
Kapcsoló üzemű tápegység megvalósítása ARM alapú mikrovezérlővel és Linux-szal

Készítette: Fuszenecker Róbert

Budapest, 2007. november

Köszönetnyilvánítás

Ezúton szeretném kifejezni köszönetemet konzulenseimnek, **dr. Schuster György főiskolai docensnek**, és **Krüpl Zsolt okleveles villamosmérnöknek**, akik áldozatos munkájukkal, ötleteikkel és nélkülözhetetlen szakmai tanácsaikkal segítettek a dolgozat megírásában.

Szintén hálával tartozom a lektoroknak, **Körmendi Zita kommunikációs szakértőnek**, **Gagyi Endre okleveles villamosmérnöknek** és **Gnandt András okleveles villamosmérnöknek**, akik erejükön felül teljesítve azon fáradoztak, hogy a dolgozat mindenfajta – helyesírási, szakmai, és logikai – hibától mentesen kerülhessen az Olvasó elé.

Tartalomjegyzék

Köszönetnyilvánítás.....	3
Bevezetés.....	7
1. Az áramkörök specifikációja.....	9
2. Az áramkörök megtervezése és megépítése.....	11
2.1. Elvi kapcsolási rajzok és NYÁK-tervek.....	11
2.2. Paraméterek megállapítása: elméleti számítások.....	15
2.3. Alkatrészlista.....	17
3. A szoftver környezet.....	19
3.1. C fordító IA32 architektúrára.....	19
3.2. C fordító ARM architektúrára.....	19
3.3. JTAG programozó szoftver: az OpenOCD.....	20
3.4. Teljes mintaprogram.....	22
4. A mikrovezérlő hardver eszközeinek használata	27
4.1. Hibakonzol (DBGU).....	27
4.2. LED-ek.....	29
4.3. PWM.....	30
4.4. ADC.....	31
4.5. IRQ – megszakításkezelés.....	32
4.6. A főprogram.....	35
4.7. Az „I” szabályozó algoritmus.....	37
5. Kommunikáció a PC és a mikrovezérlő között.....	39
6. Felhasználói interfészek.....	45
6.1. Karakteres.....	45
6.2. Grafikus.....	47
Összefoglalás, végkövetkeztetés.....	49
Felhasznált irodalom.....	51
Felhasznált szoftverek.....	53

Bevezetés

Ez a dolgozat a Budapesti Műszaki Főiskola Kandó Kálmán Műszaki Főiskolai karán oktatott „Gyártórendszerek prodzsekt I. laboratórium” tantárgy keretein belül elkészített önálló feladat teljes leírását tartalmazza.

A mű tartalmaz minden olyan információt, amely segítségével az áramkör és annak működtető szoftvere (firmware) reprodukálható. Mindvégig szem előtt tartottam azon szándékomat, hogy az elektronikában kevésbé jártas érdeklődők is haszonnal forgathassák ezt az írásművet.

Nem tagadhatom azonban, hogy elsődleges célom a főiskola követelményeinek maximális kielégítése, így gyakran építék a kedves Olvasó előzetes ismereteire: elsősorban a gyakorlati elektronikában és az előző féléves tanulmányok során előfordult szaktudásra gondolok (PWM, ADC, programozás C nyelven, stb.).

Az áramkör magját képező mikrovezérlő felépítését, működését és a használatával kapcsolatos részletes információkat nem kell ismernie az Olvasónak, hiszen ezek nagymértékben eszközspecifikusak, és megtalálhatók a félvezető lapka leírásában.¹

Magának a prodzsektnek a végső célja az, hogy a rádióamatőrök (és az elektronikát hobbiként művelő érdeklődők) „házi” laboratóriumát, műhelyét olyan berendezésekkel lássa el, melyek könnyen utánépíthetők, mégis teljesítenek bizonyos minőségi követelményeket². Az eszközök egymással összekapcsolhatók, így egy – minden háztartásban megtalálható, Linux-ot futtató – személyi számítógép segítségével vezérelhetők. Megfelelő programozási ismeretekkel rendelkező szakemberek az eszközökből akár bonyolultabb mérési elrendezéseket is építhetnek, így nem okoz majd problémát például egy hangfrekvenciás erősítő frekvenciamenetének felvétele, vagy egy PLL áramkör szűrőjének tranziens állapotbeli vizsgálata.

Jelenleg a prodzsekt még a tervezés fázisában van, de remélhetőleg találok néhány lelkes mérnök-aspiránst, akik segítségemre lesznek a függvénygenerátor, AC-DC feszültség- és árammérő, a logikai analízátor vagy az oszcilloszkóp modul tervezésében, kivitelezésében.

Ezen álmom teljes megvalósulása még rengeteg időbe és munkába kerül. Mégis remélem, hogy az Olvasó kedvet kap ahhoz, hogy csatlakozzon a munkához, ennyivel is közelebb segítve a rádióamatőr mozgalmat a XXI. századhoz.

A dolgozat megírása során azon elv vezérelt, hogy hasznára legyen azoknak, akik valóban érdeklődnek áramkörök építése és fejlesztése iránt. Felajánlom tehát ezt a dolgozatot a közösség számára:

Ez a dokumentum szabad szoftver, szabadon terjeszthető és/vagy módosítható a GNU General Public License 3-ban leírtak szerint.

Váljon ez a dolgozat minden élőlény javára!

A szerző

1 A legfrissebb hardver leírás (datasheet) és a hozzá kapcsolódó alkalmazási segédletek (application notes) letölthetők a gyártó honlapjáról

2 A pontos specifikációkat a megfelelő fejezet tartalmazza

1. Az áramkörök specifikációja

Ebben a részben arról írok, hogy a különböző berendezésektől milyen általános és speciális funkciókat várok el, vagy milyen minőségi paramétereknek kell megfelelniük.

Az megvalósítandó készülékek az alábbi paraméterekkel, funkciókkal rendelkezzenek:

1. Általános szempontok
 - hálózatba köthető eszközök, gyűrű topológia: TokenRing-RS232 (továbbiakban TR-232)
 - vezérlő szoftver (konzol): egy Linux-ot futtató gépen, karakteres és grafikus (GTK+) felhasználói felület
 - vezérléshez szükséges függvénykönyvtár biztosítása (C nyelven, esetleg Python modulként³). Ennek hiányában használható a soros port natív kezelése is, a parancsformátum – mint látni fogjuk – nagyon egyszerű és kényelmes
 - JTAG csatlakozás nyomkövetéshez és firmware frissítéshez
2. Tápegység
 - szekunder oldali kapcsoló üzemű működés, bemeneti feszültsége stabilizálatlan egyenfeszültség (például egy már megépített, nem szabályozható tápegységből)
 - 3 szabályozott kimenet, minden kimenet a többitől függetlenül kezelhető: szoftveres szempontból külön-külön „egycsatornás” tápegységek látszik
 - 3 × 1 A-es maximálisan megengedett terhelőáram
 - szoftveres áramhatárolás, szoftveresen „cserélhető” biztosíték
 - 0-16 V-ig szabályozható kimeneti feszültség
 - a bementi feszültség maximális értéke 16 V
 - „I” szabályozó algoritmussal kompenzált szabályozási kör az optimális beállási idő és a minimális túllendülés elérése céljából
 - LED-ek a hibás vagy a megfelelő működés jelzése céljából
 - RS-232 kivezetés (3 vezetékes) hibaelhárításra (debug), távvezérlésre és kézi működtetésre (konzol)
 - ARM7TDMI processzorra épülő mikrovezérlő: ATMEL AT91SAM7S64⁴

Talán felmerül az Olvasóban a kérdés: „Miért éppen ARM7TDMI?” A válasz nem egyszerű, de az alábbi lista a teljesség igénye nélkül felsorolja az ARM alapú mikrovezérlők legfontosabb tulajdonságait:

- valódi 32 bites működés, akár operációs rendszer is futtatható az eszközön
- 4 gigabájtos lineáris címtartomány
- 16 ... 512 kb-ot FLASH programtár
- 4 ... 64 kb-ot RAM
- 55 MHz maximális órajel-frekvencia
- DMA
- 2 USART, DeBuG Unit
- SPI, TWI (I²C), USB
- 16 bites számlálók / időzítők
- PWM vezérlő (4 kimenet)
- watchdog
- 10 bites A/D átalakítók (8 csatorna)
- integrált RC oszcillátor

3 Python modul a távolabbi jövőben

4 Ez egy 64 lábú LQFP tokban kapott helyet, amit én már csak mérsékelt lelkesedéssel forrasztok be

1. Az áramkörök specifikációja

- BOR, POR, kvarcoszcillátor, PLL
- Boot program: SAM-BA
- TQFP tokozás

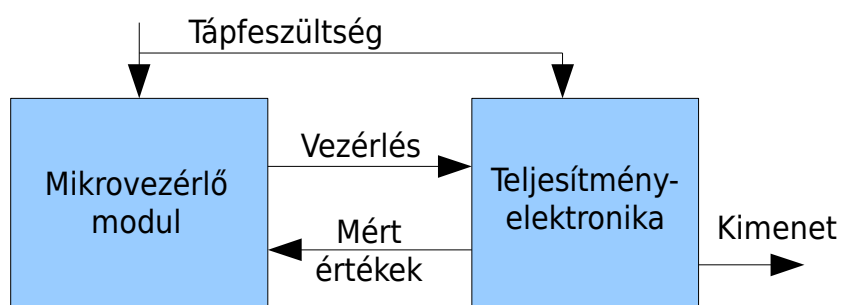
Azt hiszem, hogy a fenti lista magáért beszél. Mégis, a legnyomósabb érv az ARM mellett a JTAG interfész. Ez ugyanis lehetővé teszi, hogy ne vakon programozzak, hanem folyamatosan nyomon tudjam követni a processzor működését, a kódom futását. Nagyon sokat segít a hibák eliminálásában, ha tudom, hogy a program mely része nem működik helyesen.

2. Az áramkörök megtervezése és megépítése

2.1. Elvi kapcsolási rajzok és NYÁK-tervek

Ebben a fejezetben összefoglalom azon elvárásokat, melyek felmerülnek az áramkör elvi rajza és a nyomtatott áramköri tervének elkészítése során. Már az elején elmondhatom, hogy a kész áramkört két alapvetően eltérő részre fogom bontani. Ennek oka alapvetően abban keresendő, hogy a NYÁK-tervező programom nem engedi fél-Európa kártyánál nagyobb munkaterület használatát, meghatározva ezzel a részegységek maximális méretét.

Más okból is célszerű két részre bontani a munkát: az egyik modul alapvetően a mikrovezérlőt és annak kiegészítő áramköri elemeit tartalmazza, míg a másik részegység a teljesítményelektronikai feladatok megoldásában vesz részt. Ezen megfontolásokból kiindulva azon kell elgondolkodnom, hogy milyen interfész használata szükséges a két modul között. Könnyen belátható, hogy egyik modul sem működik tápenergia nélkül, így ennek megosztása szükségszerűnek tűnik. Ehhez kapcsolódik még az is, hogy a mikrokontroller vezérlő és ellenőrző jelekkel kommunikál a végrehajtó-beavatkozó szervként működő teljesítményelektronikai modullal.



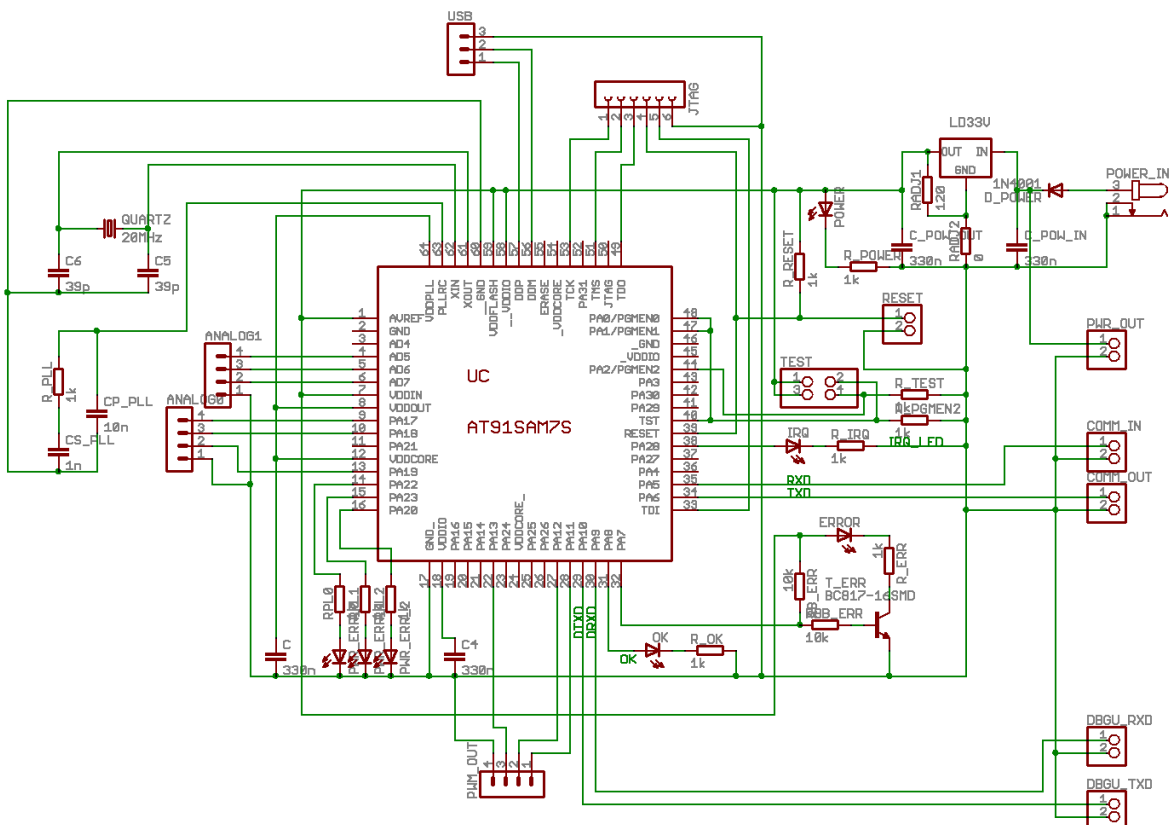
Hosszas gondolkodás, dokumentációolvasás és internetes keresgélés után arra a megállapításra jutottam, hogy a mikrovezérlő modul a következő részegységeket kell, hogy tartalmazza:

