

**Rīgas Tehniskā universitāte
Datorzinātnes un informācijas tehnoloģiju fakultāte
Datorvadības, automātikas un datortehnikas institūts
Datoru tīklu un sistēmas tehnoloģijas katedra**



**Harvardes arhitektūras RISC-procesoru ar sadalītu datu un
programmas atmiņas piekļuvi**

(Laboratorijas darbi, mācību līdzekļi)

V.Zagurskis,

R.Kuzmenkovs

2008g.

Laboratorijas darbs 1

Uzdevums

Laboratorijas darba mērķi

- Iepazīties ar Charon II izstrādāšanas plati;
- Saprast kā sagatavot darbavietu;
- Saprast plates programmēšanu;

Teorija

Visu laboratorijas darbu apraksti tiek piedāvāti mapītē ar tekošu laboratorijas darbu (laboratorijas darba numurs un tas tēma), kurā būs:

1. Pats laboratorijas darba apraksts (laboratorijas darba numurs) .doc formātā;
2. Mapīte „*Materials*”, kurā būs laboratorijas darba izpildīšanas nepieciešamas informācijas minimums;
3. Mapīte „*Code*” (ne vienmēr), kurā būs piedāvāta programmas daļa;

Pēc katras nodaļas pabeigšanas (kurā ietilpst daži laboratorijas darbi) ir jātaisa atskaite, kurā jāatbild uz uzdotiem jautājumiem.

Šī laboratorijas darbā būs jāinstalē darbavieta lai būtu iespēja strādāt (piem. ieprogrammēt) ar Charon II platēm. Charon II plates sastāv no firmas Atmel mikrokontrolera Atmega128 un ethernet mikrosēmas RTL8019AS.

Atmega128 – tas ir RISC AVR arhitektūras mikrokontroleris, kuru mēs programmēsim.

RTL8019AS –tas ir Full-Duplex Realtek firmas ethernet kontroleris, kurš aptver divus pirmos (MAC+PHY) OSI modeļa līmeņus.

Charon II plate ir iespraudāmā izstrādāšanas platē (angl. Development board), kas kopā veido Charon II Development Kit, kurš sastāv no (skat. „*Materials*” mapītē ”*Charon2_dk_en.pdf*” failu):

1. Charon II moduļa (plates)
2. Izstrādāšanas plates, uz kuras ir:
 - a. LCD displeja savienotāju
 - b. Perifērijas tiltslēgu lauku (angl. Jumper Field)
 - c. AVR ISP (iekšsistēmas programmators (angl. In-system programmer))
 - d. RS-232 seriālas pieslēgvietas 0 (angl. Serial port 0)
 - e. RS-232 seriālas pieslēgvietas 1 (angl. Serial port 1)
 - f. AVR JTAG ICE pieslēgvietas (atkļūdošanas ierīce)
 - g. Barošanas pieslēgvietas
 - h. Ethernet RJ45 savienotāja

- i. 8x gaismas diodes
 - j. 1-Wire kopnes savienotāja
 - k. Paralēlas ievades pārbīdes reģistra slēdzi (angl. Shift register parallel input switch)
 - l. Paralēlas izvades pārbīdes reģistra diožu matrica
- 3. HW ISP programmatora
- 4. LCD displeja
- 5. 1-Wire DS18B20 sensora

Mūsējos laboratorijas darbos būs nepieciešamās sekojošās lietas:

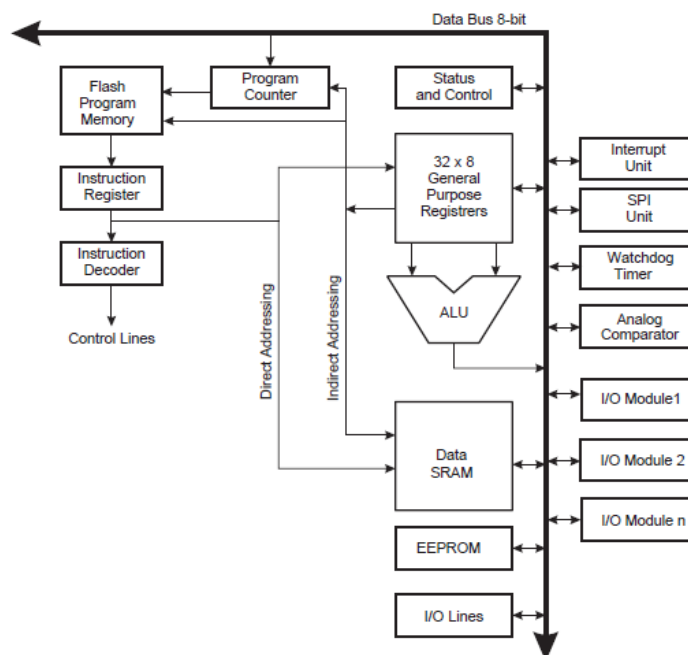
1. Plati Charon II modulis ar izstrādāšanas plati;
2. HW ISP programmers;
3. WinAvr kompilators (speciāls c valodas kompilators priekš AVR arhitektūras mikrokontrolleriem);
4. Nut/OS (operētājsistēma ar ievestu TCP/IP steku)
5. LCD displejs;
6. 1-Wire DS18B20 sensors;

Tā kā Charon II plate ir taisīta uz Atmega128 mikrokontrollera bāzes, tad ir vērts sīki aprakstīt šī mikrokontrollera arhitektūru.

levads AVR arhitektūrā

AVR arhitektūra apvieno sevī jaudīgu Harvardes arhitektūras RISC-procesoru ar sadalītu datu un programmas atmiņas piekļuvi (bāzes AVR komandu kopa sastāv no 120 instrukcijām), 32. vispārējās vajadzības reģistrus, katrs no kuriem var strādāt kā reģistrs-akumulators, un paplašinātu komandas sistēmu ar fiksētu 16 bitu garumu (zīmējums 1.). Komandu vairākums izpildās vienā taktī ar vienlaicīgu komandas izpildīšanu un nākamās komandas ienesi. Tas nodrošina ap 1 MIPS veikspēju uz katru taktu frekvences MHz..

Visiem AVR mikrokontrolleriem ir iebūvēta FLASH ROM ar iekšsistēmas programmēšanas iespēju caur 4. līniju seriālu interfeisu. AVR mikrokontrollera perifērija sastāv no: taimeriem-skaitītājiem, PWM, ārēju pārtraukumu atbalsta, analogiem komparatoriem, 10 bitu 8. kanālu ACP, paralēliem portiem (no 3. līdz 48. ievades izvades līnijām), UART un SPI interfeisiem, apsardzības taimera (angl. Watchdog timer). Visas šīs īpašības pārvērš AVR mikrokontrollerus par jaudīgiem instrumentiem mūsdienīgu augstāzīgu daudznolūku kontrolleru būvēšanā. AVR mikrokontrolleris atbalsta guļošu režīmu un mikropatēriņa režīmu. Guļoša režīmā tiek apstāts centrāla procesora kodols, tajā laikā kad reģistri, taimeri-skaitītāji, apsardzības taimeris turpina funkcionēt. Mikropatēriņa režīmā tiek saglabāts reģistru saturs, tiek apstāts taktu ģenerators, tiek aizliegtas visas mikrokontrollera funkcijas līdz tam brīdim, kamēr nebūs ārēja pārtraukuma signāla vai aparaturatgriešanas. Atšķirība no AVR mikrokontrollera modeļa, tie strādā 2,7 – 6 V vai 4 – 6 V barošanas diapazonā.



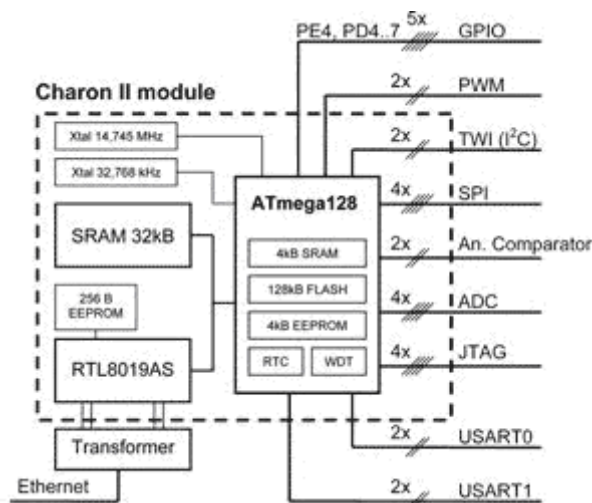
Zīmējums 1. AVR arhitektūras bloku diagramma.

Atmega128 īpašības

Atmega128 mikrokontrollerim ir:

1. 53 programmējamās līnijas, kuras sastāv no:
 - a. 8. A porta līnijām
 - b. 8. B porta līnijām
 - c. 8. C porta līnijām
 - d. 8. D porta līnijām
 - e. 8. E porta līnijām
 - f. 8. F porta līnijām
 - g. 5. G porta līnijām
2. 128 KB programmējamas atmiņas FLASH ROM
3. 4 KB MCU SRAM
4. 4 KB programmējamas atmiņas EEPROM

Zemāk ir parādīts zīmējums (zīm. 2), kurā ir attēlota Charon II moduļa shematiska pieslēgšana pie Atmega128 mikrokontrollera.



