Linux a gyakorlatban: mikrovezérlő-programozás

A Linux kapcsán legtöbbször vagy a számítógépes hálózati kiszolgálókról, vagy a felhasználói programokról esik szó.

Az a tény, hogy a számítógép elsősorban eszköz, eléggé háttérbe szorul. Az alábbiakban egy gyakorlati feladat megoldásának ürügyén mutatom be a nyílt forrású programok használatát: egy PIC mikrovezérlő program megírását, kipróbálását és eszközbe töltését követhetjük figyelemmel.

A feladat és az eszköz

Munkám során egy irányítástechnikai feladat – adatgyűjtés, egyszerű folyamatvezérlés – merült fel. Általában a Microchip által gyártott PIC mikrovezérlőket szoktam alkalmazni, mivel sokféle változat közül könnyen lehet az adott feladathoz illőt találni. Az újakban ráadásul flashmemória van, ami lehetővé teszi a törlés nélküli sokszoros (legalább ezer) újraprogramozást, így könnyebb a programfejlesztés, mert az alakuló programot magában az eszközben is ki lehet próbálni, nem csak a szimulátorban.
Mivel kis sorozatú vagy egyedi készülék esetén a lapkák közötti kisebb árkülönbség nem játszik jelentős szerepet, az egyik legnagyobb tudású mikrovezérlőt választottam, a 16F877-et.
A Microchip cég oldaláról le lehet tölteni egy fejlesztőrendszert az általa gyártott mikrovezérlőkhöz. Ez tartalmazza az assemblert, a különböző általuk gyártott vagy ajánlott programozó készülékek illesztőfelületét, valamint az összes vezérlőhöz szimulátort. Emellett a honlapjukon minden eszköz adatlapja (pdf formátumban), példaprogramok és tanulmányok is fellelhetők. Ugyanakkor két hátrányt meg kell említenem: a programok csak Windows alatt működő változatban léteznek, valamint az assembleren kívül más programnyelv csak külön díj ellenében szerezhető be.
A fentiek miatt elhatároztam, hogy megpróbálok találni valamilyen programot, eszközt, ami lehetővé teszi a könnyebb programfejlesztést, esetleg Linux alatt is.

A gputils programok és a gpsim

A keresés nem várt gyors eredménnyel járt, ugyanis a ➲ http://www.gnupic.org címen ***Scott Dattalo*** igen jó, PIC mikrovezérlőkkel kapcsolatos címlistájára akadtam rá. Ezt az oldalt használhatjuk kiindulópontként a programok letöltésekor. A legalapvetőbb programok a `gputils` csomagban találhatók. Mint kiderült, ezeknek a programoknak az egyik fő fejlesztője szintén Scott Dattalo. A következő főbb programok tartoznak hozzá: a `gpasm`, az assembler-fordító, `gplink` és `gplib`, amik az áthelyezhető kódként fordított programok könyvtárakba való szervezését, majd futtathatóvá szerkesztését teszik lehetővé. A fejlesztők célja az, hogy a gpasm fordítóforráskód szintjén teljes egészében helyettesíteni tudja az eredeti Microchip programot, az `mpasm`-t. Mivel a forráskód szabad, tetszőleges rendszerre lefordítható, amit Linux, Mac és Windows esetén már meg is tettek. Eddig eljutva jó érzéssel töltött el, hogy Linux alatti programfejlesztéshez is léteznek programok, azonban a fent vázolt célhoz nemigen jutottam közelebb. Szerencsére mindjárt a gputils honlapján megtaláltam (amúgy a gnupic oldalon is elérhető) a következő lépéshez vezető utat: a gpsim honlapjának a hivatkozását. Ez tulajdonképpen nem része a gputils csomagnak, de mivel ezt is Scott fejlesztette, nem csoda, hogy szervesen illeszkedik a gpsimhez. A célja az, hogy más fordítóprogramokkal és más mikrovezérlőkkel is együtt tudjon működni, de egyelőre a PIC-változat a legkidolgozottabb. Ezekből a legtöbb gyakran használt típust tudja szimulálni, nagyrészt kifogástalanul, az összes külső eszközzel együtt.