- mikrovezérlő
- tápfeszültséget előállító áramkör: a mikrovezérlőnek 3,3 V-ra és 1,85 V-ra van szüksége; ez utóbbit az eszköz maga állítja elő, így nekem csak a 3,3 V-os tápfeszültségről kell gondoskodnom.
- RESET áramkör: ezt biztosítja a mikrovezérlő. Egy felhúzóellenállást azért célszerű beépíteni, hogy még a zajos ipari környezet se okozzon problémát
- csatlakozók a kommunikációs interfészek (TR-232, DBGU, JTAG) és a jelvezetékek számára
- a panelen elhelyeztem néhány tűskét: ezek segítségével kényszerítem a mikrovezérlőt arra, hogy programozási-nyomkövetési (JTAG, SAM-BA) üzemmódba lépjen
- LED-ek: **a hibás működést jelző piros LED, ami még akkor is világít, ha a mikrovezérlő nem indult el megfelelőképpen** (erről egy FET-es kapcsolás gondoskodik, amit a mikrovezérlőnek „szándékosan” ki kell kapcsolnia ahhoz, hogy a LED kialudjon), a megszakításrutin megfelelő működését jelző **IRQ LED (belépéskor bekapcsolom, kilépéskor kikapcsolom, így tudom, hogy milyen gyakran fut le a szubrutin, és mennyi időt vesz el a főprogramtól)**, valamint a készenléti állapotot jelentő zöld LED-ek (globálisan és csatornánként)
- szűrőkondenzátorok: ezek nagyon fontosak az áramkör tartós és megbízható működéséhez

2. Az áramkörök megtervezése és megépítése

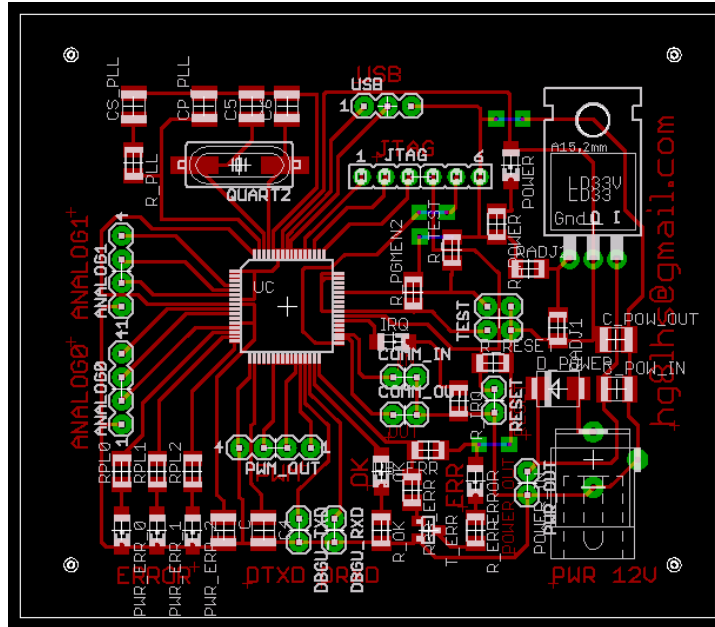
- PLL szűrője arra szolgál, hogy megfelelő módon húzzon be a PLL frekvenciaszorzó áramkör, amikor bekapcsolom azt. Ebben a műszerben nem találtam szükségesnek a használatát, de beépítettem, hogy szükség esetén rendelkezésre álljon
- kvarckristály és kondenzátorok a kvarcoszcillátor (főoszillátor) működtetéséhez. Ennek nagyon fontos szerepe van, hiszen a mikrovezérlőn belül minden esemény (időben) ehhez igazodik, tehát a kvarckristály frekvenciája döntő jelentőséggel bír. A mostani munkámban egy 20 MHz-es kvarcot használok, és ehhez igazítottam minden időzítést. Ha az Olvasó más frekvenciájú kristállyal szeretne dolgozni, akkor újra végig kell számolnia minden frekvenciafüggő paramétert...

Az előbbi követelményrendszer figyelembe vételével megtervezett kapcsolási rajz látható az alábbi ábrán:



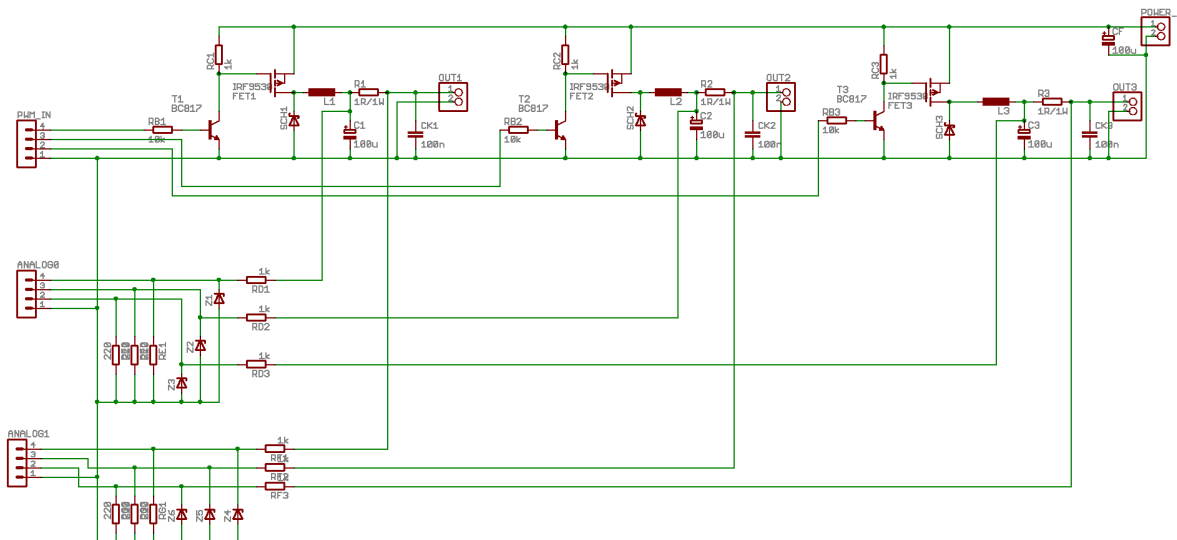
Az elvi kapcsolás megtervezésekor azt is szem előtt tartottam, hogy viszonylag könnyű legyen a nyomtatott áramköri rajz elkészítése, hiszen kétoldalas NYÁK-lemez hiányában szóba sem kerülhet átvezetések, átmenő alkatrészslábak, stb. alkalmazása.

Néhány órányi munka eredményeként elkészült a NYÁK-terv is. A végeredményt a következő ábra mutatja:



De nem szabad megfeledkezni a teljesítményelektronikai modulról sem. Ennek felépítése jóval egyszerűbb, mivel ez csak kapcsolóelemeket (p-csatornás MOSFET-ek, Schottky-diódák), szűrőket és passzív elemeket tartalmaz.

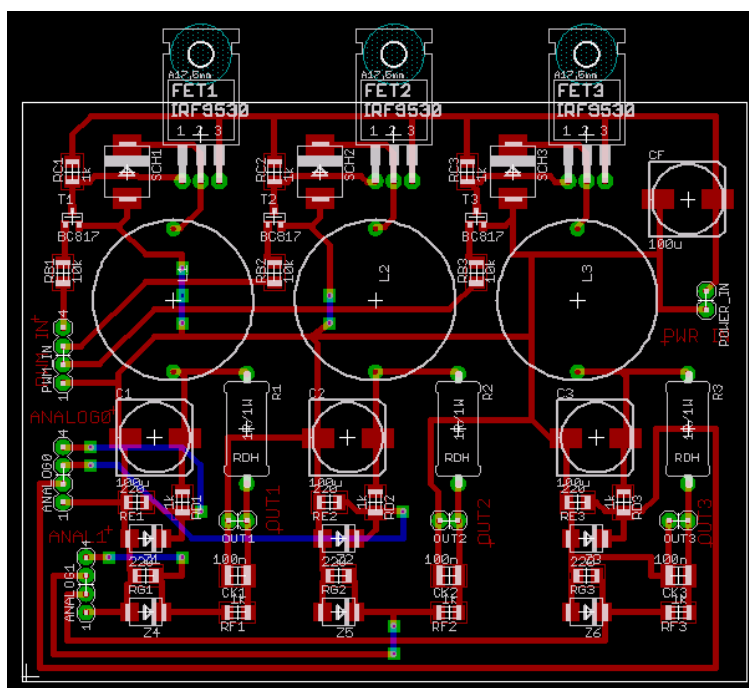
A kapcsolójel, mint bejövő jel, és a mért jelek külön-külön csatlakozókon érkeznek, ezzel megnövelem a tervezés szabadsági fokát, másrészt biztosítom magam számára annak lehetőségét, hogy az analóg bemeneteket nem veszélyeztetve tesztelhetem a PWM kimeneteket, kapcsolóelemeket, stb.



Az áramkör felső része ismerős kell, hogy legyen: ezek csak tranzisztorokkal meghajtott kapcsoló üzemű tápegység megoldások.

Az alsó rész olyan feszültségosztókat tartalmaz, melyek kimeneti feszültségét párhuzamosan kötött Zéner-diódák határolják. Ezek azért szükségesek, hogy megvédjék a mikrovezérlő analóg bemeneteit az esetleges túlfeszültségtől. Határolás hiányában könnyen tönkremehet a mikrovezérlő, annak cseréje pedig rengeteg plusz munkába kerülhet, jóval többre, mint néhány Zéner-dióda és ellenállás beépítése. A mikrovezérlő könnyen pótolható, de az elvesztegetett időt gyakran nehéz behozni.

A fenti áramkör NYÁK-terve az alábbi ábrán látható:



Bizonyára az Olvasó is észrevette, hogy a három FET „lelóg” a panelről. Ez nincs így, azok ugyanis állnak, hogy szükség esetén hűtőbordát tudjak szerelni a hátukra. Erre természetesen nem kerülhet sor, hiszen a kapcsoló üzemű működésből az következik, hogy a félvezetők nem melegedhetnek jelentősen. Ha igen, akkor ott a tervezőt kell vallatóra fogni.

Nem ejtettem szót az alkatrészek méretezéséről. Ennek oka az, hogy a következő fejezet éppen azon az úton vezet végig az Olvasót, amit magamnak is be kellett járnom, míg kezembe vehettem az azóta már elkészült, és kiválóan működő áramkört.

2.2. Paraméterek megállapítása: elméleti számítások

Ebben a fejezetben félvezető kapcsolók és a szűrőtagok tervezését ismerheti meg az Olvasó. A méretezés fontos részét képezi a mérnöki munkának, ezért ennek a résznek az áttanulmányozása feltétlenül ajánlatos.

A kapcsolóelemeken valójában nem sokat tudunk méretezni: a mikrovezérlő kimenetein 3,3 V csúcsfeszültségű impulzusok jelennek meg, ehhez illesztettem egy tranzisztorból álló áramkört, amely az előbbi jelet (impulzussorozatot) „erősíti”. Ez az erősített jel biztosítja a FET számára a gate-source feszültséget. A pontos számításokat mellőzöm, egy földelt emitteres inverter tervezése a digitális technika alapjainak tárgyalásakor került elő.

A FET kiválasztásakor a következő megfontolások vezettek: p-csatornásnak kell lennie, hogy a tápegység kimenő jele a földhöz képest jelenjen meg; valamint akkora csatornaellenállással kell rendelkeznie, hogy semmiképpen ne okozzon gondot a kapcsolt áram szaggatása (ez maximálisan 1 A). Arról sem szabad megfeledkezni, hogy zárt állapotban a FET source-a és drain-je között megjelenő feszültség nem okozhat maradandó károsodást: olyan típust szükséges választani, melynek letörési feszültsége nagyobb, mint 32 V. A felsorolt követelményeknek igen sokféle FET-típus tesz eleget, ebben az áramkörömben az IRF 9530-at használtam.

