Egy kommunikációs protokoll

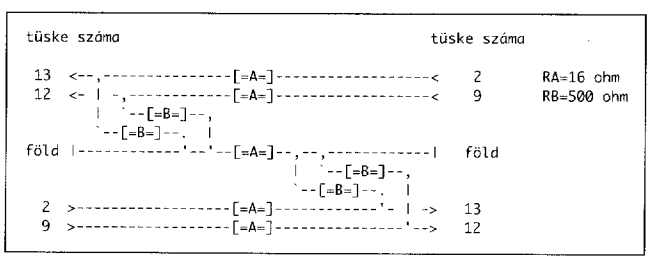
CSIRMAZ ELŐD, CSIRMAZ LÁSZLÓ

E-mail: csirmaz@renyi.hu

Számítógépek közötti adatcsere gyakran használt eszköze a flopi (hajlékony mágneslemez). Különösen így van ez hordozható gépek (laptopok) esetén. A laptop és az asztali gép adatainak összevetése – szinkronizálása – meglehetősen fáradságos: flopi be, adatok felírása, flopi ki, be a másik gépbe, adatok leolvasása, flopi ki, vissza a laptopba és így tovább. Persze csak akkor, ha nincs adathiba, és az átviendő állomány sem túlságosan nagy. A két gép között kábeles összeköttetést használva mindez sokkal egyszerűbb. Számtalan kitűnő program létezik, melyek a soros portokon keresztül kommunikálva valósítják meg az adatcserét. Az általunk használt újabb gépeken azonban soros portok helyett kizárólag a jóval többet tudó USB portok vannak, és ezekhez a régi kábelek természetesen nem csatlakoztathatók, így ezeket a programokat sem tudjuk használni. Ott van viszont a nyomtatásra használt párhuzamos port, fel tudjuk ezt használni két gép közötti kommunikációra? Mennyire megbízható az átvitel, és milyen sebességet tudunk elérni? Próbaképpen összeállítottunk egy összekötő kábelt, és megírtuk a kommunikációt megvalósító programot. A munka legtanulságosabb része a kommunikációs protokoll tervezése volt, ezt ismertetjük. A szerzett tapasztalatok – reményeink szerint – mások számára is hasznosak lehetnek.

1. Az összekötő kábel

Az kábel a két gép párhuzamos portját köti össze. A portot eredetileg egyirányú kommunikációra tervezték: a számítógép ezen a porton keresztül küld adatokat egy másik készüléknek, tipikusan egy printernek. Az átvitel során egyszerre nyolc bitet lehet átküldeni (innen az elnevezés: az adatokat egyszerre, párhuzamosan küldjük), a fogadó készülék néhány vezérlő lábon keresztül tud visszajelezni. A csatlakozó 25 érintkezős, ebből 12 kimenő (közötte 8 adat és négy vezérlőjel), 5 bemenő, a többi pedig föld. Az összekötéshez öteres kábelt használtunk. Az öt ér közül egy a föld; kettő az egyik irányú, kettő pedig a másik irányú kommunikációt biztosítja. A kábel középen egy ellenállásokat tartalmazó doboz van, végeire pedig egy-egy 25 érintkezős csatlakozó került az 1. ábrán látható vázlat szerint.



1. ábra

Az RA ellenállások arra szolgálnak, hogy egy esetleges rövidzár esetén se menjen tönkre a portot kezelő elektronika. Az RB ellenállások biztosítják, hogy az input lábak ne lebegjenek.

A párhuzamos port kezelése három egymás után következő portcímen történik, ezeket b+0, b+1 illetve b+2-vel jelöljük. A b (bázis) értéke általában 378h (decimálisan 888) vagy ritkábban 3BDh (decimálisan 956). A b+0 portra kiírt byte bitjei jelennek meg a 2–9 tüskéken, a b+1 portról beolvasott byte 16-os, illetve 32-es helyi értékű bitje adja meg, hogy a 13-as, illetve a 12-es input lábakon 0 vagy 1 áll-e. A kábel használatakor 00, 01, 10, illetve 11 átviteléhez a b+0 portra a 0, 1, 128, illetve 129 számokat kell kiírni. Ezek az értékek a bemenő oldalon a b+1 portról olvasott értéket (16+32)-vel maszkolva rendre 0, 16, 32, illetve 48-ként jelennek meg. Az összekötő kábellel tehát mindkét irányban két-két bitet tudtunk átvinni.