Zīmējums 2. Charon II moduļa pieslēgšana pie Atmega128 mikrokontrollera.

Darba vietas uzstādīšana

Ir divas lietas, kas vajag izdarīt vispirms, lai nodrošinātu normālu darbu ar izstrādāšanas platēm, līdzīgām Charon II. Tas ir komunikācijas nodrošināšana (dators --- Charon II) un programmēšanas valodas un kompilatora izvēlē. Tā kā mūsos laboratorijas darbos būs nepieciešama operētājsistēma (OS), to arī jāinstalē kopā ar kompilatoru.

Komunikācijas nodrošināšana

1. Izpakojiet plati. Pievienojiet barošanu (8-15 volti, optimāla ir 9 volti).
2. Darbam ar Charon II plati caur COM (UART) portu ir nepieciešams krusteniskais RS-232 kabelis, viņš tiek piedāvāts komplektā. Kā arī ir nepieciešama programma, kura ļauj darboties ar COM portu, jūs variet izmantot iebūvētu Windows operētājsistēmā HyperTerminal. To vajag īpaši sakonfigurēt lai darboties ar demo projektu, kurš ir pieejams pēc noklusējuma:
 - a. Connect using: (Izvelēties to COM portu, pie kurā tiek pievienots datu kabelis)
 - b. Bits per second: 9600 (ja ir ielādēts demo projekts)
 - c. Data bits: 8
 - d. Parity: None
 - e. Stop bits: 1
 - f. Flow control: None

Ja plate nebija iepriekš programmēta, tad, pārstartējot viņu, uz termināla loga būs ziņojums par veiksmīgu darbavietas konfigurēšanu.

Kompilatora un OS instalēšana

1. Instalējiet WinAVR GNU C kompilatoru, viņš atrodas uz CD diska.
2. Instalējiet Nut/OS priekš Win32 pielikumiem, viņa atrodas uz CD diska.
3. Instalējiet AVR Studio 4 lai būtu iespēja iedarbināt STK500 programmatūru, viņa atrodas uz CD diska.
4. Instalējiet SP2 priekš AVR Studio 4, viņš atrodas uz CD diska.

Ja viss ir izdarīts veiksmīgi, nākamais solis būs Nut/OS konfigurēšana.

1. Pāriet uz: *Start->Programs->Ethernut-4.4.0->Configurator*.
2. Ielādētajā logā izvelēt *charon2.conf*, uzspiest Ok.
3. Pēc konfigurācijas ielādēšanas pāriet uz *Edit -> Settings...*
4. Ielādētajā logā ieliktnī *Build*, izvēlnē *Platform* izvelēt *avr-gcc*. Pāriet uz izvēlni *Tools, Tool Paths* laukā atstāt *d:\ethernut-4.4.0\nut\tools\win32* (te ir minēts „d” diska numurs, ja jums viņš atšķiras, atstājat jūsēju); Pēc tam pāriet uz izvēlni *Samples, Programmer* laukā izvelēt *avr-dude*. Aizvērt uzstādījumu logu, nospiežot *Ok*.
5. Konfigurēšanas logā pāriet uz *Build->Build Nut/OS*, parādīsies divi logi, kuros jūs varēsiet vēlreiz pārbaudīt izvēlētu konfigurāciju. Šī komanda kompilēs Nut/OS ar izvēlētu konfigurāciju.
6. Konfigurēšanas logā pāriet uz *Build->Create Sample Directory*. Šī komanda kompilēs Nut/OS piemērus.

Pēdējais ir jānokopē *libusb0.dll* failu (viņš atrodas uz CD diska) iekš *Windows->System32*. Ja viss notika bez kļūdām, tad var uzskatīt, ka darbavietas uzstādīšana tiek veikta veiksmīgi.

Nut/OS koda piemēra ielādēšana mikrokontrollerī

Lai sāktu rakstīt kompilēt un iešūt savu kodu, jākonfigurē makefails – tas ir speciāls fails, kurš tiek izmantots lai WinAVR varētu nokompilēt jūsu programmu ar make.exe utilīti, kurā ir WinAVR\utils\bin mapītē. Make.exe kontrolē no programmas koda ģenerētus izpildes failus. Kā bija minēts agrāk, šīs utilītas vadību nodrošina makefili, kuri paziņo kompilatoram, kādas komandas palaist, kādus failus kompilēt un linkēt, kādu izejkodu ģenerēt u.t.t.

Mūsu gadījumā makefaili ir jau izveidoti pēc Nut/OS kompilēšanas, bet tos jāredīģē:

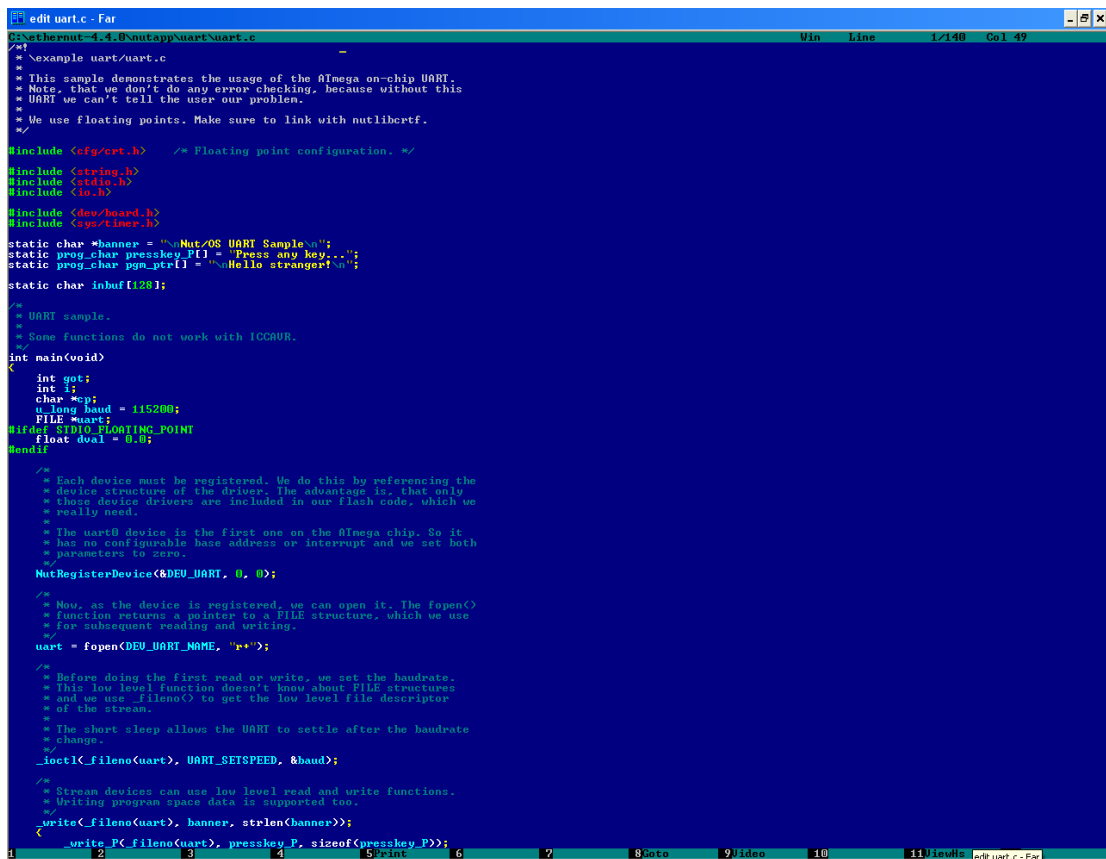
1. Pāriet uz (Nut/OS)->nut->app, atvērt priekš rediģēšanai failu *Makeburn.avr-dude*;
2. Tajā failā izmainīt COM porta numuru uz to, pie kura ir pieslēgts programmators (*BURNPORT=com4*, ja programmators ir pieslēgts pie com4);
3. Izmainīt rindiņu *AVRDUDE_PROGRAMMER=stk500v2* uz *AVRDUDE_PROGRAMMER=stk500* ;
4. Saglabāt uz aizvērt failu;

Tagad mūsu programmēšanas vide un Charon II ir gatavi darbam.

Pāriet uz (Nut/OS)->nutapp mapīti, tur jūs redzēsiet daudzās mapītes – tas ir programmu piemēri, kurus mēs kompilējam no *Nut/OS Configuratora* (*Build->Create Sample Directory*). Atveriet „uart” mapīti.

Ieteicams lietot *Far Manager* programmu ar instalētu „Colorer” spraudni (angl. plugin). Tad no tā ļoti viegli palaist kompilēšanas komandas, un rediģēt jūsu programmu, jo „Colorer” atbalsta C koda pagaismošanu.

