interrupt`), alarme (`time`), serijsku vezu (`uart`) i podsustava za početno konfiguriranje mikrokontrolera i inicijalizaciju podataka (`loader`). Podsustavi sloja arhitekture nisu namijenjeni da ih poziva sam korisnik, već su namijenjeni kao veza između jezgre i sklopovlja sustava.

Sloj arhitekture postavi kazaljku stoga, inicijalizira podatke i poziva sloj jezgre iz koje se rade daljnje inicjalizacije.

4.1.2. Sloj jezgre

Sloj jezgre (engl. *Kernel layer*) brine se za postavljanje sustava u način rada prikladan za korisničke programe. Sadrži podsustave za obradu prekida i alarma, kontrolu komunikacijom preko serijske veze i dinamičko dodjeljivanje memorije korisničkim programima.

Većina koda jezgre preuzeta je iz BENU operacijskog sustava [8]. Detaljnije o BENU može se pronaći u [9]. Iteracija BENU jezgre korištena u ovom sustavu je `Chapter_03_Interrupts / 04_Interrupts`. Na jezgru je dodana podrška za alarme i komunikaciju serijskom vezom. Iz jezgre su izbačeni dijelovi za standardni ulaz i izlaz jer su zauzimali previše mjesta u Flash memoriji mikrokontrolera.

Sloj jezgre povezan je sa slojem arhitekture preko deklaracija funkcija u direktoriju `include/arch`. Sloj arhitekture mora implementirati sve funkcije navedene u tim datotekama da bi sloj jezgre mogao pravilno raditi.

4.1.3. Sloj za programe

U sloju za programe (engl. *Programs layer*) nalaze se korisnički programi. U sustavu se nalazi primjer programa, `hello_world`, koji korisnik može iskoristiti za pisanje svojih programa.

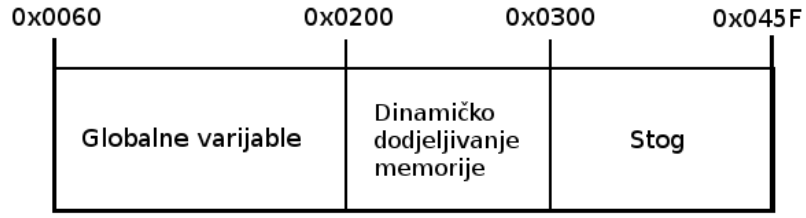
Detaljniji opis izrade korisničkih programa nalazi se u 6. poglavlju.

4.2. Upravljanje memorijom

Cijeli sustav zauzima 5451 B Flash memorije. Zbog ograničene veličine Flash memorije mikrokontrolera od 8 KB, za korisničke programe ostaje slobodno 2741 B memorije (uključujući instrukcije i globalne varijable).

Sadržaj Flash memorije definiran je u skripti povezivača koda (engl. *linker*) koja se nalazi u `linker.ld` datoteci sloja arhitekture. Na početnoj adresi (`0x0000`) nalaze se instrukcije `loader.S` datoteke u kojoj se nalazi kod koji se mora prvi izvesti kod pokretanja sustava. Nakon `loader.S` slijede instrukcije iz svih ostalih datoteka sustava. Poslije instrukcija dolaze inicijalizirane i neinicijalizirane globalne varijable. Iako se globalne varijable nalaza odmah iza programskog koda, podešene su za prebacivanje na adresu `0x800060` koja označava početak RAM memorije.

RAM memorija sustava prikazana je na slici 4.1. Adresni prostor memorije kreće se od adrese `0x0060` do adrese `0x045F`. Memorija je podijeljena u tri dijela: prostor za globalne varijable, prostor za dinamičko dodjeljivanje memorije i prostor za stog.



Slika 4.1: Raspored RAM memorije

Početak prostora za dinamičko dodjeljivanje memorije (engl. *heap*) nalazi se na adresi `0x0200` i ima veličinu od 256 okteta. Koristi se kada korisnički program tijekom rada zatraži od jezgre memorijski prostor. Početak i veličina definirani su u datoteci `memory.c` sloja arhitekture — `HEAP_START` označava adresu početka, a `HEAP_SIZE` veličinu prostora u oktetima.