Néhány szó a gpsimről

A `gdb` hibakeresőhöz hasonlóan alapvetően parancssori működésű, azonban a gtk-extra csomagra épülve grafikus felületet készítettek hozzá (ha ezt akarjuk használni, X11 alatt indítsuk a programot). Az alapvető hibakereső szolgáltatások megtalálhatóak benne: léptetés, függvényátugrás, töréspont és futtatás. A mikrovezérlő tulajdonságainak megfelelően megtekinthetjük a programmemória, az adatmemória, vagyis a regiszterek (mivel ezek a vezérlők Harvard felépítésű RISC processzorokat tartalmaznak, így a kettő teljesen különálló, sőt ebben az esetben bitszélességük sem azonos), az esetleges belső EEPROM memória, valamint az áramkör lábainak értékét, illetve állapotát. Emellett külön állományban meg lehet adni az úgynevezett stimulusokat, amik a külső világot hívatottak szimulálni a mikrovezérlő számára. Ilyenek például az adatok a soros vonalon, a bemenetek változása, a számláló impulzus. A lefuttatott szimulációs ciklusról részletes nyomkövető állományt tud előállítani, amit utólag (mintegy offline) tanulmányozni lehet. Ezzel a programok fordítása és hibakeresése tekintetében gyakorlatilag elértem azt a szintet, amit az eredeti Microchip programokkal, és mindezt Linux alatt. Azonban még nem találtam megoldást a program eszközbe (lapkába) történő írására. Nos, ilyenre is rá lehet akadni a gnupic oldalon.

A beprogramozó

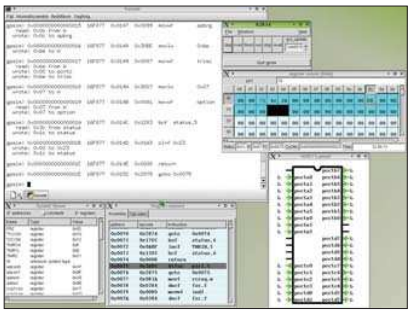
Többféle megoldást is fellelhető: saját építésű programozó (nyomtatókapus és USB-változat), valamint nyomtatókapus gyári programozókhoz való program, és nagy örömömre található Linux program a Microchip Picstart Plus soros illesztésű programozójához is. Annak idején vásároltam is egy ilyen programozókészüléket az alábbi előnyök kedvéért:

- viszonylag olcsó,
- szervesen illeszkedik a Microchip fejlesztőrendszerbe (ez Linux alatt már nem játszik szerepet),
- a Microchip honlapjáról mindig letölthető a legújabb gyári program (firmware), amit egy 17C44 vezérlőbe beprogramozva és ezt a készülékbe helyezve a legújabb gyártmányú lapkákhoz is mindig használható marad.

Korábban bosszankodtam, hogy a cég, jóllehet más tekintetben nagyon megbízható és a kisfelhasználóknak is minden útmutatást megad, a Picstart protokollját nem hozza nyilvánosságra, így linuxos meghajtóprogramját nem tudtam (és eddig más se) megírni. Tulajdonképpen nyomós oka van annak, hogy a cég ezt miért nem teszi meg: így nincs

```
include libnibble.fs  \ karakter --> hexa-szÆmjegyÆtalak t
include piceeprom.fs  \ eeprom mem ria rÆs, -olvasÆs
include picflash.fs       \ programmem ria-kezelØs
include libstrings.fs     \ sz vegek tÆrolÆsa Øs kezelØse

macro

: transmit-byte ( byte -- )
    begin txif bit-clr? while repeat
    txreg !
;

: transmit-nibble ( nibble -- ) nibble>hex transmit-byte ;

target

variable recv-state
variable recv-char
variable recv-dat

: receive-command ( -- )
    begin
        begin rcif bit-clr? while  repeat
        \ '!' vagy '?' parancsot vezet be, a t bbi
          ➥karaktert egyszerßen visszhangozzuk
        rcreg @ dup dup transmit-byte
        [char] ? = if drop 1 recv-state ! recurse exit then
        dup
        [char] ! = if drop 11 recv-state ! recurse exit then
        hex>nibble
        \ elsı adat karakter
        recv-state @ 1 = if
            swapf-tos recv-char ! 2 recv-state ! recurse
            ➥exit then
        \ mÆsodik adat karakter
        recv-state @ 2 = if
         \ olvasÆs
            recv-char @ or ee@ dup [char] = transmit-byte
            ➥4 rshift transmit-nibble 0f and
            ➥transmit-nibble 0 recv-state !
            $d transmit-byte recurse exit then
          \ elsı c m karakter
        recv-state @ 11 = if
            swapf-tos recv-char ! 12 recv-state ! recurse
            ➥exit then
          \ mÆsodik c m karakter
        recv-state @ 12 = if
            recv-char @ or recv-char ! 13 recv-state !
[char] = transmit-byte recurse exit then
          \ elsı adat karakter
        recv-state @ 13 = if
            swapf-tos recv-dat ! 14 recv-state ! recurse
            ➥exit then
         \ mÆsodik adat karakter
        recv-state @ 14 = if
         \  rÆs
            recv-dat @ or recv-char @ ee! 0 recv-state !
            ➥$d transmit-byte recurse exit then
        drop
    again
;
```