Atveriet *uart.c* failu, jūs redzēsiet standartu C valodas kodu (zīm. 3). Jūs variet izlasīt par šī kodu programmas augšējā pusē. Šajā laboratorijas darba ietvaros, jums vajag tikai nokompilēt dotu kodu.



```
edit uart.c - Far
C:\Program Files\Far Manager\4.4.0\nutapp\uart\uart.c
Win Line 1/140 Col 49

/*
 * Example uart/uart.c
 *
 * This sample demonstrates the usage of the ATmega on-chip UART.
 * Note, that we don't do any error checking, because without this
 * UART we can't tell the user our problem.
 * We use floating points. Make sure to link with nutlibrtf.
 */

#include <config.h> /* Floating point configuration. */
#include <string.h>
#include <stdio.h>
#include <io.h>
#include <device.h>
#include <sys/timer.h>

static char *banner = "Nut/OS UART Sample\n";
static prog_char presskey_P1 = "Press any key...";
static prog_char pgm_ptr1 = "Hello stranger!\n";
static char inbuff[256];

/*
 * UART sample.
 *
 * Some functions do not work with ICCAUR.
 */
int main(void)
{
    int got;
    int i;
    char *cp;
    unsigned long baud = 115200;
    FILE *uart;
    float dval = 0.0;

    /*
     * Each device must be registered. We do this by referencing the
     * device structure of the driver. The advantage is, that only
     * these device drivers are included in our flash code, which we
     * really need.
     *
     * The uart0 device is the first one on the ATmega chip. So it
     * has no configurable base address or interrupt and we set both
     * parameters to zero.
     */
    NutRegisterDevice(&DEV_UART, 0, 0);

    /*
     * Now, as the device is registered, we can open it. The fopen()
     * function returns a pointer to a FILE structure, which we use
     * for subsequent reading and writing.
     */
    uart = fopen(DEV_UART_NAME, "r+");

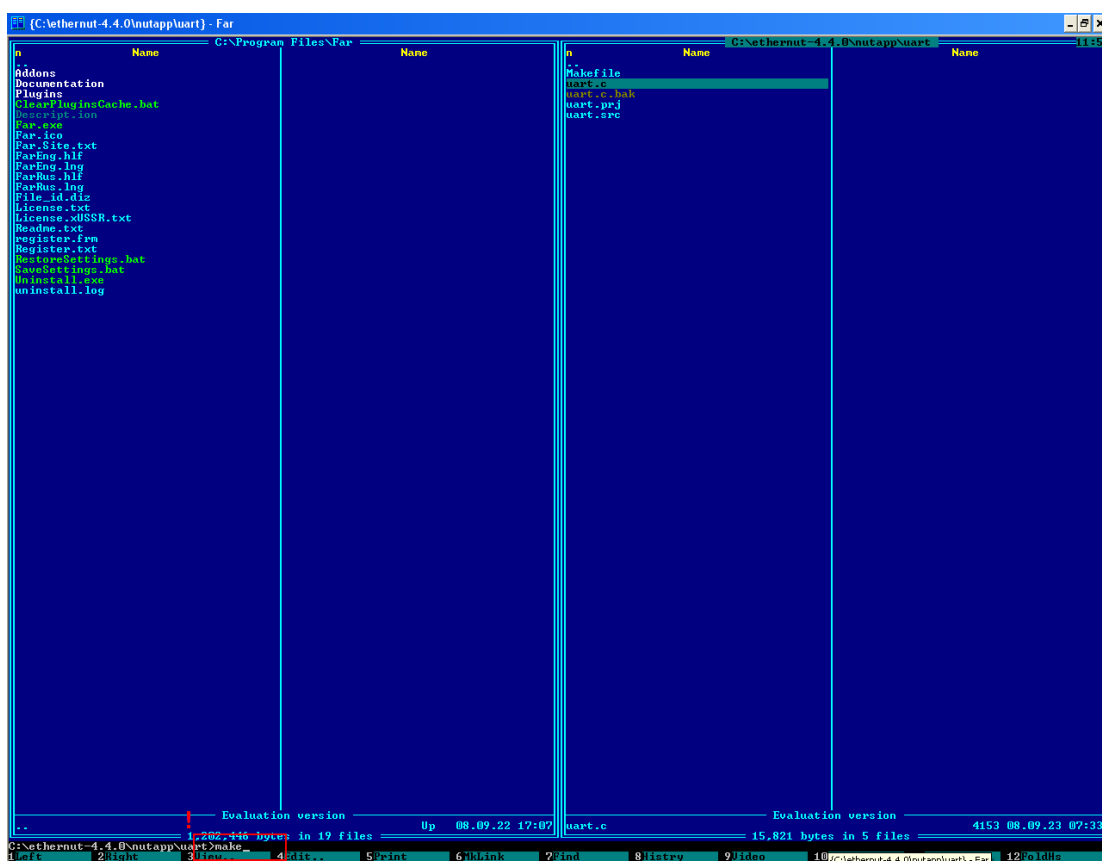
    /*
     * Before doing the first read or write, we set the baudrate.
     * This low level function doesn't know about FILE structures
     * and we use _fileno() to get the low level file descriptor
     * of the stream.
     *
     * The short sleep allows the UART to settle after the baudrate
     * change.
     */
    _ioctl(_fileno(uart), UART_SETSPEED, &baud);

    /*
     * Stream devices can use low level read and write functions.
     * Writing program space data is supported too.
     */
    write(_fileno(uart), banner, strlen(banner));

    write_P(_fileno(uart), presskey_P, sizeof(presskey_P));
}
```

Zīmējums 3. uart.c fails, atvērts no Far Manager

Lai izdarītu programmas kompilēšanu no mapītes, ar programmu un *makefile*, jāizsauc komandu **make** (zīm. 4.)



Zīmējums 4. Programmas kompilēšana ar make komandu.

Ja programmas kodā nebija pielaistas kļūdas, tad mapītē ar programmas kodu parādīsies jauni faili, un kompilēšanas brīdī uz ekrāna nebūs izvadīti paziņojumi par kļūdām.

Ja kompilēšana bija pabeigta veiksmīgi, tad jāpalaiž **make burn** komanda, kura iesūtīs programmas kodu (iepriekš sakompilētu) Atmega128 FLASH atmiņā. Viss, jauna programma tiek iesūta mikrokontrolerī. Palaidiet *Putty* vai *HyperTerminal* (ja viņš nav vēl atvērts), un pieslēdzieties pie Charon II plates, tā kā bija aprakstīts „komunikācijas nodrošināšanas” sadaļā, bet ievadot tādus parametrus:

- g. Connect using: (Izvelēties to COM portu, pie kurā tiek pievienots datu kabelis)
- h. Bits per second: 115200**
- i. Data bits: 8
- j. Parity: None
- k. Stop bits: 1
- l. Flow control: None

Uz termināla ekrāna jābūt paziņojumam par programmu („Nut/OS UART Sample”), programma paprasīs nospiegt jebkuru tastatūras taustiņu („Press any key...”), un pēc tam pajautās ievadīt jūsu vārdu („Enter your name:”). Ievadiet to, un nospiediet Enter pogu. Rezultāta programma jūs pasveicinās ar „Hello (jūsu vārds)!”, vai ar „Hello stranger!”, ja nekas nebija ievadīts.

Ja viss panākts veiksmīgi, līdz ar to beidzas 1. laboratorijas darbs.

Laboratorijas darbs 2

Uzdevums

Izmantojot *uart.c* programmu pārrakstīt viņu sekojošā veidā:

- Izveidot trīs buferus (*char* tipa masīvus);
- pieprasīt no lietotāja pirmo skaitli;
- pieprasīt no lietotāja otro skaitli;
- pieprasīt no lietotāja matemātiskās operācijas zīmi (+, -, *, /);
- secināt par operācijas zīmi, izskaitļot vērtības un izvadīt rezultātu un terminālu (kopā ar ievadītājiem cipariem, piem. $2+2=4$);

Laboratorijas darba mērķi

- saprast, kā uz C valodas programmēt mikrokontrollerus;
- saprast, kā izmantot iebūvējamu OS, kā lietot viņas funkcijas;

Teorija

Kompilēšanas norādījumi

Vispirms, jāsaglabā *uart.c* faila kopija, zem Far Managera to var panākt, ievietojot mārkeru virs *uart.c* faila un nospiežot Shift+F5, un ievadot jaunu faila vārdu (peim. *uart.c.bak*). Kad rezerves kopija tiek uztaisīta, jāpārsauc *uart.c* fails (nospiežot taustiņu F6) šāda veidā:

uart_JŪSUVĀRDS.c. Pārsaucot, pamēģiniet sakompilēt programmu (komanda **make** komandas rindā), lai ieraudzētu kļūdas no Far Managera nospiediet taustiņu kombināciju Ctrl+O – šī kombinācijas paslēps logus un jūs redzēsiet savas kļūdas (nospiediet Ctrl+O vēlreiz, lai atgriezt logus). Ekrāna lejā būs redzams uzraksts (*make.exe ceļš*): ***** No rule to make target 'uart.o', needed by 'all'. Stop.** Šis uzraksts liecina par to, kā ar makefilu kaut kas nav kartībā.

Atveriet failu *Makefile* uz rediģēšanu (ievietojiet mārkeru virs *Makefile* un nospiediet F4), pāriet uz faila pēdējām rindām. Sameklējiet tādu rindu: „**PROJ = uart**” – to jāpamaina uz „**PROJ = uart_JŪSUVĀRDS**”, citādi utilīta *make.exe* meklēs jūsu projekta mapītē failu ar nosaukumu *uar.c*, bet jūs jau pārsauciet to par *uart_JŪSUVĀRDS.c*, nav dīvaini, kā tāda faila nav un kļūdas paziņojums tiek atgriezts.