Prostor za globalne podatke smješten je na početku RAM memorije (na početnoj adresi memorije, `0x0060`) i proteže se do početka prostora za dinamičko dodjeljivanje memorije. Ukupna veličina prostora je 416 okteta. Koristi se za spremanje svih globalnih varijabli u sustavu.

Stog kreće od kraja memorijskog prostora RAM memorije (adresa `0x045F`) i raste prema manjim memorijskim lokacijama. Maksimalna veličina stoga određena je veličinom prostora za dinamičko dodjeljivanje memorije. Kraj prostora za dinamičko dodjeljivanje memorije nalazi se na adresi `0x0300` što znači da je veličina prostora za stog 351 oktet.

4.3. Inicijalizacija globalnih varijabli

Globalne varijable sustava spremaju se zajedno sa instrukcijama u Flash memoriju. Kod pokretanja sustava potrebno je podatke kopirati u RAM memoriju da bi im sustav mogao pristupati.

Kopiranje podataka u RAM nalazi se u datoteci `loader.c` u sloju arhitekture, funkcija `arch_init_data` koja se poziva kod podizanja sustava. Skripta za povezivanje sustava u izvršni kod (`linker.ld`) podržava definiranje varijabli koje je moguće koristiti iz C koda [16]. Koriste se dvije varijable: `_etext` koja označava adresu kraja instrukcija i varijabla `_edata` koja označava adresu gdje završavaju podaci. Funkcija `arch_init_data` koristi te dvije varijable da bi saznala početak

i veličinu prostora s globalnim varijablama u programskoj memoriji. Za čitanje iz programske memorije koristi se funkcija `copy_from_prg` definirana u `loader.S` datoteci sloja arhitekture. Funkcija koristi posebnu instrukciju procesora (`lpm` [3]) za dohvaćanje jednog okteta iz programske memorije.

4.4. Vanjski prekidi

Vanjski prediti implementirani su u sloju arhitekture u datotekama `interrupt.c` i `interupt.S`. Kod ATmega8 mikrokontrolera moraju se podesiti prekidni vektori na početku Flash memorije. U datoteci `loader.S` postavljaju se prekidni vektori za prekide INT0, INT1, prekid kod preljeva brojača vremena i prekid za primanje sa serijske veze. Tablica 4.1 sadrži popis prekida i odgovarajućih vektora koji su podržani u sustavu. Detaljan opis prekidnih vektora može se naći u [3].

Tablica 4.1: Prekidni vektori

Adresa vektora	Opis
`0x000`	Reset
`0x001`	INT0
`0x002`	INT1
`0x009`	Preljev brojača vremena T0
`0x00B`	Primljen podatak sa serije

Kada mikrokontroler primi prekid, sprema se trenutna adresa koja se izvodi i skače se na lokaciju odgovarajućeg vektora. U datoteci `interrupt.S` definirana je funkcija koja sprema trenutni kontekst (sve registre) i zove odgovarajuću obradu za prekid. Nakon obrade prekida obnavalja se kontekst i nastavlja se s normalnim izvođenjem. Funkcije za obradu prekida implementirane su u datoteci `interrupt.c`.

Funkcija `arch_init_interrupts` u datoteci `interrupts.S` postavlja prekide sustava i konfigurira konektore mikrokontrolera da prihvaćaju vanjske prekide.

4.5. Serijska veza

Serijska veza implementirana je u datoteci `uart.S` u sloju arhitekture i `uart.c` u sloju jezgre. U sloju arhitekture nalaze se funkcije za slanje i primanje jednog okteta

podataka sa serije, a u sloju jezgre implementirane su funkcije koje mogu primati i slati polje znakova.

Kod slanja jednog okteta, čeka se da se očisti izlazni registar serijske veze nakon čega se šalje oktet. Primanje podataka sa serije radi se preko prekida implementiranog u `interrutpts.c` u sloju arhitekture. Kada se pošalje oktet podataka na mikrokontroler, generira se prekid u kojem se u međuspremik stavljaju svi znakovi primljeni sa serije. Korisnik preko sučelja sustava može dohvatiti podatke iz međuspremnika, a ako je međuspremnik prazan, korisnički poziv se blokira i čeka se znak sa serije.