a lista folytatását lásd a következő oldalon



A gpsim futás közben

megkötve a keze, és a különböző gyári változatoknak nem kell egymással felcserélhetőnek maradniuk: hiszen mindig csak a pillanatnyi `mplab`, illetve `mpasm`-változathoz kell illeszkedniük. Nos, ***Andrew Pines*** most (jórészt) megoldotta a feladatot, ugyanis egy különleges soros kábelt készített, és miközben Windowsból használta a programozót, rákötött linuxos gépével mentette az adatokat, és ezek révén nagyon sok részt megfejtett a protokollból. Jelenleg adott egy olyan működő program, amellyel a legtöbb programbetöltési feladat megoldható. Olykor előfordulnak hibák, de azért használható.

A program lefordítása egyszerű: használatakor meg kell adnunk a soros kaput, amihez a programozó csatlakoztatva van. Ügyeljünk arra, hogy semmilyen eszköz ne használja ezt. (Azért tudom ilyen pontosan, mert mint kiderült, én elfelejtettem, hogy az egyik soros kapun a modem, a másikon egy soros terminál volt beállítva. Igaz, hogy régóta nem használtam egyiket sem, de a megfelelő programok futottak, és bekavartak az eseményekbe). A programozóban legalább a 3.00.04-es gyári program szerepeljen, ha ennél régebbi van, frissíteni kell (a Chipcad Kft. honlapján akár a felprogramozott 17C44 is megrendelhető). Ha minden össze van kötve, akkor a

```
picp /dev/ttyS0 16f877 -v
```

parancsra kiírja a gyári program változatszámát. A használt soros kapunak és a programozóban lévő vezérlőnek megfelelően módosítani kell a parancsot. Lehetőség nyílik az eszköz programmemóriájának és beállítási szavának olvasására és írására (`-wp`, illetve `-rp` vagy `-wc` és `-rc`), a nem flasheszközök üres állapotának ellenőrzésére és a flasheszközök törlésére. Egyelőre az EEPROM adatmemória írása és olvasása nincs megvalósítva. Az „adatlopó” program is benne rejlik a forrástárban, továb-

a lista folytatása az előző oldalról

```
: print ( -- ) begin str-char dup while transmit-byte
➥repeat drop ;
: nyito ( -- ) c" PIC soros teszt v0.1 (c)
➥Havranek Ferenc " print $d transmit-byte ;

: init ( -- )
    $06 adcon1 !          \ A/D Ætalak t  tiltÆsa
    $ff trisa !           \ porta bemenet
    0 trisb !             \ portb kimenet
    0 portb !             \ alacsony szint (ki)
    $90 rcsta !           \ RS232 engedØlyezØse, folyamatos
                            ➥fogadÆs
    $24 txsta !           \ nagysebessØgß bitrÆta-lØtrehoz
    $0c spbrg !           \ 19200 baud 4 MHz-nØl
    $be trisc !           \ C6 kapu TX
    $07 option_reg !
    0 recv-state !        \ kiindul  Ællapot

;

main : main ( -- ) init nyito receive-command ;

\ LapkabeÆll tÆsok

fosc-hs set-fosc
false set-boden
false set-wdte
```

bá a megfelelő kábel leírása is, így akár magunk is megpróbálkozhatunk az utólagos felderítéssel.
Ezzel már valóban eljutottunk arra a szintre, hogy majdnem minden olyat meg tudunk tenni, amit az eredeti fejlesztőeszközökkel. De a célkitűzésem az volt, hogy valamilyen, a programfejlesztést jelentősen megkönnyítő eszközt is keressek. Nos, hála ***Samuel Tardieu***-nek, találtam is ilyet.