Logikailag nem ide illik, mégis itt tárgyalom a kimeneti áramot mérő ellenállás kiválasztásának szempontjait. Szerencsére egyszerű kiszámolni ennek paramétereit: mivel a kimeneti áram maximális értéke 1 A, és az 1 V-os feszültségesés éppen megfelelő az áramméréshez, ezért az ellenállás 1 Ω -os, és legalább 1 W-os legyen.

A kimeneti szűrőkondenzátorokat a lehető legnagyobbra célszerű választani, ugyanis a nagyobb értékű kondenzátor (és fojtótekercs) hatékonyabban szűri a kimeneten megjelenő zavarjelet. Jelen áramkörben három darab 100 μ F-os kondenzátor kapott helyet.

A kimeneten látható 100 nF-os kondenzátorok is zajszűrésre szolgálnak, ezek a terhelésről érkező nagyfrekvenciás zavarokat szűrik.

Mindkét kondenzátortípus (tartósan!) el kell hogy viselje a kimeneti feszültséget, mint üzemi feszültséget, ezért ennek megfelelően kell az alkatrészeket kiválasztani.

Az áramkörök legnehezebben számítható alkatrészéhez, a fojtótekercshez érkeztünk. Azért mondom ezt, mert a tervezés során számos, egymásnak ellentmondó követelményt kell kielégíteniük: például a menetszámot végtelen nagyra lenne célszerű választani, hogy minél jobb legyen a szűrés hatásfoka, de akkor a mágneses indukció értéke is végtelen lenne, ami – nagy valószínűséggel – telítésbe vinné a vasmagot (tehát nullára esne vissza a tekercs induktivitása).

Abból indultam ki, hogy még a terhelőáram legnagyobb értékénél sem mehet a vasmag telítésbe (a maximálisan megengedhető vasmagindukció eszközfüggő és katalógusadat, de a leírásokban eddig csak $B_{max} = 0,4$ T maximális indukciót láttam, tehát ez jó kiindulási alap lehet). A relatív mágneses permeabilitás szintén katalógusadat, nálam ennek értéke éppen 500-ra jött ki (10 menetes tekercs induktivitásából – ami mérhető – számítottam ki).

$$B_{max} = \frac{\mu_0 \cdot \mu_r \cdot N \cdot I_{nom}}{d \cdot \pi}$$

ahol N a tekercs menetszáma, I_{nom} a terhelőáram névleges (pontosabban maximális értéke), d pedig a vasmag (gyűrű) közepes átmérője. A képletből a menetszám $N = 30$ -nak adódott 1 A maximális terhelőáram esetén.

Ezek után kiszámítható a tekercs induktivitása:

$$L = \frac{\mu_0 \cdot \mu_r \cdot N^2 \cdot A}{d \cdot \pi}$$

A kijelölt műveletek elvégzése után arra jutottam, hogy a tekercs 530 μH induktivitású. Ez az adat ahhoz szükséges, hogy megállapítsam, hogy mennyi idő alatt alakul ki a kimenő áram maximális értékével megegyező áram a tekercsen, ha a tápfeszültség teljes értékét rákapcsolom (ez ugye a legrosszabb eset, ami a tekercssel történhet az áramkörben). Ennyi ideig ugyanis a tekercs még nem megy telítésbe – induktivitásként viselkedik –, tehát megfelelően használható.

Ha a tekercs telítésbe megy, akkor az induktivitása nullára csökken, mert a hiszterézisgörbe meredeksége (ami a relatív permeabilitással arányos) nullára csökken.

Ismeretes, hogy

$$U_i = -L \cdot \frac{dI}{dt}$$

Ha a feszültség értéke konstans, és $T = 0$ -ból indul az időmérés, akkor igaz az, hogy

$$U_t = L \cdot \frac{I_{nom}}{T}$$

ahol U_t a tápfeszültség értéke, T pedig az az időtartam, amíg a telítés nem következik be, vagyis a kapcsolójel periódusidejének maximális értéke.

Az előbbi összefüggésekből kiszámoltam, hogy az áramkörömnek legalább 33 kHz-es kapcsolójel kell szolgáltatnia. A megépített készülék kb. 40 kHz-cel dolgozik, tehát ez a feltétel teljesül (nem melegszi a kapcsoló-FET, ami arra utal, hogy a számítások megfelelnek a valóságnak).

A végére hagytam a feszültségosztók méretezését: a maximális 16 V-os kimenethez 3,3 V analóg bemeneti feszültség tartozik. Ha az osztók „földhöz közelebbi” komponensei 220 Ω -osak, míg a felső ellenállások 1 k Ω -osak, akkor a 16 V kimenő feszültséghez 2,88 V bemenő jel tartozik.

Mivel az osztón folyó áram értéke elhanyagolható (mA nagyságrendű), így alkalmazhatók a kis terhelhetőségű SMD ellenállások is.

2.3. Alkatrészlista

Ez a rész az áramkör megépítéséhez szükséges alkatrészek teljes listáját tartalmazza. A táblázat alapját a NYÁK-tervező program szolgáltatta, amit kiegészítettem a csatlakozókkal, a dobozzal, stb.

Az alkatrészek a piaci ára kb. 3500 Ft, a mikrovezérlőt (1440 Ft) és a műszerházat (1300 Ft) külön kell megvásárolni. A teljes összeg 6000-6500 Ft, ami nem sok a kereskedelmi forgalomban kapható hasonló paraméterekkel rendelkező tápegységek árához képest.

4x	100u/SMD
4x	330n/1210
2x	39p/1210
1x	10n/1210
1x	1n/1210
2x	100n/1210
3x	IRF9520 (p-csat. MOSFET)
1x	78L05 (SO-8)
1x	LD33V
7x	BC817-SMD
3x	B 340, Schottky dióda 1A, SMD
8x	Zener 3.3V MELF
5x	1N4001/SMD
4x	chipléd (2x zöld, 1x piros, 1x sárga)
1x	nagy fényerejű piros LED (5 mm)
4x	nagy fényerejű zöld LED (5 mm)
3x	ferritgyűrű (20x10x10 mm)
2x	tűskecsor (40 pól.)
2x	hűvelysor (40 pól.)
	zsugorcső
2x	mono 2.5 mm Jack aljzat, házba építhető
2x	mono 2.5 mm Jack dugó, lengő
1x	DC aljzat, 2.1/5.5, házba építhető
1x	DC dugó, 2.1/5.5, lengő
1x	DSUB 9 Female + ház
6x	banánhüvely (= 3x fekete, 3x piros)
2x	banándugó (= 1x fekete, 1x piros)
5x	LED foglalat
1m	telefonkábel (4 ér)
3x	1R/2W
6x	10k/1206
25x	1k/1206
8x	220
1x	2.2k/1206
20x	0R/1206
1x	20MHz kvarc, SMD
1x	műszerház
1x	AT91SAM7S64 (ebből célszerű többet beszerezni)

3. A szoftver környezet

Az előző részben azt az utat mutattam meg, hogy miként lehet az ötlettől a kész nyomtatott áramkörtől eljutni. Ebben a fejezetben azt írom le, hogy hogyan lehet összeállítani a (szoftver)fejlesztői környezetet, és ehhez milyen komponensek szükségesek.

Feltételezem, hogy az Olvasó ismeri a Linux programozásának rejtélyeit, ha mégsem így lenne, akkor javaslom, hogy ismerje meg, mert területi okokból nincs lehetőségem a szükséges programozási tudást átadni.

Egyik régebbi munkámban így írok a C fordítóról:

„Mivel a számítógép és a mikrovezérlő egyaránt 32 bites, vagyis az egy lépésben kezelhető adatok 32 bitesek, ezért a forráskódok – ott, ahol ez lehetséges – megegyeznek egymással (elsősorban header fájlok, kommunikációs rutinok).

A XXI. században már nem vagyok hajlandó assembly nyelven programozni, ezért a programokat C fordítóprogram segítségével fogom elkészíteni. Szerencsére a GNU prozsekt gcc-je fordít Intel IA32 és ARM architektúrára, így ugyanazt a fordítóprogramot használhatom mindkét processzorra.

A gcc szabad szoftver, szabadon beszerezhető a <http://www.gnu.org/gcc> oldaltól. Célszerű a forráskódját letölteni, és azt lefordítani a célgépre. Szükség lesz továbbá a binutils programcsomagra is. Az elkészült programot valahogyan a mikrovezérlőbe kell juttatni. Erre szolgál az OpenOCD (Open On-Chip Debug) nevezetű program. Természetesen ez is letölthető az internetről.”

3.1. C fordító IA32 architektúrára

Mivel én kizárólag POSIX szabványnak megfelelő operációs rendszert (aktuálisan Linuxot) használok, ezért nagyon egyszerű dolgom van: a gcc minden Linux disztribúciónak része. Ez a fordítóprogram a személyi számítógép számára készít futtatható állományokat, így a felhasználói program (konzol) elkészítéséhez fog hozzájárulni.

Maga a telepítés rendkívül egyszerű, Debian alapú Linux disztribúciók esetén (Pl. Ubuntu) nem tartogat semmilyen meglepetést. Léteznek különböző grafikus telepítőprogramok, én viszont a legegyszerűbb módszerrel installáltam fordítóprogramomat:

```
$ sudo apt-get install gcc make
```

Ezzel a paranccsal utasítást adtam a számítógépnek, hogy töltsse le, majd telepítse a C fordítót és a fordítást automatizáló „make”-et. Ha minden rendben megy, akkor néhány másodperc múlva ki is próbálhatom:

```
$ gcc
gcc: no input files
```

Ha eddig minden rendben történt, akkor nem lesz problémám a számítógép programjának elkészítésével. Fontos azt is tudni, hogy ezzel a C fordítóval fogom a gcc-t lefordítani úgy, hogy az új fordító (a keresztfordító) az ARM processzor számára generáljon futtatható állományt, ezért a gcc megfelelő működése alapvető fontosságú.

3.2. C fordító ARM architektúrára

A már előzőleg letöltött *binutils* és *gcc* forrást le kell fordítani. Ezt az alábbi módon célszerű megtenni:

3. A szoftver környezet

- kicsomagolom a tömörített állományt:

```
tar xjvf binutils-2.17.tar.bz2|
```

- belépek a *binutils-2.17* könyvtárba
- elvégezem a konfigurálást:

```
./configure --target=arm-elf --prefix=/usr/local/arm
```

- lefordítom a forrást a „make” paranccsal (ez néhány percre tart)
- végül feltelepítem a „sudo make install” paranccsal

Ezt követi a **C fordító** fordítása:

- kicsomagolom a másik tömörített állományt is:

```
tar xjvf gcc-4.0.3.tar.bz2
```

- belépek a *gcc-4.0.3* könyvtárba, létrehozom az *armgcc* könyvtárat, és belelépek:

```
cd gcc-4.0.3; mkdir armgcc; cd armgcc
```

- elvégezem a konfigurálást:

```
export PATH+="/usr/local/arm/bin";  
../configure --prefix=/usr/local/arm --target=arm-elf --enable-languages=c
```

- ezután már csak a fordítást („make”) és az installálást („sudo make install”) kell elvégeznem

A fordítás és telepítés után a keresztfordító (ami már ARM architektúrára fordít) a **/usr/local/arm/bin** könyvtárban található, és **arm-elf-gcc** névre hallgat.

3.3. JTAG programozó szoftver: az OpenOCD

Az *OpenOCD*⁵ egy nyílt forráskódú program az ARM alapú mikrovezérlők programjának feltöltésére és a program futásának ellenőrzésére. Fordítása a következőképpen történhet:

- kicsomagolom az *openocd* tömörített állományát, vagy letöltöm az aktuális fejlesztői forrást
- belépek a *trunk* könyvtárba
- elvégzem a konfigurálást:

```
./bootstrap;  
./configure --enable-parport  
■ vagy  
./configure --enable-parport --enable-parport_ppdev
```

Ez utóbbi (két) argumentum azért szükséges, mert alapértelmezés szerint nincs engedélyezve a párhuzamos (nyomtató) porton való kommunikáció.

USB-s JTAG interfész használata esetén (a konfigurálást megelőzően) fel kell telepíteni **libusb** és **libftdi** függvénykönyvtárakat is. Konfigurálásnál kötelező az „--enable-ft232r_libftdi” argumentum megadása is!

5 Open On-Chip Debugger

Az USB-s JTAG áramkörök kapcsolási rajza egy régebbi TDK munkám első részének végén található.

- lefordítom a forrást a „make” programmal, majd installálom a már megismert módon

A lefordított program a `/usr/local/bin/openocd` nevet viseli.

Az *OpenOCD*-t úgy tervezte a programozója, hogy *telnet* program segítségével lehessen irányítani. Ez azért hasznos, mert a világ bármely pontjáról elérem mikrovezérlőm programozó szoftverét. A másik érdekessége az *OpenOCD*-nek, hogy össze lehet kapcsolni a *gdb*-vel, így közvetlenül a C forrás segítségével ellenőrizhetem a mikrovezérlőt és a lefordított-feltöltött programot. Ez bizony nagyon nagy segítség bonyolultabb algoritmusok esetén.