4.6. Alarmi

Alarmi su implementirani u sloju arhitekture u datotekama `time.c` i `time.S`. U sloju jezgre implementirana je podrška za dodavanje korisničkih funkcija koje će se izvršavati u određenim vremenskim intervalima.

U funkciji `arch_time_init` u datoteci `time.S` podešen je 8 bitni brojač vremena T0 da generira prekid kod svakog preljeva. Postavljen je na dijelitelj frekvencije od 256, što znači da s frekvencijom takta od $4MHz$ broji 15625 puta u sekundi. 8 bitni brojač broji do ukupno $2^8 = 256$ što znači da se preljev događa svakih $0.016384s$ što je približno $16ms$. Svaki puta kada se dogodi preljev generira se prekid u kojem se provjerava koje je korisničke funkcije potrebno izvesti. Svaka korisnička funkcija ima zadano vrijeme u milisekundama do njezinog sljedećeg pozivanja. Na svaki prekid to se vrijeme smanjuje za 16 i kada je manje ili jednako nuli, funkcija se izvodi i resetira se vrijeme čekanja.

Zbog obrade alarma u diskretnim vremenskim trenutcima, najveća razlučivost alarma je $16ms$ što znači da se korisničke funkcije izvode u zadanim vremenskim intervalima uz najveću moguću grešku od $\pm 16ms$.

Ako se mijenja frekvencija takta mikrokontrolera ili se mijenja konfiguracija brojača vremena (mijenjanjem djelitelja frekvencije) da bi se postigla drugačija razlučivost alarma, potrebno je podesiti `TIMEINTERVAL` u datoteci `time.h` u `include/kernel` direktoriju. `TIMEINTERVAL` označava vrijeme u milisekundama između dva prekida brojača vremena i točno podešena vrijednost osigurava pravilan rad alarma.

4.7. Ulazno-izlazne naprave

Ulaz i izlaz odvija se preko PC0–PC5 konektora mikrokontrolera. Sve funkcije za rad s konektorima implementirane su u `io.S` datoteci sloja arhitekture.

`PORTC` označava konektore PC mikrokontrolera i pisanjem u taj registar postavljaju se konektori, a čitanjem iz tog registra dobivaju se trenutne vrijednosti na konektorima. Pisanjem u `DDRC` registar postavlja se konektor za ulaz ili za izlaz [3].

5. Sučelje sustava prema korisniku

5.1. Sučelje za prekide

Sučelje za vanjske prekide deklarirano je u datoteci `<arch/interrupt.h>` koju je potrebno uključiti u program koji koristi prekide.

Prekidi se povezuju s korisničkim funkcijama koristeći funkciju

```
void arch_register_interrupt_handler
    (unsigned int inum, void *handler);
```

Parametar `inum` označava vrstu prekida, a `handler` označava korisničku funkciju koja se povezuje s prekidom.

Za raskid veze prekida i korisničke funkcije koristi se funkcija

```
void arch_unregister_interrupt_handler
    (unsigned int irq_num, void *handler);
```

koja prekida izvođenje korisničke funkcije `handler` kod prekida `irq_num`.

Oznake prekida potrebne kod ovih funkcija mogu se pronaći u datoteci `interrups.h` sloja arhitekture.

Funkcije koje se povezuju s vanjskim prekidima moraju biti oblika

```
void ime_funkcije(int irq);
```

Nakon što se funkcija poveže s određenim prekidom, bit će pozvana svaki puta kada se zadani prekid dogodi i kao argument će primiti oznaku prekida.

5.2. Sučelje za alarme

U sustav je moguće dodati maksimalno 10 periodičkih funkcija. Jednom kada se funkcija doda na izvođenje, ona će se neograničeno puta periodički izvoditi.

Za dodavanje alarma koristi se funkcija

```
void sys__add_timer(const time_t *when);
```

koja je deklarirana u datoteci `<kernel/time.h>` koju je potrebno uključiti u izvorni kod programa koji koristi alarme.