A forth fordító

Mielőtt (röviden) ismertetném a programot, néhány szó a forth nyelvről, mivel az utóbbi időben – megítélésem szerint – érdemtelenül mellőzve van.
A forth egy láncolt kódot előállító értelmező nyelv. Legalábbis az volt, amikor ***Charles Henry Moore*** egy csillagvizsgáló teleszkóp irányításának programozását megkönnyítendő megalkotta az első megvalósítását. Ő egyébként azóta is lelkes híve maradt ennek az ötletnek: már több ilyen elven működő processzort készített, részben a saját maga által alkotott (és szintén forthban írt) áramkörtervező programmal. Érdekességként: az általa alkotott processzor az adott feladatot végrehajtó program futtatása esetén azonos teljesítményt nyújtott, mint egy (a szóban forgó időpontban korszerű) 486-os processzorú gép, miközben a megvalósított tranzisztorszolgáltatások száma a lapkán kevesebb mint annak egytizede volt. A forth nyelv fő jellemzője, hogy úgynevezett szavakból áll, valamint az adatokat egy veremtáron keresztül továbbítja közöttük. Ami igazán széppé teszi, az az, hogy tetszőleges szavakat adhatunk hozzá, és a továbbiakban ugyanúgy használhatjuk őket, mint a nyelv alapszavait (a C-függvényekhez hasonlóan). Mivel a forth értelmező- és fordítóprogramok általában láncolt kódot állítanak elő, rendkívül kis méretű, elfogadható sebességű programot kapunk eredményül. Az ügyesen megírt programok még olvashatók is.
Az általam megtalált program a picforth, tulajdonképpen nem is program a szó ma használatos értelmében, azaz nem önállóan végrehajtható. Inkább parancsfájlnak nevezhetnénk, mivel a futásához szüksége van egy, a futtató gépre készült forth-értelmezőre. Ebben az esetben erre a célra a gforth (GNU forth) használható, ami az ANS forth szabvány egy nyílt forrású megvalósítása.
A program lényegi része a ***picforth.fs*** fájlban bújik meg. Emellett létezik még néhány segédfájl, amikben a mikrovezérlő EEPROM és programmemóriájának írását, illetve olvasását megkönnyítő szavak találhatók, valamint táblázatok és szövegek tárolását és előhívását lehetővé tevő segédletek. Ezenkívül egy rövid leírás mellett több, a forth esetén különösen fontos, működő (azaz kipróbált) példaprogram. A `Makefile` jól használható, végeredményként a picp programmal beégethető ***.hex***, az assembler listát tartalmazó ***.disasm*** és a szavak címeit tartalmazó ***.map*** állományt állít elő.
A ***.hex*** fájl a gpsim programba betölthető (előbb ki kell adni benne a `processor pic16f877` parancsot), és ki is próbálható. Ez látható a képernyőn is az alábbi program (egy soros vonalon keresztül a vezérlő EEPROM memóriájának írását és olvasását egyszerű parancsok segítségével megvalósító) lefordítása után, lásd *listánkon*.
Már csak egyetlen dolog hiányzik: hogy ki tudjuk próbálni a programot. Ehhez természetesen szükség van egy PIC kártyára, 16f877-tel, 4 MHz-es oszcillátorral és megvalósított soros (RS232) ki- és bemenettel. Ezenkívül szükségünk lesz valamire, ami nyers adatot tud küldeni és fogadni (megjeleníteni) a soros vonalon. Némi keresgélés után kiderült, hogy a Linux Programmers Guide példaprogramjai között akad egy egyszerű soros terminálszimulátor program: a miniterm. Ennek forrásába be kell írni a soros kapun kívánt értékeket (melyik például ttyS0, vagy a sebesség: 19200 baud), majd le kell fordítani, és már futhat is!
Mint látjuk, ismét kiderült, hogy a Linux szinte minden feladat megoldására alkalmas, valamint arra is fény derült, hogy legyen bármilyen a program- és eszközeszményképünk, a világban biztosan találunk olyan embereket, akik hozzánk hasonlóan gondolkodnak, hasonló megoldásokat keresnek, és sok esetben munkájuk eredményét közkinccsé is teszik.
Ha valakit érdekel a mikrovezérlő, akár kedvtelésből, egy későbbi cikkemben ismertetek egy olyan kártyatervező és szerkesztőprogramot, amivel a vezérlőnkhöz elkészíthetjük a nyomtatott áramkör tervét. Vajon hol is alkalmazhatjuk őket? Lássuk csak: hőmérsékletmérésre lakásban, udvaron, számítógépházban, de akár lakásautomatizálásra is.

Havránek Ferenc
Automatikamérnökként dolgozik. Kedvtelései közé tartozik mindenféle kétkerekű járművön való közlekedés. Szívesen tölti idejét programozással, nemcsak PC-s, hanem egyéb környezetben.