Kad fails *Makefile* pieredīgēts, pamēģiniet vēlreiz nokompilēt programmu ar komandu **make**.

Visam jānotiek pareizi.

Lai izdzēst sakompilētu kodu (.elf, .hex, .lst, .map, .o, .bin failus), pie tām jūsu programmā nebūs aiztikta, jāizpilda komanda: **make clean**.

Programmas koda apraksts

Pirms koda analizēšanas atcerēties, ka C valodā ir savi svarīgi momenti, kurus jāatceras rakstot programmu:

1. Programma sākas ar funkciju „*main*”
2. Lietojiet `#include <kaus.h>` direktīvus, ja gribiet izmantot funkcijas no citiem C valodas kodiem;
3. Globālus mainīgus vajag definēt ārpus funkcijām.
4. Ja mainīgais (-ie) ir definēts (-i) lokāli (funkcijas ietvaros), tad funkcijai pabeidzot strādāt, šie mainīgie tiks likvidēti (nebūs pieejami un dati tajos būs nederīgi).
5. Ja mainīgam/funkcijai priekšā pievienot modifikatoru *static*, tad viņi būs pieejami tikai tā objekta ietvaros, kur ir definēti. Ja ar *static* ir definēta funkcija, tad viņa būs redzama tikai dota faila ietvaros. Ja mainīgais būs lokāli (iekš funkcijas) definēts un viņam būs *static* modifikators, tad šis mainīgais būs inicializēts tikai vienu reizi (pirmā funkcijas izsaukšanas reizē), un otrreiz, izsaucot funkciju, šis mainīgais nebūs inicializēts (viņš saturēs veco vērtību).

Tā kā šī laboratorijas darba uzdevums ir aprēķināt lietotāja ievadītu informāciju, kuru jāievieto trīs masīvos, tad ir jēga atgādināt par vienkāršāku datu struktūru: **masīviem**.

Masīvi

Masīvs ir salikts objekts, sastājošs no objektiem-komponentēm, kurus dēvē par viena un to paša tipa elementiem. Parasta masīva noteikšana:

```
Datu tips x[n1][n2]...[nk],
```

Kur **x** – identifikators, kurš noteic masīva nosaukumu, **ni** – masīva dimensija. **k** masīvs ir dēvēts par **k – dimensiju masīvu**, ar **datu tipa** tipa elementiem.

Piemēram:

```
int page[10]; /*viendimensiju masīvs, no 10 no 0 līdz 9
              sanumurētiem (ne vērtības) elementiem*/
char line[81];
float big[10][10], sales[10][5][8]; /*divu dimensiju un
                                     trīs dimensiju
                                     masīvi*/
```

Atsauci uz **k dimensiju x** masīva elementu taisa šādi:

```
page[5]
line[i+j-1]
big[i][j]
```

Tas ir obligāts minimums, ko jāatceras, pildot šo laboratorijas darbu.

Atveriet jūsu programmas kodu priekš rediģēšanai. Faila sākumā ir redzams komentārs (teksts iekļauts `/*` un `*/` simbolos), kur ir rakstīts, par piedāvātu piemēru (programmu). Zemāk seko `#include <kauk_kas.h>` direktīvas, kuras parada kompilatoram, kur meklēt trūkstošas funkcijas. Tālāk ir globālo mainīgo deklarēšana un inicializēšana. Un mūsu galvenā funkcija – **main funkcija** (no kuras sākas programmas izpilde). Par viņu parunāsim atsevišķi.

Galvena funkcija

Funkcijas sākumā ir definēti mainīgie, kuri tālāk būs izmantoti projektā:

```
int got;
int i;
char *cp;
u_long baud = 115200;
FILE *uart;
#ifdef STDIO_FLOATING_POINT
float dval = 0.0;
#endif
```

Ka arī, te ir izmantots makroapzīmējums `#ifdef`. Tam nepievēršat uzmanību, jo tas mūs neinteresē un to mēs neizmantosim.

Tālāk kodā tiek izmantota Nut/OS API funkcija:

```
NutRegisterDevice(&DEV_UART, 0, 0);
```

Katrai ierīcei jābūt reģistrētai, tas ir izdarīts lai iekļautu jūsu rakstītā kodā (programmā) tikai tas ierīces draiverus, kuru jūs izmantojāt.

Kad ierīce (mūsu gadījumā tas ir UART ierīce) ir reģistrēta, mēs to varam atvērt.

```
uart = fopen(DEV_UART_NAME, "r+");
```

`fopen()` funkcija atgriež rādītāju (angl. pointer) uz FILE tipa struktūru, kuru Nut/OS izmanto priekš datu apmaiņai (sūtīšana, saņemšana) ar ierīcēm (mūsu gadījumā ar UARTu). Pirms pirmās lasīšanas/rakstīšanas operācijas UARTam jāpiešķir ātrums, kurā notiks komunikācija.

```
ioctl(filen(uart), UART_SETSPEED, &baud);
```

Līdz ar to UART inicializācijas ar Nut/OS palīdzību pabeidzas.

Tālāk kodā tiek parādītas dažas datu sūtīšanas un saņemšanas funkcijas, kurus mēs neaplūkosim, jo priekš tiem nolūkiem izmantosim citas funkcijas.

Pēc lasīšanas/rakstīšanas instrukcijām seko „bezgalīgais” *for* cikls. Programmējot mikrokontrollerus ir pieņemts, savu funkcionālu koda daļu iekļaut „mūžīgajā (bezgalīgajā)” ciklā, jo citādi, pēc programmai izpildes, mikrokontrolleris apstāsies, un atkārtotai programmas izpildīšanai to jāpalaiž no jauna (atvieno un pēc tam pievieno barošanu).

for ciklā tiek veiktas sekojošās darbības:

1. „Enter your name:” teksta rindas sūtīšana caur UARTu un gaide, kamēr visi simboli būs izvadīti;
2. Ievadīta teksta saņemšana (teksts, kuru ievada lietotājs, apstiprinot to nospiežot taustiņu Enter);

```
fputs("\nEnter your name: ", uart); //Izvadīt paziņojumu uz
                                   lietotāja terminālu
fflush(uart); //Gaidīt, kamēr visa informācija būs pārsūtīta
               lietotājam
fgets(inbuf, sizeof(inbuf), uart); //Nolasīt lietotāja
                                   ievadītus datus
```

3. „\n” (pāriet uz nākamo rindu, simbols formējas nospiežot Enter taustiņu) simbola no saņemtas teksta rindas apgriešana;

```
cp = strchr(inbuf, '\n'); //Saņemtā tekstā (inbuf) atrast „\n”
                           simbolu, piešķirt cp tās atrašanas
                           vietas adresi, vai 0, ja „\n” nav
                           atrasts
if (cp) //Ja „\n” bija atrasts (cp != 0 (!= - nav vienāds))
    *cp = 0; //Pozīcijā (adresē) cp, aizvietot simbolu (*cp) ar
            0 (rindas beigšanās kods)
```

4. Pārbaude uz informācijas esamību (vai lietotājs ir ievadījis savu vārdu vai vienkārši nospieda un Enter taustiņu) saņemtā tekstu rindā. Ja kaut kāda informācija tur (saņemtās informācijas buferī) ir, tad izvadīt paziņojumu „Hello <ievadītājs_lietotāja_vārds>”, ja informācijas tur nav, tad izvadīt paziņojumu: „Hello stranger!”

```
if (inbuf[0]) //Ja saņemtā tekstā (inbuf) pirmais simbols ir
              (inbuf[0])
    fprintf(uart, "\nHello %s!\n", inbuf); //Tad izvadīt
                                           paziņojums „Hello
                                           ievadītājs_lietotāja_vārds>”
else { // Citādāk
    fputs_P(pgm_ptr, uart); //Izvadīt paziņojumu „Hello s
                             stranger!”
}
```

5. Te arī ir parādīta iespēja, kā strādāt ar cipariem ar peldošo komatu. Tas teksta rindas mūs neinteresē, līdz ar to, viņus neapskatīsim.

Tagad jums ir nepieciešamās zināšanās lai izpildītu otru laboratorijas darbu.

Laboratorijas darbs 3

Uzdevums

Izmantojot 2. laboratorijas darba programmu, modernizēt to šādā veidā:

- trīs buferu vietā izveidot vienu *char *buf*;
- pieprasīt no lietotāja veselu matemātiskas izteiksmes rindu (2+2);
- ievietot pieprasītu rindiņu iekš *buf*.
- atsekt pirmo un otro skaitli no matemātiskas operācijas simbola;
- secināt par operāciju un izvadīt rezultātu;

Laboratorijas darba mērķi

- saprast darbību ar radītājiem;
- saprast, kā griezties pie I/O portiem un atmiņas apgabaliem;

Teorija

Šajā laboratorijas darbā jāmodificē kods (sakarā ar uzdevumu), kurš bija uzrakstīts trešā laboratorijas darbā.

Datu masīvi un rādītāji

Atgādināsim teoriju par masīviem, lai jums būtu ērtāk to salīdzināt ar radītājiem.