Struktura `time_t` koristi se za definiranje perioda izvođenja i korisničke funkcije koja će se izvoditi.

```
typedef struct _time_ {
    int ms;
    int _ms_save;
    void (*f)(void);
} time_t;
```

Vrijednost `ms` označava period izvođenja u milisekundama, a varijabla `f` označava funkciju. Varijablu `_ms_save` nije potrebno postavljati, ona se koristi u internom radu sustava.

Funkcije koje će se izvoditi u određenim vremenskim intervalima moraju biti oblika

```
void ime_funkcije(void);
```

5.3. Sučelje za dinamičko dodjeljivanje memorije

Sustav podržava dinamičko dodjeljivanje memorije korisničkim programima. Jezgra *BENU* sadrži implementacije dva algoritma: metoda prvi odgovarajući (*first fit*) i GMA (*Grid Memory Allocator*) [9]. Izbor algoritma radi se u `Makefile` datoteci, varijabla `MEM_ALLOCATOR`.

Funkcije za dinamičko dodjeljivanje memorije deklarirane su u `<api/malloc.h>` koju je potrebno uključiti u korisničke programe.

Zauzimanje memorije veličine `size` ostvaruje se s funkcijom

```
void *kmalloc ( size_t size );
```

koja vraća adresu zauzetog memorijskog prostora.

Oslobađanje zauzete memorije radi se funkcijom

```
int kfree ( void *chunk );
```

Detaljan opis algoritama i način implementacije dinamičkog dodjeljivanja memorije opisan je u [9].

5.4. Sučelje za ulaz i izlaz

Sustav podržava postavljanje i čitanje sa konektora PC0–PC5. Funkcije za upravljenje ulazom i izlazom deklarirane su u `<arch/io.h>` i datoteku je potrebno uključiti u korisničke programe koji koriste ulaz ili izlaz.

Sve funkcije rade sa oktetom podataka u kojem točno određeni bitovi označavaju pojedine konektore sustava. Počevši brojenje od najmanje značajnog bita, 1. bit označava konektor PC0, a 6. bit konektor PC5.

Izbor koji će konektori biti ulazni, a koji izlazni radi se pomoću funkcije

```
void arch_io_init(char config);
```

koja kao argument prima jedan oktet s konfiguracijom konektora. Izbor načina rada konektora radi se stavljanjem jedinice za izlaz ili nule za ulaz na odgovarajuće mjesto za svaki konektor.

Postavljanje konektora obavlja se funkcijom

```
void arch_io_outb(char c);
```

koja šalje oktet `c` na izlaz konektora. Jedinica na mjestu za određeni konektor označava postavljanje visoke razine, a nula postavljanje niske razine.

Čitanje sa konektora obavlja se s

```
char arch_io_inb(void);
```

koja vraća jedinice za konektore koji su u visokoj razini, a nule za konektore u niskoj razini.

5.5. Sučelje serijske veze

Sučelje serijske veze nalazi se u datoteci `<api/uart.h>` koju je potrebno uključiti u programe koji koriste komunikaciju serijskom vezom.

Čitanje polja znakova[1] sa serije obavlja se funkcijom

```
unsigned int uart_get_string(char *buf);
```

[1]Znak u ovom kontekstu označava jedan oktet podataka primljenih ili poslanih sa serijske veze. Serijskom je vezom moguće slati i primati proizvoljne binarne formate, a ne samo znakovne nizove.

Funkcija stavlja sve znakove primljene sa serije u međuspremnik `buf` i vraća broj primljenih znakova.

Moguće je čitanje samo jednog znaka sa serije koristeći funkciju

```
char uart_get_char(void);
```

koja vraća jedan znak pročitan sa serije.

Ako prije poziva ovih funkcija nema primljenih podataka na seriji, funkcije blokiraju dok se ne primi barem jedan znak koji se odmah vraća korisniku.

Za slanje znakova serijskom vezom koristi se funkcija

```
void uart_put_string(const char *buf);
```

koja šalje niz znakova iz međuspremnika `buf` dok ne naiđe na nul-znak. Funkcija je namijenjena slanju znakovnih nizova (engl. *string*). Slanje binarnih podataka tom funkcijom nije praktično jer se mora posebno voditi računa o nul-znaku (oktet vrijednosti `0x00`) koji neće biti poslan.

