Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Drolc

Daljinsko upravljanje preko SMS z 8-bitnim mikrokrmilnikom

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Patricio Bulić

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00164/2011
Datum: 09.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ DROLC**

Naslov: **DALJINSKO UPRAVLJANJE PREKO SMS Z 8-BITNIM MIKROKRMILNIKOM**

REMOTE CONTROL VIA SMS AND AN 8-BIT MICROCONTROLLER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Razvijte sistem za daljinsko upravljanje s SMS sporočili z uporabo Atmelovih 8-bitnih mikrokrmilnikov. Sistem naj bo zasnovan okrog Atmelovega mikrokrmilnika ATmega 168A. Izberite ustrezne komponente sistema, kot so GSM pretvornik, temperaturni senzor, pretvornik nivojev za RS232 ipd. Razvijte testno aplikacijo za nadzor temperature ter krmiljenje stikal.

Mentor:
doc. dr. Patricio Bulić

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo! Glej tudi sam konec Poglavja **??** na strani **??**.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Drolc, z vpisno številko **63070468**, sem avtor diplomskega dela z naslovom:

Daljinsko upravljanje preko SMS z 8-bitnim mikrokrmilnikom

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. marca 2012 Podpis avtorja:

Kazalo

Seznam kratic

BCD - Binarno kodirana cifra desetiškega sistema (*Binary-Coded Decimal*).

EEPROM - Električno zbrisljiv in programirljiv bralni pomnilnik (*Electrically Erasable Programmable Read-Only Memory*).

EPROM - Zbrisljiv in programirljiv bralni pomnilnik (*Erasable Programmable Read-Only Memory*).

GPRS - Mobilna podatkovna storitev v okviru standarda GSM. (*General Packet Radio Service*).

GSM - Svetovni standard mobilnih komunikacij (*Global System for Mobile Communications*).

ISR - Prekinitveno servisna rutina (*Interrupt Service Routine*).

MIPS - Milijon ukazov na sekundo (*Million Instructions Per Second*).

MSB - Najpomembnejši Bajt (*Most Significant Byte*).

RAM - Bralno-pisalni pomnilnik z naključnim dostopom (*Random Access Memory*).

RISC - Računalnik z naborom omejenih ukazov (*Reduced Instruction Set Computer*).

ROM - Bralni pomnilnik (*Read-Only Memory*).

SMS - Kratko tekstovno sporočilo (*Short Message Service*).

Povzetek

V diplomski nalogi je predstavljen razvoj sistema za daljinsko upravljanje s SMS sporočili in njegove ključne komponente kot so krmilnik ATmega168A, GSM terminal GS64, pretvornik napetosti MAX232 in 1Wire senzor za temperaturo DS18B20. Opisani so cilji, ki jih komponente zadovoljujejo, navedeni pa so tudi argumenti za izbiro posameznih komponent skupaj z njihovim prednostmi in slabostmi. Podrobno je opisano razvojno okolje, tako strojna kot programska oprema. V vseh poglavjih smo poudarek postavili na opis tematike v splošnem, ki mu sledi razlaga korakov in implementacij, ki so se nam zdeli pomembnejši iz razlogov netrivialne rešitve ali morebitnih nepredvidenih problemov. Predstavljeni so različni protokoli npr. protokol 1Wire, serijski protokol RS-232, Hayesov protokol AT, protokol za upravljanje s senzorjem DS18B20. Z diagrami poteka, programsko kodo v zbirnem jeziku AVR ali besedno so opisane njihove implementacije v našem programu. Na koncu je naveden primer uporabe sistema v praksi in možnosti izboljšav sistema.

Abstract

This B.Sc. thesis revolves around the development of a system that provides remote control functionality for different electronic appliances over the short message service component of the mobile communication system(GSM). Individual components of the system like the ATmega168A microcontroller, GS64 GSM terminal, MAX232 signal converter and 1Wire temperature sensor DS18B20 are described. The reasons for the choice of these components are presented as well. Our development environment and hardware tools are described in detail. In all chapters a general description of the subject is followed by a detailed description of implementations and steps. Emphasis was put on nontrivial challenges or solutions that were introduced in order to solve an unexpected problem. Also different protocols are introduced. Our implementations of these protocols are explained through means of flow charts, AVR assembly code examples or a simple description. In the end we provide an example of how our system can be used in practice.

Poglavje 1

Uvod

Uporaba kratkih tekstovnih sporočil (SMS sporočil) je danes nekaj običajnega. Zaradi visoke podpore storitvi s strani mobilnih operaterjev in mobilnih telefonov predstavljajo SMS sporočila zelo priročen in poceni način komunikacije med uporabniki mobilnih omrežij. Taka sporočila se lahko poleg komunikacije med ljudmi uporabijo tudi za avtomatizacijo sistemov, predvsem pa so uporabna za upravljanje sistemov na daljavo. Na trgu je zaslediti nekaj rešitev, ki ponujajo upravljanje s SMS sporočili, vendar so cene takih rešitev za uporabo v gospodinjstvih relativno visoke. Poleg tega je njihova uporaba omejena na peščico primerov uporabe, ki jih je porizvajalec predvidel in ne omogočajo dodatnih razširitev ali nadgradenj. Zato smo se odločili razviti lastno rešitev za oddaljeno upravljanje preko SMS sporočil, ki bo cenovno dostopna, hkrati pa bo omogočala kasnejše nadgradnje oz. prilagotive bolj specifičnim primerom uporabe z nadgradnjo strojne in programske opreme.

Glavni komponenti sistema predstavljata GSM (svetovni standard mobilnih komunikacij) modul, ki pošilja in prejema SMS sporočila in 8-bitni mikrokrmilnik, ki procesira sporočila in upravlja GSM modul. Tak mikrokrmilnik predstavlja bistven prihranek pri stroških materiala za izdelavo. Na krmilnik smo priklopili temperaturne senzorje saj tej bistveno razširijo področje uporabe sistema, strošek pa je zanemarljiv. Sistem omogoča oddaljeni nadzor temperature in vžiga ali izklopa štirih stikal na katera lahko povežemo

poljubno zunanjo napravo. Krmilnik ima na voljo še 12 nepovezanih vhodno/izhodnih nogic, programska koda obsega 2000 vrstic AVR zbirnika kar zasede 20% programskega pomnilnika, uporablja 50% delovnega pomnilnika (RAM). Možnosti za razširitve periferije in programske kode je torej dovolj, kar je pomembno za nadaljne nadgradnje v prihodnosti.

Poglavje 2

Atmelov 8-bitni krmilnik

Pri izvedbi naloge nam je večinski delež časa vzelo programiranje mikrokrmilnika. Tega smo se zavedali že od začetka, zato je izbira arhitekture in modela krmilnika bila ena najpomembnejših odločitev v pripravah na nalogo. Krmilnik je moral biti cenovno dostopen, preprost za programiranje, dovolj zmogljiv za zahtevano nalogo in z obilico prosto dostopne dokumentacije. Upoštevajoč omenjene kriterije smo za najbolj primerne ocenili 8-bitne mikrokrmilnike iz serije AVR proizvajalca Atmel in 8-bitnike serije PIC proizvajalca Microchip. Odločili smo se za serijo AVR saj je ta bolj priljubljena in uživa večjo podporo na Fakulteti za računalništvo in informatiko in Fakulteti za elektrotehniko v Ljubljani. V primeru težav bi tako bilo lažje poiskati pomoč. Eden izmed krmilnikov, ki ustrezajo vsem naštetim pogojem je bil ATmega168A [1], zato smo ga pri izdelavi tudi uporabili.

2.1 AVR arhitektura

AVR označuje računalniško arhitekturo in serijo mikrokrmilnikov, zasnovanih v skladu z omenjeno arhitekturo. Gre za 8-bitne RISC mikrokrmilnike z modificirano Harvardsko arhitekturo. To pomeni, da je nabor ukazov omejen, in podatke pa sta ločena, vsak ima svoje vodilo in ločen naslovni prostor [1]. Posledično krmilnik za dostop do programskega pomnilnika uporablja

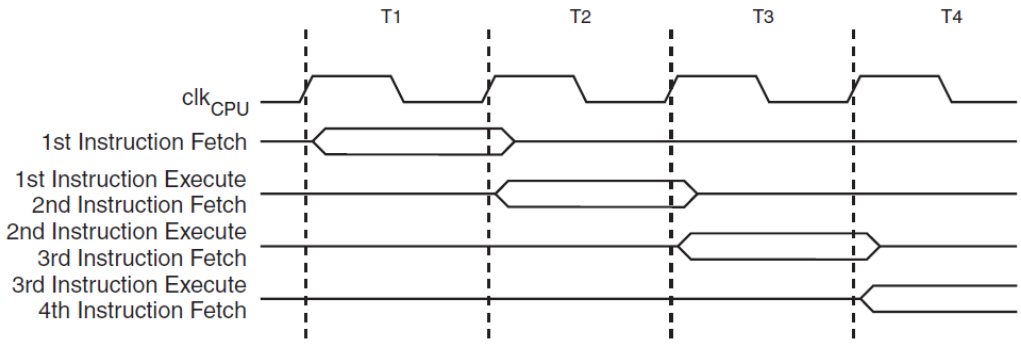
ločene ukaze. AVR platformo je podjetje Atmel razvilo leta 1996. Zanjo je značilno, da je bila ena prvih, ki je uporabljala vgrajen bliskovni spomin kot pomnilnik za program, kar je v nasprotju z ostalimi rešitvami kot so ROM (bralni pomnilnik), EPROM (zbrisljiv in programirljiv bralni pomnilnik) in EEPROM (električno zbrisljiv in programirljiv bralni pomnilnik), dovoljevalo večkratno programiranje krmilnika in poleg branja tudi pisanje v programski pomnilnik med izvajanjem samega programa.

Programski pomnilnik je torej bliskovni, omogoča večkratno pisanje in brisanje, vanj lahko posega centralna procesna enota med izvajanjem, zapisan program se ohrani brez električnega napajanja. Najmanjša enota, ki jo lahko naslovimo (beseda) je 8 bitov oz 1 Bajt. Kljub 8-bitni arhitekturi so ukazi mikrokrmilnika dolgi 16 bitov (nekateri tudi 32 bitov), naslavljanje istega pomnilnika je zato s stališča programskega števca organizirano v 16-bitne besede. V programu moramo biti pazljivi, če izvajamo dostope do programskega pomnilnika na podlagi vrednosti programskega števca in vrednost števca pretvoriti v naslov 8-bitne besede z množenjem z 2. Razširitev programskega pomnilnika z zunanjim ni mogoča.

Podatkovni pomnilnika je organiziran v 8-bitne besede. Naslovni prostor podatkovnega pomnilnika je razdeljen in naslavlja več komponent. Na začetek je prezrcaljenih 32 delovnih registrov, sledijo vhodno/izhodni registri in kontrolni registri nato pa notranji statični RAM pomnilnik (bralno-pisalni pomnilnik z naključnim dostopom). Nekateri krmilniki podpirajo tudi dodajanje zunanjega RAM pomnilnika do 64 KBajtov in imajo zanj rezervirane dodatne naslove na koncu naslovnega prostora (slika 2.3).

Večina krmilnikov ima vgrajen tudi notranji EEPROM pomnilnik, ki je namenjen za trajnejšo hrambo podatkov. Pomnilnik nima naslovnega prostora in vanj pišemo ali beremo, kot če bi bil zunanja naprava, preko posebnih naslovnih registrov in pisalno/bralnih ukazov. Tak dostop je bistveno počasnejši hkrati pa je tudi število pisalnih ciklov zaradi omejitev tehnologije omejeno na okoli 100 tisoč pisalnih ciklov.

Ukazi se izvršujejo preko dvostopenjskega cevovoda [1]. To pomeni, da



Slika 2.1: Prikaz dvostopenjskega cevovoda za izvrševanje ukazov

se med izvajanjem trenutnega že nalaga naslednji, kar je prikazano na sliki 2.1. Večina ukazov nad delovnimi registri traja 1 procesorski cikel (2 ob uporabi takojšnjih operandov), kar je v primerjavi z ostalimi 8-bitniki malo. Vejanje in bralni ali pisalni dostopi do podatkovnega pomnilnika trajajo dva cikla. AVR mikrokrmiliniki lahko torej dosežejo do 1 MIPS (milijon ukazov na sekundo) procesorske moči pri frekvenci 1 MHz. Krmilniki večinoma podpirajo frekvence v območjih od 1 do 10 MHz ali 1 do 20 MHz, obstajajo pa izvedbe, ki dosegajo tudi 32 MHz. Razlog v relativno velikih razponih frekvenc je v naraščanju električne porabe s frekvenco. Vse moderne izvedbe vsebujejo tudi notranji oscilator, ki generira procesorsko uro. Procesorsko uro lahko dodatno zmanjšamo z uporabo delilnika različnih vrednosti, tudi do 1024.