Masīvi

Masīvs ir salikts objekts, sastājošs no objektiem-komponentēm, kurus dēvē par viena un to paša tipa elementiem. Parasta masīva noteikšana:

```
Datu tips x[n1][n2]...[nk],
```

Kur **x** – identifikators, kurš noteic masīva nosaukumu, **ni** – masīva dimensija. **k** masīvs ir dēvēts par **k** – *dimensiju masīvu*, ar *datu tipa* elementiem.

Piemēram:

```
int page[10]; /*viendimensiju masīvs, no 10 no 0 līdz 9
              sanumurētiem (ne vērtības) elementiem*/
char line[81];
float big[10][10], sales[10][5][8]; /*divu dimensiju un
                                     trīs dimensiju
                                     masīvi*/
```

Atsauci uz **k** dimensiju **x** masīva elementu taisa šādi:

```
page[5]
line[i+j-1]
big[i][j]
```

Rādītāji

Par rādītājiem dēvē dota tipa komponentu, kurš ir atsauce uz kādu atmiņas apgabalu. Rādītāja definējums ir šāds:

```
Datu tips *id1, *id2,_, *idn
```

Mainīgo **id1, *id2,_, *idn* tips noteicas kā rādītāju tips uz *datu tipu*. Šie mainīgie kalpo par atsaucēm uz objektiem ar *datu tipu*. Tādu tipu sauc par mainīgo-rādītāju *bāzes tipu*. Zemāk seko daži rādītāju noteikšanas piemēri:

```
int *pi, *qi; /* rādītājs uz veseliem objektiem */  
char *c;      /* rādītājs uz teksta objektu */
```

Atsauce uz iepriekš definētiem objektiem

Rādītāji var nodrošināt atsauci uz iepriekš definētiem objektiem. Tāda objekta adrese var būt noteikta lietojot adresācijas operatoru *&* (*address of operator*). Piemēram, apskatīsim mainīgu *i* un *pi*, definētus kā:

```
int i, *pi;
```

Piešķire:

```
pi = &i;
```

pi ļauj atsaukties uz objektu ar vārdu *i* arī ar rādītāju *pi*, tikai lietojot apzīmējumu **pi*. Šī gadījumā vārdus *i* un **pi* dēvē par pseidovārdiem. Operators *&* arī ir standarts parametra pēc atsauces nodošanas rīks.

Atsauce uz patvaļīgu atmiņas šūnu.

Ar tiešas pārveidošanas palīdzību var dabūt atsauci uz patvaļīgu atmiņas šūnu. Piemēram, pieņemsim, ka *pt* ir *T** tipa atsauce. Tad atsauce uz atmiņas šūnu *0777000* var dabūt sekojoši:

```
pt = (T*)0777000;
```

Vēršana pie konkrētām atmiņu šūnām bieži ir nepieciešams programmās, kuras sadarbojas ar aparatūru, piemēram, ierīču draiveros, kad priekš ierīču vadībai jābūt pieejai pie tādām atmiņu šūnām, kā stāvokļu reģistri vai ierīces buferu šūnas.

Laboratorijas darbs 4

Uzdevums

Šajā laboratorijas darbā ir iespējami divi uzdevumi:

1. Izmantojot 2. laboratorijas darba programmu, modernizēt to šādā veidā:

- trīs buferu vietā izveidot vienu *char *buf* (rezervēt tam atmiņas apgabalu);
- pieprasīt no lietotāja veselu matemātiskas izteiksmes rindu (2+2);
- ievietot pieprasītu rindiņu iekš *buf*.
- atsekt pirmo un otro skaitli no matemātiskas operācijas simbolu;
- secināt par operāciju un izvadīt rezultātu uz LCD;

2. Izmantojot piedāvātu kodu, izveidot programmu:

- Izmānīt LCD simbolu atmiņu tā, lai būtu pieejami latviešu vai krievu valodas burti.

Laboratorijas darba mērķi

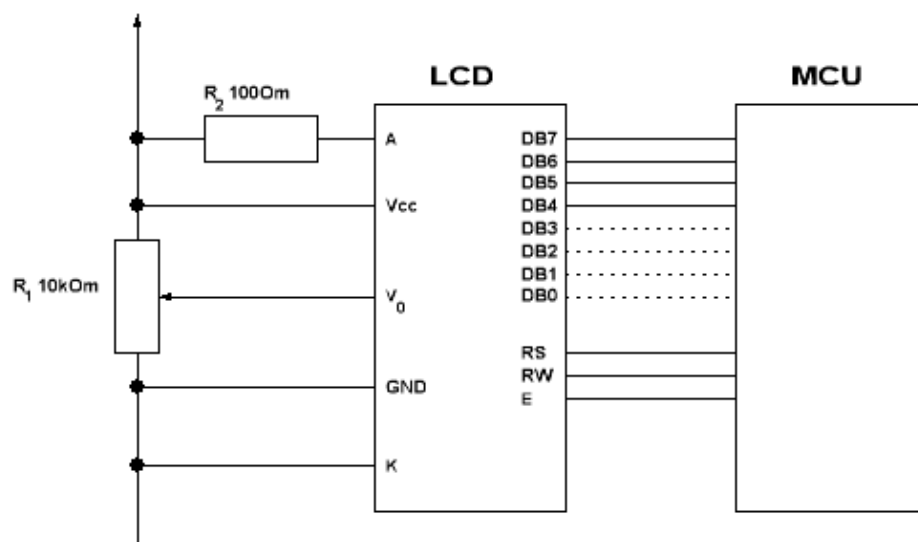
- saprast darbību ar radītājiem;
- saprast kā strādā LCD un kā ar to strādāt;

Teorija

Mūsu gadījumā LCD ir uztaisīts uz HD44780 mikroshēmas bāzes. Šī mikroshēma ir ļoti populāra, jo ir diezgan ātrdarbīga ($F_{osc} = 250\text{kHz}$). Kā arī tā ļauj diezgan viegli ieprogrammēt LCD un pārsūtīt tam datus. Zemāk sekos informācija par HD44780 darbības principiem, kā arī būs aprakstīts realizētas komunikācija algoritms.

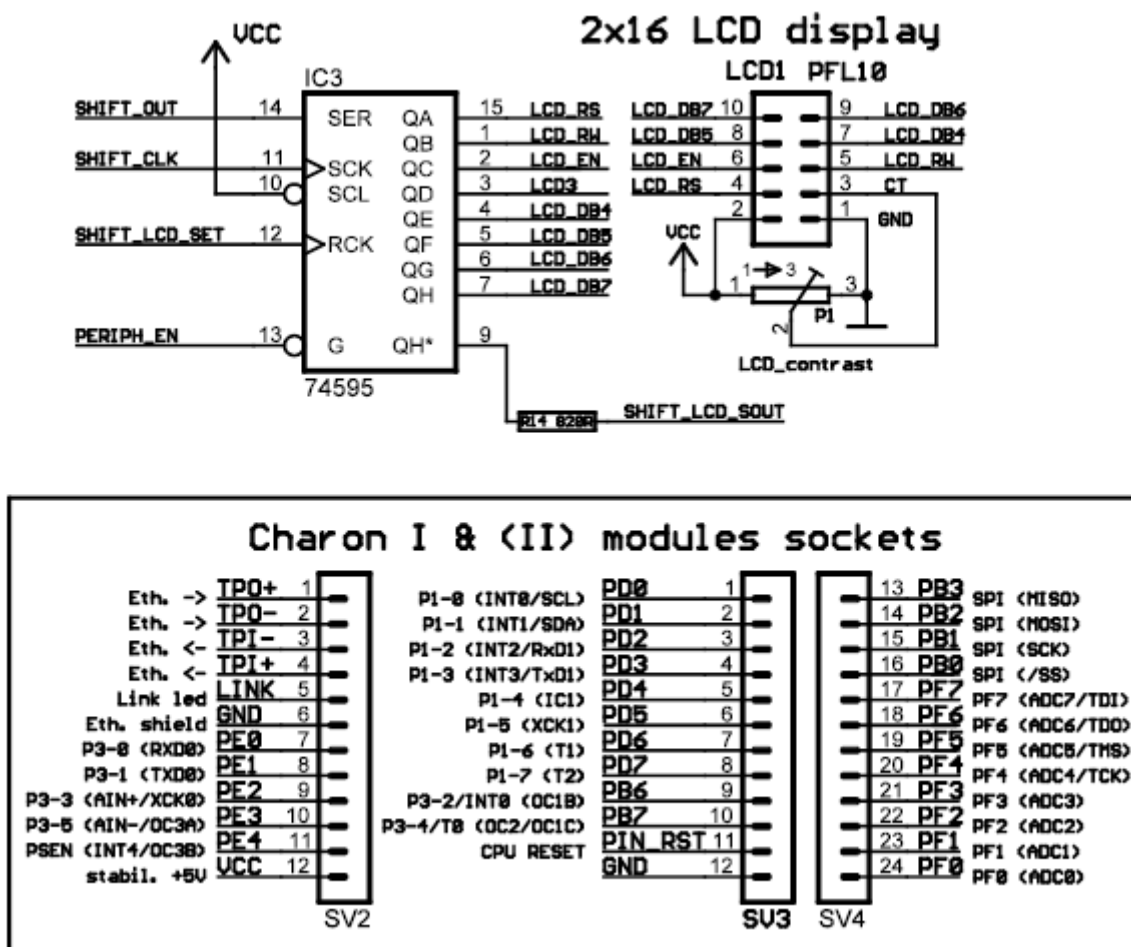
LCD apraksts

LCD uz HD44780 bāzes var būt pieslēgts tieši pie kontroliera portiem un var izmantot gan 4.-bitu gan 8.-bitu paralēlu datu pārraidi (zīm. 1). 4.-bitu pārraidei ir ievērojama priekšrocība – viņa ekonomē četrus kontroliera izvadus.