Pojedinačno slanje okteta serijskom vezom može se napraviti funkcijom

```
void uart_put_char(char c);
```

6. Izrada korisničkih programa

Korisnički programi nalaze se u direktoriju `programs`. Za svaki program treba stvoriti novi direktorij u koji je potrebno staviti sve datoteke s izvršnim kodom programa.

U `Makefile` datoteci treba podesiti varijablu `DIRS_P`. Treba navesti sve direktorije gdje se nalaze izvršni kodovi korisničkih programa. Varijabla mora sadržavati samo one programe koji su potrebni korisniku da bi se uštedilo na memorijama sustava.

Deklaraciju glavne funkcije programa potrebno je dodati u datoteku `prog_info.h` u direktoriju `include/api`.

Svaki se program može sastojati od više funkcija raspoređenih u više datoteka. Ime glavne funkcije programa stavlja se u datoteku `startup.c` u sloju jezgre gdje se nalazi izbor i redosljed programa koji će se izvesti.

Primjeri nekih jednostavnijih korisničkih programa nalaze se u 7. poglavlju.

6.1. Važna pravila kod pisanja korisničkih programa

Funkcije i globalne varijable korisničkih programa mogu imati proizvoljno odabrana imena, jedino je bitno da se ne zovu isto kao i neka globalno vidljiva funkcija ili varijabla sustava jer će doći do konflikata kod prevođenja izvornog koda.

Maksimalna veličina prostora za globalne varijable je 416 okteta. Nakon prevođenja izvornog koda sustava potrebno se uvjeriti da veličina ne prelazi maksimalnu dopuštenu veličinu. Veličina dijela sa globalnim varijablama može se saznati naredbom

```
$ avr-objdump -h build/rtcs.elf
```

Potrebno je zbrojiti veličinu `.data` i `.bss` dijela. Ako ukupna veličina prelazi dopuštenih 416 okteta potrebno je smanjiti količinu globalnih varijabli ili smanjiti veličinu prostora za dinamičko dodjeljivanje memorije namiještanjem `HEAP_START` i `HEAP_SIZE` u datoteci `memory.c` sloja arhitekture.

Ako se neke funkcije žele implementirati u asembleru, potrebno je paziti na kompatibilnost s asemblerskim kodom koji generira prevoditelj C koda. Prevoditelj pretpostavlja da je u svako vrijeme registar r1 očišćen (postavljen na nulu) pa ga je u asemblerskom kodu potrebno očistiti nakon svake upotrebe. Registri r18–r27 i r30–r31 mogu se slobodno koristiti unutar asemblerskog koda i nije ih potrebno spremati. Ostale registre, ako se koriste, potrebno je spremiti i nakon korištenja ponovno vratiti njihove vrijednosti. [2]

Argumenti funkcija prenose se preko registara r25–r8 (gledajući s lijeva na desno u deklaraciji funkcije). Svi argumenti počinju na parnim registrima[1] (uključujući i `char` od 8 bitova koji ima iznad sebe prazan registar). Svi ostali argumenti koji ne stanu u navedene registre prenose se preko stoga. [2].

Povratne vrijednosti funkcije spremaju se u r24 ako su 8 bitne, 16 bitne spremaju se u r25-r24, 32 bitne vrijednosti u r22-r25, a 64 bitne u r18-r25.

Veličina tipova podataka koje prevoditelj koristi navedeni su u tablici 6.1.