AVR krmilniki se delijo v tri osnovne družine:

- tinyAVR (ATtiny v imenu krmilnikov):
  - 0,5 do 8 kB programskega pomnilnika,
  - 6 do 32 nogic,
  - zmanjšan nabor periferije.
- megaAVR (ATmega v imenu krmilnikov):
  - 4 do 256 kB programskega pomnilnika,

  - 28 do 100 nogic,
  - dodatni procesorki ukazi za podporo množenju in delu z večjimi programskimi pomnilniki,
  - razširjen nabor periferije.

- XMEGA (ATxmega v imenu krmilnikov):
  - 16 do 384 kB programskega pomnilnika,
  - 44, 64 ali 100 nogic,
  - največji nabor periferije (vključno z digitalno/analognimi pretvorniki).

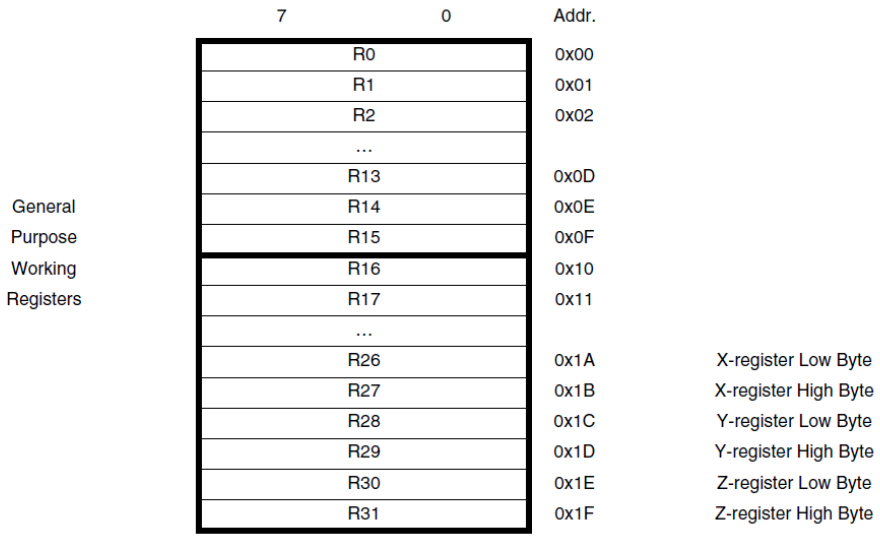
2.2 ATmega168A

Kot že omenjeno, smo za razvoj našega projekta uporabili mikrokrmilnik ATmega168A. Ker pripada družini megaAVR krmilnikov (kar je razvidno že iz imena), zanj veljajo vse v poglavju 2.1, ki se prične na strani 3, navedene lasnosti AVR arhitekture in megaAVR družine. V tem poglavju bodo opisane podrobnejše lastnosti tega krmilnika.

2.2.1 Delovni registri

ATmega168A ima 32 8-bitnih delovnih registrov (slika 2.2), ki jih označujemo s črko r in zaporedno številko registra: od r0 do r31. Namenjeni so splošni uporabi pri izvajanju programa. To pomeni, da se uporabljajo za shranjevanje operandov, rezultatov, vmesnih rezultatov, spremenljivk in tako naprej. Poleg osnovnega nabora ukazov, ki je podprt pri vseh registrih, posamezni skupki registrov podpirajo še dodatne nabore ukazov za specifične operacije:

- Register r0 je privzeti ciljni register za določene operacije, ki vračajo 8-bitni rezultat (npr. LPM - *load program memory*, ki naloži 1 Bajt iz programskega pomnilnika v r0)



Slika 2.2: Diagram 32 delovnih registrov

- Par registrov r1:r0 je privzeti par ciljnih registrov za različne operacije množenja, ki vračajo 16-bitni rezultat (npr. MUL - *multiply unsigned*, ki množi dva registra)

- Gornjih 16 registrov (r16 do r31) podpira operacije v kombinaciji s takojšnjimi operandi (npr. LDI - *load immediate*, ki v ciljni register naloži 1 Bajt iz naslova, zapisanega v konstanti kot kaže programska koda 2.1)

- Najvišji štirje pari registrov (r24 do r31), to so pari r25:r24, r27:r26 (imenovan tudi X), r29:r28 (imenovan tudi Y), r31:r30 (imenovan tudi Z) podpirajo 16-bitne operacije (npr. ADIW *add immediate to word*, ki 16-bitni vrednosti v paru registrov prišteje neko konstanto

- Najvišji trije pari registrov (r26 do r31) so 16-bitni registri X, Y in Z. Ti se lahko uporabljajo v navezi z ukazi, ki berejo in pišejo v naslovni prostor podatkovnega pomnilnika, kjer predstavljajo kazalec na naslov. Priročni so za indeksno naslavljanje, kjer lahko v posamezni iteraciji z

enim ukazom preberemo podatek in povečamo kazalec, kar predstavlja dodatno pohitritev izvajanja. Primera takih ukazov sta ukaza LD in ST.

- Najvišji par registrov r31:r30 (16-bitni register Z) se uporablja za dostop do programskega pomnilnika, kjer register Z predstavlja kazalec na programski pomnilnik. Ukaza, ki se v tem primeru uporabljata sta LPM in SPM.

Programska koda 2.1: Primer uporabe konstante 0x0111 v ukazu

```
1 ldi r16, 0x0111 ;prenesi 1 Bajt iz 0x111 v r16
```

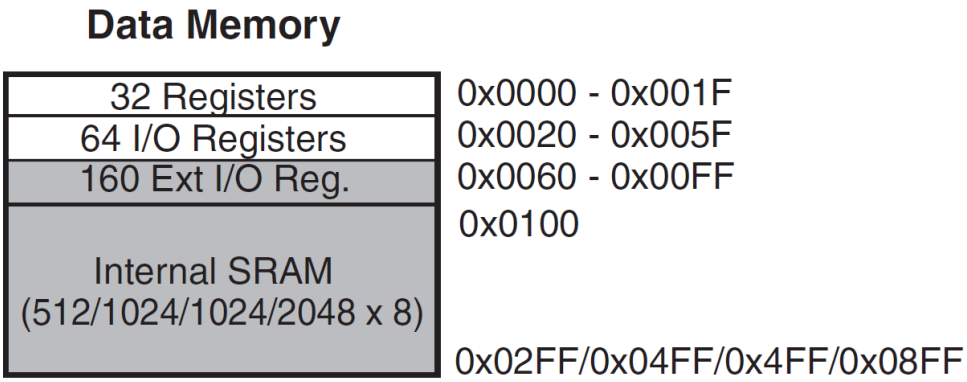
Čeprav gre za 8-bitni krmilnik, so določene operacije tudi 16-bitne. V primerih, kjer se pari registrov uporabljajo kot 16-bitni register, sta Bajta razporejena po principu tankega konca (little endian) kar pomeni, da je v višjem registru zapisan pomembnejši Bajt - *most significant Byte* - MSB).

2.2.2 Podatkovni pomnilnik

Naslovni prostor krmilnika ATmega168A naslavlja več komponent. Kot je razvidno is slike 2.3, prvih 32 naslovov predstavlja delovne registre, sledijo vhodno/izhodni in kontrolni registri, ki zasedajo nadaljnih 224 naslovov, nato pa se na heksadecimalnem naslovu 100 prične statični ram.

V začetnem območju, ki predstavlja delovne registre lahko dostopamo do vsakega bita posebej. Za nastavljanje bitov uporabimo ukaza SBI in CBI, za preverjanje vrednosti bita pa SBIS in SBIC.

Skupek vhodno/izhodnih in kontrolnih registrov je preslikan na heksadecimalne naslove 20 do FF. Tukaj nastavljamo delovanje krmilnika vključno z upravljanjem celotne vgrajene periferije in pisanjem oz. branjem na zunanje naprave. Omenjeni prostor se deli na dve sekciji v katerih veljata različna režima naslavlja in dostopanja. Do registrov v obeh sekcijah lahko dostopamo z ukazi, ki veljajo za celoten naslovni prostor podatkovnega pomnilnika: ST, STS, STD, LD, LDS, LDD. Za prvo sekcijo, ki predstavlja prvih 64 registrov,



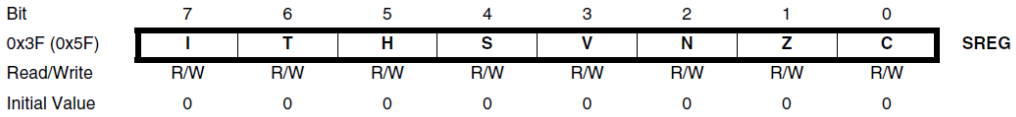
Slika 2.3: Naslovni prostor podatkovnega pomnilnika pri ATmega168A

pa za dostop in pisanje lahko uporabimo tudi ukaza za delo z vhodno/izhodnimi registri: IN in OUT, ki sta krajša in se hitreje izvedeta. Drugo sekcijo predstavlja dodatnih 160 registrov pri katerih pa nazadnje omenjenih ukazov ne moremo uporabiti.

Za indeksno naslavljanje podatkovnega pomnilnika se uporabljajo registri X, Y in Z.

2.2.3 Statusni register

Statusni register (*status register*, na kratko SREG) je najpomembnejši register krmilnika. Operacije v centralni procesni enoti spreminjajo vrednosti bitov v statusnem registru. Vrednosti statusnega registra ne obravnavamo kot celoto, ampak vsak bit predstavlja svojo nastavitev ali označitev. Takim bitom zato pravimo tudi zastavice. Ker gre za najpomembnejše zastavice procesne enote, ima vsaka za sinonim izbrano črko. Črke so naštete v nadaljevanju. Preko omenjenih zastavic je zasnovano delovanje osnovne logike procesne enote: gre za nastavljanje zastavic ob izvajanju aritmetičnih in logičnih operacij. Na podlagi njihovih vrednosti centralna procesna enota ob različnih ukazih za vejanje odloči, katera veja programa se bo izvedla. S kombinacijami operacij in ukazov za vejanje dobimo mehanizme za vpliv na



Slika 2.4: Statusni register AVR krmilnika

potek programa.

Statusni register sestavljajo (razpored bitov je prikazan tudi na sliki 2.4):

- Bit 7 predstavlja zastavico I, ki se včasih pojavlja tudi kot kratica GIE (*Global Interrupt Enable*). Kot že angleško ime pove, gre za zastavico, ki globalno omogoči ali onemogoči proženje prekinitev. Več o prekinitvah je napisano v poglavju 5.3. Če je zastavica postavljena (ima vrednost 1) potem so prekinitve omogočene, če pa je vrednost zastavice 0, potem procesor prekinitev ne bo prožil. Za nastavljanje zastavice se uporabljata ukaza SEI in CLI.

- Bit 6 predstavlja zastavico T. T se uprablja kot ciljni bit pri operacijah, ki operirajo z enim samim bitom. Ukaza BLD in BST kopirata vrednost iz željenega bita v delovnem registru v zastavico T in obratno.

- Bit 5 predstavlja zastavico H (Half-Carry), ki se uporablja pri 4-bitnih operacijah nad desetiškimi števili v BCD formatu.

- Bit 4 predstavlja zastavico S, ki se izračuna kot ekskluzivni ali (XOR) med zastavicama N in V. Uporablja se pri primerjavi predznačenih števil.

- Bit 3 predstavlja zastavico V, ta nam pove ali je prišlo do preliva dvojiškega komplementa in spremembe predznaka (ob seštevanju enako predznačenih števil dobimo drugačen predznak in napačen rezultat, ki je posledica pomanjkanja bitov). Če seštevamo dve različno predznačeni števili do preliva ne more priti.

- Bit 2 predstavlja zastavico N, ta nam pove ali je najpomembnejši bit (bit 7) v rezultatu operacije 1 ali 0.

- Bit 1 predstavlja zastavico Z, ta nam pove ali je rezultat operacije vrednost 0, skupaj z operacijo odštevanja se uporablja za ugotavljanje enakosti dveh števil.

- Bit 0 predstavlja zastavico C, ta nam pove ali je pri seštevanju prišlo do prenosa na bit 8 (vsota je prevelika za 8-bitni zapis, zastavica predstavlja 9. bit).

2.2.4 Varovalke

Varovalke (*fuses*) so posamezni biti v treh Bajtih (Low, High, Extended) v permanentnem pomnilniku, neodvisnem od električnega napajanja in z neomejenim številom prepisov. Varovalke vplivajo na delovanje mikrokrmilnika, z njimi določamo stvari kot so:

- frekvenca procesorke ure,

- električna napetost,

- vir procesorske ure (zunanji ali notranji),

- tip zunanjega oscilatorja (npr. kvarčni kristal ali keramični oscilator),

- delilnik procesorke ure,

- vklop/izklop debugWIRE razhroščevanja,

- ohranitev EEPROM spomina ob reprogramiranju,

- vklop/izklop reset nogice,

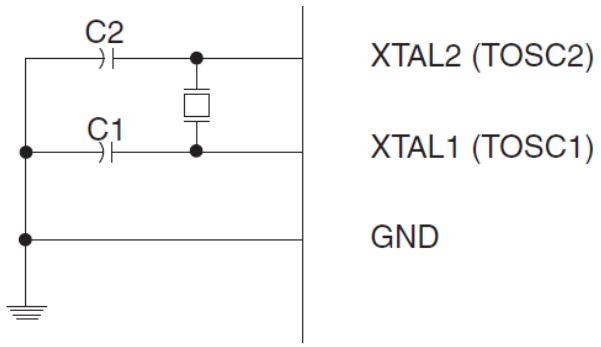
- vklop/izklop SPI načina.