A mikrovezérlő programjának áttöltéséhez el kell indítani az *openocd*-t. Az *openocd* egy szöveges konfigurációs fájlból veszi az indulási paramétereket. A fejlesztés során használt konfigurációs fájl így néz ki:

```
telnet_port 4444
gdb_port 3333
interface parport
parport_port 0x378
parport_cable wiggler
jtag_speed 0
reset_config srst_only
jtag_device 4 0x1 0xf 0xe
target arm7tdmi little reset_halt 0 arm7tdmi
flash bank at91sam7 0 0 0 0
```

Az első két sor a *telnet* és a *gdb* hálózati TCP portját adja meg. Ezt követi a programozó interfész megadása. A példánál maradva ez a párhuzamos port, ami a 0x378-as I/O porton van. A kábel típusa „hg8lhs” lesz, vagyis egy általam definiált kiosztás. Ez teljesen megegyezik a Wiggler kiosztással, de én – Wigglerrel ellentétben – nem invertálom a RESET jelet. Úgy láttam a forráskódban, hogy mostanában már ő sem.

A párhuzamos portra csatlakozó JTAG programozó sebessége legyen 0 (valójában nincs szó semmilyen programozóról, így a sebesség értéke nem befolyásol semmit, de szintaktikailag szükséges).

Beállítom a RESET viselkedést is: mivel jelen áramkörben nincs TRST (target RESET, JTAG RESET), csak SRST (System RESET, rendszer alaphelyzetbe állítás) bemenet, ezért ezt is tudatni kell a programmal. A következő sorban azt határozom meg, hogy a legalacsonyabb JTAG szinten milyen hosszú a parancs regiszter, és melyik parancs szolgál az eszköz azonosítására. Ezekből egy szót sem értek.

A processzor megadása kötött formát követ. A FLASH memória definiálása sem hagy sok lehetőséget a változtatásra, az *openocd* minden paramétert a lapkából kér le.

Csatlakoztatva a mikrovezérlőhöz a JTAG kábelt, rövidre zárom a RESET kapcsolót (anélkül nem lép DEBUG üzemmódba), újraindítom a mikrovezérlőt, és kiadom következő parancsot:

```
$ sudo openocd -f config
Info: openocd.c:86 main(): Open On-Chip Debugger (2007-05-30 17:45 CEST)
```

Amennyiben üdvözlő szöveg után az *OpenOCD* nem tér vissza hibaüzenettel, akkor minden sikeres volt. Ha nem ezt látom, akkor egy üzenet tájékoztat a hiba okáról. Ennek oka többnyire a mikrovezérlő tápfeszültségének hiánya, a RESET átkötés rövidre nem zárása, esetleg hibásan kivitelezett JTAG kábel. Ezekon kívül persze a hibák száma végtelen, de hely hiányában nem tudom az összes hibát (és azok elhárításának módját) tételesen felsorolni.

Innentől kezdve két dolgot tehetek: a *telnet* program segítségével belépek az *openocd*-be, vagy elindítom a *gdb*-t. Véleményem szerint az *openocd* több lehetőséget biztosít a

hibafelderítésre, ezért a *gdb* használatát még röviden sem tárgyalom.

Sajnos időbeli és terjedelmi okokból nem áll módomban tovább foglalkozni az *openocd*-vel, némi gyakorlás által az Olvasó is megfelelő jártasságot szerezhet a program használatában. A kiadható parancsok rövidített listája megtalálható az *openocd* dokumentációjában.

3.4. Teljes mintaprogram

Ebben a részben egy rendkívül egyszerű, de teljes és működő mintaprogrammal örvendeztetem meg a kedves Olvasót.

Legnagyobb bánatomra minden részletet nem ismertethetek, az ARM processzor működésének alapjait az Olvasónak magának kell megszereznie. A szoftverkomponensek használatát viszont teljes részletességgel bemutatom: egyrészt áttekintem a mikrovezérlő „vezérlőként” való alkalmazását, másrészt összefoglalom a programletöltés lépéseit.

Szükség lesz egy C fordítóra, ami ARM architektúrára fordít. Erről már a „A szoftver környezet” fejezetben volt szó, így ezzel nem kívánok foglalkozni.

Ezen kívül el kell készíteni a forrásprogramot és a megfelelő *Makefile*-t. Ez utóbbi arra szolgál, hogy forgatókönyvként szolgáljon a *make* számára.

A C forráskód (*main.c*) tartalma a következő:

```
#include "AT91SAM7S64.h"

// Definíciók
#define IRQ_FLAG          0x80
#define FIQ_FLAG          0x40
#define THUMB_FLAG        0x20

#define USER_MODE         0x10
#define FIQ_MODE           0x11
#define IRQ_MODE           0x12
#define SUPERVISOR_MODE   0x13
#define ABORT_MODE         0x17
#define UNDEF_MODE         0x1b
#define SYSTEM_MODE        0x1f

#define IRQ_STACK_SIZE    1024
#define FIQ_STACK_SIZE    IRQ_STACK_SIZE

#define DELAY      500

// Speciális függvények: assembly fv. és megszakításkezelő rutinok
void _start() __attribute__((naked));
void _stop() __attribute__((naked));
int main() __attribute__((naked));
void fiq() __attribute__((interrupt("FIQ")));
void irq() __attribute__((interrupt("IRQ")));

// Assembly belépési pont
void _start() {
    asm("b main");
    asm("b _stop");
    asm("b _stop");
    asm("b _stop");
    asm("b _stop");
    asm("b _stop");
    asm("ldr pc, [pc, #-0xF20]");
    asm("b fiq");
}

// Itt álljon meg a processzor hiba esetén
void _stop() {
    for(;;);
}

void delay() {
    unsigned int a;
```

```

        for (a = 0; a < DELAY; a++)
            asm("nop");
    }

// C program belépési pontja
int main() {
    *AT91C_PIOA_PER = 0xffffffff;
    *AT91C_PIOA_OER = 0xffffffff;

    *AT91C_PIOA_PPUER = 0xffffffff;
    *AT91C_PIOA_MDDR = 0x00000000;

    while (1) {

// LED be:
        *AT91C_PIOA_CODR = 0xffffffff;
        *AT91C_PIOA_SODR = 0xaaaaaaaa;
        delay();

// LED ki:
        *AT91C_PIOA_CODR = 0xffffffff;
        *AT91C_PIOA_SODR = 0x55555555;
        delay();

    }

    return 0;
}

// Megszakításkiszolgáló rutinok
void fiq() {
}

void irq() {
}

```

A forráskód legelején az ARM processzor néhány konstansát (bitek, processzor üzemmódok) definiáltam. Ezután következik néhány „rendszer”-függvény deklarálása: a **„naked”** attribútum arra utasítja a fordítót (gcc specifikus), hogy hagyja el a függvény assembly preambuláját (használt regiszterek mentése, lokális változók létrehozása és inicializálása, stb.) és lezárását (regiszterek visszaolvasása, verem visszaállítása, stb.). Az **„interrupt”** attribútum tájékoztatja a fordítót, hogy a függvény preambuluma és lezárása nem a szokásos függvényekével azonos, hanem speciális: a használt regisztereken kívül menteni kell a gépi állapotregisztert, és vermet kell váltani, ugyanis a főprogram, az IRQ kiszolgáló rutinja és a FIQ⁶ rutinja más-más vermet használ, nehogy összekeveredjenek az adatok (ami még biztonsági rés is lenne).

A **„_start()”** függvény a program belépési pontja, a linkernek szüksége van erre a címkére. Eredetileg ez egy **assembly függvény**, de én **„naked” C függvényt** csináltam belőle, ami ugyanaz a megoldás – egy régebbi TDK munkám első részében a **„_start()”**-ot még assembly rutinként tartottam számon.

A **„_start()”** 8 db utasítást tartalmaz, ezek a processzor kivételkezelő rutinjainak⁷ belépési pontjai. Látható, hogy az utasítások mindegyike ugró utasítás, hiszen 1 db utasítás ritkán elég egy kivétel lekezeléséhez. Érdekes a **„ldr pc, [pc, #-0xF20]”** utasítás, mert az ugrás címét egy hardver elemtől, a fejlett megszakításkezelőtől (Advanced Interrupt Controller) kérdezi le, így választja ki a különböző hardver eseményekhez tartozó (különböző) kiszolgáló rutinokat. Nagyon elmés megoldás. Ezt én is fogom használni, de nálam – lévén egyetlen megszakításforrás – kevésbé lesz jelentősége.

A **„main()”** függvény a C forrás belépési pontja (tehát **„naked”**), ide ugrik RESET után a

6 Fast interrupt – gyors megszakítás. Bankolja a regiszterek egy részét, így azok mentésével és visszaállításával nem kell foglalkoznia a processzornak. Ezzel a megoldással jelentősen csökken a kiszolgáló rutin lefutásának ideje

7 Processzor (újra-)indítás (RESET), nem definiált utasítás (UNDEFINED_INSTRUCTION), szoftver megszakítás (SWI), előolvasási hiba (PREFETCH_ABORT), adatolvasási hiba (DATA_ABORT), ezt a processzor nem használja, normál megszakítás (IRQ), gyors megszakítás (FIQ)

3. A szoftver környezet

processzor: („b main”). Első lépése a veremk létrehozása, majd beállítja a kimeneti-bemeneti portokat, és néhány Hz frekvenciájú négyszögjelet szolgáltat azokon. Kiválóan tesztprogram ez: felváltva villognak a LED-ek, így könnyű kideríteni, hogy jó-e a mikrovezérlő, a kód és a LED.

Az utolsó két függvény csak szintaktikai okokból található ebben a kódban, elvileg ezek a megszakítások kiszolgáló rutinjai. Most nincs megszakítási forrás, így nincs mit kiszolgálni.

Az előbb megírt kódot le kell fordítani, hogy aztán a mikrovezérlőbe tölthessem. Arra mutatok most példát, hogy hogyan célszerű elkészíteni a *Makefile*-t:

```
PREFIX = /usr/local/arm/

CC      = $(PREFIX)/bin/arm-elf-gcc
LD      = $(PREFIX)/bin/arm-elf-ld
CFLAGS  = -Wall -g -mlittle-endian -marm
CFLAGS += -O3
#CFLAGS += -O1
#CFLAGS += -O0
#CFLAGS += -Os
LDFLAGS = -Ttext 0x100000 -Tdata 0x200000
LIBGCC  = $(PREFIX)/lib/gcc/arm-elf/4.0.3/libgcc.a

OBJCOPY = $(PREFIX)/bin/arm-elf-objcopy
OCFLAGS = -j .text -O

OBJDUMP = $(PREFIX)/bin/arm-elf-objdump
ODFLAGS = -h -j .text -j .data -dS

PROG     = program
OBS      = main.o

all:      clean $(OBS)
          $(LD) $(LDFLAGS) -o $(PROG) $(OBS) $(LIBGCC)
          $(OBJCOPY) $(OCFLAGS) binary $(PROG) $(PROG).bin
          $(OBJCOPY) $(OCFLAGS) ihex $(PROG) $(PROG).hex
          $(OBJDUMP) $(ODFLAGS) $(PROG) > $(PROG).list

clean:
          rm -f *core *.o *.hex *.bin *.list $(PROG) messages.h
```

A *Makefile* nem követ bonyolult szintaktikát. Az elején beállítom azt, hogy hol található az ARM architektúrára fordító komponensek, majd meghatározom a C fordító és a linker nevét. Ezt követi az argumentumok megadása (a részletes leírást a *gcc* kézikönyve tartalmazza). Ami viszont nagyon fontos: a FLASH memória a 100000_{HEX} címen kezdődik, a RAM terület pedig a 200000_{HEX} címen található. Ezt tudatni kell a linkerrel, hogy a végül minden oda kerüljön a memóriában, ahova kerülnie kell.

A *libgcc*-t célszerű hozzáfűzni a kódhoz. Ez tartalmazza például a lebegőpontos műveleteket és az osztást végző függvényeket.

Az *objcopy* a különböző formátumok közötti konvertálást segíti (ELF ⇒ bináris, ELF ⇒ HEX).

Az *objdump* segítségével készíthetünk olyan állományt (listing fájl), ami keverve tartalmazza a C kódsorokat, és az azokból generált assembly utasításokat. Nagyon jól használható ez hibakeresésre és -elhárításra.

A fordítás takarítással kezdődik, majd előállítatom a tárgykódú állományokat. A tárgykódok és a függvénykönyvtárak összefűzéséből keletkezik a bináris végeredmény (program⁸ és program.bin), amit a későbbiekben a FLASH-be töltök. Időközben előáll az előbb már említett listing fájl is.

8 ELF: executable and linkable format – futtatható és összefűzhető formátum; bináris, de még van fejléce, ebből állítom elő a tényleges bináris végeredményt (program.bin – fejléc nélküli, nyers)

A *telnet* programmal bejelentkezek az *openocd*-be, majd a *program.bin*-t a mikrovezérlő 0. bankjának 0. offset címére töltöm:

```
halt  
flash write 0 program.bin 0  
resume 0
```


4. A mikrovezérlő hardver eszközeinek használata

Az előző részben bemutatam egy kicsi, de jól használható mintaprogramot. Ebben a fejezetben a mikrovezérlő hardvereszközeinek programozásával fogok foglalkozni. Az így elkészített alapfüggvényekből – mint építőkockákból – fel tudom építeni a berendezést vezérlő teljes kódot.