Zīmējums 1. LCD pieslēgšana pie MCU ar 8.-bitu datu maģistrāli.

Tā kā mēs izmantojam izstrādāšanas plati Charon II, tam ir 4.-bitu pārraides interfeiss ar LCD, kurš ir pieslēgts ne pie kontroliera izvadiem, bet pie secīg-paralēla porta izvadiem (zīm. 2). Tās ir uztaisīts tāpēc, ka izstrādātāji ekonomēja kontroliera izvadus jo bez LCD ir daudz ko pievienots pie Charon II plates.



Zīmējums 2. LCD pieslēgšana pie Charon II plates

LCD uz HD44780 kontroliera loģiska struktūra

Kontrolierim ir savs vadības bloks, kurš apstrādā gan komandas, gan atmiņu. Atmiņa tiek dalīta uz trim veidiem:

DDRAM – displeja atmiņa. Viss ko būs ierakstīts iekš DDRAM būs izvadīts uz displeju.

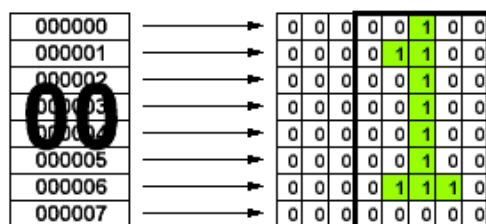
Piemēram, ierakstīsim tur **0x31** – uz ekrāna būs vieninieks („1”), tā kā **0x31** ir „1” ASCII kods.

Parasti **DDRAM** atmiņa ir lielāka apjoma nekā uz displeja redzamais diapazons. Parasti

DDRAM satur **80 šūnas** – 40 vienā rindā, 40 otrā. Arī pastāv kursora jēdziens – tas vieta, kur būs ierakstīts nākamais simbols, t.i. adreses skaitītāja pašreizējā vērtība. Kursors var būt gan redzams, gan neredzams.

CGROM – simbolu tabula. Kad iekš DDRAM tiek ierakstīts baits, no tabulas tiek ņemts simbols un attēlots uz ekrāna. CGROM nedrīkst modificēt.

CGRAM – arī simbolu tabula, bet to ir iespēja mainīt, ierakstot savus simbolus (zīm. 3).



Zīmējums 3. Simbola formēšana iekš CGRAM šūnas

Adresēšana notiek lineāri, t.i. sākumā seko pirmā simbola 8 biti, tie ir ierakstīti viens pēc otra – katrs bits ir punkts uz ekrāna (zīm. 3). Pēc tam otru simbolu un t.t. Tā kā mēs izmantosim **5x8** lielus simbolus, tad **vecāki trīs biti nav vajadzīgi**. Kopumā **CGRAM**ā var būt 8 simboli, attiecīgi **CGRAM** satur **64**. baitu. Šiem programmējamiem simboliem ir kodi no 0x00 līdz 0x08. Tad, ierakstot pirmajos **8. CGRAM baitos** kaut-kādu simbolu un ierakstot **DDRAM**ā nulli (**CGRAM** pirmā simbola adrese) uz ekrāna dabūsim mūsu simbolu.

Komandu sistēma

Par to, kā ir sūtīta komanda displeja kontrolierim paziņos izvads RS=0. Pati komanda sastāv no vecāka bita, kurš noteica par ko atbild dotā komanda un parametru bitiem, noradošiem HD44780 ko izpildīt.

Tabula1. Komandu tabula

DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Vērtība
0	0	0	0	0	0	0	1	Ekrāna notīrīšana. Adreses skaitītāju uz nulles DDRAM pozīciju (atgriezt kursoru sākumpozīcijā).
0	0	0	0	0	0	1	-	Adreses skaitītāju uz 0 (atgriezt kursoru sākumpozīcijā). Nobīdes atgriešana.
0	0	0	0	0	1	I/D	S	Ekrāna un kursora nobīdes uzstādīšana.
0	0	0	0	1	D	C	B	Attēlošanas režīma uzstādīšana.
0	0	0	1	S/C	R/L	-	-	Kursora vai ekrāna nobīde, attiecība pret bitiem.
0	0	1	DL	N	F	-	-	Kopnes līniju skaita, kopnes platuma un simbola izmēra izvēle.
0	1	AG	AG	AG	AG	AG	AG	Pārslēgties uz CGRAM adresāciju un uzdod CGRAM adresi.
1	AD	AD	AD	AD	AD	AD	AD	Pārslēgties uz DDRAM adresāciju un uzdod DDRAM adresi.

Bitu nozīmes apraksts:

- **I/D** – adreses skaitītāja palielināšana vai samazināšana. Pēc noklusējuma ir uzstādīts 0 – palielināšana. Tas nozīmē, ka katrs nākamais baits tiks ierakstīts n+1 šūnā.
- **S** – ekrāna nobīde. Ja uzstādīt 1, tad izvadot jaunu simbolu, ekrāns būs nobīdīts līdz tam, kamēr nebūs sasniegtas DDRAM beigas. Varbūt būs derīgi izvadot ļoti garu teksta rindu.

- **D** – ieslēgt displeju. Ja tajā ierakstīt 0, tad attēlojums pazudis. Bet lai informācija būtu redzama, tad to bitu jāuzstāda vieniniekā.
- **C** – attēlot kursoru. Ierakstot 1 kursors būs redzams.
- **B** – izveidot kursoru melnā kvadrāta veidā.
- **S/C** – kursora vai ekrāna nobīde. Ja ir uzstādīta 0, tad tiek nobīdīts kursors. Ja 1, tad ekrāns. Viena reizē pēc komandas.
- **R/L** – noteic kursora un ekrāna nobīdes virzienu. 0 – pa kreisi, 1 – pa labi.
- **D/L** – datu kopnes platuma noteikšanas bits. 1 – 8 biti, 0 – 4 biti.
- **N** – rindu skaits. 0 – viena rinda, 1 – divas rindas.
- **F** – simbola izmērs. 0 – 5x8 punktu, 1- 5x10 punktu (sastopams reti).
- **AG** – atmiņas adrese **CGRAMā**.
- **AD** – atmiņas adrese **DDRAMā**.

LCD darbības algoritms

Inicializācija

Kā jebkuru ierīci, tā arī LCD jāinicializē. Dažreiz dažiem LCD veidiem ir noklusējuma stāvokli (8 biti, kursoru pārvietot uz sākumu), viņiem vajag tikai padod barošanu. Bet tomēr, labāk inicializēt LCD tā, kā to grib izstrādātājs.

Inicializācijas piemērs:

1. **00111000** – 8. bitu datu kopne, 2. rindas.
2. **00000001** – Ekrāna attīrīšana
3. **00000100** – Adreses palielināšana. Ekrāns nekustas.

1. **00001100** – Ieslēgt displeju (D=1) .
2. **00000001** – Notīrīt displeju. Atmiņas radītājs tiek uzstādīts uz DDRAM.
3. **00010100** – Kursora nobīde (S/C=0) pa labi (R/L=1).
4. **00110001** – Ierakstīt datus (RS līnija = 1) „1” kods 0x31

Darbība ar CGRAM

Pieņemsim, ka displejs ir inicializēts un gatavs datu saņemšanai.

1. **01001001** – Izvelēsimies CGRAMā adresi 0x09 (0x09 – otra simbola sākums, uz vienu simbolu aiziet 8 baiti).
2. **00000001** – Sāka ievadīt datus. (**RS=1**)
3. **00000010** – Zīmēsim zibeni
4. **00000100**
5. **00001000**
6. **00011111**
7. **00000010**
8. **00000100**
9. **00001000** – Pēdējais datu baits
10. **10000000** – Pārslēgsimies uz DDRAM, un uzstādām rādītāju uz **0000000** adresi – pirmais simbols pirmā rindā.
11. **00000010** – Un atkal dati (**RS=1**), kods 02 – tieši viņa mēs ierakstījām mūsu zibeni

LCD kontroliera HD44780 lasīšanas/rakstīšanas algoritms

Portu inicializācija:

1. RS, RW, E – ieejas režīmā.
2. DB7..DB0 – ieejas režīmā

Ierakstīšana komandas

1. RS=0 (komanda)
2. RW=0 (ierakstīšana)
3. E=1 (Uzmanību!)
4. Pauze
5. E=0 (Aiziet!)

Datu ierakstīšana

1. RS=1 (Dati)
2. RW=0 (Ieraksts)
3. E=1 (Uzmanību!)
4. Izvadīt uz portu datus.
5. Pauze.
6. E=0 (Aiziet!)

Ar 4. bitu kopni viss ir tas pats, tikai katra ierakstīšanas/nolasīšanas operācijas ir veikta ar divām strobu maiņām.