Spreminjanje varovalk poteka preko vmesnika v AVR Studiu, ob nastavljanju pa smo si pomagali s kalkulatorjem varovalk, ki je v različnih izvedbah prosto dostopen na spletu. Primer takega kalkulatorja je dostopen na naslovu http://www.engbedded.com/fusecalc/. Spremenili smo varovalke v najnižjem Bajtu. Tovarniška vrednost je bila 0x62, postavili smo jo na 0xF7. S tem smo izklopili delilnik procesorske ure in vklopili uporabo zunanjega kvarčnega kristala kot vira procesorske ure. Zunanji oscilator frekvence 18,43MHz smo uporabili iz dveh razlogov:

- frekvenca procesorja vpliva na serijsko komunikacijo, ki je realizirana na strojnem nivoju, navedena frekvenca omogoča teoretično točno komunikacijo, brez napak,
- uporaba rahroščevalnega načina debugWIRE z uporabo vgrajenega oscilatorja ni mogoča.

Zunanji kvarčni oscilator smo s krmilnikom povezali kot kaže slika 2.5.

6. bit srednjega Bajta je varovalka DWEN, ki vklaplja debugWIRE razhroščevanje. Varovalko vklopimo z zagonom debugWIRE razhroščevanja v AVR Studiu pri tem pa moramo zagotoviti, da krmilnik uporablja zunanji oscilator in uporabljamo SPI programiranje. Če prvi pogoj ni izpolnjen in varovalko DWEN vklopimo, bo krmilnik nehal delovati, saj ne bo imel vira procesorske ure in ga bo potrebno prevezati po shemi za paralelno programiranje (slika A.2 v prilogi A).



Slika 2.5: Shema vezave kvarčnega oscilatorja

Poglavje 3

Razvojno okolje

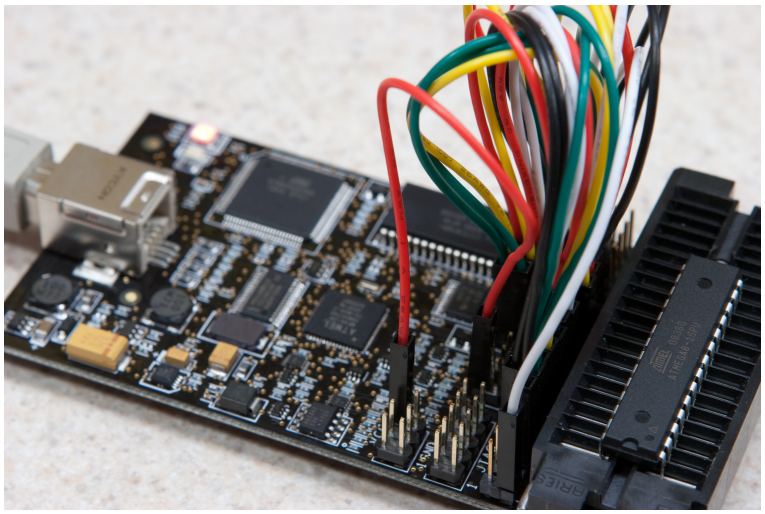
Za programiranje ATmega168A sta potrebna zbirnik AVR zbirnega jezika in programator. Za prvo nalogo smo uporabili Atmelov AVR Studio [12], ki je priporočeno razvojno orodje za AVR mikrokrmilnike in združuje urejevalnik kode, zbirnik, prevajalnik in preprocesor. Za programator smo uporabili Atmelov AVR Dragon [4] saj podpira razhroščevanje preko protokola debugWIRE in programiranje v ISP ali PP načinu.

3.1 Strojna oprema

Strojno opremo, ki je bila uporabljena pri razvoju sistema za oddaljeni nadzor preko SMS sporočil, predstavljajo programator AVR Dragon in dva adapterja RS-232 , ki se priklopita v USB vodilo. Strojno opremo, ki je del same rešitve pa sestavljajo GSM terminal GS64 [7], ATmega168A, MAX232 [**?**] pretvornik nivojev električnih signalov, dva DS18B20 [**?**, 6]enzorja za temperaturo in ostali električni elementi.

3.1.1 AVR Dragon

AVR Dragon je programator zadnje generacije, ki podpira širok nabor krmilnikov serije AVR. Prednost pred ostalimi programatorji s podobno ceno ima v:



Slika 3.1: AVR Dragon z ZIF podnožjem in nogicami za povezavo krmilnika z zunanjimi napravami in Dragonom

- podpori paralalnemu programiranju (PP) [4], ki v nasprotju z načinom ISP, omogoča preprogramiranje narobe nastavljenih varovalk in tako zelo olajša reševanje omenjenega problema,
- podpori razhroščevalnega načina debugWIRE [4], tj. razhroščevanja na samem čipu (*on-chip debugging*) in je zaradi interakcije z zunanjo napravo (GSM terminal) nepogrešljiv,
- programske posodobitve samega programatorja so enostavne, saj se izvedejo avtomatsko preko programa AVR Studio

Treba je poudariti, da paralelno programiranje ni nadomestek za ISP, ker ne omogoča programiranja krmilnika, ki je priklopljen v vezje. Zato smo se ga poslužili samo ob reševanju narobe sprogramiranih varovalk, v splošnem pa smo uporabljali ISP način, ki omogoča programiranje v vezje umeščenega krmilnika in debugWIRE protokol. Shemi vezave za PP in ISP način programiranja sta priloženi v prilogi A na slikah A.1 in A.2.

Programator smo uporabili kot preprosto razvojno ploščo, pri tem smo na ploščico prispajkali ZIF (*Zero Insertion Force*) podnožje, ki omogoča pre-

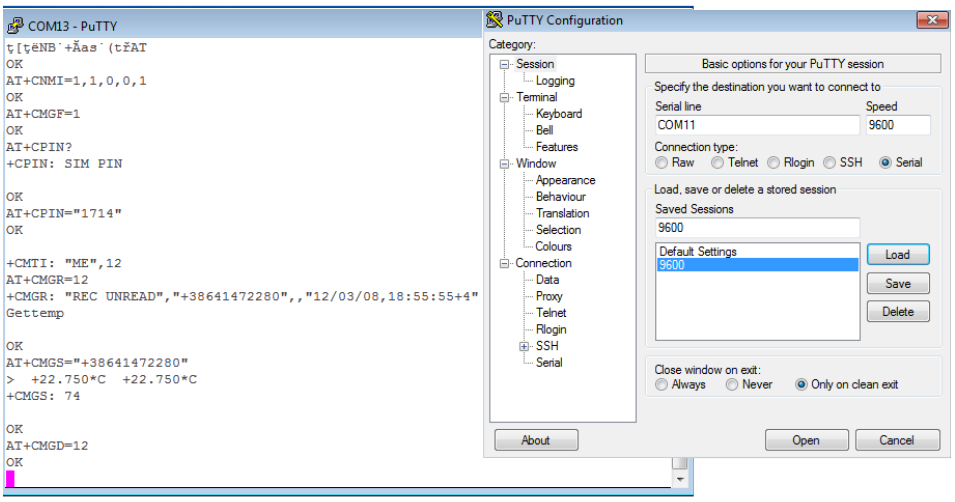
Slika 3.2: Prisluškovanje krmilniku in terminalu z dvema RS-232 adapterjema

prosto nameščanje in odstranitev čipa ter vertikalne nogice z namenom povezovanja krmilnika z zunanjo periferijo. Oboje je prikazano na sliki 3.1.

3.1.2 RS-232 adapter

Upravljanje z GSM terminalom GS64 poteka preko serijske povezave preko standarda RS-232. Terminal ima za to namenjen DE-9 konektor. Za povezavo terminala z namiznim računalnikom smo uporabili RS-232 adapter, ki se priklopi v USB vodilo. Za komunikacijo smo uporabili program Putty [10], kjer smo nastavili ustrezen COM port (gre za operacijski sistem Windows) in hitrost prenosa simbolov (*baud rate*) 9600 (enako hitrost smo kasneje uporabili tudi na mikrokrmilniku), ostale nastavitve smo pustil na privzetih vrednostih. Nastavitve programa Putty prikazuje tudi slika 3.3. Ker GSM terminal podpira avtomatsko detekcijo hitrosti prenosa simbolov je komunikacija stekla brez težav.

V kasnejši fazi razvoja, smo uporabili dva RS-232 adapterja za prisluškovanje komunikaciji med krmilnikom in terminalom. Pri tem smo tx nogico termi-



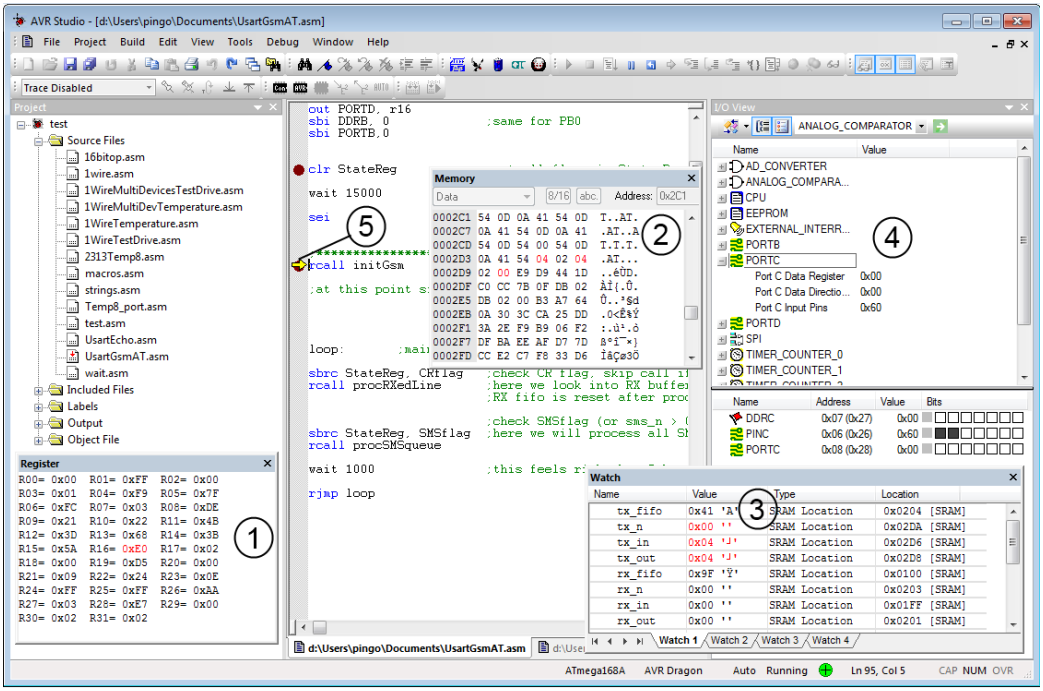
Slika 3.3: Serijska komunikacija preko programa Putty, vključno z nastavitvami

nala povezali z rx nogico na 1. adapterju, tx nogico iz MAX232 pretvornika pa na rx nogico 2. adapterja. Adapterja sta prikazana na sliki 3.2. Tako smo v dveh oknih programa Putty spremljali pošiljanje podatkov v obe smeri.

3.2 Programska oprema

Programsko opremo razvojnega okolja je predstavljal predvsem AVR Studio. Ker protokol debugWIRE v navezi s programatorjem AVR Dragon ne deluje v AVR Studio 5, je bilo potrebno uporabiti verzijo 4. Za uspešno migracijo na nižjo verzijo je potrebna nadgradnja programja na programatorju v nižjo različico *downgrade*), kar AVR Studio 4 izvede samodejno. AVR Studio vsebuje preprocesor, zbirnik za AVR zbirni jezik, urejevalnik kode in razhroščevalnik z grafičnim vmesnikom za spremljanje vrednosti podatkovnega pomnilnika in možnostjo določitve ustavitvenih točk (*breakpoint*). Na sliki 3.4 je prikazan uporabniški vmesnik programa AVR Studio 4 med razhroščevalno sejo. Številka 1 označuje okno za spremljanje vrednosti delovnih registrov, številka 2 okno za vpogled v vsebino podatkovnega pomnilnika, številka 3 okno za spremljanje vrednosti v podatkovnem pomnilniku na fiksnih lokacijah (spremenljivk), številka 4 okno z vrednostmi vhodno/izhodnih registrov, številka 5 pa označuje rumeno puščico in rdečo piko. Rumena puščica nam pove kateri ukaz se trenutno nahaja v programskem števcu, rdeča pika pa ozačuje točko ustavitve, kjer naj razhroščevalni način ustavi izvajanje programa in nam ponudi vpogled v aktualno stanje v krmilniku. Po izvedbi posameznega ukaza, nam uporabniški vmesnik olajša ugotavljanje sprememb tako, da spremenjene vrednosti v registrih ali pomnilniku obarva z rdečo barvo.

