Tér-idő feldolgozás linuxos stílusban

Ha új generációs vezeték nélküli kommunikációs készülékeket szeretnénk fejleszteni, akkor szükségünk lesz FPGA fejlesztői eszközökre, egy fürtre a szimulációkhoz, valamint egy beágyazott operációs rendszerre a mintapéldányokhoz. A Linux mindegyik célra megfelel.

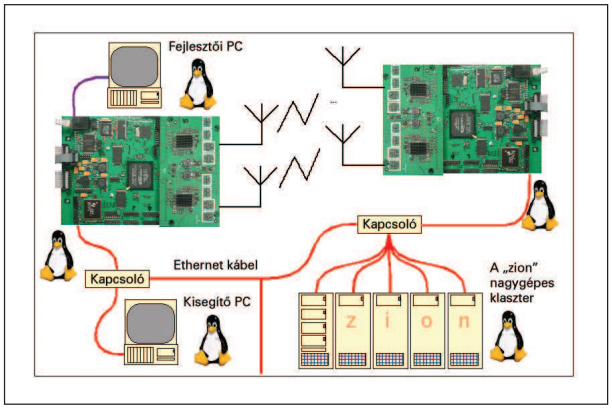
Munkahelyemen, az új-zélandi ***Christchurchben*** lévő ***Tait Electronics Group Researchnél*** az elmúlt időszakban szép csendesen kidolgoztunk egy fejlett vezeték nélküli hálózati megoldást, amit ***tér-idő (space-time, ST)*** feldolgozásnak neveztünk el. Az ***ST***-t sokan a vezeték nélküli rendszerek következő, a harmadik generáció utáni megoldásának tartják. A tér-idő lényege a mögöttes matematika, ebből fakadóan megvalósítása rendkívül bonyolult. Nem egy akadémiai kutató dolgozik ezen a területen, mégis talán mi vagyunk azok, akik ***ST*** tömbkutató (***STAR***) alaprendszerünkre építkezve el tudtuk készíteni a két első gyakorlati megvalósítást. Ezek az időinverziós ***tér-idő blokk-kódolás (time-reversal space-time block coding, TR-STBC)*** és a nyelvtörőnek is beillő nevű ***egyhordozós, adaptív többváltozós döntés-visszacsatolásos, kiegyenlített több bemenetű, több kimenetű (single carrier, adaptive multivariate decision feedback equalised multiple-in, multiple-out, SC-AMV-DFE-MIMO)*** kódolási séma. Kétségtelen, hogy eredményeinket jelentős mértékben a ***Linux*** kiváló teljesítményének és alkalmazkodókészségének köszönhetjük.

A továbbiakban szeretnénk elmondani, a ***Linux*** milyen – amúgy kulcsfontosságú – szerepet játszott matematikai megoldásaink kidolgozásában, illetve hogyan alkalmaztuk beágyazott feldolgozási és futási idejű vezérlési célokra. Olyan szerteágazó témákat fogunk érinteni, mint az ***üzenetátadó felület (message passing interface, MPI)***, a fürtözött adatfeldolgozás, a beágyazott ***Linux***, a ***PHP***, a héjparancsfájlok, a hálózati fájlrendszer (***NFS***), az ***SMB*** (kiszolgáló üzenetblokk) és a rendszermagmodulok használata és futtatása ***ARM***, ***Alpha*** és ***Intel*** géptípusokon. A fejlesztés részleteit át fogjuk ugrani, ám a rendszer jelenlegi működését ismertetni fogjuk, kiemelve azokat a valós életbeli előnyöket, amelyeket a Linux biztosított számunkra.

Alapesetben minden ***STAR*** eszköz egy többcsatornás adóegységből és egy többcsatornás vevőegységből áll, mindkettő egy megosztott ***LAN***-hoz csatlakozik, és mikrohullámú frekvenciákon bármilyen adattípust legfeljebb 200 Mbit/s sebességgel képes továbbítani. Jelenleg minden egységben 23 nyomtatott áramkör található, ezek megtervezése és legyártása 15 embernek egy évig tartott.