4.1. Hibakonzol (DBGU)

Habár elsőre nem tűnik túl fontosnak, mégis ez a leglényegesebb része a mikrovezérlőnek. A DBGU eszköz arra szolgál, hogy nyomkövetési (debug) üzeneteket küldjön a mikrovezérlő a számítógépnek, és várja a reakciót a számítógéptől.

Tulajdonképpen nem másról van szó, mint egy 3 vezetékes (TXD, RXD, GND) RS-232 portról. Mivel a PC-m elég réginek mondható, ezért biztosan találom a hátlapján RS-232 kivezetést. Az RS-232 interfész programozása igen egyszerű, mind Linux alatt, mind a mikrovezérlő oldaláról. Ennek alátámasztására álljon itt egy idézet egy korábbi munkámból:

„Ahhoz, hogy a számítógép, mint mérésvezérlő utasításokat adhasson a mikrovezérlőnek, szükséges, hogy valamiféle adatátvitel jöjjön létre a két eszköz között. Ennek egyik legegyszerűbb, és ezért legelterjedtebb módja az RS-232 aszinkron soros átvitel. Aszinkron, mert nem visszük át az órajelet, így azt mindkét fél maga állítja elő az adatfolyam alapján, másrészt soros, mert a bitek nem egyszerre, hanem egymás után kerülnek át egyik gépről a másikra.

A hardvert leíró részben már láttuk, hogy rendkívül egyszerű összekábelezni a két berendezést: az egyik készülék TXD lábát a másik RXD lábára kell kötni, és viszont. Természetesen szükségünk lesz egy földvezetékre, így összesen 3 kábelt alkalmazunk a kommunikációhoz.

Az adatátvitel paramétereit (sebesség, adatbitek száma, hibaellenőrzés) szoftveres úton állíthatjuk be: az ipari gyakorlatnak megfelelően válasszuk a 9600 bit/sec sebességű kommunikációt 8 bit adathosszal, hibaellenőrzés nélkül és 1 stop bittel (9600 8N1). A következő programrészlet azt mutatja be, hogy egy Linux-os gépen mindezt hogyan tehetjük meg a legegyszerűbben: ...”

```
#include <termios.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "commands.h"
#include "usart.h"

FILE *setup_usart0(char *tty_name) {
    FILE *tty;
    struct termios term;

    tty = fopen(tty_name, "r+");

    if (tty == NULL) {
        return NULL;
    }

    tcgetattr(fileno(tty), &term);
    cfsetispeed(&term, B9600);
    cfsetospeed(&term, B9600);
    term.c_cflag &= ~(CSIZE|PARENB);
    term.c_cflag |= CS8;
    tcsetattr(fileno(tty), TCSANOW, &term);

    return tty;
}
```

4. A mikrovezérlő hardver eszközeinek használata

A fenti függvény paraméterként a soros portot reprezentáló fájl nevét (tipikusan „/dev/ttyS0”, vagy „/dev/ttyUSB0”) várja, visszatérési értéke a fájlkezelést lehetővé tevő adatstruktúra mutatója (FILE *).

A soros port megnyitása után beállítja az átvitel paramétereit: sebességet, adathosszat és a stopbitek számát.

Linux alatt (szinte) minden eszköz fájlként kezelendő, ezért minden művelet végső soron fájlművelet. Nincs ez másként a soros port írásával és olvasásával kapcsolatban sem:

Az írás (adat küldése a mikrovezérlőnek):

```
fwrite(packet, sizeof(struct Packet), 1, tty);
```

Az olvasás (adat fogadása a mikrovezérlőtől):

```
fread(packet, sizeof(struct Packet), 1, tty);
```

A mikrovezérlő oldaláról nézve sem túl összetett a feladat, hiszen annak is pontosan ezeket a paramétereket kell beállítania. Mivel a megfelelő előkészítő, beállító függvényeket az ATMEL programozója már megírta, ezért ezek újraírását feleslegesnek tartom, és mellőzöm. Helyette az ATMEL függvényeinek használatára koncentrálok, és remélem, hogy a programozó nem tévedett.

```
#include "AT91SAM7S.h"
#include "lib_AT91SAM7S.h"

inline void dbgu_init() {
    AT91F_DBGU_CfgPMC();
    AT91F_DBGU_CfgPIO();
    AT91F_US_Configure((AT91PS_USART) AT91C_BASE_DBGU,
        20000000, AT91C_US_ASYNC_MODE, 9600, 0);
    AT91F_US_EnableRx((AT91PS_USART) AT91C_BASE_DBGU);
    AT91F_US_EnableTx((AT91PS_USART) AT91C_BASE_DBGU);
}
```

Röviden összefoglalom, hogy mit is csinál a „dbgu_init()” függvény, annak ellenére, hogy a függvénynevek elég beszédesek:

Először rákapcsolja az órajelet a DBGU perifériára. Ez nem kell a beállításhoz, de az adatkommunikációhoz nagyon fontos, ezért jobb ezt még az elején megtenni. Ezt követi a port lábak hozzárendelése a perifériára kimeneteihez.

A következő sor – a 20 MHz-es órajelhez igazodva – beállítja az átviteli sebességet és a többi paramétert (8 adatbit, nincs paritás, 1 stopbit).

Végül engedélyezi az adatok vételét és adását.

Ha már a vételnél tartunk, bemutatom az adatok vételét és adását végző függvény:

```
inline char dbgu_recv_char() {
    while (!AT91F_US_RxReady((AT91PS_USART) AT91C_BASE_DBGU));
    return AT91F_US_GetChar((AT91PS_USART) AT91C_BASE_DBGU);
}

inline void dbgu_send_char(char ch) {
    while (!AT91F_US_TxReady((AT91PS_USART) AT91C_BASE_DBGU));
    AT91F_US_PutChar((AT91PS_USART) AT91C_BASE_DBGU, ch);
}
```

A két függvény annyi pluszt tartalmaz az ATMEL eredetihez képest, hogy mindkettő megvárja, míg a vételi regiszterben megjelenik az adat, illetve az adási regiszter kiürül (1-1 „while”-ciklus, ami a megfelelő állapotbitet figyeli, blokkolva a hívó program futását).

4.2. LED-ek

Ezek az apró alkatrészek valóban nagyon fontosak. Mivel az ember vizuális lény, jogos a gondolat, hogy a mikrovezérlő az állapotáról jól láthatóan tájékoztassa felhasználót.

A LED-ek többnyire közvetlenül a mikrovezérlő lábaira vannak kötve, így tulajdonképpen a portok programozását kell áttekinteni. Az ATMEL programozója rengeteg hasznos függvényt biztosított, így azokon keresztül elérhetem a LED-eket.

Minden LED állapotát egy bit reprezentálja a kimeneti port regiszterben. Azt, hogy pontosan melyik bit, a „ports.h” fájl tartalmazza:

```
#ifndef __PORTS_H__
#define __PORTS_H__

#define ERROR_LED 7
#define OK_LED 8
#define IRQ_LED 28

#define PWR1_LED 22
#define PWR2_LED 23
#define PWR3_LED 20

#define PWM1 13
#define PWM2 12
#define PWM3 11

#endif
```

A PWMx értékek a PWM beállításánál kerülnek szóba, most nincs jelentőségük.

A megfelelő működés („OK”) és a hibás működés („ERROR”) állapotot a következő két (plusz egy: mindkét LED-et kikapcsoló „OFF”) függvényvel könnyen kijelezhetem („report.h” fájl tartalma):

```
#ifndef __REPORT_H__
#define __REPORT_H__

#include "ports.h"

inline void report_error() {
    AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << ERROR_LED);
    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, 1 << OK_LED);
}

inline void report_ready() {
    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, 1 << ERROR_LED);
    AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << OK_LED);
}

inline void report_off() {
    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, 1 << ERROR_LED);
    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, 1 << OK_LED);
}

#endif
```

4. A mikrovezérlő hardver eszközeinek használata

Tudni érdemes, hogy a többi LED kezelése könnyen elvégezhető az alábbi módon:

```
// Bekapcsolás:
AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 LED_ÉRTÉK);

// Kikapcsolás:
AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, LED_ÉRTÉK);
```

4.3. PWM

A PWM (impulzusszélesség-modulátor) nagyon hasznos periféria: ez teszi lehetővé, hogy kapcsoló üzemű tápegységet építsen a mikrovezérlő köré. A PWM működését nem tárgyalom, annak részletes leírása megtalálható egy régebbi munkámban (a Philips-szel [NXP-vel] ellentétben az ATMEL nem invertál).

A bekapcsolt állapot idejét „duty cycle”-nak nevezi az ATMEL, a két felfutó él közötti időt pedig periódusidőnek. A kimeneten megjelelő **átlag** feszültség értéke a duty cycle és a periódusidő arányától függ (lineárisan).

Ahhoz, hogy a már előzőleg kiszámított 33 kHz-es kapcsolási frekvenciát tartani tudjam, a periódusidőt a kvarcoszcillátor periódusidejének 512-szeresére kell állítani. Ehhez kb. 40 kHz-es kapcsolási frekvencia tartozik, ami több, mint a 33 kHz, tehát bőven megfelel a célnak. Ehhez a PWM vezérlőben a következő beállításokat kell eszközölni:

```
#ifndef __PWM_H__
#define __PWM_H__

#define DEFAULT_PWM_MODE 0 /* f_clock = MCK, left aligned,
                             period starts at high level,
                             CUPD modifies period */

#define PWM_PERIOD 512

inline void pwm_init(unsigned int period) {
    AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, 0, AT91C_PA11_PWM0);
    AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, 0, AT91C_PA12_PWM1);
    AT91F_PIO_CfgPeriph(AT91C_BASE_PIOA, 0, AT91C_PA13_PWM2);

    AT91F_PWMC_CfgPMC(); // Enable perif. clock.

    *AT91C_PWMC_MR = 0; // Prescaler turned off

    AT91F_PWMC_CfgChannel(AT91C_BASE_PWMC, 0, DEFAULT_PWM_MODE, period, 0);
    AT91F_PWMC_CfgChannel(AT91C_BASE_PWMC, 1, DEFAULT_PWM_MODE, period, 0);
    AT91F_PWMC_CfgChannel(AT91C_BASE_PWMC, 2, DEFAULT_PWM_MODE, period, 0);

    AT91F_PWMC_StartChannel(AT91C_BASE_PWMC, 0x7); // Enebles Channel 0, 1, 2
}

inline void pwm_set_value(int id, int duty_value) {
    AT91F_PWMC_UpdateChannel(AT91C_BASE_PWMC, id, duty_value);
}

#endif
```

A „pwm_init()” függvény először engedélyezi a perifériát (PWM vezérlőt), ellátja órajellel, majd kikapcsolja az előosztót, vagyis a kvarcoszcillátor jele közvetlenül a PWM vezérlőre jut. Ezt követően beállítja a periódusidőt és a „duty cycle”-t.

Ahhoz, hogy a PWM csatornákat használni lehessen, engedélyezni kell azok működését. Erre szolgál az utolsó sor.

A „pwm_set_value()” függvény a duty cycle értékét módosítja. Ez egy **inline** függvény: a fordító az inline függvény törzsét behelyettesíti (bemásolja) a hívó kód azon részébe, ahol az

`inline` függvényre hivatkozás történt. Vagyis az egész függvénytörzs alig néhány assembly utasítássá alakul. Aki nem hiszi, meggyőződhet erről a listing fájl tanulmányozásával (gyorsan hozzáteszem, szinte követhetetlen, hogy a `gcc` hogyan állítja össze a megfelelő címeket, adatokat. Tény, hogy jól csinálja, mert néhányszor átszámoltam, és kijönnek az értékek).

```
// main.c-ben: pwm_set_value(channel, power_supply[channel].ui / (PI_MUL));

inline void pwm_set_value(int id, int duty_value) {
    AT91F_PWMC_UpdateChannel(AT91C_BASE_PWMC, id, duty_value);

102960: e5912000    ldr r2, [r1]
102964: e0822fa2    add r2, r2, r2, lsr #31
102968: e1a020c2    mov r2, r2, asr #1

    {
        pPWM->PWMC_CH[channelId].PWMC_CUPDR = update;

10296c: e1a0328e    mov r3, lr, lsl #5
102970: e243390d    sub r3, r3, #212992 ; 0x34000
102974: e5832210    str r2, [r3, #528]

    }
```

Azt tényleg nem tanácsolom senkinek, hogy egy optimalizált kódot tanulmányozzon. Ha már assembly szinten kell hibamentesíteni, akkor a „-O0” (ejtsd: ó-null) `gcc`-kapcsoló használatát javaslom. Eredménye (ebben az *inline* használata hatástalan):

```
10510c: ebfffc5b    bl 104280 <pwm_set_value>
```

Látszik, hogy tényleges függvényhívás történik (**a visszatérés címe az LR regiszterbe kerül (ami gyorsabb, mint a verem)**). Ebből is látszik, hogy az ARM processzort nagyon jól tervezte meg az ARM cég.