Za spremljanje serijske komunikacije smo uporabili program Putty, nastavitve za povezavo so opisane na strani 18 na sliki 3.3.



Slika 3.4: Uporabniški vmesnik razvojnega okolja AVR Studio 4

Poglavje 4

Merjenje temperature

Za meritve temperatur smo uporabili senzor DS18B20. Za omenjeni senzor smo se odločil iz več razlogov:

- maksimalna napaka temperature je 0,5°C,
- na skupno vodilo lahko priklopimo več senzorjev,
- podatkovno vodilo temelji na protokolu, ki za komunikacijo potrebuje samo 1 žico,
- za senzorje in protokol, ki ga uporabljajo je na voljo veliko dokumentacije, saj so zelo razširjeni.

4.1 Tehnologija 1Wire

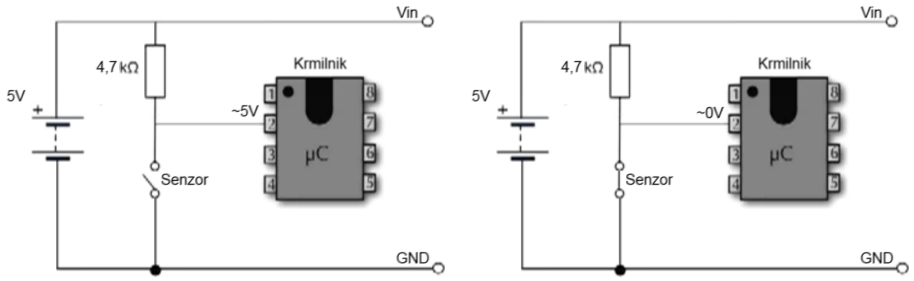
4.1.1 Opis tehnologije

1Wire [13] je ime za tehnologijo, ki omogoča komunikacijo med dvema napravama preko podatkovnega vodila v obliki ene same žice. Tehnologijo je razvila korporacija Dallas Semiconductor. Za komunikacijo preko 1Wire vodila se uporablja 1Wire protokol. Ta loči naprave na 1 gospodarja (*master*) in 1 ali več sužnjev (*slave*). Gospodar je naprava, ki prične komunikacijo na vodilu, sužnji pa lahko odgovarjajo le v primeru, ko jim gospodar to ukaže s

primernim ukazom. Vlogo gospodarja ima običajno krmilnik, kar je veljalo tudi v našem primeru. Suženj je običajno nek senzor, ki ob pozivu gospodarja sporoči željeni podatek. Cilj tehnologije je uporaba čim manj žic za komunikacijo med dvema ali več napravami. Posledično naprava suženj za delovanje potrebuje samo tri ali dve žici za delovanje. Dve žici sta namenjeni napajanju naprave (plus in minus), tretja pa predstavlja podatkovno vodilo. Preproste senzorje je mogoče povezati tudi v parazitnem načinu (*parasite power* [6]). To pomeni, da se senzor napaja kar preko podatkovnega vodila. Senzor v parazitnem načinu deluje tako, da se napaja iz vodila, ko je to v visokem stanju (stanje pozitivne napetosti). To je privzeto stanje vodila in je prisotno tudi ob mirovanju vodila. V takem stanju je električni tokokrog, ki napaja senzor, sklenjen preko podatkovnega vodila. V režimu pozitivne napetosti senzor tudi polni notranji kondenzator. Ta je uporabljen kot vir napetosti kasneje, v primeru aktivnosti na vodilu, ko ob nizkem stanju vodila vir napetosti usahne. Iz tega izhaja, da senzorji ne smejo imeti prevelike elekrične porabe saj bi to privedlo do prehitre izpraznitve kondenzatorja oz. zelo dolgih polnilnih ciklov.

Privzeto stanje vodila je visoko. Naprave s spuščanjem nivoja napetosti na vodilu v nizko stanje signalizirajo različne stvari. Privzeto visoko stanje je doseženo tako, da je na vodilu pred napravami vezan dvižni upor (*pull-up resistor*), ki poskrbi za dovod napetosti do naprav preko vodila hkrati pa omejuje pretok in prepreči kratek stik v primeru nizkega stanja vodila. Nizko stanje vodila je doseženo, če katera od naprav mostiči vodilo in maso, to povzroči padec napetosti, ki ga zaznajo vse povezane naprave. Mehanizem dviganja in spuščanja je nazornejše prikazan na sliki 4.1. Na sliki je označeno stikalo senzorja, ki dviga in spušča napetost na vodilu in krmilnik, ki zaznava in interpretira vhodno napetost. Komunikacija v obratno smer je realizirala na enak način, le vlogi senzorja in krmilnika na sliki je potrebno zamenjati. 1Wire protokol določa 6 različnih signalov:

- pulz za ponastavitev (*reset pulse*),
- prisotnostni pulz (*presence pulse*),



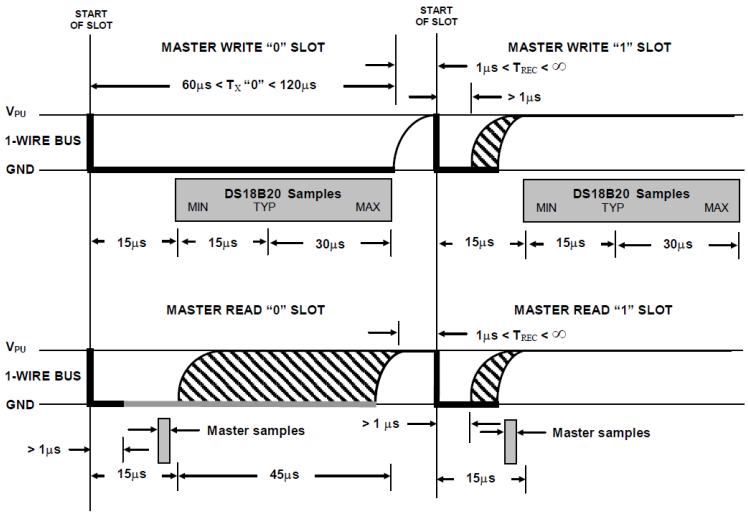
Slika 4.1: Visoko stanje vodila na levi in nizko stanje na desni

- zapiši 0,
- zapiši 1,
- preberi 0,
- preberi 1 [5].

Z izjemo prisotnostnega pulza vse signale sproži gospodar.

Če želi gospodar komunicirati z eno od naprav mora izvesti postopek inicializacije. To stori tako, da vodilo povleče na nižjo napetost za vsaj 480μs nato pa vodilo sprosti in prične poslušati za morebitni odgovor. Ker je vodilo sproščeno, napetost preko dvižnega upora prične naraščati. Ko suženj zazna naraščujočo strmino napetosti (*rising edge*), počaka 15μs do 60μs, nato pa sproži prisotnostni pulz tako, da napetost na vodilu povleče navzdol za 60μs do 240μs.

Pisanje in branje bitov je realizirano z bralno - pisalnimi časovnimi okvirji (*time slot*), ki so dolgi 60μs ali več, med njimi pa mora biti vsaj 1μs premora. Pisalni časovni okvir traja 60μs in se prične, ko gospodar povleče napetost na vodilu navzdol. Če želi zapisati vrednost 1, mora vodilo sprostiti v roku 15μs, za zapis vrednosti 0 pa vodilo drži v nizkem stanuju celoten pisalni časovni okvir - 60μs (lahko več). Suženj vrednosti bere tako, da ko zazna pričetek pisalnega časovnega okvirja (vodilo gre v nizko stanje), počaka 15μs nato pa interpretira signal kot se pojavlja med 15. in 60. μs pisalnega

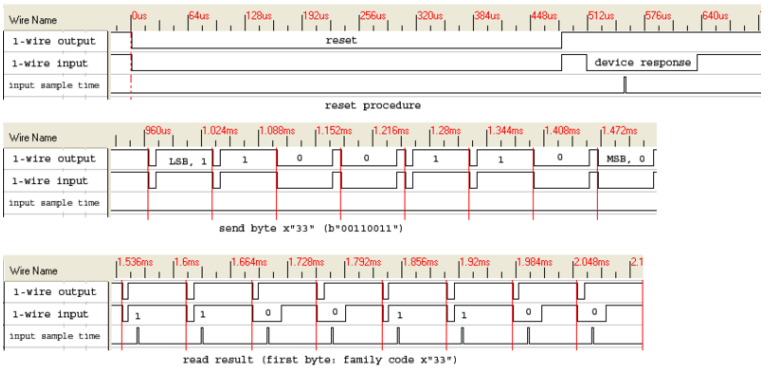


Slika 4.2: Tipični časovni okvirji protokola 1Wire

časovnega okvirja. V tem obdobju suženj naredi več meritev in po potečeni 60. μs interpretira signal v vrednost 0 ali 1 [6]. Večkratne meritve zmanjšajo možnost napačne interpretacije signala.

Tudi branje sproži samo gospodar. Suženj lahko gospodarju pošilja podatke samo v primeru, da gospodar prične bralni časovni okvir. Bralni časovni okvir traja vsaj 60μs, med branji pa mora biti vsaj 1μs premora. Gospodar signalizira bralni časovni okvir tako, da napetost na vodilu povleče nizko, nato pa vodilo takoj sprosti. Ko suženj zazna dvig napetosti, pošlje vrednost 1 tako, da ne naredi nič (napetost vodila pusti v visokem stanju) v celotnem preostanku bralnega časovnega ovkirja (blizu 60μs). Vrednost 0 pošlje tako, da napetost na vodilu povleče navzdol vsaj za 15μs, nato pa mora pred potekom okvirja vodilo nazaj sprostiti. Če hoče gospodar pravilno interpretirani poslane vrednosti, mora opraviti meritve v roku 15μs po signalizaciji bralnega časovnega okvirja. Ker odzivnost sužnja ni instantna, gospodar opravi več meritev na intervalu 15μs in jih interpretira v signal.

Napisano je predstavljeno tudi na sliki 4.2 kjer so skicirani tipični časovni okvirji pisanja in branja pri senzorju DS18B20. Odebeljena črna črta predstavlja nizko napetost vodila, ki jo povzroča gospodar. Odebeljena siva črta



Slika 4.3: Posnetki signalizacije med gospodarjem in sužnjem s pomočjo logičnega analizatorja

predstavlja sužnja, ki vleče napetost navzdol. Tanka črta predstavlja rast napetosti in visoko stanje napetosti na vodilu, posledico sprostitve vodila in dvižnega upora.

Pisanje ali branje več bitov oz Bajtov hkrati izvedemo z zaporednim nizanjem bralnih ali pisalnih časovnih okvirjev, kot je to prikazano na sliki 4.3. Tak mehanizem komunikacije omogoča prenos podatkov s hitrostjo do 16,3 kbit/s (glede na podatke v tehničnih specifikacijah na naslovu http://datasheets.maxim-ic.com/en/ds/DS2450.pdf).

4.1.2 Implementacija 1Wire na krmilniku

Krmilnik ATmega168A sicer ponuja strojno podporo za komunikacijo preko protokola 1Wire s pomočjo vgrajene komponenete USART (univerzalni asinhroni oddajnik in sprejemnik), vendar smo to komponento uporabili za komunikacijo z GSM terminalom prek vodila RS-232. Protokol 1Wire smo implementirali sami z uporabo ene od generičnih vhodno/izhodnih nogic krmilnika, ki je prevzela vlogo gospodarja 1Wire vodila. Za uspešno generiranje signalov je bilo potrebno pravilno nastaviti ustrezne bite v kontrolnih registrih, ki vplivajo na delovanje nogice.

Vhodno/izhodne nogice nastavljamo z uporabo vhodno/izhodnih regi-

Tabela 4.1: Nabor nastavitev za vhodno/izhodno nogico z zaporedno številko x

	DDRx = 0	DDRx = 1
PORTx = 0	Branje v PINx (nogica je v stanju visoke impendance - notr. dvižni upor ni povezan)	Izhod 0 (masa)
PORTx = 1	Branje v PINx (nogica je povezana preko notranjega dvižnega upora)	Izhod 1 (+5V)

strov DDRB, DDRC, DDRD, PORTB, PORTC, PORTD, nabor nastavitev je prikazan v tabeli 4.1. Vhodne vrednosti na nogici pa nastavimo preko registrov PINB, PINC in PIND. DDRx registri nastavljajo ali se nogica uporabi kot vhod ali izhod. PORTx registri nastavljajo izhodne vrednosti nogice, preko registrov PINx pa beremo vhodne vrednosti na nogici.

Za nastavljanje napetosti na vodilu smo preklapljali med dvema nastavitvama nogice. Vodilo mora biti na privzeti vrednosti visokega stanja, ki ga lahko po potrebi vsaka naprava spremeni (napetost povleče navzdol). Da bi to dosegli, bi potrebovali dvižni upor, katerega imajo vse nogice v notranjosti čipa tudi vgrajenega in je priklopljen pri nastavitvah DDRx=0 in PORTx=1. Taka nastavitev nogice načeloma omogoča branje vhodnih vrednosti, vendar v našem primeru integriran dvižni upor ni zadostoval. Zato smo za branje uporabili nastavitev DDRx=0, PORTx=0, kar je nogico postavilo v posebno stanje visoke impendance (*tristate* oz. *Hi-Z state*) in uporabili zunanji dvižni upor, ki ustreza zahtevam v dokumentaciji temperaturnega senzorja (4,7 kΩ).