1. kódrészlet A fürtbeli csomópontok IP-címét SMB megosztáson tároló parancsfájl

```
#!/bin/sh
#
# Saját IP kiírása SMB megosztásra
#
# Központi SMB megosztás befűzése
smbmount //pelda/proba /mnt/proba
          ➥-o username=nev,password=jelszo >&
          ➥/dev/null#IP kiírása
#Az IP-cím lekérdezése az ifconfiggal
address/sbin/ifconfig | grep Bcast
  ➥| sed ´s/^.*addr://;s/Bcast.*//´ >
  ➥/mnt/proba/$HOSTNAME.ip
```



1. ábra Minden STAR alaprendszerben a kétirányú rádiós összeköttetés egy többcsatornás adót és egy többcsatornás vevőt használ

Az *1. ábrán* a rendszer egy megosztott ***Ethernet*** hálózathoz kétirányú rádiófrekvenciás összeköttetésként csatlakoztatva látható. Ez az összeállítás természetesen csak kísérleti célokat szolgál, a tényleges termékeknél az adó és a vevő között nem lesz megosztott ***Ethernet*** hálózat.

A rádiójelek feldolgozása a digitális kártyán történik, de nem az *ARM* processzor, hanem egy dedikált, ***egyedileg programozható kaputömb (field programmable gate array, FPGA)*** és egy ***digitális jelfeldolgozó processzor (DSP)*** végzi. Az *FPGA*-ban lévő kódot belső programnak nevezzük, esetében elég nehéz eldönteni, hogy program vagy vas alapú megvalósításról van-e szó. Ezzel a könnyed megoldással megengedhetjük magunknak, hogy mindenféle programozási és vastervezési szabványt figyelmen kívül hagyjunk.
A múlt év közepén, amikor a rendszert terveztük, a legnagyobb, leggyorsabb és legdrágább *FPGA*-t és *DSP*-t választottunk, amit csak be tudtunk szerezni, de azóta további két nagyméretű *FPGA*-t adtunk hozzá. Az *ARM* processzort alacsony szintű műveletekre nem használjuk, mert az *ST* feldolgozásokhoz több mint 50000 *MIPS* teljesítményre van szükségünk. Ilyen szintű bonyolultságnál még a leggyorsabb építőelemek is lassúnak bizonyulnak, ezért elég hamar eldöntöttük, hogy többprocesszoros működésre képes rendszerben gondolkodunk.
A *STAR* egységeket nagy sebességű, ***alacsony feszültségű, differenciális jelzéskezelésű (low-voltage differential signaling, LVDS)***, legfeljebb 1 Gbit/s sebességre képes kapcsolatokkal lehet bővíteni. Minden kártyán két darab ötcsatornás, kétirányú *LVDS* összeköttetés áll rendelkezésre a szomszédos kártyákkal történő kapcsolatteremtésre. Ugyanakkor minden kártya rendelkezik *Ethernet* kapcsolattal is. A nagy sebességű adatátvitel az *LVDS* összeköttetéseken, a vezérlőadatok továbbítása pedig az *Ethernet* kapcsolaton keresztül történik.