2. Kapcsolatfelvétel

Mielőtt az adatok átvitele megkezdődne, a két összekapcsolt gépnek meg kell bizonyosodnia arról, hogy a megfelelő program a másik gépen is fut, továbbá a két programnak egyet kell értenie abban, hogy az átvitelt meg lehet kezdeni. A kapcsolatfelvevő protokollt a következő feltételekkel terveztük:

a) nem tudjuk, hogy melyik gépen indul el először a kommunikációs program,
b) a program indulásakor a kimenő bitek értéke akármi lehet.

2.1. Első próbálkozás

Első protokollunk szimmetrikus (vagyis mindkét fél ugyanazt csinálja), hátránya viszont, hogy a programnak

tudnia kell, indulásakor hogyan áll az általa vezérelt két kimenő bit. (Ez a soros port elektronikájától függően esetleg megtudható a b+0 port kiolvasásával.) A kapcsolatfelvevő protokollok leírásánál a két fél közül az egyiket E-vel (Egyik), a másikat M-mel (Másik) jelöljük.

```
1.  leolvassa az input és az output biteket;
2.  megváltoztatja az output biteket:
      ha az eddig 01 volt, akkor 10-ra, egyébként pedig 01-re;
3.  vár addig, míg az input megváltozik az 1-ben leolvasotthoz képest;
4.  kiír 00-t;
5.  vár addig, míg az input 00 lesz.
```

2.1 protokoll

Indulás után mindkét gép a 3. lépésben vár arra, hogy a másik megváltoztassa az input lábakon bejövő értéket. Mivel a protokoll szimmetrikus, feltehetjük, hogy E indul hamarabb. Beállítja az outputot 01-re vagy 10-ra, és a 3. lépésben vár, hogy az input megváltozzon. Közben M is elindul, leolvassa saját inputját (amin vagy az E által beállított értéket találja, vagy azt, ami E indulása előtt volt ott), és megváltoztatja a saját outputját. Ezt a változást E észreveszi, és kiír 00-t (4. lépés). M közben a 3. lépésben vár az E outputjának megváltozására. Ez most megtörtént, így M is továbbmegy a 4. lépésre és kiírja a 00-t. E az 5. lépésben erre várt, tehát számára a protokoll ezennel véget ér. M az 5. lépésben vár a 00-ra, ez megjött, vagyis M is befejezi a protokollt. Mindkét fél tudja, hogy a kapcsolat létrejött: kezdődhet a kommunikáció.

A protokoll helyesen működik akkor is, ha végrehajtását pontosan egyszerre kezdik el, vagy ha az egyik fél sokkal lassabban dolgozik, mint a másik. Mindkét fél a 3. lépésen csak akkor jut tovább, ha a másik vagy a második, vagy a negyedik lépést már végrehajtotta – és így biztos lehet benne, hogy a másik működik. A protokoll végére a két fél majdnem egyszerre jut el, a különbség akkora idő, ami alatt a bejövő inputot le lehet olvasni és ellenőrizni.