Tablica 6.1: Veličine tipova podataka

Tip podataka	Veličina u bitovima
`char`	8
`int`	16
`long`	32
`long long`	64
`float` i `double`	32
Kazaljke (engl. *pointers*)	16

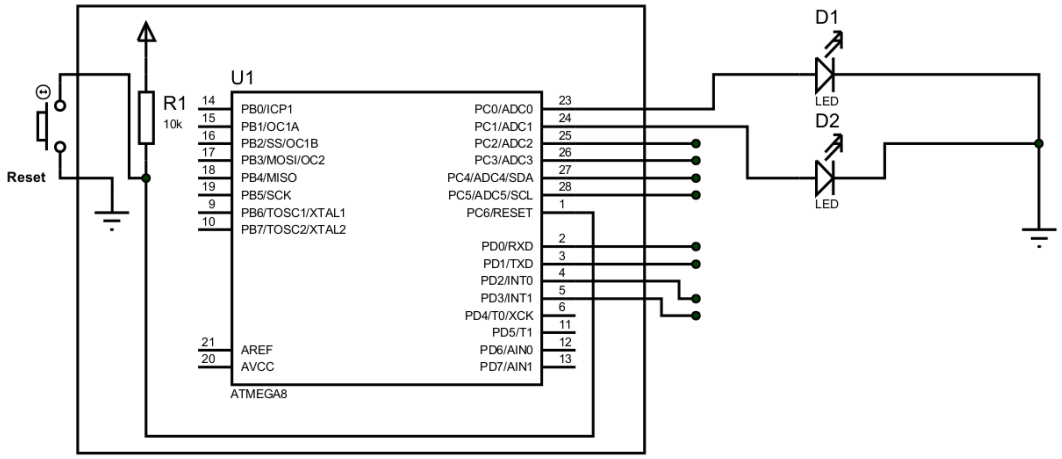
[1] ATmega8 koristi *little-endian* format zapisa što znači da ako je prvi argument funkcije *int* od 16 bitova, u registru r25 bit će 8 manje značajnih bitova, a u registru r24 8 više značajnih bitova.

7. Primjeri korisničkih programa

Svi primjeri nalaze se u izvornom kodu [18] u grani `oszur_kernel2` u direktoriju `programs`. Za pokretanje primjera potrebno je spojiti sustav prema opisanim shemama i u `startup.c` datoteci odabrati željeni program.

7.1. Primjer alarma

Sustav treba spojiti prema shemi na slici 7.1. Izvorni kod nalazi se u direktoriju `primjer_alarm`.



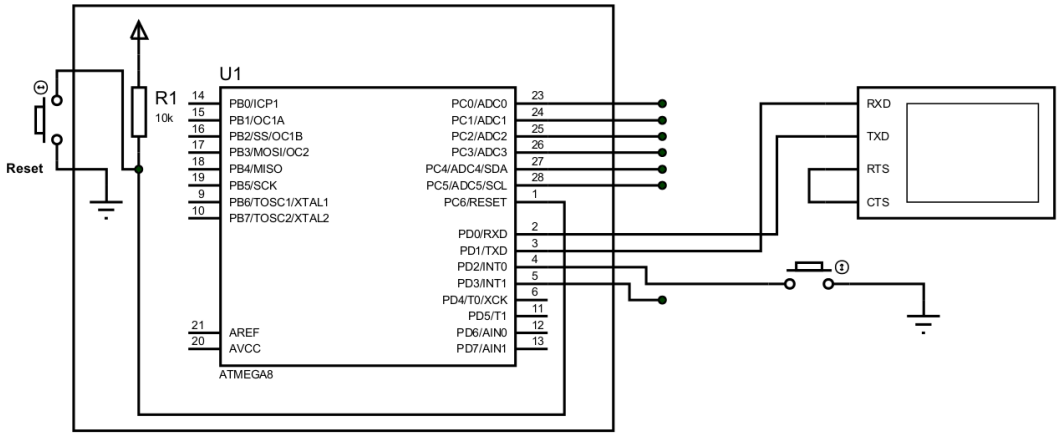
Slika 7.1: Shema sustava za primjer alarma

Izlazi konektora PC0 i PC1 povezani su s LED diodama koje svakih pet sekundi mijenjaju svoja stanja.

Program koristi alarme i izlaz preko konektora PC0 i PC1. Postavljen je jedan alarm koji svakih $5s$ izvodi funkciju `alarm`. Funkcija postavlja izlaz konektora PC0 i PC1 tako da invertira njihova trenutna stanja. Trenutno stanje izlaza nalazi se u varijabli `lampice` koja je na početku postavljena na vrijednost `0x01`.