Fejlesztés

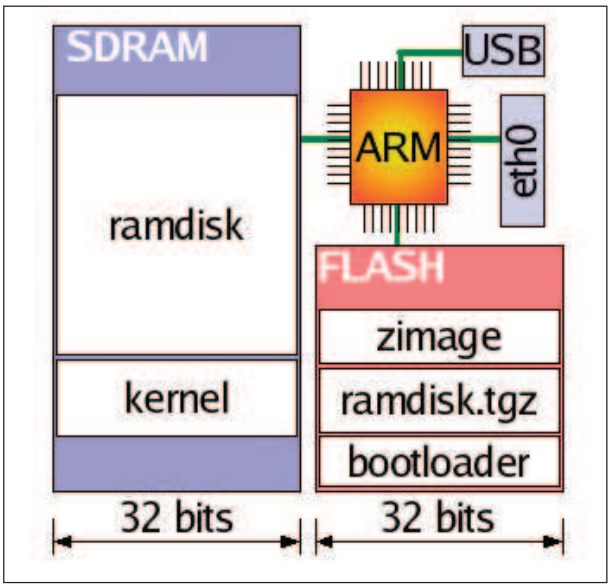
A feldolgozás túlnyomó része az *FPGA*-ban folyik, a kód *VHDL*-ben készült. *Linux* alá egyre több *VHDL* eszköz érhető el (lásd a széljegyzetet), mi az *Altera Quartust* használtuk. Rendszerünkben az algoritmusokat *GNU-Octave* alatt fejlesztettük ki, majd átültettük őket *VHDL* alá. Az *Octave* és a vele többé-kevésbé együttműködni képes *MATLAB Linux* és *Microsoft Windows* alá egyaránt elérhető, ám az *Octave*-nak van egy *MPI*-képes változata is, mely fürtökön is futtatható. Jó öreg *DEC Alpha* gépek fürtjére fordítottuk le, ezt egyébként *zionnak* neveztük el. Az *MPI-Octave* annak ellenére is nagyon szépen teljesített a *zionon*, hogy leggyorsabb processzora is csak 500 MHz-es volt.
A digitális kártyákhoz egy a *handhelds.org*-on talált *ARM Linux* eszközláncot használtuk a frissen foltozott 2.4.18-as rendszermagforrások lefordítására. *Russell King* az 1990-es évek elején *Acorn* számítógépekhez készített terjesztést, ekkor kezdett el dolgozni az *ARM Linuxon*. Jelenleg az *ARM* az egyik legjobb linuxos támogatású processzor, ami számunkra nagy könnyebbséget jelentett. Mindössze három napra volt szükségünk, hogy a *Linuxot* átültessük egyedi eszközeinkre, igaz, az *Ethernet* illesztőprogram beüzemelése még eltartott néhány napig. Az *ARM* a világon a legnagyobb példányszámban eladott processzorfajta, a *Linux* rajta való futtatásához elképesztő mennyiségű támogatást lehet szerezni. Ennek köszönhető, hogy a *Tait Electronics* úgy döntött, a jövőben is *ARM* processzorokat fog használni.
Amikor az *ARM Linux* már elindult, létrehoztunk neki egy *RAM*-lemezt. Az *ARM* 16 MB *SDRAM*-mal és 2 MB Flash

2. kódrészlet writeport.c
Egyszerű program 32 bites egész szám megadott fizikai memóriahelyre írására

```
#include <stdio.h>
#include <fcntl.h>      //az O_RDWR és az O_SYNC
                          ➥miatt szükséges
#include <sys/mman.h>   //a PROT_READ stb. miatt
                          ➥szükséges
#define GRAB_SIZE 1024UL
#define GRAB_MASK (GRAB_SIZE - 1)

int main(int argc, char **argv)
{
   void *grab_base, *virt_addr;
   unsigned int md, read_result, writeval;
   off_t phys_addr = strtoul(argv[1], 0, 0);
   /*memóriakezelő felület megnyitása*/
   if((md = open("/dev/mem", O_RDWR | O_SYNC))
   ➥== -1)
   {
      printf("HIBA - /dev/mem megnyitása
      ➥sikertelen\n");
      exit(1);
   }
   /* Lap leképezése a megadott fizikai címre*/
   if(grab_base = mmap(0, GRAB_SIZE, PROT_READ |
     PROT_WRITE, MAP_SHARED, md, phys_addr &
     ~GRAB_MASK), grab_base == (void *) -1)
   {
      printf("HIBA: leképezés sikertelen\n");
      exit(1);
   }
   /*írás a kért fizikai címre leképezett
   ➥képzetes memóriába*/
   *((unsigned long *)grab_base + (phys_addr &
     GRAB_MASK)) = strtoul(argv[2], 0, 0);
   /*memóriakezelő felület lezárása*/
   if(munmap(grab_base, GRAB_SIZE) == -1)
   {
      printf("HIBA - leképezés megszüntetése
      ➥sikertelen\n");
      exit(1);
   }
   close(md);
}
```