2.2. Lehet másképp?

A 2.1 protokoll során mindkét fél arra vár, hogy a másik változtassa meg az outputját. De lehet-e a protokollt úgy módosítani, hogy induláskor ne kelljen tudni, hogyan állt az output? Az ötlet az, hogy minkét fél az általa látott input és az általa vezérelt output között teremt kapcsolatot úgy, hogy a két összefüggés egyszerre ne állhasson fenn. Ezért valamelyik fél biztosan megváltoztatja az általa vezérelt biteket, amit a másik fél felismer. A protokollt a 2.2 táblázat mutatja, és ez is szimmetrikus. Most is a 00 értéket használjuk arra, hogy a felek jelezzék: a kapcsolat létrejött.

```
1. leolvassa az input biteket xx-be
2. az input értékétől függően határozza meg az output biteket:
      xx:         00   01    10    11
      output:     01   10    11    01
3. vár addig, míg az input megváltozik az xx-hez képest;
4. az újonnan leolvasott input yy. Ha yy=00, akkor kiír 00-t, és a
     protokollt a 7. lépésnél folytatja. Ha yy ettől különböző, akkor
     yy-tól és az előző outputtól függően az outputot az alábbi módon
     változtatja meg:

            előző output:      01        10        11
        -----------------+------------------------
                  01     |    10        01        01
            yy:   10     |    10        11        10
                  11     |    11        11        01

 5. vár addig, míg az input megváltozik a 3-ban leolvasotthoz képest;
 6. kiír 00-t;
 7. vár addig, míg az input 00 lesz.
```

2.2 protokoll

A protokoll második lépésében mindkét fél az outputját 00-tól különbözőre állítja, és csak akkor ír ki 00-t (a negyedik vagy hatodik lépésben), ha előzőleg az inputja megváltozott, vagyis meggyőződött arról, hogy a másik féllel tud kommunikálni. Ha tehát a protokoll végére érnek, akkor mindkét fél kész a kommunikációra, és mindkettő tudja ugyanezt a másikról. Csak azt kell ellenőriznünk, hogy a protokoll véget ér.

Az nem lehet, hogy mindkét fél a harmadik lépésben várakozzon. Ha ugyanis E itt vár, akkor E output bitjeit a második lépésben állította be, így az csak 01, 10 vagy 11 lehet. Ugyanígy M is beállította output bitjeit a 01, 10 vagy 11 értékekre. Tehát E ezen három érték valamelyikét olvasta be az első lépésben (mivel az input bitek azóta nem változtak). Ha tehát M bitjei 01, 10 vagy 11, akkor E outputja rendre (a második lépésben megadott táblázat szerint) 10, 11 illetve 01. Most ugyanez igaz M-re is: ő az 10, 11, illetve 01 értékeket olvasta le az első lépésben, tehát a második lépésben kiírt értékek 11, 01, valamint 10. Ez pedig lehetetlen, hiszen abból indultunk ki, hogy az M output bitjei rendre 01, 10 vagy 11. Így valamelyik fél biztosan továbbmegy a harmadik lépésen; mondjuk elsőként E.

A harmadik lépésben E által beolvasott (megváltozott) érték csak az lehet, amit M a második lépésben kiírt – hiszen feltételünk szerint M még nem ment túl a harmadik lépésen. Tehát M már mindenesetre elindult. M az első lépésben vagy E outputjának eredeti állapotát olvasta le, vagy pedig azt az értéket, amit E a második lépésben kiírt. A negyedik lépésben E megváltoztatja az outputját úgy, hogy az mindenképpen más legyen, mint amit M látott. M előbb-utóbb rátér a harmadik lépésre, ahol észreveszi, hogy az inputja megváltozott. (Ez a megváltozott érték vagy a most kiírt, vagy pedig még a második lépésben kiírt output is lehet.) Mindenesetre M megváltoztatja az outputját (negyedik lépés). E csak erre vár az ötödik lépésben, kiír 00-t (ezzel ismét megváltoztatva M inputját). M felismeri a változást, ő is kiírja a 00-t, és a protokoll véget ér.

Ezek szerint a negyedik lépésben a leolvasott yy érték nem lehet nulla, tehát annak kezelésére nincs is szükség. Továbbá azt is láthatjuk, hogy a protokollt mindkét fél nagyjából egyszerre fejezi be, csakúgy, mint a 2.1 protokoll esetében.