7.2. Primjer serijske veze i korištenja prekida

Izvorni kod nalazi se u direktoriju `primjer_serija_prekid`. Sustav treba spojiti prema shemi na slici 7.2.



Slika 7.2: Shema sustava za primjer serije i prekida

INT0 konektor sustava potrebno je spojiti sa niskom razinom napajanja preko sklopke. Na RXD i TXD konektore treba spojiti serijski terminal.

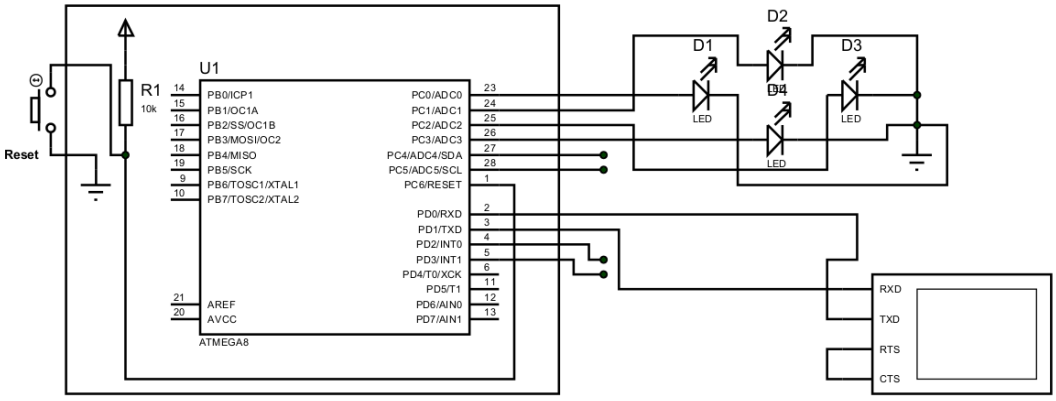
Napravljen je *echo server* koji sve znakove koje primi sa serije vraća nepromijenjene natrag na serijsku vezu. Kod pokretanja programa, na serijsku se vezu prvo ispiše `Server started` nakon čega znamo da je sustav spreman za primanje znakova.

Pritiskom na sklopku generira se prekid INT0 i izvodi se funkcija `handler`. Funkcija na seriju dodatno ispisuje zadnji znak koji je sustav primio.

7.3. Primjer zadavanja naredbi serijskom vezom

Izvorni kod nalazi se u direktoriju `primjer_lampice`. Sustav treba spojiti prema shemi na slici 7.3.

Program periodički pali četiri LED diode u krug. Slanjem znaka `'+'` preko serije povećava se brzina kojom se LED diode izmjenjuju, a znak `'-'` smanjuje tu brzinu. Paljenje LED dioda obavlja se pomoću alarma koji se izvodi u zadanom intervalu od $200ms$, a duljina zadržavanja na svakoj LED diodi određuje se varijablom `trenutno`. Funkcija koju izvodi alarm smanjuje vrijednost varijable `trenutna` za jedan i tek kada se dođe do nule, gasi se trenutna LED dioda i pali sljedeća u nizu.



Slika 7.3: Shema sustava za primjer zadavanja naredbi serijskom vezom

8. Zaključak

Sustav opisan u ovom radu temelji se na jednostavnom mikrokontroleru koji je dovoljan za upravljanje manjim procesima. Sustavom se može brzo i bez zamaranja s detaljima rada mikrokontrolera izvesti željeno upravljanje.

Za sustav je moguće izgraditi jednostavan i jeftin programator. Za simulaciju i razvoj sustava mogu se koristiti programi otvorenog izvornog koda što znači da je projektiranje i razvoj programa temeljenih na ovom sustavu moguć bez dodatnih troškova osim nabavljanja sklopovlja.

Izgrađen operacijski sustav podržava alarme, vanjske prekide, komunikaciju serijskom vezom i spajanje na vanjske uređaje. Sustav se može spojiti na razne prekidače i serijski terminal preko kojih će primati naredbe. Na izlaz sustava mogu se spojiti relej ili tranzistor koji će kontrolirati vanjski uređaj kojim se želi upravljati. Za izvođenje akcija u točno određenim intervalima mogu se koristiti alarmi sustava.