memóriával gazdálkodott, ezért úgy döntöttünk, hogy a tömörített rendszermag és a RAM-lemez egyaránt legfeljebb 1024 KB-os legyen, noha a tömörítetlen RAM-lemez mérete 4 MB. Mindkettőt a Flash memória tárolja, és külső beavatkozás nélküli rendszerindítást tesznek lehetővé.
A gyakran használt eszközöket, mint az `ls`, a `cd`, a `mount`, az `insmod` és a `ping` a ***BusyBoxból*** gyűjtöttük ki, a bejelentkezések és jelszavak kezelését pedig a ***TinyLoginra*** bíztuk. További segédprogramok végzik a memórialeképezett külső eszközök és a Flash kezelését, a ***TinyLogin*** szolgáltatásait használó ***telnet*** démon pedig a ***netkit-base*** csomagból szár-

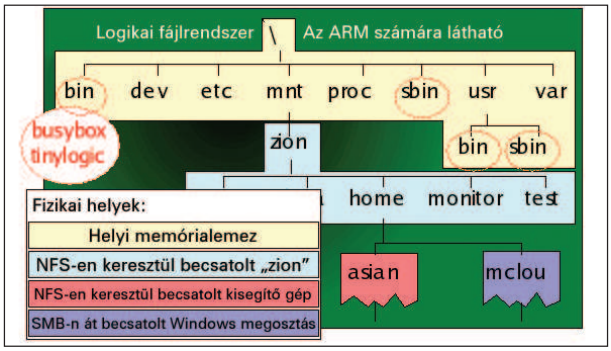


2. ábra Minden egység rendelkezik Flash alapú és RAM alapú fájlrendszerrel is

mazik. ***Linux-ARM*** fejlesztéseihez a ***Tait Electronics*** vásárolt egy ***MAC***-címtartományt az ***IEEE***-től, az egyes kártyák ***MAC***-címét a Flash memória tárolja.
Emellett jó néhány apróbb segédprogramot is készítettünk, illetve az ***Abyss*** webkiszolgálót is működésre bírtuk, de természetesen mindez egyszerre már nem fér el a Flash memóriában. A ***zionról*** ugyanakkor ***NFS***-en keresztül minden ***ARM***-on be tudunk fűzni egy könyvtárat, amivel több GB-nyi tárhelyhez jutunk. A 2. *ábrán* a fájlrendszerek elrendezése látható.
A ***zion*** alól ***SMB*** befűzésekkel a felhasználók megosztott meghajtóit is elérhetővé tettük, ezekhez az ***ARM*** kártyák szintén ***NFS***-en keresztül férhettek hozzá. Ha az ***ARM*** kártyákon futtattuk a webkiszolgálót, akkor végeredményként szerteágazó kapcsolatokkal rendelkező rendszert kaptunk.
A ***Windows***-felhasználók saját meghajtóikat webkiszolgálón keresztül is el tudták érni, és ez a webkiszolgáló olyan beágyazott ***ARM***-on futott, amely ***NFS***-en keresztül egy távoli fürtkiszolgálóhoz kapcsolódott.