A 2.2 protokoll elemzése azt is mutatja, hogy az a fél, aki az ötödik lépést később hajtja végre, inputján már 00-t lát, azaz neki a 7. lépésben már nem kell várakoznia. Azt, hogy ez melyik fél lesz, a véletlen (a programok indítása, a gépek sebessége) dönti el. Adatátviteli protokolljaink feltételezik, hogy először a küldő állítja outputját 00-ra, és erre válaszol a fogadó azzal, hogy

outputját 00-ra állítja. Hogy ezt el tudjuk érni, a 2.2 protokoll szimmetriáját meg kell bontanunk. A két fél K (küldő), illetve F (fogadó), ők a 2.2a protokollt hajtják végre.

```
K:   1--5. lépések ugyanazok, mint a 2.2 protokollban;
     6. az újonnan leolvasott input zz. Ha zz nem 00, akkor a
        protokollt a 11. lépésnél folytatja, egyébként a 7.
        lépésnél;
     7. kiír 00-t;
     8. megvárja, míg az input 11 lesz;
     9. kiír 11-t;
    10. megvárja, míg az input 01 lesz;
    11. kiír 00-t;
    12. megvárja, míg az input 00 lesz.

F:   1--5. lépések ugyanazok, mint a 2.2 protokollban
     6. az újonnan leolvasott input zz. Ha zz=00, akkor a protokollt a
        13. lépésnél folytatja, egyébként a 7. lépésnél;
     7. kiír 00-t;
     8. megvárja, míg az input 00 lesz;
     9. kiír 11-et;
    10. megvárja, míg az input 11 lesz;
    11. kiír 01-et;
    12. megvárja, míg az input 00 lesz;
    13. kiír 00-t.

                        2.2a protokoll
```

Ha az ötödik lépésben K által leolvasott érték nem 00, akkor ő állítja az outputját hamarább 00-ra, és a protokollt rövid úton be is fejezheti. Ha viszont K 00-t olvas le, akkor F előrébb van, és a nyolcadik lépésben már arra vár, hogy K 00-t írjon ki. Ezt K meg is csinálja a hetedik lépésben, mire F 11-et ír ki. Erre K 11-gyel válaszol, mire F 01-et ír ki, és most állítja K output bitjeit 00-ra. Végül erre válaszul a 13. lépésben F is ezt teszi. Azt, hogy mikor mi történik, az alábbi idő ábra szemlélteti arra az esetre, amikor F a hatodik lépésben nem 00-t olvas be.

```
lépés:  4.   6.   7.   8.   9.   10.  11.  12.
    K:  zz   ..   00   ..   11   ..   00   ..
    F:  ..   00   ..   11   ..   01   ..   00
lépés:  6.   7.   8.   9.   10.  11.  12.  13.
```

3. Az adatátvitel

Miután a kapcsolat létrejött, megkezdődhet az adatok átvitele. Nekünk nem volt szükségünk kétirányú adatátvitelre, minden esetben a küldő fél küld át egy teljes adatállományt; az állomány átvitele után a kapcsolatot megszakítjuk. A kapcsolat megszakítására nincs semmilyen külön eljárás, egyszerűen kilépünk a programból. Mivel egyértelmű, hogy ki küld és ki fogad, nem kell külön gondoskodnunk arról, hogy mi történjen, ha mindkét fél egyszerre, vagy nagyon kis időkülönbséggel kezd el forgalmazni.

3.1. Bitenkénti küldés

A kapcsolatfelvétel után mindkét bitpár értéke 00. Amikor K küldeni akar, ezt az értéket megváltoztatja. A változást F észreveszi, és saját bitpárját megváltoztatva jelzi, hogy az adat megérkezett. Erre K visszaállítja a 00 értéket, és megvárja, míg F ugyanezt teszi. Egy ilyen lépés során K háromféle értéket tud átvinni, attól függően, hogy mire változtatja az outputját. Az eljárás időbeli lefolyását a 3.1 ábra mutatja. Az xx jelzi a 01, 10 vagy 11 valamelyikét; F azzal jelzi az adat megérkezését, hogy a vonalra 11-et ír.

```
K:  00 ..  xx  ..  00  ..
F:  .. 00  ..  11  ..  00

     3.1 Egy bit küldése
```

A protokoll segítségével biteket tudunk továbbítani (például 01 jelenti a 0-t, 10 pedig az 1-et); az átvitt bitekből kell F-nek összeállítani a teljes adatállományt.