Ierakstīšana:

1. E=1

2. Pauze
3. Izvadīt uz portu vecāku tetrādi.
4. $E=0$
5. Pauze
6. $E=1$
7. Pauze
8. Izvadīt uz portu jaunāku tetrādi.
9. $E=0$

Laboratorijas darbs 5

Uzdevums

Jums tiek piedāvāts 1-Wire kopnes apstrādāšanas pirmkods, lietojot funkcijas no šī koda uzprogrammēt temperatūras vērtības nolasīšanu no DS18B20 sensora (izveidot *main* funkciju). Nolasītu vērtību izvadīt uz LCD.

Papilduzdevums: pieprasīt no lietotāja (no termināla) uz kādu interfeisu izvadīt nomērītu temperatūru (uz LCD vai uz terminālu, vai uz abiem).

Laboratorijas darba mērķi

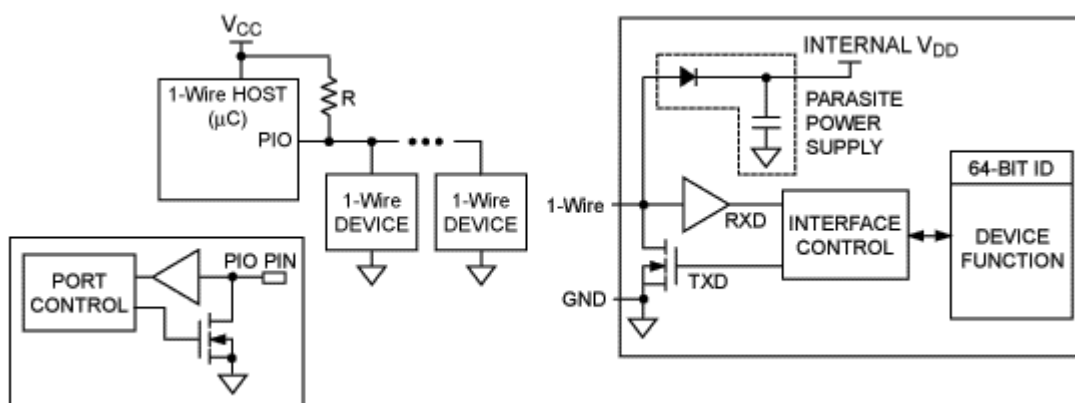
- Saprast interfeisu programmēšanas principus, lietojot mikrokontroleru;
- Saprast 1-Wire interfeisa darbības principus;

Teorija

Uz plates Charon II realizēts 1-Wire interfeiss, bet nav ieprogrammēts. Šī laboratorijas darbā mēs aplūkosim 1-Wire interfeisa darbības principus un pamēģināsim to realizēt.

Kas tas ir 1-Wire tehnoloģija?

1-Wire tehnoloģijas būtība ir **virtnes protokols** izmantojošs vienīgu datu līniju kopā ar zemi, lai nodrošinātu komunikāciju. 1-Wire **vedējs** uzsāk un vada komunikāciju ar vienu vai vairākām 1-Wire **sekotāju** ierīcēm uz 1-Wire kopnes (zīm. 1). Katrai 1-Wire vedējierīcei ir unikāla, neizmaināma uz rūpnīcas iešūta **64-bitu ID** (identifikācijas numurs), kurš kalpo par ierīces adresi uz 1-Wire kopnes. 8-bitu saimes kods, 64-bitu ID apakškopa, nosaka ierīces tipu un funkcionalitāti. Pārsvārā 1-Wire vedējierīce strādā barošanas diapazonā no 2.8V līdz 5.25V. Vairākām 1-Wire ierīcēm nav barošanas izvada, viņi barojas no 1-Wire kopnes (parazītiska barošana (angļu Parastie supply)).



Zīmējums

s 1. 1-Wire vedēja/sekotāja konfigurācija izmanto vienu datu līniju kopā ar zemi (GND).

Kas īpašs ir 1-Wire tehnoloģijā?

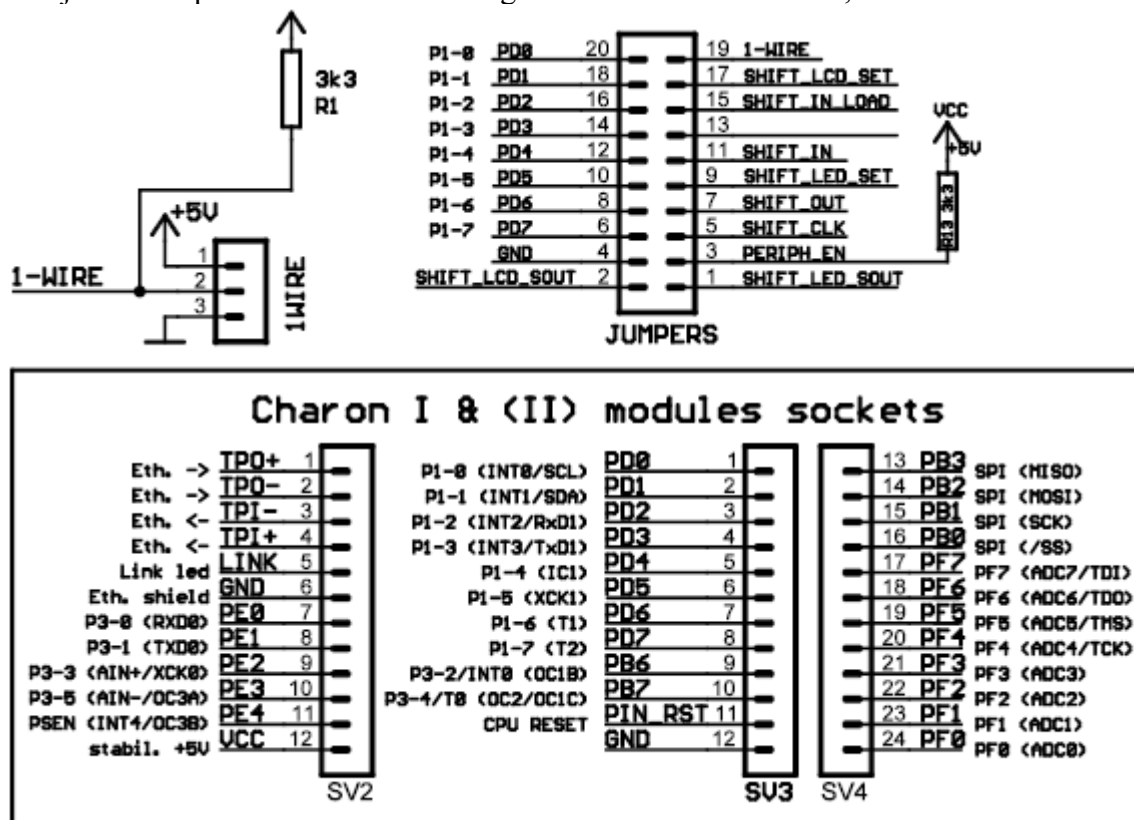
1-Wire vienīga uz voltāžas bāzēta sistēma, kura izmanto divus kontaktus, dati un zeme, half-duplex divvirzienu komunikācijas realizēšanai. Salīdzinājumā ar citiem virknes interfeisiem, tādiem kā I²C vai SPI™, 1-Wire ierīces izstrādāti lai izmantotu saskarnei ar apkartēju vidi.

Temperatūras sensors DS18B20

DS18B20 termometrs nodrošina no 9. bitu līdz 12. bitu Celsiju temperatūras mērīšanu un satur trauksmes funkciju. DS18B20 strādā uz 1-Wire kopnes, kurai pēc definīcijas vajag tikai vienu datu līniju (un zeme) komunicējot ar mikroprocesoru. Sensors var strādāt temperatūras diapazonā no -55°C līdz +125°C ar precizitāti ±0.5°C pie nomērītas temperatūras diapazona no -10°C līdz +85°C.

Kā bija aprakstīts agrāk, katram DS18B20 ir unikāls 64-bitu sērijas kods, kurš ļauj daudzām DS18B20 ierīcēm komunicēt uz vienas 1-Wire kopnes. Tāpēc ir ļoti vienkārši izmantot vienu mikrokontrolieru lai vadītu daudzus DS18B20 sensorus izplātītus lielā laukumā. Pielikumi ir daudzās nozarēs, piemēram, temperatūras kontrole ēkās, aprīkojumā, HVAC (*Heating, Ventilation, & Air Conditioning*) vadībā. Šī kopnes sistēmā mikroprocesors (vedēja ierīce) identificē un adresē ierīces uz kopnes, izmantojot katras ierīces unikālu 64-bitu kodu. Tā kā katrai ierīcei ir unikāls kods, tad adresējamo ierīču skaits virtuāli var būt neierobežots.

Zīmējumā 2. ir parādīta 1-wire ar Atmega128 savienošanas shēma, kura ir realizēta mūsu platē.



ms 2. 1-Wire „dzelziska” realizācija Charon II platē.