Zion

2001 vége felé megvettük a kiöregedőben lévő ***DEC Alpha*** gépeket, és kíváncsian vártuk, mit tudunk kezdeni velük. Először is, módosítottuk a beépített rendszerindítót, és működésre bírtuk a 7.1-es ***Red Hatet***. Két fontosabb változtatást hajtottunk végre. Először kiválasztottunk egy mester gépet, majd hat darab ***SCSI*** merevlemezt és egy ***DVD-ROM***-meghajtót szereltünk bele. Öt darab lemezből egy ***RAID-5***-ös tömb lett (a ***raidtools*** segítségével), ez tárolja a kísérletek adatait, a fennmaradó lemez pedig a rendszerindítás és a helyreállítás céljait szolgálja.
A második módosítás az ***MPI*** telepítése volt az összes gépre. Bár maga a telepítés egyszerű volt, az összes IP-címet ***DHCP***-n keresztül kellett kiosztani, és ezzel bizony megszenvedtünk. Végül a ***zion*** nevű mester gépnek adtunk egy állandó IP-címet, a többi gépen pedig egy olyan indító parancsfájlt futtatunk, amely ***RPC*** használatával egy ***SMB*** megosztásra naplózza az egyes gépek címét.



3. ábra A fájlrendszerfa az egység oldaláról nézve

Amikor az ***MPI*** működőképes volt, letöltöttük a legújabb ***Octave*** változatot, és felraktuk rá az ***Octave-MPI*** foltot. Azóta az ***Octave-MPI*** fejlesztését a ***Transient Research*** vette át. (Lásd az internetes források részt.) ***MPI*** rendszerünket egy parancsfájl segítségével helyezzük üzembe, ez összegyűjti a már említett parancsfájl által elmentett IP-címeket, megpingeli őket, majd létrehoz egy ***rhosts*** fájlt. Amikor az ***rhosts*** dinamikus előállítása befejeződött, akkor a 4+1 gépen az alábbi parancsokkal indítjuk el az ***Octave-MPI***-t:

```
recon -v
lamboot
mpirun -v -c4 octave-mpi
```

Ha a rendszer életre kelt, az ***Octave*** terhelése eloszlik a fürt tagjain. Észrevettük, hogy az ***Alpha*** processzorok sokkal jobb lebegőpontos teljesítménnyel rendelkeznek, mint a ***Pentiumok***, ám az ***MPI***-üzenetek ***Ethernet*** feletti továbbítása lassította a fürtöt. Teljesítménytesteket ugyan nem végezünk, de az tény, hogy egy olyan szimuláció, amely egy 2 GHz-es Compaq PC-n néhány óra alatt fejeződik be, körülbelül 10 százalékkal gyorsabban futott az első, négy darab Alpha alapú gépből álló fürtön.
A ***GNU-Octave*** numerikus szimulációk elvégzésére kiváló eszköznek bizonyult. A `-traditional`, vagy, ha úgy tetszik, `-braindead` kapcsolóval futtatva a legtöbb ***MATLAB*** parancsfájlt képes értelmezni. Egyes esetekben az ***Octave*** jobb szolgáltatásokat nyújt, mint a ***MATLAB***; igaz, a ***MATLAB*** megjelenítési képességei fejlettebbek, mint az ***Octave*** által alapesetben használt ***gnuplot*** motorei.
Néhány mérnökünk inkább a windowsos fejlesztést kedvelte, ezért az ***MPI-Octave*** webes felületet is kapott. ***JavaScript*** alapú ***telnet*** ügyfelet használ, amelyet a ***zionon*** futó ***Apache*** és néhány háttérben dolgozó parancsfájl szolgáltat. A parancsfájlokat egy hálózati többszereplős játék motorjából vettük. A windowsos felhasználók megosztott meghajtóit a ***telnet*** ügyfél önműködően befűzi a ***zion*** fájlrendszerébe, majd ***telneten*** keresztül futtatja az ***Octave***-ot, valamint a rajzolást úgy állítja be, hogy a rajzok ***PNG*** formátumban a webkiszolgáló egyik könyvtárába kerüljenek, ahonnan böngészővel lehet megtekinteni őket. A felhasználók dönthetnek úgy is, hogy a rajzokat közvetlenül megosztott meghajtójukra mentik.
Hibakeresési célokra az ***ARM*** el tudja érni az ***FPGA*** nagyméretű hibakereső pufferét. Az ***FPGA***-t 32 bites, nagysebességű, aszinkron hozzáféréssel leképeztük a memóriába, így