Moguće primjene sustava su upravljanje koračnim motorom ili kontrola rasvjete u nekom prostoru. Sustav ima malu potrošnju, može se spojiti na bateriju i iskoristiti za mjerenja koja se poslije preko serijske veze mogu prebaciti na računalo za daljnju obradu.

Sustav bi se mogao nadograditi podrškom za analogni ulaz (iz različitih senzora), ostvarenjem infra-crvenog protokola za primanje naredbi preko daljinskog upravljača i mogućnošću spajanja LCD ekrana. Time bi se dodatno povećale mogućnosti sustava i načini na koji bi se sustav mogao iskoristiti.

LITERATURA

[1] avr libc. *Data in Program Space*, . URL http://www.nongnu.org/avr-libc/user-manual/pgmspace.html.

[2] avr libc. *What registers are used by the C compiler?*, . URL http://www.nongnu.org/avr-libc/user-manual/pgmspace.html.

[3] Atmel Corporation. *ATmega8/L datasheet*. URL http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf.

[4] Brian S. Dean. *AVRDUDE - AVR Downloader/UploaDEr*. URL http://www.nongnu.org/avrdude/.

[5] Labcenter Electronics. *Proteus*. URL http://www.labcenter.com/.

[6] Electronics-DIY.com. *AVR Programmer*. URL http://electronics-diy.com/avr_programmer.php.

[7] Inc. Free Software Foundation. *Debugging with gdb*. URL http://sourceware.org/gdb/current/onlinedocs/gdb/.

[8] Leonardo Jelenković. *Benu, Operating System Increments for Education and Research*. URL http://sourceforge.net/projects/oszur13/.

[9] Leonardo Jelenković. *Operacijski sustavi za ugrađena računala*. skripta za predavanja, 2013. URL http://www.zemris.fer.hr/~leonardo/oszur/OSZUR-skripta.pdf.

[10] Claudio Lanconelli. *PonyProg2000*. URL www.lancos.com/prog.html.

[11] Michel Pollet. *simavr, AVR simulator za Linux*. URL http://gitorious.org/simavr.

[12] GNU Project. *GNU Compiler Collection*, . URL http://gcc.gnu.org/.

[13] GNU Project. *Host/Target specific installation notes for GCC*, . URL http://gcc.gnu.org/install/specific.html#avr.

[14] GNU Project. *GDB: The GNU Project Debugger*, . URL http://www.gnu.org/software/gdb/.

[15] Klaus Rudolph. *SimulAVR*. URL http://www.nongnu.org/simulavr/.

[16] Ian Lance Taylor Steve Chamberlain. *The GNU linker*. Red Hat Inc. URL http://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/Linker.pdf.

[17] Linus Torvalds. *Git, free and open source distributed version control system*. URL http://git-scm.com/.

[18] Marko Turk. *RTCS izvorni kod*. URL http://git.markoturk.info/rtcs.

Izgradnja mikrojezgre za ugradbeni sustav

Sažetak

U radu je prikazano ostvarenje jednostavanog operacijskog sustava za ATmega8 mikrokontroler. Projektirano je sklopovlje za rad sustava i opisan je način spajanja vanjskih uređaja. Opisan je način izvedbe programatora i svih potrebnih dijelova za pokretanje opisanog sustava. Opisan je način rada operacijskog sustava i sučelje sustava za korištenje u korisničkim programima. Navedeni su simulatori i razvojna okolina koji se koriste za pokretanje i testiranje izgrađenog sustava.

Ključne riječi: mikrokontroler, ATmega8, AVR, operacijski sustav, programator mikrokontrolera, simulator, alarmi, prekidi, serijska veza

Building a microkernel for embedded systems

Abstract

A simple system was built around ATmega8 microcontroller and an operating system was designed to run on this hardware. This work describes internal structure of the system and how external devices are connected and controlled by the system. The environment for developing user programs based on this system and simulators which can be used while developing are described. All necessary parts to run the system, including building a simple programmer for the microcontroller, are also described in this work.

Keywords: microcontroller, ATmega8, AVR, operating system, microcontroller programmer, simulation, timers, interrupts, serial communication