Zīmēju

Temperatūras sensors DS18B20 atmiņa

1-Wire kopnes darbības princips

Signālu pārraide kopnē tiek sadalīt laiku slotos, kuru ilgums ir 60 mks. Vienā laiku slotā tiek pārraidīts tikai viens bits. Tā kā sekotāju datu apstrāde var ilgst ļoti ilgi, jāievēro visas laiku robežas, aplūkotas tālāk.

Kopnes pamata signāli

Katra bita pārsūtīšana tiek realizēta tikai ar vedēja palīdzību. To var panākt, uzstādot zemu līmeni uz kopnes, kurš sinhronizē visu parējo ierīču loģiku. Sakaram caur 1-Wire kopni pastāv piecas komandas: „Ierakstīt loģ. 1.”, „Ierakstīt loģ. 0.”, „Lasīšana”, „Atgriešana”, „Klātiešana”.

Signāls „Ierakstīt log. 1.”

Signāls „Ierakstīt log. 1.” tiek parādīts zīmējumā 3. Vedējs uzstāda zemu līmeni uz laiku 1..15 mks. Pēc tam, palīkušajā laika slota laikā atbrīvo kopni.



Zīmējums 3. Signāls „Ierakstīt log. 1.”

Signāls „Ierakstīt log. 0.”

Signālu „Ierakstīt log. 0.” tiek parādīts zīmējumā 4. Vedējs formā zemu līmeni laikā ne mazāk, ka 60 mks, bet neilgāk par 120 mks.



Zīmējums 4. Signāls „Ierakstīt log. 0.”

Signāls „Lasīšana”

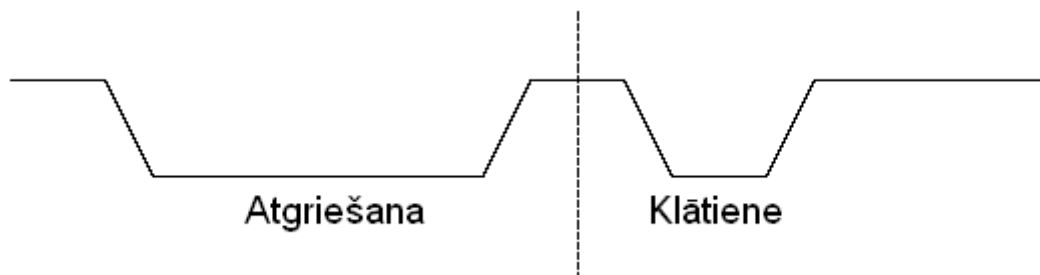
Signāls „Lasīšana” tiek parādīts zīmējumā 5. Vedējs uzstādā zemu līmeni uz laiku no 1 līdz 15 mks.. Pēc tam sākotājs uztur līniju zemā līmenī, ja grib nodod loģisko 0. Ja ir nepieciešams nodod loģisko 1., tad viņš vienkārši atbrīvo līniju. Kopnes skanēšanu jāizdara pēc 15 mks pēc zema līmeņa uzstādīšanas uz līnijas.



Zīmējums 5. Signāls „Lasīšana”

Signāli „Atgriešana” un „Klātiene”

Signāli „Atgriešana” un „Klātiene” tiek parādīti zīmējumā 6. Ievērojiet to, ka šo impulsu laika intervāli ir atšķirīgi. Vedējs uzstāda zemu līmeni 8. laika slotu laikā (480 mks), bet pēc tam atbrīvo kopni. Tāda ilguma signāls saucas par „Atgriešanu”. Jā uz kopnes ir sēkotājs, tad viņam jāatbild uz signālu (pēc ta) ar 60 mks ilgu kopnes uzturēšanu zema līmenī – tāda atsaukšana saucas par signālu „Klātiene”. Ja tāda signāla nav, vedējam jāuzskata, kā uz kones nav neviena sēkotāja.



Zīmējums 6. Signāli „Atgriešana” un „Klātiene”

ROM funkciju konadas

Katrai mikroshēmai (M) no 1-wire saimes ir ROM, kurā ir saglabāts unikāls 64.-bitu identifikācijas kods (IK). Šis kods var būt izmantots priekš kadas, uz kopnes esošas, konkrētas M adresēšanai vai identificēšanai. IK sastāv no trīs daļām: 8 saimes koda biti, 48 sērijas numura biti, 8 CRC koda biti, kurš tiek izskaitļots no pirmiem 56. bitiem. Arī te ir neliels komandu skaits, kuras strādā ar 64. bitu IK. Tādas komandas tiek dēvētas par ROM funkciju komandas. Tabulā 1 tiek piedāvāts šādu komandu saraksts.

Tabula 2. – ROM komandas.

Komanda	Kods	Uzdevums
READ ROM (ROM lasīšana)	33H	Identifikācija
SKIP ROM (izlaist ROM)	CCH	Adresācijas izlaišana
MATCH ROM (ROM sakrišana)	55H	Pakļautas ierīces adresācijas
SEARCH ROM (ROM meklēšana)	F0H	Identifikācijas datu par visām ierīcēm uz kopnes saņemšana.
OVERDRIVE SKIP ROM	3CH	Paātrināta SKIP ROM versija
OVERDRIVE MATCH ROM	69H	Paātrināta MATCH ROM versija

Komandu apraksts

„ROM lasīšanas” komanda

„ROM lasīšanas” komanda var būt lietota ar vienu sekotāju uz kopnes, lai nolasītu viņu 64. bitu IK. Jā uz kopnes ir vairāki sekotāji, tad komandas rezultāts būs loģiskais UN starp visu ierīču IK.

„ROM izlaišanas” komanda

„ROM izlaišanas” komanda var būt lietojama, ja nav vajadzības konkrētas ierīces adresācijā (pēc viņa unikāla IK). Kopnē ar vienu sekotāju „ROM izlaišanas” komanda ir vispiemērotākā šo sekotāja adresēšanai. Kopnē ar vairākiem sekotājiem šī komanda var būt izmantota visu ierīču adresēšanai.

Tas ir ērti, ja pastāv kopējas komandas atsūtīšanas vajadzība, piemēram, vienlaicīga temperatūras pārveidošanas palaišana visos sensoros.

„ROM sakrišanas” komanda

„ROM sakrišanas” komanda tiek izmantota konkrēta sekotāja adresācijai (uz kopnes). Pēc „ROM sakrišanas” komandas izpildīšanas tiek pārraidīts 64. bitu IK. Pēc pabeigšanās, tikai šim sekotājam (kurš saņēma savu IK) tiek atļauts atbildēt pēc nākoša „atgriešanas” impulsa saņemšanas.

„ROM meklēšanas” komanda

„ROM meklēšanas” komanda tiek izmantota, ja sekotāju identifikatori iepriekš nav zināmi. Komanda atļauj atrast visu ierīču, pieslēgtu pie kopnes, identifikatorus.

DS18B20 funkcionālas komandas

Tabula 2. - DS18B20 funkcionālas komandas

Komanda	Apraksts	Kods	1-Wire kopnes darbība pēc komandas izpildes	Piezīme
Temperatūras pārveidošanas komandas				
„Convert T” (Pārveidot t°)	Inicializē temperatūras pārveidošanu	44h	DS18B20 pārraida temperatūras pārveidošanas statusu vedējam	1
Atmiņas komandas				
„Read Scratchpad” Brīžatmiņas lasīšana	Lasīt iekšējo brīžatmiņu kopā ar CRC baitu.	BEh	DS18B20 pārraida vedējam līdz pat 9. datu baitus.	2
„Write Scratchpad” Rakstīšana brīžatmiņā	Ierakstīt datus brīžatmiņas 2., 3., un 4. (T _H , T _L un konfigurēšanas reģistrā) baitos.	4Eh	Vedējs pārraida 3 datu baitus DS18B20.	3
„Copy Scratchpad” Kopēt brīžatmiņu	Kopēt T _H , T _L , un konfigurācijas reģistrus no brīžatmiņas EEPROMā.	48h	Nav	1

Piezīmes:

1. DS18B20iem ar parazītisku barošanu, vedējam jāaktivizē spēcīgu pullup rezistoru 1-Wire kopnē temperatūras konvertācijas un no EEPROM uz brīžatmiņu kopēšanas laikā. Nekādam citai aktivitātei nedrīkst būt kopnē.
2. Vedēj var pārtraukt datu pārsūtīšanu jebkurā brīdī izsaucot „atgriešanas” komandu.
3. Visiem trīs baitiem jābūt ierakstītiem pirms „atgriešanas” komandas izsaukšanās.

Tipiska saites seansa secība

Visas vienvada ierīces ievēro saites pamatsecību:

1. Vedējs atsūta „atgriešanas” impulsu.
2. Sekotājs(i) atbild ar „klātienēs” impulsu.
3. Vedējs atsūta ROM komandu, adresējas vai pie vienas vai pie visām sekotājierīcēm.
4. Vedējs atsūta atmiņas komandu.

Pievērsiet uzmanību tam, kā pāriešanai pie nākamā soļa, iepriekšējām jābūt pabeigtām. Bet nav nepieciešamības visas komandas secības pabeigšanā. Piemēram, pēc ROM komandas var atsūtīt jaunu „atgriešanas” komandu līdz ar to iniciēt jaunu saites seansi.