4.4. ADC

Az egyik legérdekesebb periféria kétségkívül az analóg-digitális átalakító. Ez az a pont, ahol az analóg világunk analóg értékei számokká alakulnak. Kicsit misztikus dolog ez.

Szerencsére az ADC használata sem bonyolultabb, mint az eddig megismert eszközöké, ezt bizonyítja az „adc.h” fájl tartalma:

```
#ifndef __ADC_H__
#define __ADC_H__

inline void adc_init() {
    AT91F_PIO_CfgPullup(AT91C_BASE_PIOA, 1 << 17);
    AT91F_PIO_CfgPullup(AT91C_BASE_PIOA, 1 << 18);
    AT91F_PIO_CfgPullup(AT91C_BASE_PIOA, 1 << 19);

    AT91F_ADC_CfgPMC();

    AT91F_ADC_CfgTimings(AT91C_BASE_ADC, 20, 5, 200, 600);
    AT91F_ADC_EnableChannel(AT91C_BASE_ADC,
        (1 << 0) | (1 << 1) | (1 << 2) | (1 << 5) | (1 << 6) | (1 << 7));
}

inline void adc_start_conversion() {
    AT91F_ADC_StartConversion(AT91C_BASE_ADC);
}

inline unsigned int adc_get_value(int channel) {
    switch (channel) {
        case 0:
```

```
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 0)));  
        return AT91F_ADC_GetConvertedDataCH0(AT91C_BASE_ADC);  
    case 1:  
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 1)));  
        return AT91F_ADC_GetConvertedDataCH1(AT91C_BASE_ADC);  
    case 2:  
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 2)));  
        return AT91F_ADC_GetConvertedDataCH2(AT91C_BASE_ADC);  
    case 5:  
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 5)));  
        return AT91F_ADC_GetConvertedDataCH5(AT91C_BASE_ADC);  
    case 6:  
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 6)));  
        return AT91F_ADC_GetConvertedDataCH6(AT91C_BASE_ADC);  
    case 7:  
        while (!(AT91F_ADC_GetStatus(AT91C_BASE_ADC) & (1 << 7)));  
        return AT91F_ADC_GetConvertedDataCH7(AT91C_BASE_ADC);  
    default:  
        return 0;  
    }  
}  
  
#endif
```

Az ADC alaphelyzetbe állítására (konfigurálására) az „adc_init()” függvényt használom. Ez engedélyezi a perifériát, majd beállítja az időzítési paramétereket: 5 MHz-es órajelet, 20 µs-os felébredési időt (alvó (idle) állapotból), és 600 ns-os mintavételi-tartási időt. Minden paraméter katalógusadat: a pontos értékeket mikrovezérlő adatlapjának „36.7 ADC Characteristics” nevű fejezete tartalmazza.

Az ADC nem konvertál folyamatosan. Ha szükség van az analóg értékekre, akkor egy trigger eseményt kell szolgáltatni az ADC számára. Ez lehet automatikus (időzítő), vagy felhasználói kérés. A felhasználói kérést az „adc_start_conversion()” függvénnyel jelzem.

Amikor kíváncsi vagyok valamelyik ADC csatorna bemenetén mérhető jel konvertált értékére, akkor meghívom a „adc_get_value()” függvényt, paraméterként átadva a csatorna azonosítóját. A függvény megvárja, amíg az átalakítás teljesen lezajlik, csak azután tér vissza a konvertált értékkel.

4.5. IRQ – megszakításkezelés

A mikrovezérlőnek ez a legnehezebben használható, de egyben legrugalmasabb része. Tisztelettel adózom a mérnöknek, aki tervezte.

Az ARM magnak csak egy darab IRQ bemenete van (és egy FIQ bemenete, de azt most nem használom). Periféria viszont rengeteg került a mikrovezérlőben. Ezt az ellentmondást úgy oldotta meg az ATMEL mérnöke, hogy olyan megszakításvezérlőt tervezett, aminek 32 bemenete van, és két kimenete: az egyik jelzi az ARM magnak, hogy megszakítás érkezett, míg a másik azt a lineáris memóriacímet szolgáltatja, ahol a megszakítás kiszolgáló rutinja található. Ezt a címet persze nem „drótozta bele” a mérnök, hanem meghagyta annak lehetőségét, hogy az inicializáló függvény egyenként beállítsa a különböző hardver megszakításokhoz tartozó kiszolgáló rutinok címét.

Az ARM mag megszakításkiszolgáló rutinjának (1 db utasítás) már csak annyi dolga van, hogy ugorjon a megszakításkezelő által megadott címre. Gépi nyelven ez rövidebben hangzik:

```
ldr pc, [pc, #-0xF20]
```

Ez volt a „renitens” ugró utasítás a mintaprogram elején (a többi utasítás egyszerű „b”⁹ volt; mivel csak egy megszakításforrás van, használhatnák egyszerű „b irq”-t is, mert minden esetben ugyanazt a címet adja vissza a megszakításkezelő).

⁹ Feltétel nélküli ugrás, BRANCH

A fenti utasítás hatására az ARM mag a PC¹⁰-be tölti a kapott címet, ami a PC aktuális értékéhez képest relatívan a -F20_{HEX} címen állt elő (ott található a megszakításkezelő hardver egyik regisztere).

A megszakításkiszolgáló rutinnak kicsit többet kell tennie, mint egy normál függvénynek: jeleztem a gcc-nek, hogy mely függvények a megszakításkiszolgáló rutinok.

```
void fiq()    __attribute__((interrupt("FIQ")));
void irq()   __attribute__((interrupt("IRQ")));
```

Egy normál függvény részlete (eleje és vége, optimalizálatlan kód!):

```
1045b8:  e1a0c00d    mov ip, sp
// Használt regiszterek mentése
1045bc:  e92dd800    stmdb sp!, {fp, ip, lr, pc}
1045c0:  e24cb004    sub fp, ip, #4 ; 0x4
// 1 db lokális változó (int outputs) létrehozása a veremben
1045c4:  e24dd004    sub sp, sp, #4 ; 0x4
...
// Regiszterek tartalmának visszaállítása és
// visszatérés a függvényből (PC visszaállítása által)
104608:  e89da808    ldmia sp, {r3, fp, sp, pc}
```

És az „irq()” részlete:

```
104cdc:  e52dc004    str ip, [sp, #-4]!
104ce0:  e1a0c00d    mov ip, sp
// regiszterek mentése
104ce4:  e92dd81f    stmdb sp!, {r0, r1, r2, r3, r4, fp, ip, lr, pc}
104ce8:  e24cb004    sub fp, ip, #4 ; 0x4
// 2 db lokális változó (int channel, dummy) létrehozása a veremben
104cec:  e24dd008    sub sp, sp, #8 ; 0x8
...
105160:  e24bd020    sub sp, fp, #32 ; 0x20
// regiszterek tartalmának visszaállítása
105164:  e89d681f    ldmia sp, {r0, r1, r2, r3, r4, fp, sp, lr}
105168:  e8bd1000    ldmia sp!, {ip}
// visszatérés a megszakításból
10516c:  e25ef004    subs pc, lr, #4 ; 0x4
```

Hogyan kell felprogramozni a megszakításkezelő hardvert? Erre mutat példát a következő kódrész (*intr.h*):

```
#ifndef __INTR_H__
#define __INTR_H__

void intr_init() {
    AT91F_AIC_Open(AT91C_BASE_AIC, irq, fiq, irq, (void *) 0, 0);
}
```

10 Programszámláló, az aktuális utasítás címe

```
AT91F_PWMC_CfgPMC();
AT91F_PWMC_CfgChannel(AT91C_BASE_PWM, 3, 7, 150, 1);
AT91F_PWMC_InterruptEnable(AT91C_BASE_PWM, (1 << 3));

AT91F_AIC_ConfigureIt(AT91C_BASE_AIC, AT91C_ID_PWM,
    AT91C_AIC_PRIOR_LOWEST, AT91C_AIC_SRCTYPE_HIGH_LEVEL, irq);

AT91F_AIC_EnableIt(AT91C_BASE_AIC, AT91C_ID_PWM);

AT91F_PWMC_StartChannel(AT91C_BASE_PWM, (1 << 3));

// Enable interrupts in CPU
asm("mov r1, %0" :: "i" (SYSTEM_MODE));
asm("msr cpsr, r1");
}

#endif
```

Elsőként az alapértelmezett kiszolgáló függvények címét állítom be: ha másként nem rendelkeznek, akkor ezek a függvények hívódnak meg megszakításkérés esetén. Később felülbírálok az alapértelmezett értékeket, mert a 4. PWM csatorna (PWM#3) paramétereinek beállítása után – ez adja az időalapot az „I” szabályozásnak és az áramhatárolásnak – azt állítom be, hogy PWM megszakítás esetén az „irq()” függvény fusson le.

Ez még nem elegendő a megszakításrendszer megfelelő működéséhez, engedélyezni kell a PWM megszakítások fogadását a megszakításkezelőben, továbbá az IRQ fogadását az ARM magban. Megszakítás csak akkor keletkezik, ha a PWM csatorna működik (bekapcsolt állapotban van), ezt is engedélyezni kell.

Nem szabad arról sem megfeledkezni, hogy a megszakítást nyugtázni kell, mind a megszakításkezelő felé, mind a PWM vezérlő felé:

```
dummy = *AT91C_PWM_ISR;
AT91F_AIC_AcknowledgeIt(AT91C_BASE_AIC);
```

Az első sor fél napi munkám eredménye. Nem a begépelés folyamata tartott fél napig, hanem az, hogy rájöttem: ki KELL olvasni a megfelelő státuszregiszter értékét, hogy nyugtázzam a megszakítást a PWM vezérlőben is, különben az folyamatosan jelez a megszakításvezérlőnek.

Nagyon hasznos, ha a megszakításrutin elején bekapcsolok egy LED-et, a végén pedig kikapcsolom azt. Lévéen ez digitális jel, meg tudom mérni a megszakítás frekvenciáját, vagyis azt, hogy a függvény másodpercenként hányszor fut le, emellett a LED kapcsolójele PWM jel is, tehát időben integrálva (pontosabban szűrve) azt mutatja, hogy mennyi időt tölt a processzor a megszakításrutin végrehajtásával, és mennyit a főprogram futtatásával. Hasznos és fontos statisztikai jellemző ez, mert a túl hosszú megszakítási rutin lehetetlenné teszi a főprogram feladatának elvégzését (jelen esetben a soros port figyelését).

Szélsőséges esetben a megszakításkérések egymásra is futhatnak, vagyis a megszakítási rutin még nem fut le, pedig már a következő megszakítást kellene kiszolgálni...

```
AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << IRQ_LED);

...

AT91F_PIO_ClearOutput(AT91C_BASE_PIOA, 1 << IRQ_LED);
```

4.6. A főprogram

A „main()” függvényről nem írok részletesen. Ennek oka az, hogy a kezdeti értékek beállítása után kizárólag a TR-232 csomagok vételével és adásával foglalkozik. A csomagok forgalmazásának leírása a „Kommunikáció a PC és a mikrovezérlő között” című fejezetben található meg.

A kezdeti értékek beállítása, az inicializálás eképpen történik:

```
// r0 = End_of_RAM
asm("mov r0, %0" :: "i" (AT91C_ISRAM));
asm("add r0, r0, %0" :: "i" (AT91C_ISRAM_SIZE));

// IRQ_stack_pointer = End_of_RAM
asm("mov r1, %0" :: "i" (IRQ_MODE | IRQ_FLAG | FIQ_FLAG));
asm("msr cpsr, r1");
asm("mov sp, r0");

// FIQ_stack_pointer = End_of_RAM - IRQ_stack_size = r0 - IRQ_stack_size
asm("sub r0, r0, %0" :: "i" (IRQ_STACK_SIZE));
asm("mov r1, %0" :: "i" (FIQ_MODE | IRQ_FLAG | FIQ_FLAG));
asm("msr cpsr, r1");
asm("mov sp, r0");

// System_mode_stack_pointer = End_of_RAM - IRQ_stack_size - FIQ_stack_size =
// = FIQ_pointer - FIQ_stack_size
asm("sub r0, r0, %0" :: "i" (FIQ_STACK_SIZE));
asm("mov r1, %0" :: "i" (SYSTEM_MODE | IRQ_FLAG | FIQ_FLAG));
asm("msr cpsr, r1");
asm("mov sp, r0");
```

Az első, és legfontosabb a verem beállítása. Nem túlzás a többes szám használata, a főprogram és a megszakításkezelő rutinok külön-külön vermet használnak.

Önkényesen definiáltam: a verem visszafelé növekednek. Megtehetem ezt, mert az **ARM processzorban nincsen veremkezelő utasítás, így a verem úgy működik, ahogy akarom:**

```
stmdb sp!, {fp, ip, lr, pc}
```

Vagyis „store multiple data (into stack), decrement SP before”. Ez azt jelenti, hogy a verem visszafelé növekszik (a RAM végétől kezdve), és a regiszterek tartalmának mentése előtt gyorsan csökkenti SP értékét. Megvan ennek a párja is:

```
ldmia sp, {r3, fp, sp, pc}
```

Ennek jelentése: „load multiple data (from stack), increment after”, vagyis olvassa ki az adatokat a regiszterekbe, és utána növelje SP regiszter értékét.