Če povzamemo - PORTx smo ob inicializaciji fiksno nastavili na 0, stanje vodila smo brali preko PINx. Ob signaliziranju nam za visoko stanje vodila ni bilo potrebno storiti ničesar, saj je za to poskrbel dvižni upor, za nizko stanje pa smo DDRx postavili na 1 in na vodilo mostičili maso.

Za uspešno implementacijo singaliziranja vodilu s preklapljanjem nastavitev nogice (*bit banging*) je bistvena časovna usklajenost s časovnimi okvirji protokola 1Wire. Za to smo poskrbeli s posebnimi rutinami, ki zapravijo točno preračunano količino časa. V primeru programske kode 4.1 iz našega programa je zapisana rutina, ki poskrbi za inicializacijo vodila s signalom za ponastavitev in preverjanjem prisotnosti naprav na vodilu.

V vrsticah 2 in 3 v register X naložimo vrednost DVUS(480). DVUS(x)

je preprocesorski makro, definiran s formulo 4.1 Makro je zastavljen tako, da izračuna potrebno število iteracij, ki jih je potrebno izvesti v rutini Wait4xCycles(ta zapravi štiri cikle na iteracijo), da bo procesor zapravil x μs časa.

Formula 4.1

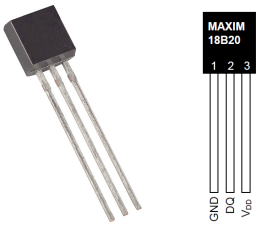
$$DVUS(x) = 18432000/100000/4 * x \tag{4.1}$$

V vrstici 4 nastavimo bit 5(ker uporabljamo 5. nogico) v registru DDRC na 1, kar povleče napetost vodila navzdol. Klic rutine v 5. vrstici sproži čakanje v obsegu 480μs. V 8. vrstici bit 5 v DDRC nastavimo na 0, kar sprosti vodilo v visoko stanje. Sledi čakanje 70μs. V 10. vrstici nastavimo zastavico T, sledi preverjanje vodila. Če je vodilo v visokem stanju, bo ukaz clt izpuščen, v nasprotnem primeru pa bo clt zastavico T postavil nazaj na 0. Dodatno čakanje poskrbi na koncu poskrbi, da se sužnji lahko pripravijo na naslednji signal. Določeni intervali čakanja, ki smo jih uporabili v naši implementaciji 1Wire komunikacije, odstopajo od navedenih v poglavju 4.1.1. V programu smo uporabili vrednosti, priporočene v priročniku, ki je med literaturo zaveden pod zaporedno številko 5.

Po podobnem kopitu smo napisali tudi rutine za pisanje in branje bitov z vodila. Napisali smo še rutini za zapis in branje 1 Bajta podatkov.

Programska koda 4.1: Primer uporabe konstante 0x0111 v ukazu

```
1  OWReset:
2  ldi             XH, HIGH(DVUS(480))
3  ldi             XL, LOW(DVUS(480))
4  sbi             DDRC, 5
5  rcall   Wait4xCycles
6  ldi             XH, HIGH(DVUS(70))
7  ldi             XL, LOW(DVUS(70))
8  cbi             DDRC, 5
9  rcall   Wait4xCycles
10 set
11 sbis    PINC, 5
12 clt
13 ldi             XH, HIGH(DVUS(410))
```



Slika 4.4: Senzor DS18B20

```
14 ldi              XL, LOW(DVUS(410))
15 rcall    Wait4xCycles
16 ret
```

4.2 Temperaturni senzor DS18B20

Temperaturni senzor DS18B20 je proizvod korporacije Maxim Integrated Products. Merilno območje senzorja je od -55°C do +125°C z natančnostjo ±0.5°C. Vsebuje tudi dodatne funkcije kot je proženje alarma previsoke ali prenizke temperature.

Za komunikacijo uporablja 1Wire tehnologijo (kot suženj), na 1Wire vodilo pa ga lahko priklopimo tudi v parazitnem načinu, ki je razložen v poglavju 4.1.1. Ima tri nogice, ki so označene na sliki 4.4. Prva in tretja sta namenjeni napajanju, sredinska pa priklopu na vodilo. V primeru parazitne vezave, je potrebno nogici za plus in minus zvezati z žico, ki predstavlja minus v tokokrogu. Vsak senzor ima unikatno 64-bitno serijsko kodo, ki omogoča naslavljanje več senzorjev na skupnem vodilu. V našem sistemu smo na skupno vodilo priklopili dva senzorja, po potrebi pa se na vodilo lahko priklopi tud več senzorjev. V tem primeru je potrebno v programu dodati 64-bitne identifikacijske kode dodanih senzorjev.

Senzor upravljamo z uporabo različnih ukazov, ukaz pošljemo tako, da na vodilo zapišemo Bajt z ustrezno vrednostjo. Za uspešno meritev temperature v senzorju najprej sprožimo meritev temperature in konverzijo v digitalni zapis. To storimo z naslednjim postopkom:

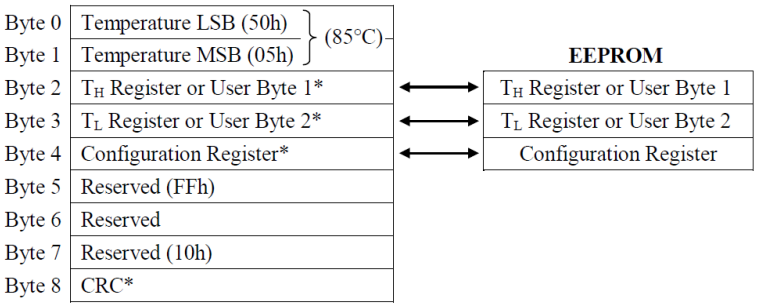
1. inicializacija vodila (ponastavitveni pulz, pri tem preverimo še prisotnost naprave),
2. ukaz "Match rom command"(na vodilo zapišemo vrednost 0x55),
3. na vodilo zapišemo 8 Bajtno identifikacijsko kodo senzorja, ki ga želimo nasloviti,
4. ukaz C̈onvert Temperature"(na vodilo zapišemo vrednost 0x44).

Nobenega ukaza ne smemo izpustiti, izvršeni pa morajo biti v navedenem zaporedju. Po izvedbi zgornjega zaporedja se v senzorju prične merjenje in konverzija temperature v digitalni zapis. Senzor lahko temperaturo zapiše z od 9 do 12-bitnim zapisom. 12-bitni zapis omogoča večjo resolucijo vendar to pomeni daljšo konverzijo temperature. Če od senzorja zahtevamo temperaturo, zapisano z 9 biti, bo konverzija trajala do 93.75ms, pri 12-bitnem zapisu pa do 750ms. Zaradi narave naše rešitve, kjer je nekaj sekundni zastoj sprejemljiv, smo se odločil za uporabo 12-bitnega zapisa. Ob tem se je potrebno zavedati, da se časi dostopov do več senzorjev pri trenutni zasnovi našega programa seštevajo. Če na vodilo priklopimo 10 senzorjev, bo pridobivanje vseh temperatur trajalo do 7 sekund in pol.

Po prejetju ukaza za konverzijo temperature, senzor temperaturo izmeri in pretvori ter zapiše v notranji spomin. Kot kaže slika 4.5, ta sestoji iz 8 Bajtov. Temperatura se zapiše v prvi par Bajtov, preostali prostor je namenjen konfiguraciji senzorja (Bajt 4), nastavljanju vrednosti za prožitev temperaturnega alarma (Bajta 2 in 3) nekaj spomina pa je rezerviranega za interno uporabo. Bajti 2, 3 in 4 se hranijo v EEPROM spominu in preživijo izpade električnega napajanja.

Po pretečenem maksimalnem času konverzije temperature, program to prebere z naslednjim postopkom:

1. inicializacija vodila (ponastavitveni pulz, pri tem preverimo še prisotnost naprava),
2. ukaz "Match rom command"(na vodilo zapišemo vrednost 0x55),



Slika 4.5: Interni spomin senzorja DS18B20

3. na vodilo zapišemo 8 Bajtno identifikacijsko kodo senzorja, ki ga želimo nasloviti,

4. ukaz "Read scratchpad"(na vodilo zapišemo vrednost 0xBE),

5. sprožimo branje dveh Bajtov.

Omenjena Bajta vsebujeta izmerjeno temperaturo. Prenos se prične z najmanj pomembnim bitom (bit 0) in konča z najpomembnejšim bitom (bit 15). Branje zaključimo s ponovno inicializacijo vodila. Ko smo temperaturo prebrali, jo je zaradi lažjega izpisa v programu potrebno pretvoriti v ascii zapis. Negativne temperature so zapisane v dvojiškem komplementu zato najprej preverimo najpomembnejši bit (bit 11). Če ima ta vrednost 1, gre za negativno vrednost, kar pomeni da v pomnilnik zapišemo ascii vrednost minusa, nato pa izračunamo dvojiški komplement dane vrednosti tako, da jo odštejemo od 0. Dobljeno vrednost moramo dodatno obdelati. Najprej jo razbijemo na prve štiri bite in na preostali Bajt. Bajt predstavlja celo število. Za konverzijo v ascii zapis, smo napisali rutine za celoštevilsko deljenje z ostankom (*modulo*). S pomočjo ostankov pri deljenju s števili potence 10 (konretno 100 in 10), program celoštevilsko vrednost razbije v posamezne cifre, te pa nato pretvori v ascii vrednosti s prištevkom heksadecimalne vrednosti 30. Po obdelavi Bajta, se program loti preostalih štirih bitov. Ti biti predstavljajo šestnajstinke celote. Ker strojne podpore za deljenje na AVR krmilnikih ni, implementacija pa bi bila kompleksna in neučinkovita,

vrednost pretvorimo z množenjem po formuli 4.2. V formuli nastopa deljenje s 4, katero je bilo veliko lažje implementirati s pomočjo bitnih rotacij in pomikov(*shift*). Dobljeno število program obdela na enak način kot Bajt s pomočjo deljenja z ostankom in prištevanja 0x30 zapiše ascii vrednosti decimalnega števila. Po končani obdelavi je na znani lokaciji v ramu zapisana temperatura v ascii formatu, ki jo krmilnik servira v SMS odgovoru.

Formula 4.2 *Decimalna komponenta temperature pomnožena s tisoč je enaka vrednosti v prvih štirih bitih, pomnoženi z 250 in deljeni s 4*

$$T_{dec} * 1000 = T_{0:3} * 250/4 \tag{4.2}$$

Poglavje 5

Upravljanje GSM terminala

Pri iskanju primernega GSM terminala smo si postavili naslednje kriterije:

- terminal mora biti cenovno dostopen,
- omogočati mora komunikacijo preko serijskega vodila,
- podpirati mora Hayesov protokol AT,
- podpirati mora pošiljanje SMS sporočil v tekstovnem načinu.

Na podlagi omenjenih kriterijev smo se odločili za uporabo GSM terminala GS64.

5.1 Opis terminala GS64

Terminal GS64 lahko deluje na frekvencah 900, 850, 1800 in 1900 MHz. Podpira tudi prenos podatkov preko protokola GPRS [7].

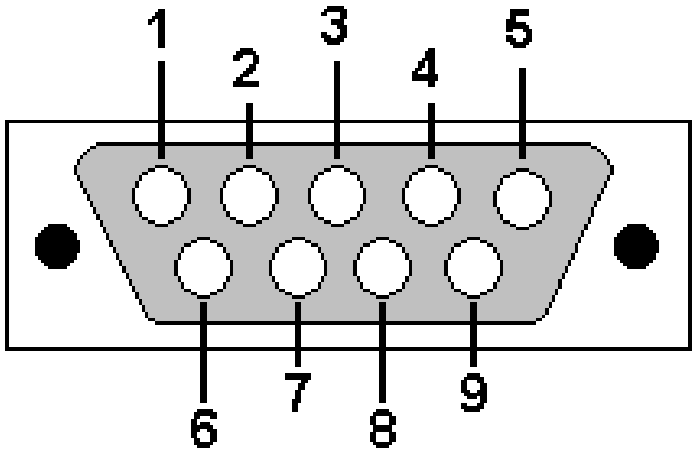
Terminal varuje robustno črno plastično ohišje dimenzij 77 x 67 x 26 mm. Na ohišju se nahajajo:

- konektor DE-9 (slika 5.2) za serijsko komunikacijo,
- konektor za zunanje električno napajanje (RJ11),
- konektor za zunanjo GSM anteno,

Slika 5.1: Terminal GS64

- plastična vratca z zatičem, ki varujejo sim kartico,
- rdeča in zelena led dioda.