felhasználói és rendszermag térből egyaránt elérhetővé vált az *ARM* számára. Nem meglepő, hogy a rendszermag térben fájlként elért karakteres illesztőprogram modult használunk. Ez löket módban akár 1024 adatszó nagysebességű továbbítására is alkalmas – természetesen folyamatos átvitelnél nincs ilyen jó sebessége –, miközben gondoskodik a jelzéskezelésről is. A felhasználói hozzáférést az `mmap` segítségével, a */dev/mem* felületen keresztül biztosítjuk – persze nem árt, ha nem feledkezünk meg a */dev/mem* létrehozásáról a beágyazott fájlrendszer alatt.

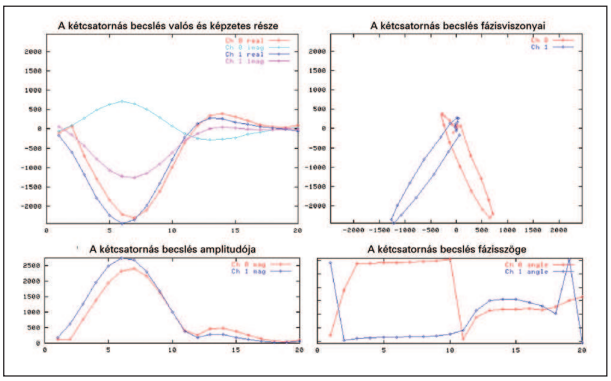
Mindezen eszközök segítségével fel tudjuk tölteni az ismert, az *Octave* által a *zionra* mentett tesztvektorokat a hibakereső pufferbe. Az *ARM* vezérlése alatt a hibakereső puffer kimenetét a tesztelés alatt álló blokk bemenetére irányítjuk, futtatjuk a rendszert néhány órajel erejéig, majd ugyancsak a hibakereső pufferben fogadjuk a kimenetet. Az eredményt az *Octave*-val elemezve el tudjuk dönteni, hogy a blokk működik-e.

A megjelenítést fontos tényezőnek tartottuk. Az első fontosabb lépések elvégzése után meghívtunk néhány embert, és megmutattuk nekik a rendszert. Hogy mit láttak? Zümmögő dobozokat, amelyeken néhány zöld *LED* jelezte, hogy minden rendben működik. Nem nagyon akartak lelkesedni, holott ez a műszaki megoldás nálunk működött elsőként a világon – kénytelen-kelletlen beláttuk tehát, hogy valami még hiányzik. Kitaláltuk, hogy – nagyjából valós időben, alkalmazásuk szerint – megjelenítjük a csatornamodelleket. A csatorna egy összetett útvonal a vevő és az adó között, különféle visszaverődésekkel, többszörös utakkal, szóródásokkal stb. Rendszerünk próbajelekkel vizsgálta a csatornát, mielőtt megkezdte volna az adatok továbbítását. A próbák révén egyfajta képet lehetett alkotni a csatornáról, és úgy döntöttünk, hogy ezt fogjuk megjeleníteni. Készítettünk egy *ARM* alapú parancsfájlt, ez időnként elindított egy programot, amely a vevőoldali *FPGA* hibakereső pufferéből kinyerte a csatornaadatokat, megformázta őket, és egy az *Octave* által is kezelhető *.mat* fájlba mentette a *zionra*. Az *Octave* a *zionon* nem interaktív módban futva rendszeresen kiolvasta a csatornaadatokat, elemezte őket, majd négy darab *PNG* képfájl formájában elkészítette a rajzokat. Ezeket a *zionon* futó webkiszolgáló egy *PHP* alapú weblappal jelenítette meg, négy másodperces frissítéssel. (*4. ábra*)