Mondanom sem kell, létezik az STM és az LDM utasítás minden kombinációja az IA (increment after), IB (increment before), DA (decrement after), DB (decrement before) posztfixumokkal.

Visszatérve a verem beállításához: a verem „eleje” a RAM terület végén (AT91C_ISRAM_SIZE) található. Itt „kezdődik” az IRQ verem.

A RAM végének címéből levonva az IRQ vermének nagyságát (IRQ_STACK_SIZE = 1024) megkapom a FIQ verem címét.

Ebből levonva a FIQ verem nagyságát (FIQ_STACK_SIZE = IRQ_STACK_SIZE = 1024) eljutok a főprogram vermének címéhez.

Az előbb kiszámított címeket a főprogram beállítja úgy, hogy belép a megfelelő CPU üzemmódba, és menti az értéket az R13 (SP – Stack Pointer, veremmutató) regiszterbe. Legvégül visszalép *Supervisor* (felügyelő) üzemmódba, és abban futtatja a főprogramot.

4. A mikrovezérlő hardver eszközeinek használata

Minden programnak a verem beállításával kell kezdődnie. Enélkül nem hívhatók meg a perifériainicializáló és -kezelő függvények:

```
// WDT (Watchdog timer)11 tiltása
AT91F_WDTSetMode(AT91C_BASE_WDTC, AT91C_WDTC_WDDIS);

// Kvarcoszcillátor bekapcsolása
AT91F_CKGR_CfgMainOscillatorReg(AT91C_BASE_CKGR,
    OSC_DELAY << 8 | AT91C_CKGR_M0SCEN);

// Várok, amíg a kvarcoszcillátor stabilizálódik
while (!(AT91F_CKGR_GetMainClockFreqReg(AT91C_BASE_CKGR) & AT91C_CKGR_MAINRDY));

// Ha stabilizálódott, akkor ez legyen az órajelforrás
AT91F_PMC_CfgMCKReg(AT91C_BASE_PMC, AT91C_PMC_CSS_MAIN_CLK);

// Ki- és bemeneti portok kiválasztása, beállítása
ports_init();

// Egy kis visszajelzés: a portokat sikerült beállítani
AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << PWR1_LED);

// A már megismert DBGU beállító függvény
dbg_u_init();

// Újabb visszajelzés: ez is sikerült
AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << PWR2_LED);

// PWM csatornák beállítása, már erről is volt szó
pwm_init(PWM_PERIOD);

// Tápegység alapértelmezett adatainak beállítása
// Áramhatárolás legyen 1 amper
power_supply[0].current_limit = 1000.0 / ADC_MUL;
power_supply[1].current_limit = 1000.0 / ADC_MUL;
power_supply[2].current_limit = 1000.0 / ADC_MUL;

// Még nincs túlterhelés
power_supply[0].over = 0;
power_supply[1].over = 0;
power_supply[2].over = 0;

// Alapértelmezett kimenő feszültségek: 12V, 5V, 3,3V
power_supply[0].reference = 1000 * 12 / ADC_MUL;
power_supply[1].reference = 1000 * 5 / ADC_MUL;
power_supply[2].reference = 1000 * 3.3 / ADC_MUL;

// ADC és AIC beállítása
adc_init();
intr_init();

// Visszajelzés, nagyon hasznos
AT91F_PIO_SetOutput(AT91C_BASE_PIOA, 1 << PWR3_LED);

// Végül megnyugtatom a felhasználót: minden sikerült,
// a tápegység készenléti állapotban van
report_ready();
```

A főprogram hátralevő része a csomagok vételével és továbbküldésével foglalkozik. Erről a következő fejezetekben talál további információkat az Olvasó.

¹¹ Olyan számláló, ami túlszordulása esetén RESET-eli a mikrovezérlőt. Hogy ne csorduljon túl, megadott időközönként nullázni kell, vagyis jelezni kell, hogy még megfelelően fut a kód

4.7. Az „I” szabályozó algoritmus

A megszakításkezelő rutin egyik feladata a szabályozó algoritmus megvalósítása. Ennek megértéséhez ismerni kell a szabályozási körök tulajdonságait és kompenzálásuk módjait. Mindezek leírására nincs módom, a háttérinformációk elsajátítását az Olvasóra bízom. Ebben a témakörben rengeteg szakirodalom áll rendelkezésére.

Annyit azonban elmondhatok, hogy az integráló tag kimenő jele a bemenő jel idő szerint integrálja. Diszkrét időtartományban úgy mondhatnám, hogy a kimenő jelhez hozzá kell adni az aktuális bemenő jel k -szorosát („ k ” egy valós szám).

Ehhez meg kell állapítani a bemenő jel nagyságát:

```
adc_start_conversion();

power_supply[0].analog0 = adc_get_value(0);
power_supply[1].analog0 = adc_get_value(1);
power_supply[2].analog0 = adc_get_value(2);

power_supply[0].analog1 = adc_get_value(5);
power_supply[1].analog1 = adc_get_value(6);
power_supply[2].analog1 = adc_get_value(7);
```

Ezt követi az áramhatárolás-ellenőrzés és az „I” szabályozás megvalósítása (csatornánként):

```
for (channel = 0; channel < 3; channel++) {
// Túláramvédelem: ha (analog1-analog0) / 1 Ohm > áramhatár
    if (abs(power_supply[channel].analog0 - power_supply[channel].analog1)
        > power_supply[channel].current_limit) {
// Növelek egy túláram-számlálót
        power_supply[channel].over++;
// Ha még mindig túláram van, és már régóta az van (250 ms)...
        if (power_supply[channel].over == MAX_OL_TIME) {
// ... akkor kikapcsolom a csatornát (visszaveszem a PWM értékét 0-ra)
            power_supply[channel].reference = 0;
// és kikapcsolom a csatorna LED-jét
            switch (channel) {
                case 0:
                    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,
                                            (1 << PWR1_LED));
                    break;
                case 1:
                    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,
                                            (1 << PWR2_LED));
                    break;
                case 2:
                    AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,
                                            (1 << PWR3_LED));
                    break;
            }
        }
// De ha elmúlt a túláram, akkor minden rendben, törlöm a számlálót
    } else power_supply[channel].over = 0;

    /* I-alg. */

// Maga az I algoritmus: az aktuális végrehajtójelhez hozzáadom a hibajel negyedét
    power_supply[channel].ui += (power_supply[channel].reference
        - power_supply[channel].analog1) >> 2);
}
```

```
// Határolni kell a végrehajtó jel nagyságát
    if (power_supply[channel].ui < 50)
        power_supply[channel].ui = 50;

    if (power_supply[channel].ui > 1023)
        power_supply[channel].ui = 1023;

// Végül beállítom az aktuális végrehajtó jelet
    pwm_set_value(channel, power_supply[channel].ui / (PI_MUL));
}
```

A kódból már csak a megszakítások nyugtázása van hátra.

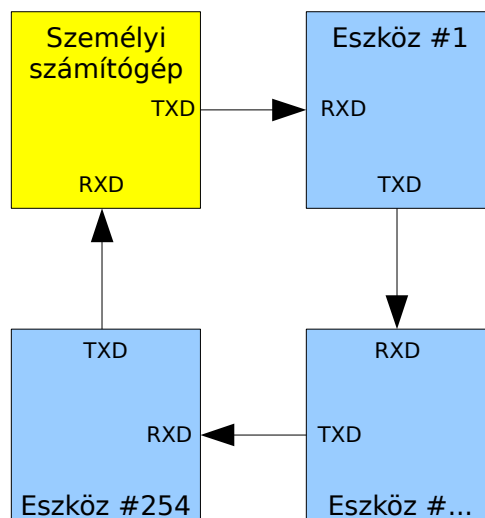
5. Kommunikáció a PC és a mikrovezérlő között

Ez az egyik legfontosabb része a dolgozatnak, mivel azt határozom meg, hogy miképpen fog egymással szót érteni a számítógép, a tápegység és a többi műszer.

A kommunikáció úgy valósul meg, hogy az eszközöket gyűrűbe kötöm, vagyis a számítógép üzenetei eszköztől eszközre vándorolva végül visszajutnak a számítógépre, remélhetőleg módosítva, ami azt jelenti, hogy valamelyik eszköz a magáénak érezte csomagot, és reagált arra.

De hogyan valósul meg az eszközök címezése? – kérdezheti valaki. A válasz egyszerű. A csomag tartalmazza az eszköz azonosítóját, ezt az azonosítót minden eszköz csökkenti, és amelyik eszköznél nullára csökken a célazonosító, az az eszköz került megcímezésre. A megcímezett eszköz végrehajtja a parancsot, és a válaszát (mérési eredmény vagy nyugta) elhelyezheti az üzenet paraméter mezéjében.

Ahhoz pedig, hogy az előbbi topológia megépíthető legyen, ily módon alakítom ki a hálózatot:



Nem új ez a topológia, a token-ring hálózat éppen így nézett ki az előző évszázadban. Ezért – kellő önbizalommal – ezt a topológiát TR-232-nek, vagyis RS-232-ből kialakított token-ring hálózatnak nevezem el.

Szoftver oldalról megközelítve a dolgot: adatbájtok küldése már nem okozhat problémát (DBGU), de valahogyan az adatbájtokból csomagokat kellene formálni. A csomag formátuma pedig a következő legyen:

```
struct Packet {
    unsigned char dst;
    unsigned char command;
    unsigned char param;
    unsigned char chsum;
};
```

Az első bájtnak a *céleszköz azonosítója*. A számozás 1-től kezdődik, és egyesével növekszik a PC-től távolodva a körüljárási iránnyal megegyező irányban.

A második bájt a *parancs*, de ennek alsó két bitje még paraméter, mert az analóg értékek 10 bitesek. A parancs alsó két bitje megfelelően látszott a 10 bites paraméterek felső két bitjének tárolására.

A „param” mező a *paraméter* (és a válasz) alsó 8 bitjét tartalmazza, a „chsum” mező pedig a *nullázó bájt*: a 4 bájt összege 0 kell, hogy legyen.

Látható, hogy egy kicsit más a Linux és a mikrovezérlő adatküldő és -fogadó függvénye, ezért ezeket újra meg kell vizsgálni:

// A Linux küldő függvénye:

```
int send_packet(FILE *tty, struct Packet *packet) {
    calc_checksum(packet);
    fwrite(packet, sizeof(struct Packet), 1, tty);
    fflush(tty);

#ifdef _DEBUG
    printf("\n\n--- Sent packet ---\n");
    printf("Destination:\t%02X\n", packet->dst);
    printf("Command:\t%02X\n", packet->command);
    printf("Parameter:\t%02X\n", packet->param);
    printf("Checksum:\t%02X\n", packet->chsum);
    fflush(stdout);
#endif

    return 0;
}
```

A függvény paraméterként a már megismert „setup_usart0()” függvény visszatérési értékét (ami nem más, mint a soros portot reprezentáló struktúra pointere) és a csomag címét kapja. Első lépésként kiszámítja a csomag ellenőrző összegét, ennek helyessége abszolút fontosságú, hibás csomagot nem fogad el a hálózat egyetlen eleme sem.

Ha ez megtörtént, akkor elküldi a teljes, 4 bájtos csomagot a már megismert módon.

Végül kiírja a nyomkövetést segítő információkat, ha a „_DEBUG” makró definiálva van (alapértelmezés szerint nincsen).

// A Linux vevő függvénye:

```
int receive_packet(FILE *tty, struct Packet *packet) {
    int c;
    unsigned char *packet_ptr = (unsigned char *) packet;
    unsigned char sum;

    fread(packet, sizeof(struct Packet), 1, tty);

#ifdef _DEBUG
    printf("\n\n--- Received packet ---\n");
    printf("Destination:\t%02X\n", packet->dst);
    printf("Command:\t%02X\n", packet->command);
    printf("Answer:\t\t%02X\n", packet->param);
    printf("Checksum:\t%02X\n", packet->chsum);
    fflush(stdout);
#endif

    for (c = 0, sum = 0; c < sizeof(struct Packet); c++)
        sum += packet_ptr[c];

    if (sum == 0) {
#ifdef _DEBUG
        printf("\n\nChecksum OK.");
#endif
    } else {
        printf("\n\nChecksum error!");
    }
}
```



```

        return -1;
    }
    return 0;
}

```

A vételi függvény egy lépésben beolvassa a csomagot, és hibát jelez, ha a bájtok összege nem nulla. Erről figyelmeztető üzenetet is küld a felhasználónak. Paraméterei pontosan ugyanazok, mint a küldő függvényé.