Z izjemo konektorja za napajanje, so vsi elementi vidni na sliki 5.1. Terminal prižgemo tako, da nanj priklopimo električno napajanje, nato pa na četrto nogico konektorja za napajanje priklopimo napetost 5V ali več za vsaj 1 sekundo. Po vklopu sledi inicializacija terminala, ki traja okoli 10 sekund. Ko je inicializacija končana, lahko pričnemo z izstavljanjem ukazov preko Hayesovega protokola AT. Hitrost prenosa simbolov preko vmesnika RS-232 je lahko od 300bps do 230400 bps in jo terminal zna prilagoditi hitrosti sogovornika, če ta ni nižja od 1200bps [7]. V našem primeru je sogovornik bil naš krmilnik s hitrostjo prenosa simbolov 9600bps, katero je GSM terminal brez težav tudi prepoznal in se ji prilagodil. To hitrost smo izbrali, ker je zelo pogosta pri serijski komunikaciji, podpira pa jo širok spekter naprav in z njo ne bi smeli imeti nikakršnih težav. Ker se preko serijskega vodila ne prenašajo velike količine podatkov (v 1 seji tj. prejetje SMS sporočila, branje, obdelava in pošiljanje odgovora se po naših meritvah v povprečju pretoči okoli 300 Bajtov podatkov) in ker v naši aplikaciji čas odgovora ni kritičnega pomena, pomislekov ob izbiri nizke hitrosti prenosa simbolov nismo imeli.



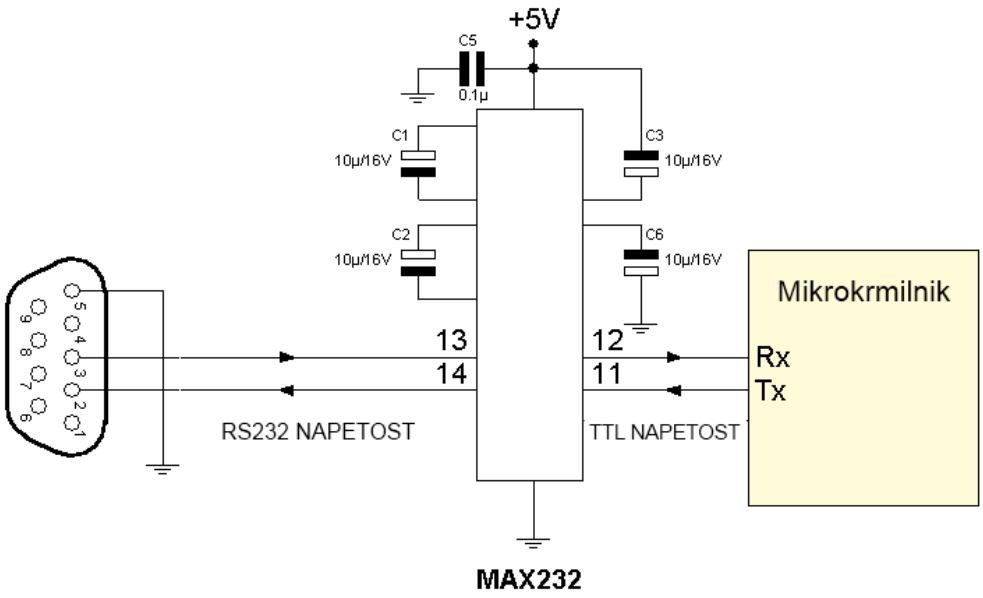
Slika 5.2: Konektor DE-9

Omeniti velja še še druge zmogljivosti terminala, ki bi lahko bile uporabljene v kasnejših nadgradnjah naše aplikacije:

- podpora protokola GPRS in TCP/IP sklada preko protokola AT,
- podpora tehnologije fax,
- podpora kodiranju znakov Unicode v SMS sporočilih.

5.2 Opis RS-232 komunikacije

RS-232 je oznaka za skupek standardov, ki definirajo oblike (*shape*) in časovne okvire (*timings*) serijske komunikacije ter konektorje, večinoma v povezavi s serijskimi priključki, ki so jih namizni računalniki nekoč imeli. Logična vrednost 0 je definirana z napetosjo od -3V do -15V, logična vrednost 1 pa z napetosjo +3V do +15V. Namizni računalniki ponavadi generirajo napetosti -12V in +12V, naš GSM terminal pa uporablja napetosti nižje od -4V in višje od +4V [7]. Serijska komunikacija med namiznim računalnikom in GSM terminalom je zelo enostavna, saj moramo napravi samo pravilno povezati, za

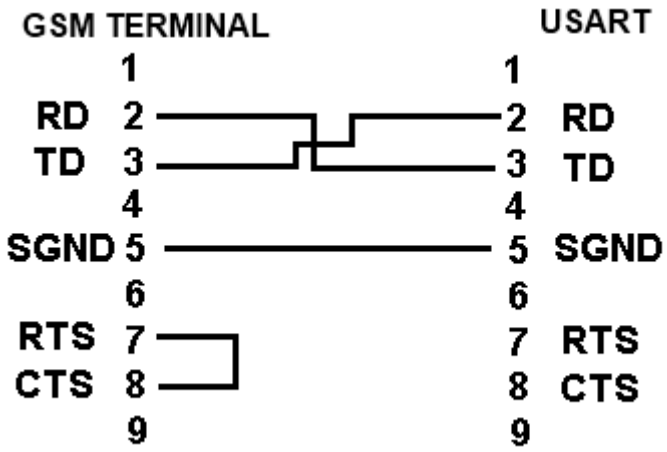


Slika 5.3: Integrirano vezje MAX232

kar uporabimo navaden RS-232 kabel z DE-9 konektorji. Naš cilj je bil upravljanje terminala s krmilnikom, ta pa ne premore takšnih napetosti. Krmilnik ATmega168A ima strojno podporo za serijsko komunikacijo v okviru komponente USART, ki je opisana v poglavju 5.3. Direktna povezava USART nogic na RS-232 vodilo ne deluje in lahko krmilnik tudi poškoduje. Krmilnik za serijsko komunikacijo uporablja medtranzistorski nabor napetosti (TTL napetost). Logična vrednost 0 je predstavljena z 0V, logična vrednost 1 pa s +5V. Problem smo rešili z uporabo integriranega vezja MAX232. MAX232 je pretvornik napetosti, ki napetosti s katerimi operira krmilnik (0V in 5V), pretvarja v napetosti, ki ustrezajo standardu RS-232. Shema vezave je prikazana na sliki 5.3

Po uspešni vezavi komunikacija še vedno ni delovala. Po podrobni preučitvi priročnika, ki je naveden med literaturo pod številko 8 smo naleteli na naslednji drobni tisk:

"Note! Regardless of how this command is configured, if 3 wire



Slika 5.4: Shema vezave serijskega vodila med GSM terminalom in krmilnikom (MAX232 ni prikazan)

> communication between the module and the application is required then the RTS and CTS lines (on the module) should be looped back on each other as the chipset requires this at a hardware level."[8]

RS-232 predvideva nogici RTS in CTS (sliki 5.2 in 5.4, nogici 7 in 8) za kontrolo pretoka (flow control). Kontrola pretoka deluje tako, da naprava 1 pred pošiljanjem podatkov zaprosi za dovoljenje tako, da spremeni logično vrednost nogice RTS (request to send) na napravi 2, če je naprava 2 pripravljena na sprejemanje, odgovori tako, da spremeni logično vrednost na nogici CTS (clear to send). Ker kontrole pretoka nismo implementirali, smo jo izklopili z mostičkom med nogicama RTC in CTS na terminalu (slika 5.4), kot predpisuje priročnik.

5.3 Implementacija RS-232 komunikacije

Serijsko komunikacijo smo v aplikaciji implementirali s pomočjo strojnih prekinitev. Mehanizem strojnih prekinitev je del krmilnika ATmega168A in

omogoča hitro asinhrono izvajanje akcij. Prekinitev omogoča, da lahko nek zunanji ali notranji dogodek prekine trenutno aktivnost centralne procesne enote, da se ta lahko posveti reakciji na ta dogodek (servisiranju prekinitve). AVR mikrokrmilniki imajo v začetku naslovnega prostora za programski pomnilnik rezervirane lokacije. Tem lokacijam pravimo prekinitveni vektorji (*interrupt vector*), predstavljajo pa vhodno točko v servisiranje prekinitve. Če želimo servisirati neko prekinitev, to storimo tako, da v ustrezni prekinitveni vektor zapišemo ukaz, ki pokliče ustrezno rutino. Rutini, ki se izvede preko klica v prekinitvenem vektorju, pravimo prekinitveno servisna rutina (*interrup service routine*) oz. na kratko ISR. Ker prekinitve upočasnjujejo delovanje programa, so privzeto izklopljene. Vklop in izklop prekinitev je realiziran na dveh nivojih. Da se prekinitev proži, mora biti vklopljena v ustreznem kontrolnem registru, hkrati pa mora biti zastavica globalnih prekinitev GIE (*global interrupt enable*) vklopljena. Omenjena zastavica vklaplja in izklaplja vse prekinitve, nastavljamo pa jo z ukazoma SEI in CLI. Globalni izklop vseh prekinitev je potreben v primerih, ko centralna procesna enota izvaja časovno kritično kodo in se ne sme pustiti motiti, prekinitve pa more hitro izklopiti. Zastavica GIE se avtomatsko izklopi tudi ob vsakem izvajanju prekinitvenega vektorja saj bi v nasprotnem primeru lahko prišlo do proženja prekinitve med servisiranjem druge, kar običajno ni zaželjeno, saj se prva prekinitev ne servisira prva, obstaja pa tudi nevarnost ciklanja. Zastavico GIE običajno vklopimo nazaj na koncu prekinitveno servisne rutine s posebnim ukazom RETI. Ukaz RETI je enakovreden ukazoma RET in SEI skupaj, z njim dosežemo vračanje iz rutine in vklop globalne zastavice prekinitev, vse skupaj pa traja 1 cikel manj. Ob pisanju prekinitveno servisnih rutin pa smo morali paziti tudi na ohranjanje vrednosti registrov. Ker prekinitve prekinjajo redno delovanje programa, smo morali poskrbeti, da njihove servisne rutine za seboj pustijo statusni in delovne registre v enakem stanju, kot je bilo tik pred klicem neke prekinitveno servisne rutine. V vsaki prekinitveno servisni rutini smo vrednosti vseh delovnih registrov (in statusnega), ki se tekom rutine zaradi različnih ukazov kakorkoli spremenijo, porinili na

sklad, pred vračanjem iz prekinitveno servisne rutine pa v obratnem vrstnem redu povlekli nazaj v ustrezne registre.

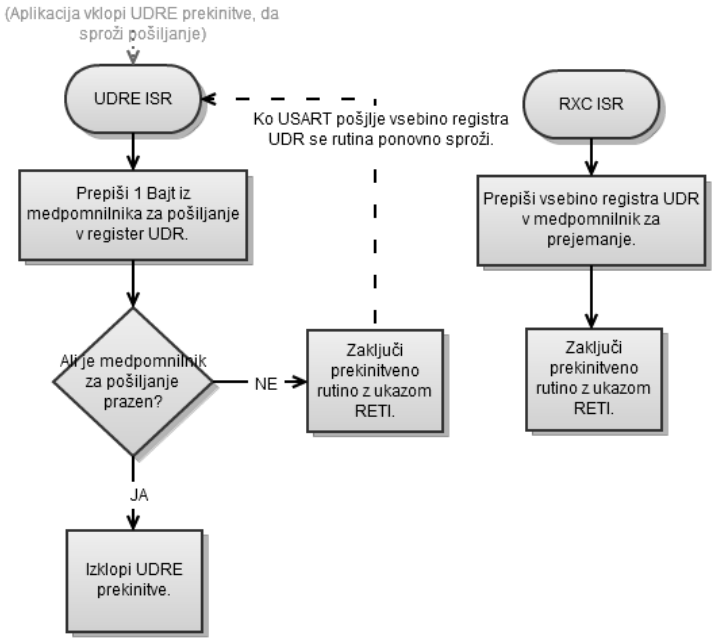
Če povzamemo so torej prekinitveno servisne rutine posebne v tem, da so prekinitve tekom njih izklopljene in jih vklopimo z RETI in v tem, da ne smejo pokvariti začetnega stanje statusnega in delovnih registrov.

V centralni procesni enoti imamo na izbiro različne prekinitve, ki se prožijo ob različnih dogodkih. V naši aplikaciji smo uporabili prekinitvi "RX Complete"(na kratko RXC) in "Uart Data Registry Empty"(UDRE). Prva se zgodi, ko komponenta USART prebere nov Bajt v register UDR (*uart data registry*). Druga se zgodi kadarkoli je register UDR prazen, uporabili smo jo za pošiljanje podatkov iz medpomnilnika za pošiljanje na serijsko vodilo.