3.2. Egyszerre egy byte-ot

Adatállományok átviteléhez jobban használható olyan protokoll, ami egy menetben egy egész byte-ot tud átvinni. Ilyet az előző módosításával kaphatunk. Alapállapotban mindkét érpáron 00 van. Egy byte összesen nyolc bit, a biteket 01 illetve 10-ként kódoljuk. K kiírja az első bitnek megfelelő értéket, majd megvárja, míg F visszaüzen: „vettem a bitet." Ez után K 11-et ír ki, jelezve hogy kész a következő bitet küldeni. F erre megváltoztatja az outputját, jelezve, hogy „jöhet a következő". Ezt csinálják, amíg a byte bitjei elfogynak. Ezek után K még küld egy paritásbitet is, vagyis olyan bitet, ami megmondja, hogy a küldött byte-ban páros vagy páratlan számú egyes volt-e. F a paritásbit vétele után vagy azt üzeni, hogy „rendben", vagy azt, hogy „hibás". Ezek után mindkét fél visszaállítja az outputját 00-ra, és kezdődhet a következő byte küldése.

A protokoll időábráján jól követhető, hogyan változnak a kiírt értékek, illetve hogy a két félnek mikor mit kell csinálnia.

```
        1.bit       2.bit       ...  8.bit       paritás
        I           I                I           I
K:  00.. bb .. 11 .. bb .. 11 .. ... bb .. 11 .. bb .. 00 ..
F:  00.. .. 01 .. 10 .. 01 .. 10 ... .. 01 .. 10 .. xx .. 00
                                                    I
                                                 rendben/hibás
                    3.2 protokoll
```

A protokoll programozásakor elkövettük azt a hibát, hogy a fogadó „rendben" üzenetét 00-val kódoltuk, azt gondolván, hogy ezzel a byte küldésének az idejét lerövidítjük. Ilyenkor ugyanis a „rendben" utáni 00 már magától ott van, nem kell külön elküldeni. Csak hosszas nyomozás után derült ki, hogy hol van a baj. Nézzük csak, mi történik ilyenkor két byte küldése között:

```
              paritás       köv byte 1.bitje
              I             I
K:     11 .. bb ..  00 .. bb
F:     .. 10 .. 00  .. .. ..
              I
            rendben
```

A küldő beállította a paritás bitet, erre F a 00-val jelezte, hogy a paritás rendben van. K kiírta a vonalra a 00-t, majd nekilátott a következő byte küldésének. Ellenőrizte, hogy az inputja 00-e (az volt), tehát a vonalra kiírta a következő byte első bitjének megfelelő értéket. Eközben F arra várt, hogy az előző byte küldése befejeződjön, vagyis a küldő vonalán megjelenjen a 00. Mivel ez csak nagyon kis ideig volt ott, F nem vette észre. hogy F még mindig a 00-ra várt, K pedig arra, hogy F igazolja vissza a következő byte első bitjét. Amikor a programba nyomkövető kiíratásokat tettünk, a protokoll rendben ment, mert ilyenkor F-nek volt ideje a byte végét jelző 00-t leolvasni. Ugyanez magyarázza azt is, hogy a kapcsolatfelvételt úgy kell befejezniük, hogy előbb K állítsa az outputot 00-ra, és F csak ezután, erre válaszul írjon ki 00-t. Csak így tudjuk biztosítani, hogy az első byte első bitjéről F nem hiszi, hogy a kapcsolatfelvevő protokoll része.

3.3. Lehet gyorsabban is?

A 3.2 protokoll kitűnően működött, csak egy kicsit lassúnak találtuk. Hogyan lehetne az átvitelt gyorsítani? Itt az egyes biteket 01, illetve 10-ként vittük át, az egyes bitek között pedig 11 az elválasztó. De miért is van szükség az elválasztójelre? Azért, mert a másik fél csak a vonal megváltozását veszi észre, tehát az outputot minden lépésben meg kell változtatni. Igen ám, de akármi is az output, azt mindig többféle módon is meg lehet változtatni – így minden egyes lépésben tudunk egy-egy bitet továbbítani. Ezt az ötletet próbáljuk kiaknázni.