Támogatás

Linuxos digitális kártyáink normál esetben összetett feldolgozó algoritmusokat futtatnak, rádiójelek és *Ethernet* segítségével tartják a kapcsolatot más rendszerekkel. Mivel még laboratóriumi körülmények között is nehéz megmondani, hogy minden összetevő megfelelően működik-e, úgy döntöttünk, hogy a *Linux* erejére építve önellenőrző és önmegfigyelő megoldásokat készítünk. A megjelenítéshez hasonlóan itt is számos összetevőt kellett rugalmasan összekapcsolni.

A nem műszaki területen dolgozó felhasználók grafikus felületet akartak, amit *GTK*, *Tk*, *Qt* vagy valami hasonló eszközzel állíthattunk volna össze, mi azonban *PHP* alapú webes parancsfájlok használatát választottuk, így ugyanis a rendszer elérése tökéletesen gépfüggetlen. A *PHP*-t a *C* stílusú írásmódnak köszönhetően a *C* programozók is



4 ábra A csatornamódok megjelenítése (összetett útvonalak az adó és a vevő között)

könnyen tudják használni, és a felhasználók által megszokott és ismert webes felület is előnyt jelent. A legtöbb alkalmazás forráskódja 50 KB-nál kisebb, ami előnyös a hibakeresésnél, illetve ebből fakadóan több bizalmat helyezhetünk az egyes kódrészletekbe.

Az ilyen jellegű felhasználói felületeknek sajnos hátrányaik is vannak. Az egyik eset az, amikor valamilyen háttérbeli eseményre kell felhívni a felhasználó figyelmét, a másik pedig az, amikor a rendszernek közvetlen párbeszédet kell folytatnia a vassal. Az első kérdést folyamatosan frissülő üzenetek külön keretben való megjelenítésével lehet megoldani. A második gondot mi úgy hárítottuk el, hogy a *PHP*-vel a kapcsolatot fájl alapú felületen keresztül tartó, apró, alacsony szintű *C* programokat írtunk.

Nem mondom, szép szövevényes rendszerünk lett. Minden egység vezérlője egy *ARM* processzor, de a vezérlők vezérlője egy *PHP* parancsfájlokat kezelő webkiszolgáló. Önellenőrzés céljából a rendszer öt másodpercenként minden *ARM*-on lefuttat egy parancsfájlt, majd az eredményeket átadja a webkiszolgálónak. Az eredmények az egyes *ARM*-ok *Ethernet* csatolójának IP-címe szerint elnevezett fájlokba kerülnek. A figyelő parancsfájl feladata annak biztosítása is, hogy minden működő rendszer órája szinkronban legyen a *zionéval*. Az *NTP* használatát elvetettük, mert viszonylag nagyméretű kód kellett volna hozzá, és az órák együttállására csak a fájlrendszerek kezelésekor jelentkező hibák elkerülése miatt volt szükségünk. A *zion* egy parancsfájl és a `data` parancs segítségével az aktuális időt és dátumot öt másodpercenként egy fájlba írja; az *ARM*-ok ezt a fájl olvassák ki szintén öt másodpercenként, és tartalma alapján beállítják órájukat. Ezzel el tudjuk kerülni az időszinkronizálási gondokat, valamint biztosítani tudjuk a fájlok időbélyegeinek pontosságát. Egyben időzítőként is szolgál a több mint egy percet csúszó kártyák alapállapotba hozatalára.