Mehanizem branja in pisanja podatkov deluje asinhrono s pomočjo omenjenih prekinitev. Na sliki 5.5 diagram poteka prikazuje delovanje mehanizmov za pisanje in branje na serijsko vodilo. Prekinitev RX Complete je vklopljena ves čas, saj ne vemo, kdaj bodo podatki prispeli. Ko se prekinitev sproži se preko prekinitvenega vektorja pokliče prekinitveno servisna rutina, ki prebere prispeli Bajt iz registra UDR in ga zapiše v medpomnilnik za prejemanje. Pisanje se zgudi nekoliko drugače. Aplikacija zapisuje po 1 Bajt v medpomnilnik za pošiljanje, zatem pa vsakič vklopi prekinitev UDRE. Rutina, ki servisira prekinitev UDRE iz medpomnilnika za pošiljanje prepiše Bajt v register UDR. Če je v medpomnilniku še podatkov, ki čakajo na prenos, rutina pusti UDRE prekinitev vklopljeno, v nasprotnem primeru pa jo izklopi. Ob prvem vklopu UDRE prekinitve smo torej pričeli cikel prekinitev, ki se bodo prožile dokler ne bodo vsi podatki iz medpomnilnika za pošiljanje poslani na serijsko vodilo.

5.4 AT ukazi

Ko smo usposobili komunikacijo med krmilnikom in GSM terminalom, je sledil razvoj dela aplikacije, ki upravlja GSM terminal. Upravljanje poteka tako, da preko vodila RS-232 pošiljamo AT ukaze [8]. AT ukazi se pričnejo



Slika 5.5: Diagram branja in pisanja na serijsko vodilo preko komponente USART

z besedo AT, vsi ukazi za delo s SMS sporočili, ki smo jih uporabili v naši aplikaciji, so razširitev osnovnega Hayesovega protokola AT, zato se pričnejo z besedo AT+. Najpreprostejši AT ukaz je sama beseda AT, ki se uporablja za preverjanje prisotnosti terminala. Če terminal uspešno prejeme ukaz AT, nazaj odgovori z besedo OK. V primeru, da ni prišlo do napake, se vsi odgovori terminala zaključijo z besedo OK. V primeru, da se je zgodila napaka, terminal odgovori z besedo ERROR, običajno pa sledi tudi opis napake.

V naši aplikaciji smo uporabili naslednje ukaze:

1. ukaz AT,
2. ukaz AT+CNMI=1,1,0,0,1
3. ukaz AT+CMGF=1
4. ukaz AT+CPIN?
5. ukaz AT+CPIN="1234"oz. ukaz AT+CPIN="12345678","1234"
6. ukaz AT+CMGR=1
7. ukaz AT+CMGS
8. ukaz AT+CMGD

Prvih pet ukazov smo v enakem zaporedju uporabili v rutini initGSM, ki se izvede, ko terminal konča z interno inicializacijo. Rutina je razložena v poglavju 5.4.1. Ko je rutina v celoti izvedena je GSM terminal nastavljen in prijavljen v omrežje, pripravljen je na delo. Zadnji trije ukazi se uporabljajo za prejemanje, pošiljanje in brisanje SMS sporočil tekom aplikacije, kar je opisano v poglavju 5.4.2. Posnetek komunikacije med krmilnikom in terminalom med inicializacijo in obdelavo sporočila je prikazan na sliki 3.3 na strani 18.

5.4.1 Inicializacija GSM terminala

Implementacija rutine initGSM je prikazana v programski kodi 5.1.

V rutini initGSM najprej izpraznemo medpomnilnik prejetih simbolov (mehanizem prejemanja in oddajanja simbolov je opisan v poglavju 5.3). Za to poskrbi vrstica 2. Sledita vrstici 3 in 4, kateri moramo obravnavati skupaj. Beseda .db je direktiva zbirniku, ki pove, da naj se na mestu pojavitve v programski pomnilnik zapiše neka vrednost oz. niz vrednosti [2]. V konkretnem primeru se bo v programski pomnilnik zapisal niz AT (zbirnik AT pretvori v vrednosti po ascii tabeli), ki mu sledijo še vrednosti 13, 10 in 0. Vrednosti 13 in 10 ustrezata ascii znakoma CR (*carriage return*) in LF (*line feed*) in predstavljata konec vrstice (in s tem ukaza). Vrednost 0 označuje konec niza vrednosti. V primeru, da je število nanizanih vrednosti neparno, zbirnik samodejno doda dodatno vrednost, ki je 0. Na to nas AVR Studio opozori s sporočilom:

> "warning: .cseg .db misalignment - padding zero byte."

Zbirnik to stori zato, da poravna ukaze, ki sledijo literalu, na naslove, ki jih programski števec lahko naslovi (glej poglavje 2.1).

Vrstica 3 pokliče rutino SendString. Ukaz rcall deluje tako, da na sklad potisne trenutno vrednost programskega števca [3] (ta v trenutku klica kaže na literal, podan z direktivo .db), nato pa programski števec nastavi na začetek rutine SendString. Ta mehanizem izkoristimo tako, da naslov takojšnjega operanda, ki se sedaj nahaja na skladu, povlečemo iz sklada in uporabimo kot parameter v naši rutini. Rutina SendString uporabi ukaz LPM za dostop do programskega pomnilnika. Ker ukaz LPM naslavlja 8-bitne besede, programski števec pa naslavlja besede dolžine 16 bitov, moramo vrednost programskega števca, ki smo jo vzeli s sklada, pomnožiti z 2. V naslednjem koraku se rutina v iteracijah sprehaja po programskem pomnilniku in na vodilo pošilja Bajt za Bajtom, dokler ne naleti na vrednost 0. Takrat se ustavi. Ker smo rutino poklicali z ukazom rcall, jo zapustimo z ukazom ret. Ukaz ret deluje tako, da iz sklada povleče in programskemu števcu nastavi

staro vrednost, ki je bila shranjena na sklad ob klicu ukaza rcall. Ker je rutina to vrednost že povlekla iz sklada, jo mora pred ukazom RET tudi vrniti nazaj. Če bi rutina na sklad vrnila originalno vrednost, bi procesorska enota pričela izvajati naš literal in aplikacija bi prenehala pravilno delovati. Zato moramo na sklad naložiti naslov prve lokacije za literalom v programskem pomnilniku, to lokacijo imamo že izračunano ob zaključku iteracij zato jo pred klicem ukaza RET delimo z 2 in potisnemo nazaj na sklad.

Po uspešni izvedbi rutine SendString, se preko mehanizma prekinitev (*interrupts*) prične pošiljanje literala iz medpomnilnika na vodilo. Vrstica 5 je makro, ki zapravi 1 sekundo procesorksega časa. Ta čas je namenjen za uspešno pošiljanje niza na vodilo, obdelavo, ki steče v terminalu in odgovor, ki ga terminal pošlje nazaj.

Vrstice od 6 do 9 preverijo, ali je krmilnik v odgovor prejel niz OK. Če se to ni zgodilo, se izvajanje rutine prestavi na začetek in pošiljanje ukaza se ponovi.

Ko izvajanje programa doseže 10. vrstico pomeni, da je terminal uspešno odgovoril z OK, zato pričnemo s pošiljanjem željenih nastavitev. AT+CNMI =1,1,0,0,1 pomeni, da nas bo terminal obvestil ob prejetju novega SMS sporočila brez, da bi ga mi po tem povprašali. AT+CMGF=1 pomeni, da bo vse delo z SMS sporočili potekalo v tekstovnem načinu, brez PDU kodiranja. Sledi ukaz AT+CPIN?, ki povpraša terminal ali je potrebno vnesti pin kodo. Če v odgovoru nastopa beseda PIN, terminalu pošljemo ukaz AT+CPIN=”1234”, kjer 1234 predstavlja pin kodo sim kartice. Če je v odgovoru beseda PUK pa odgovorimo z AT+CPIN=”12345678”,”1234”, kjer prvi niz številk predstavlja puk kodo drugi niz pa pin kodo sim kartice. Po vnosu sim kode, terminal sproži prijavo v omrežje mobilnega operaterja in je pripravljen na prejemanje in pošiljanje SMS sporočil.

Programska koda 5.1: Inicializacija GSM terminala

```
1 initGSM:
2 rcall Rx_fifo_reset
3 rcall SendString
4 .db "AT",13,10,0
```

```
5  wait 1000
6  rcall IndexOf_RXFIFO
7  .db "OK",0
8  tst r25
9  brmi initGSM
10 rcall SendString
11 .db 13,10,"AT+CNMI=1,1,0,0,1",13,10,0
12 wait 1000
13 rcall SendString
14 .db 13,10,"AT+CMGF=1",13,10,0
15 wait 1000
16 rcall SendString
17 .db "AT+CPIN?",13,10,0
18 wait 2000
19 rcall IndexOf_RXFIFO
20 .db " PIN",0
21 tst r25
22 brpl inputPinOnly
23 rcall IndexOf_RXFIFO
24 .db "PUK",0
25 tst r25
26 brpl inputPuk
27 rjmp initGSM
28 inputPuk:
29 rcall SendString
30 .db "AT+CPIN=",'"',"57141204",'"',",",'"',"1714",'"',13,10,0
31 wait 5000
32 ret
33 inputPinOnly:
34 rcall SendString
35 .db "AT+CPIN=",'"',"1714",'"',13,10,0
36 wait 5000
37 ret
```

5.4.2 Procesiranje sporočil

Po uspešni inicializaciji terminala je aplikacija v stanju pripravljenosti. To pomeni, da čaka na terminal za morebitna obvestila. O čem nas terminal obvešča je odvisno od samih nastavitev terminala. Aplikacijo smo napisali tako, da reagira na obvestila o prejetju novega sporočila. Primer takega obvestila je:

+CMTI: "ME",15.

Beseda +CMTI nam pove, da gre za prejem SMS sporočila, ME pomeni, da je sporočilo bilo zapisano v interni spomin terminala, 15 pa označuje lokacijo sporočila v spominu (indeks) [8]. Aplikacija na tako obvestilo odgovori z ukazom

AT+CMGR=15,

kar sproži branje sporočila [9]. Aplikacija prebrano sporočilo obdela - iz njega izlušči GSM številko pošiljatelja in morebitne ukaze. Če v sporočilu ne prepozna nobenega od možnih ukazov, odgovori z

AT+CMGD=15,

s tem terminalu ukaže izbris sporočila. Če sporočilo vsebuje ukaz, ki je podprt, aplikacija odpre sejo za pošiljanje SMS sporočila (odgovora) z ukazom:

AT+CMGS="+12345678987",

številka v ukazu je izmišljena, predstavlja pa GSM številko pošiljatelja SMS sporočila, kateremu odgovarjamo. Nato aplikacija požene rutine, ki so potrebne za izvedbo enega ali več ukazov iz SMS sporočila, na koncu pa rezultate zapiše v odgovor. Pošiljanje SMS sporočila potrdi tako, da terminalu pošlje heksadecimalno vrednost 1A (*substitute*) [8]. Prejeto sporočilo je sedaj obdelano, zato terminalu naročimo brisanje z zgoraj navedenim ukazom za brisanje.

Poglavje 6

Delovanje aplikacije

Razvoj aplikacije je potekal modularno. Kodo za komunikacijo s terminalom, kodo za temperaturne meritve in kodo za upravljanje GSM terminala ter procesiranje SMS sporočil smo razvili ločeno, vsako v svoji datoteki. Ko so vsi moduli delovali pravilno, smo jih vključili v osnovno kodo z direktivami .include. Napisali smo glavno zanko programa (programska koda 6.1. Ta preverja ali so prispela nova sporočila in proži njihovo procesiranje. Vrstica 2 v kodi 6.1 preverja ali je vrednost zastavice CRflag enaka 1. To zastavico postavi prekinitveno servisna rutina za branje serijskega vodila, ko naleti na ascii znak z oznako CR oz. vrednostjo 13. Ta označuje konec vrstice. CRflag aplikaciji sporoča, da je iz serijskega vodila v medpomnilnik za branje bila prebrana vsaj ena vrstica. Če je zastavica postavljena, se zgodi klic rutine procRXedLine v vrstici 3. Rutina procRXedLine najprej preveri, če je med prejetimi podatki niz znakov +CMTI, ki oznanja prejetje SMS sporočila. Če niza ni, rutina pobriše medpomnilnik za prejemanje, resetira zastavico CRflag na 0 in se zaključi. Če niz je, rutina postavi zastavio SMSflag in se loti obdelave prebranih podatkov v medpomnilniku za branje. Iz njih izlušči +CMTI obvestila skupaj s pripadajočimi indeksi oz. lokacijami sporočil v spominu terminala. Te prepiše v podatkovno strukturo vrste v ramu, nato pa medpomnilnik za prejemanje pobriše.

Vrstica 4 preverja ali je zastavica SMSflag postavljena. Ta označuje, da je

potrebno prebrati in obdelati 1 ali več sporočil. Če je zastavica postavljena, se zgodi klic rutine procSMSqueue. Ta rutina za vsako shranjeno lokacijo v znani podatkovni strukturi vrste sproži branje sporočila na terminalu. Po vsakem bralnem ukazu rutina počaka 1 sekundo, da se sporočilo pretoči po vodilu preko prekinitveno servisne bralne rutine v medpomnilnik za prejemanje. V naslednjem koraku aplikacija preveri, ali v sporočilu nastopa kateri od podprtih ukazov. Če ne obstaja, sporočilo zavrže, terminalu pa ukaže brisanje. Če ukaz obstaja aplikacija srvisira 1 ali več ukazov s klici ustreznih rutin. Te izvedejo željene akcije in poskrbijo za SMS odgovor pošiljatelju. Postopek samega procesiranja sporočila je natančneje opisan v poglavju 5.4.2.