Elsőként K a byte első bitjét küldi el, a szokásos módon 01-ként vagy 10-ként kódolva. Ha ez mondjuk 01 volt, akkor K következőként 10 és 11 között választhat, ha pedig 10 volt, akkor 11 és 01 között: ezekkel mondhatja meg, hogy a második bit micsoda. Ha pedig valamikor 11-et írt ki, akkor azt követően ismét 01 vagy 10 kódolja a következő bitet. A teljes protokoll időábrája a következő:

```
K:   00 .. b1 .. b2 .. b3 .. b4 .. b5 .. b6 .. b7 .. b8 .. b9 .. 00 ..
F:   .. 00 .. 10 .. 01 .. 10 .. 01 .. 10 .. 01 .. 10 .. 01 .. 1p .. 00
```

3.3 Byte küldés gyorsabban

A küldő a b1–b9 értékeket a következőképpen határozza meg. Az elsőnek küldött érték 01, illetve 10 attól függően, hogy a bit értéke 0 vagy 1. A többi bit küldéséhez K az alábbi táblázatot használja:

előzőleg küldött érték:	01	01	10	10	11	11
következő bit:	0	1	0	1	0	1
küldött érték:	10	11	11	01	01	10

A küldött érték egyszerűen számolható úgy is, hogy a régi érték plusz a küldendő bit értékének vesszük a hármas maradékát, és ehhez hozzáadunk egyet. Az érkező bit visszakódolásához F ugyanezt a táblázatot használhatja, vagy működik a következő eljárás is: az előzőleg küldött értéket vonjuk ki kettőből, adjuk hozzá a most kapott értéket, és ennek vegyük a hármas maradékát.

A b9 a paritásbit, és F a paritásbitre válaszul 10-et küld, ha a paritásbit megfelelő volt, és 11-et, ha nem. Ez utóbbi esetben K a teljes byte-ot újraküldi.

Ha K nem olyan értéket kap, amit vár, akkor a küldést abortálhajta azzal, hogy 00-ra állítja az outputot. Ezt F felismeri, ő is 00-t ír ki, és a byte küldését elölről kezdik. Ha viszont F talál valamilyen érthetetlen értéket, akkor amennyiben ez 00 volt, akkor ő is 00-t ír ki, és a byte fogadását elölről kezdi; ha valami más, akkor a következő outputját 11-re állítja, erre K-nak 00-val kell válaszolnia és a byte-ot újra kell küldenie.

3.4. A leggyorsabb változat

Végül utoljára hagytuk az általunk talált protokollok közül azt, ami a leggyorsabb átvitelt biztosítja. Ez a korábbiaknál jobban bízik az átvitel biztonságában, viszont még a 3.3-ban ismertetett protokollnál is majdnem kétszer gyorsabb. Ez az előzőtől két lényeges részben is különbözik. Először is K lépésenként mind a három lehetőségét kihasználja, és nem csak kettőt, másrészt a paritásbit küldését F-re bízzuk, aki azt ütemvesztés nélkül is meg tudja tenni.

A küldés során K két fordulónként összesen kilenc lehetséges értéket tud átküldeni. Ebből nyolcat három bit kódolására használunk, a kilencediket pedig annak jelzésére, hogy az előző, F által küldött paritásbit hibás volt. Két fordulóban három bitet tudunk átküldeni, tizenhat fordulóban huszonnégy bitet. A tizenhat forduló után egy szünetet tartunk, ami előtt F a vett 24 bit paritását küldi vissza. A szünetet, illetve a paritásbitet annak ellenőrzésére használjuk, hogy K és F még mindig szinkronban vannak.

```
K:   xx .. d1 .. d2 .. d3 .. d4 ..  ...  .. d15 .. d16 .. yy ..
F:   .. 00 .. 01 .. 10 .. 01 .. 10  ...  10 ..  01 ..  1p .. 00
```

3.4 protokoll -- hibátlan küldés

A tizenhat forduló előtt K outputja 00 vagy 11, F outputja pedig 00. A küldés során a vonalon található értékek a következőképpen változnak:

Ahogyan azt korábban megjegyeztük, xx értéke vagy 00 vagy 11. Amikor K a soron következő 0, 1 vagy 2 értéket akarja átküldeni, a d_i outputot az alábbi módon számítja ki: az előző outputjához hozzáad egyet, ehhez még hozzáadja az átviendő értéket, és az így adódó szám négyes maradéka lesz az új output. Azaz:

előző output	00	00	00	01	01	01	10	10	10	11	11	11
átviendő érték	0	1	2	0	1	2	0	1	2	0	1	2
új output	01	10	11	10	11	00	11	00	01	00	01	10

A befejező yy értéket lehetőség szerint 00-nak választjuk, kivéve, ha K-nak d16-os outputja 00, amikor is 11-nek. A paritásbitet F a d16-ra válaszolt értékkel tu-

datja K-val: ha a paritásbit nulla, akkor 10-et, ha pedig egy, akkor 11-et küld.

Ha K úgy találja, hogy az átvitel hibás volt, vagyis a paritás nem megfelelő, vagy pedig F-től olyan értéket kapott, ami a protokoll szerint nem jöhetett volna, akkor mindaddig 2-es kódú értéket küld, amíg F 00-ra nem állítja a saját vonalát. Két egymás utáni kettes ugyanis érvénytelen kód (hiszen 2*3+2=8, ami nem három bites szám); ezt F úgy tekinti, mint K üzenetét, hogy a jelenlegi átvitelt meg kell szakítani. Miután a hiba még az előző 24 bit átvitelekor is keletkezhetett, ilyenkor nem az aktuális, hanem a megelőző 24 bitet fogja K újraküldeni.

Ha pedig F lát valamilyen problémát, akkor outputját 11-ra állítja. Erre K-nak vagy 00-val, vagy 11-gyel kell válaszolnia. Ekkor F 00-t ír ki, amivel a megelőző 24 bit küldését indítja el.

```
K:   xx .. d1 .. d2 .. yy ..
F:   .. 00 .. 01 .. 11 .. 00
                    |
                    itt fedezi fel F a hibát
```

3.4 protokoll -- egy hibás küldés

A 3.4 protokollt használva K összesen 17-szer változtatja meg az outputját, miközben 3 byte-ot küld át F-nek. A 3.3 protokollnak ugyanennyi adat átviteléhez 30 lépésre van szüksége; a sebességnövekedés majdnem kétszeres.

4. Megvalósítás

A kábelt és a programokat elkészítettük, a kapcsolat különböző sebességű és felépítésű gépek között is kitűnően működött. A 3.3-as protokollal az átvitel sebessége Pentium I és Pentium II gépek között 7 kbyte/sec volt; míg a 3.4-es protokollal ugyanez – a várakozásainknak megfelelően – 13 kbyte/sec-nek adódott. A gépek között több megabyte adatot átküldése esetén sem észleltünk egyetlen hibát sem. Az első protokollokat BASIC-ban írtuk meg, a végleges programok C-ben készültek és DOS, illetve Windows operációs rendszer alatt próbáltuk ki őket. A programokat természetesen minden további nélkül lehet Linux alatt is futtatni.

Protokolljaink külön erénye, hogy semmilyen időzítést nem tartalmaznak. Így nem kellett foglalkoznunk azzal, mi történjen, ha programunk futása rövidebb-hosszabb időre megakad. (Ez megtörtént, de az egeret megmozgatva a kommunikáció minden egyéb beavatkozás nélkül újraindult.) Az időzítések hiánya emellett magukat a protokollokat is egyszerűbbé, áttekinthetőbbé tette.

Számunkra is meglepő volt, hogy egy ilyen látszólag egyszerű feladat (tervezendő egyirányú protokoll két kimenő és két bejövő bittel) mennyi meglepetést tartogat. A fentebb vázolt ötletek egyszerűségük ellenére igen hatékonynak bizonyultak: a kezdeti 60 byte/s sebesség helyett sikerült 13 Kbyte/s-ot is elérni. A protokollok stabilitása és hibamentessége szintén figyelemre méltó. Reméljük, a Kedves Olvasó is hasznosítani tudja módszerünket.