Az önellenőrzésen túl a parancsfájl indításkor a *RAM*-lemezek és a rendszermagok változatszámát is lekérdezi és rögzíti egy állapotfájlba. A parancsfájl utolsó feladata a kártyákhoz egyedileg megadott parancsok végrehajtása. A parancsokat a *PHP* alapú weboldal szolgáltatja, ez szabja meg, hogy melyik *ARM* kártyának mit kell tennie. Az utasítások *PHP* alól héjparancsfájl formájában érkeznek, ezeket kell az IP-cím alapján kiválasztott kártyán lefuttatni.

Minden egység ugyanolyan rendszermaggal és *RAM*-lemezekkel dolgozik, ezek tárolása a helyi Flash memóriá-

ban történik, illetve a ***zionon*** lévő másolat alapján minden indításkor megtörténik sértetlenségük ellenőrzése. Bármilyen eltérésnél a rendszer újra bemásolja a Flash memóriába a megfelelő rendszermagot és ***RAM***-lemezt. Erre a célra egyedi Flash memóriakezelő eszközöket fejlesztettünk. A gyakorlatban tárolási rendellenességeket nem tapasztaltunk, ám ezzel a megoldással felhasználói beavatkozás nélkül tudjuk telepíteni a rendszermag és a ***RAM***-lemez újabb változatait.

A ***PHP*** alapú weboldalak az ***autoload HTML*** címke használatával öt másodpercenként frissülnek, természetesen csak akkor, ha legalább egy felhasználó megnyitotta böngészőjével az adott oldalt – egyébként lehetséges, hogy az oldalon szereplő adatokra éppen nincs is szükség. A vezetőknek szánt weboldalak szinte minden hasznos adatot elrejtenek, viszont jól néznek ki. A valódi felhasználóknak szánt oldalakon ellenben lehetőség nyílik az alsóbb szintek elérésére is, egészen az egyes kártyák működését irányító parancsfájlokig.

Hibakeresés és figyelés

Az ***ARM***-on futó linuxos program az ***FPGA*** által tárolt belső program segítségével kisméretű csomagokból álló adatfolyamot továbbít a vezeték nélküli összeköttetésen keresztül az ***FPGA*** hibakereső pufferébe. Ráérő idejében az ***ARM*** kibontja ezeket, és eltárolja őket a ***zionon***. A fájlokban önműködő eljárásokkal meg lehet keresni a hibákat (például a csupa nullából álló sorozatokat), és szükség esetén a weboldalakon keresztül riasztani lehet a felhasználókat.

Összefoglalás

Legújabb tervünk az, hogy az elméleti kutatásokon túl termékek tervezésébe is belefogunk. Valószínűleg IP-csomagok küldéséről-fogadásáról lesz szó, és a termék vezérlését beágyazott ***Linuxra*** fogjuk bízni. Ennek nemcsak az az oka, hogy a fejlesztés során is a ***Linuxra*** támaszkodtunk, de az is, hogy számos kiváló lehetőséget biztosít az IP-csomagok kezelésére. A ***WinCE*** lassú és nagy, a ***VxWorks*** pedig drága és kevés protokollt támogat. A ***Tait Electronics*** nonprofit, az új-zélandi ***Christchurch*** alkalmazottait és közösségét segítő elektronikai egyesülés. ***Sir Angus Tait*** alapította, több mint 30 évvel ezelőtt. Mobil rádiós termékeinek 97 százalékát exportálja, értékesítései a világ több mint 200 országára terjednek ki.

Linux Journal 2004. szeptember, 125. szám

Ian McLoughlin 12 éve használja a Linuxot, szakterülete a jelfeldolgozás. Mielőtt kívándorolt volna Új-Zélandra, egyetemi előadó volt Szingapúrban, és mint az XSat műholdas program – indítását 2006-ra tervezik – vendégtudósa jelenleg is sokat tartózkodik ott.

Tom Scott a Mission Technologies Ltd. (www.missiontech.co.nz) igazgatója, elsősorban a dolgokhoz való lényegre törő, nyílt, szabatos és gyakorlatias hozzáállásáról ismert. Felesége és két gyermeke hittérítő.