Vrstica 5 je makro, ki zapravi sekundo procesorskega časa, in predstavlja možnost, da se morebitno pisanje na vodilo zaključi in medpomnilnik za pošiljanje sprazni pred naslednjo iteracijo galvne rutine. Tukaj se program med normalnim delovanjem zadržuje večino časa. Branje in pisanje na serijsko vodilo poteka asinhrono in ni moteno.

Programska koda 6.1: Glavna zanka programa

```
1 loop:
2 sbrc StateReg, CRflag
3 rcall procRXedLine
4 sbrc StateReg, SMSflag
5 rcall procSMSqueue
6 wait 1000
7 rjmp loop
```

Poglavje 7

Uporaba

Za uspešno uporabo sistema, ki smo ga razvili, moramo priskrbeti delujočo sim kartico, v programski kodi pa nastaviti pravilno pin in puk kodo.

30 sekund po vklopu sistema, je ta pripravljen na delovanje. Uporabnik ima na izbiro naslednje ukaze:

- Help - izpis vseh možnih ukazov
- GetTemp - izpis temperatur priklopljenih senzorjev (trenutno 2)
- SwStatus - izpis stanj štirih priklopljenih stikal
- SetSwitch - vklop posameznih stikal (SetSwitch0123 vklopi vsa stikala, SetSwitch03 pa prvo in zadnje, možne so poljubne kombinacije), odgovor je izpis stanj kot pri SwStatus
- ClearSwitch - enako kot SetSwitch, le da gre za izklop stikal

Ukazi niso občutljivi na velike in male črke (*case-insensitive*). Uporabnik lahko torej na daljavo spremlja temperature in vklaplja stikala. Na stikala lahko vežemo poljubno napravo, področje uporabe je posledično zelo široko.

7.1 Primer uporabe

Tipičen primer uporabe razvitega sistema bi bil za upravljanje ogrevalnega ali ohlajevalnega sistema hiše. Zamislimo si uporabnika, ki ima vikend na oddaljenem kraju. Vikend ima lastno ogrevanje, ki se sproži s stikalom. Uporabnik si želi obiskati vikend sredi zime, vendar ga od tega odvrača mraz v hiši, saj vikend pozimi sameva in je ogrevanje izklopljeno. Če bi uporabnik obiskal vikend, bi moral prenašati mraz precej časa, saj ogrevanje vikenda lahko traja tudi več dni. To uporabnika odvrača od zimskih obiskov. Ker ni pripravljen trpeti mraza. Namestitev našega sistema bi opisani problem rešila. Uporabnik bi nekaj dni pred načrtovanim obiskom vikenda s pomočjo sistema preveril notranjo in zunanjo temperaturo. Na podlagi prve bi se odločil, ali je potrebno prižgati ogrevanje. Na podlagi druge bi pa ocenil kakšna oblačila bo potreboval za gibanje na prostem. Če bi smatral, da je v vikendu premrzlo, bi s SMS ukazom sprožil ogrevanje. Čez nekaj časa bi ponovno peveril temperaturo in po potrebi ogrevanje izklopil. Uporabnikov problem je tako rešen.

Analogno lahko rešimo problem hlajenja oddaljenega objekta v poletnih mesecih.

Poglavje 8

Sklepne ugotovitve

Razvita rešitev je tehnologija, ki rešuje problem upravljanja na daljavo z uporabo priljubljenega medija - SMS sporočila. Rešitev podpira osnovne ukaze, ki omogočajo spremljanje temperatur in vklapljanje naprav, zasnovana pa je tako, da so nadgradnje za specifične namene možne in enostavne.

Sistem je sedaj v fazi preizkušanja in odpravljanja hroščev (zaenkrat nismo odkrili še nobenega). Vezje bo v prihodnosti potrebno prenesti iz ploščice za prototipiziranje v bolj stabilno obliko, ki bo odpornejša na zunanje vplive in rokovanje.

V rešitvi vidimo velik potencial, seznam izboljšav in razširitev pa je že sedaj dolg:

- dodati senzor za vlago,
- dodati možnost posredovanja sporočil na tretjo številko (SMS proxy),
- dodati geslo za izvajanje ukazov,
- dodati uporabniške nastavitve, ki se hranijo v EEPROM spominu,
- po implementaciji gesla in hrambe uporabniških nastavitev v EEPROM-u dodati še možnost nastavljanja gesla preko SMS sporočila in ne v kodi,

- dodati možnost temperaturnega alarma (sistem nas obvesti, ko temperatura pade pod ali zraste nad neko nastavljivo vrednost),
- dodati podporo GPRS in možnosti upravljanja preko spleta.

Ugotovitve zaključujemo z oceno, da smo z izdelavo diplomske naloge dosegli zastavljene cilje razvoja avtonomnega sistema za daljinsko upravljanje s SMS sporočili. Pri tem smo pridobili veliko novih znanj z raznih področij kot so zbirni jezik (AVR), arhitektura AVR, branje tehničnih specifikacij (*datasheet*) protokolov in naprav, spoznali smo se tudi z električnimi elementi in shemami vezav ter spajkanjem.

Literatura

[1] Atmel Corporation (2011) 8-bit Atmel Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash ATmega48A ATmega48PA ATmega88A ATmega88PA ATmega168A ATmega168PA ATmega328 ATmega328P. Dostopno na:
http://www.atmel.com/Images/doc8271.pdf

[2] Atmel Corporation (2012) AVR Assembler User Guide. Dostopno na:
http://www.atmel.com/Images/doc1022.pdf

[3] Atmel Corporation (2010) AVR Instruction Set. Dostopno na:
http://www.atmel.com/Images/doc0856.pdf

[4] (2012) Introducing AVR Dragon. Dostopno na:
http://people.ece.cornell.edu/land/courses/ece4760/AtmelStuff/dragon.pdf

[5] Atmel Corporation (2004) AVR318: Dallas 1-Wire® master . Dostopno na:
http://www.atmel.com/Images/doc2579.pdf

[6] Maxim Integrated Products (2008) DS18B20 Programmable Resolution 1-Wire Digital Thermometer Dostopno na:
http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf

[7] CEP AG (2010) CT63 Terminal Technical Description. Dostopno na:
http://www.intellicomp.de/fileadmin/Downloads/ct63/

CT63-Technische-Beschreibung-V1.4.pdf?PHPSESSID=
bdb9018598c2050f7d95fc3f9daefddf

[8] Sony Ericsson Mobile Communications International (2006) GR64 & GS64 AT Command Manual. Dostopno na:
http://www.euromobile-ukraine.com.ua/files/documents/GX64%20AT%20Command%20Manual%20P1B%20(P1A072).pdf

[9] Atmel Corporation (2006) AVR323: Interfacing GSM modems. Dostopno na:
http://www.atmel.com/Images/doc8016.pdf

[10] (2012) PuTTY: A Free Telnet/SSH Client. Dostopno na:
http://www.chiark.greenend.org.uk/~sgtatham/putty/

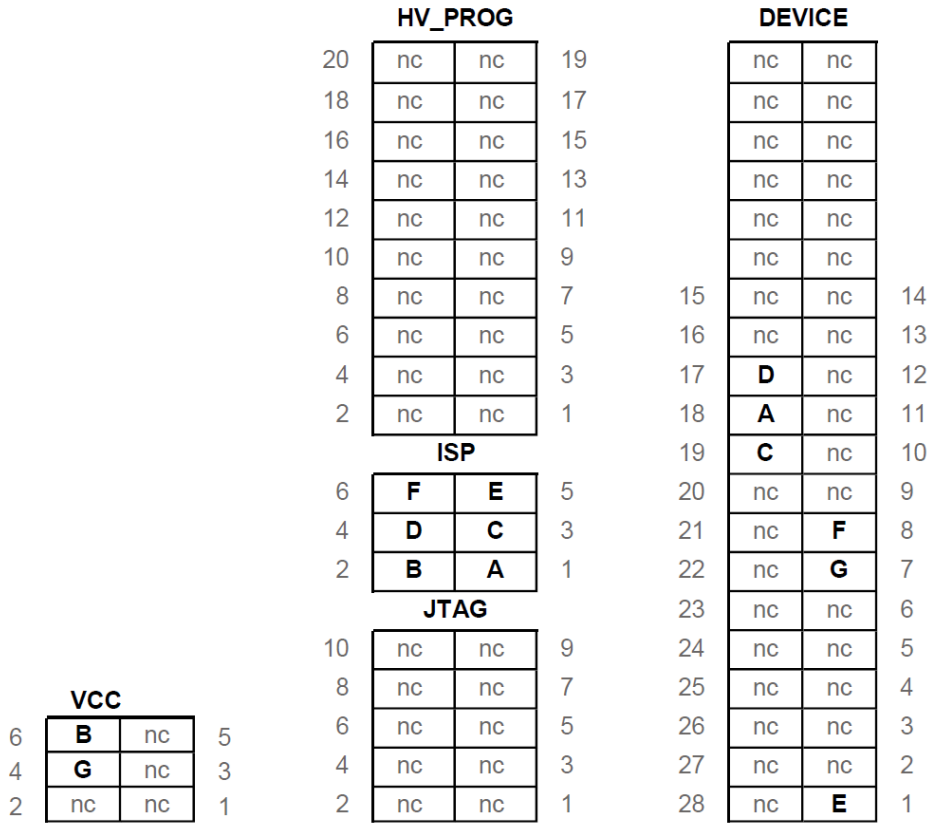
[11] (2012) AVR Studio 4. Dostopno na:
http://www.atmel.com/tools/AVRSTUDIO4.aspx

[12] Maxim Integrated Products (2010) +5V-Powered, Multichannel RS-232 Drivers/Receivers. Dostopno na:
http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf

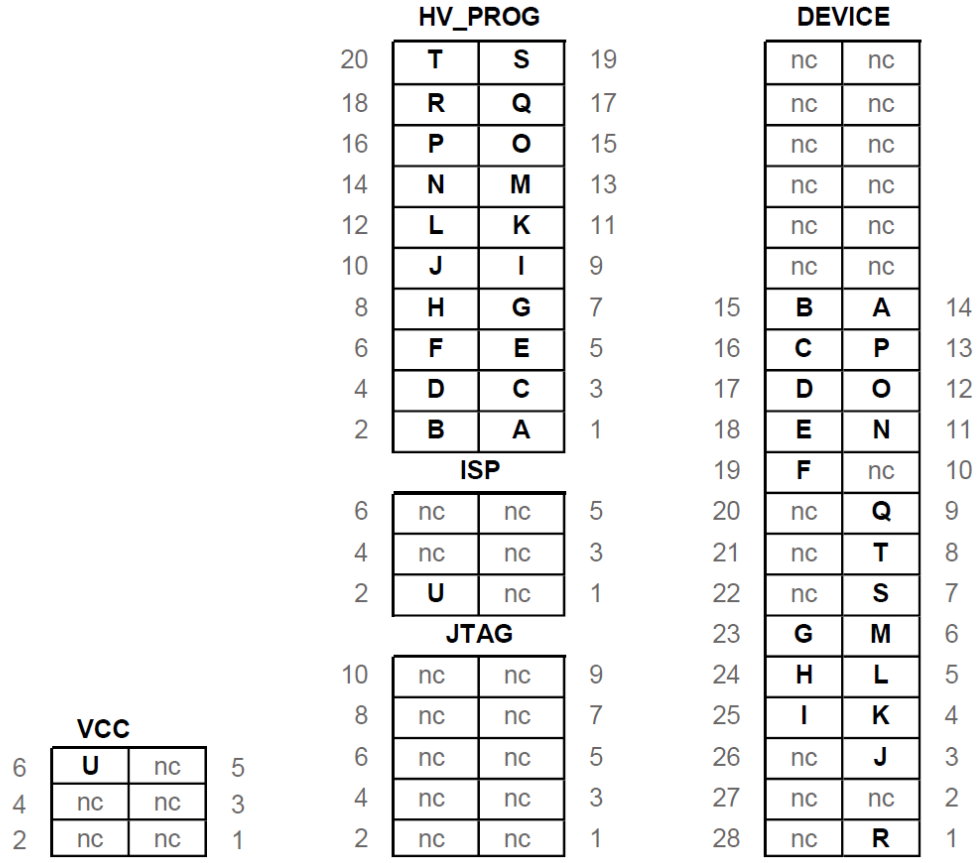
[13] (2012) 1-Wire Devices. Dostopno na:
http://www.maxim-ic.com/products/1-wire/

Dodatek A

Shemi vezav ATmega168A krmilnika s programatorjem AVR Dragon



Slika A.1: Shema vezave ATmega168A z AVR Dragonom v ISP načinu



Slika A.2: Shema vezave ATmega168A z AVR Dragonom v PP načinu