A mikrovezérlő kódja kissé eltér:

```

inline void dbgu_send_packet(struct Packet *packet) {
    int c;
    unsigned char *packet_ptr = (unsigned char *) packet;

    // Ellenőrző összeg kiszámítása
    calc_checksum(packet);

    // 4 db adatbájt küldése
    for (c = 0; c < sizeof(struct Packet); c++)
        dbgu_send_char(packet_ptr[c]);
}

```

A vevő függvény úgy tesz, mintha 3 db különálló tápegység lenne, vagyis nem 1-gyel, hanem 3-mal csökkenti a célazonosító értékét:

```

inline void dbgu_rcv_packet(struct Packet *packet) {
    int c;
    unsigned char *packet_ptr = (unsigned char *) packet;

    // Nem tér vissza, amíg nem jön olyan üzenet, ami ennek az eszköznek szól
    while (1) {

        // 4 bájt vétele
        for (c = 0; c < sizeof(struct Packet); c++)
            packet_ptr[c] = dbgu_rcv_char();

        // Ha az azonosító 0, 1 vagy 2, akkor ennek szól az üzenet -> visszatér a főprogramhoz
        switch (packet->dst) {
            case 0:
            case 1:
            case 2:
                return;

        // De ha nem ennek szól, akkor csökkenti a célazonosítót, és továbbküldi a csomagot
        default:
            packet->dst -= 3;
            dbgu_send_packet(packet);
            break;
        }
    }
}

```

A mikrovezérlő főprogramjának nincs más dolga, mint folyamatosan meghívni a csomagvevő függvényt, és ha az visszaadja a vezérlést, akkor a mikrovezérlőnek szóló csomag érkezett. A csomag feldolgozása és válasz küldése szintén a főprogram feladata.

A kommunikáció során szükséges, hogy minden résztvevő ugyanazokat a parancsokat használja. Erről a „commands.h” gondoskodik, melynek tartalma a következő:

```
#ifndef __COMMANDS_H__
#define __COMMANDS_H__

// A parancs bájt alsó két bitje még adat (paraméter)!
#define COMM_SHIFT 2

// Parancsok
#define COMM_HELLO 0
#define COMM_GET_TYPE 1
#define COMM_SET_OUTPUT_VALUE 2
#define COMM_GET_OUTPUT_VALUE 3
#define COMM_SET_CURRENT_LIMIT 4
#define COMM_GET_CURRENT_LIMIT 5
#define COMM_GET_ANALOG0 6
#define COMM_GET_ANALOG1 7

// A tápegységek ezt válaszolják, a többi eszköz mást fog majd válaszolni
#define TYPE_POWER_SUPPLY 1

#endif
```

Ez pedig a főprogram vonatkozó része:

```
while (1) {
    dbgu_rcv_packet(&packet);

    switch (packet.command >> COMM_SHIFT) {
// Lánc felderítése, eszközök megszámlálása céljából
        case COMM_HELLO:
            packet.param = 0x5a;
            dbgu_send_packet(&packet);
            break;

// Mit tud az eszköz?
        case COMM_GET_TYPE:
            packet.param = TYPE_POWER_SUPPLY;
            dbgu_send_packet(&packet);
            break;

// Kimenet referenciaértékének beállítása
        case COMM_SET_OUTPUT_VALUE:
            value = ((packet.command & ((1 << COMM_SHIFT) - 1)) << 8)
                + packet.param;

            packet.param = 0x5a;
            dbgu_send_packet(&packet);

            power_supply[packet.dst].reference =
                (value * 100) / ADC_MUL;

// Ha túláram volt, akkor visszakapcsolja a csatorna LED-et
        switch (packet.dst) {
            case 0:
                AT91F_PIO_SetOutput(AT91C_BASE_PIOA,
                    (1 << PWR1_LED));
                break;
            case 1:
                AT91F_PIO_SetOutput(AT91C_BASE_PIOA,
                    (1 << PWR2_LED));
                break;
            case 2:
                AT91F_PIO_SetOutput(AT91C_BASE_PIOA,
                    (1 << PWR3_LED));
                break;
        }
    }
}
```

```

        break;

// Kimenet referenciaértékének lekérdezése
case COMM_GET_OUTPUT_VALUE:
    value = (power_supply[packet.dst].reference * ADC_MUL) / 100;
    packet.command |= (value >> 8) & ((1 << COMM_SHIFT) - 1);
    packet.param = value & 0xff;
    dbgu_send_packet(&packet);
    break;

// Áramhatárolás beállítása
case COMM_SET_CURRENT_LIMIT:
    value = ((packet.command & ((1 << COMM_SHIFT) - 1)) << 8)
        + packet.param;

    packet.param = 0x5a;
    dbgu_send_packet(&packet);
    power_supply[packet.dst].current_limit = value / ADC_MUL;
    break;

// Áramhatárolás lekérdezése
case COMM_GET_CURRENT_LIMIT:
    value = power_supply[packet.dst].current_limit * ADC_MUL;
    packet.command |= (value >> 8) & ((1 << COMM_SHIFT) - 1);
    packet.param = value & 0xff;
    dbgu_send_packet(&packet);
    break;

// Analóg kimenetek lekérdezése
case COMM_GET_ANALOG0:
    value = power_supply[packet.dst].analog0 * ADC_MUL / 100;
    packet.command |= (value >> 8) & ((1 << COMM_SHIFT) - 1);
    packet.param = value & 0xff;
    dbgu_send_packet(&packet);
    break;

// Ez egyben a tápegység kimeneti feszültsége is
case COMM_GET_ANALOG1:
    value = power_supply[packet.dst].analog1 * ADC_MUL / 100;
    packet.command |= (value >> 8) & ((1 << COMM_SHIFT) - 1);
    packet.param = value & 0xff;
    dbgu_send_packet(&packet);
    break;

    }
}

```

A Linux-os alkalmazások várják a felhasználó parancsait, és közvetítik azokat a tápegység mikrovezérlője (vagy általánosabban: a hálózat elemei) számára.

Álljon itt a konzol főprogramjának egy részlete. Minden parancsot nem részletezek, arra majd a „Felhasználói interfészek” című részben kerül sor.

```

// Ugyancsak végtelen ciklus, ebből volt már egynehány...
do {
    printf("\n");

// Parancs beolvasása, Linuxos dolog, de jó: éljen soká a readline és a history
    line = readline("Command: ");

// Ha CTRL-D-t nyom a felhasználó, akkor vége a programnak
    if (line == NULL)
        break;

// Üres sorral nem foglalkozunk
    if (strlen(line) == 0)
        continue;

// A begépelte parancsot mentjük el a historyba, így nem kell újra begépelni

```

```
        add_history(line);

        printf("\n");

// Kiszámítjuk a parancsot a begépelte sorból
        sscanf(line, "%s", cmd);

// Ha azt mondta a felhasználó, hogy szkenneljük végig az eszközöket, akkor azt kell tenni
        if (!strcmp(cmd, "scan"))
            scan(tty);

// De ha be kell állítani egy eszköz referenciaértékét, akkor ...
        if (!strcmp(cmd, "psr")) {

// ... meg kell vizsgálni, hogy melyik számú eszközt kell megszólítani,
// és mire kell módosítani az értéket
            sscanf(line, "%s %d %lf", cmd, &id, &param);

// Beállítom a paraméter értékét
            param *= 10;

// Összeállítom a csomagot
            packet.dst = (char) id;
            packet.command = (COMM_SET_OUTPUT_VALUE << COMM_SHIFT) | (((int)
param >> 8) & ((1 << COMM_SHIFT) - 1));
            packet.param = (int) param & 0xff;

// Elküldöm a csomagot, ami kiszámítja az ellenőrző összeget is
            send_packet(tty, &packet);

// És reménytel teli szívvel várom a válaszcomagot
            receive_packet(tty, &packet);
        }

// ... ÉS A TÖBBI FELHASZNÁLÓI UTASÍTÁS FELDOLGOZÁSA ...

// Amíg a felhasználó hasznosnak találja erőfeszítéseimet
    } while (strcmp(cmd, "quit") & strcmp(cmd, "exit"));
```

Eddig tartott a mikrovezérlő és a számítógép szoftvereinek ismertetése. A hátralévő részben már nem kell a kedves Olvasónak nehezen érthető C – vagy ami még rosszabb: assembly – nyelvű kódrészleteken törnie a fejét, innentől kezdve válik igazán izgalmassá a dolog, mert életre kel a hardver, és ténylegesen azt csinálja, amit parancsként kap.

6. Felhasználói interfészek

Most, hogy már teljesen feltártam a rendszer működését, itt az ideje, hogy a felhasználó által használt segédprogramok használatáról is ejtsek néhány szót. Ebben a fejezetben összefoglalom az előbb kissé elnagyolt felhasználói interfész funkcióit és a műszer mikrovezérlőjének adható parancsokat.

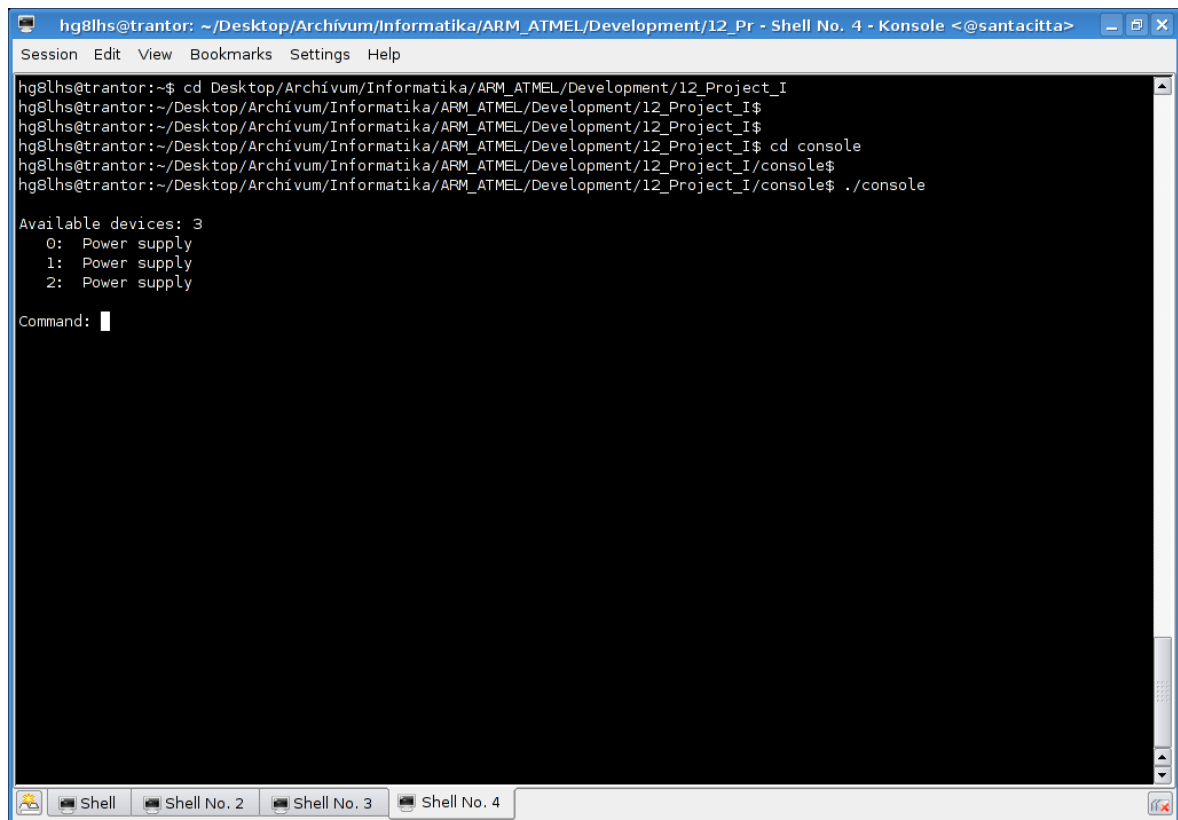
Azért, hogy mindenki megtalálja az egyéniségéhez legjobban illeszkedő felhasználói interfészt, készítettem egy karakteres felületű programot és egy grafikus alkalmazást.

A karakteres program használható szkriptek készítésére, bonyolultabb mérések (például akkumulátortöltő: ahol a töltőáramot kell beállítani és mérni az eltelt időt) elvégzésére és több eszköz integrált kezelésére.

A grafikus alkalmazás azok számára lehet hasznos, akik nem vágnak arra, hogy teljes „uralmat szerezzenek” a hálózat felett, viszont szeretnének minden paramétert egy ablakban látni.

6.1. Karakteres

A „console” alkönyvtár tartalmazza a program forrásprogramját, és fordítás után itt áll elő a futtatható program is. Elindítása után hasonló kép fogadja a felhasználót:



```
hg8lhs@trantor:~$ cd Desktop/Archivum/Informatika/ARM_ATMEL/Development/12_Project_I
hg8lhs@trantor:~/Desktop/Archivum/Informatika/ARM_ATMEL/Development/12_Project_I$
hg8lhs@trantor:~/Desktop/Archivum/Informatika/ARM_ATMEL/Development/12_Project_I$ cd console
hg8lhs@trantor:~/Desktop/Archivum/Informatika/ARM_ATMEL/Development/12_Project_I/console$
hg8lhs@trantor:~/Desktop/Archivum/Informatika/ARM_ATMEL/Development/12_Project_I/console$ ./console

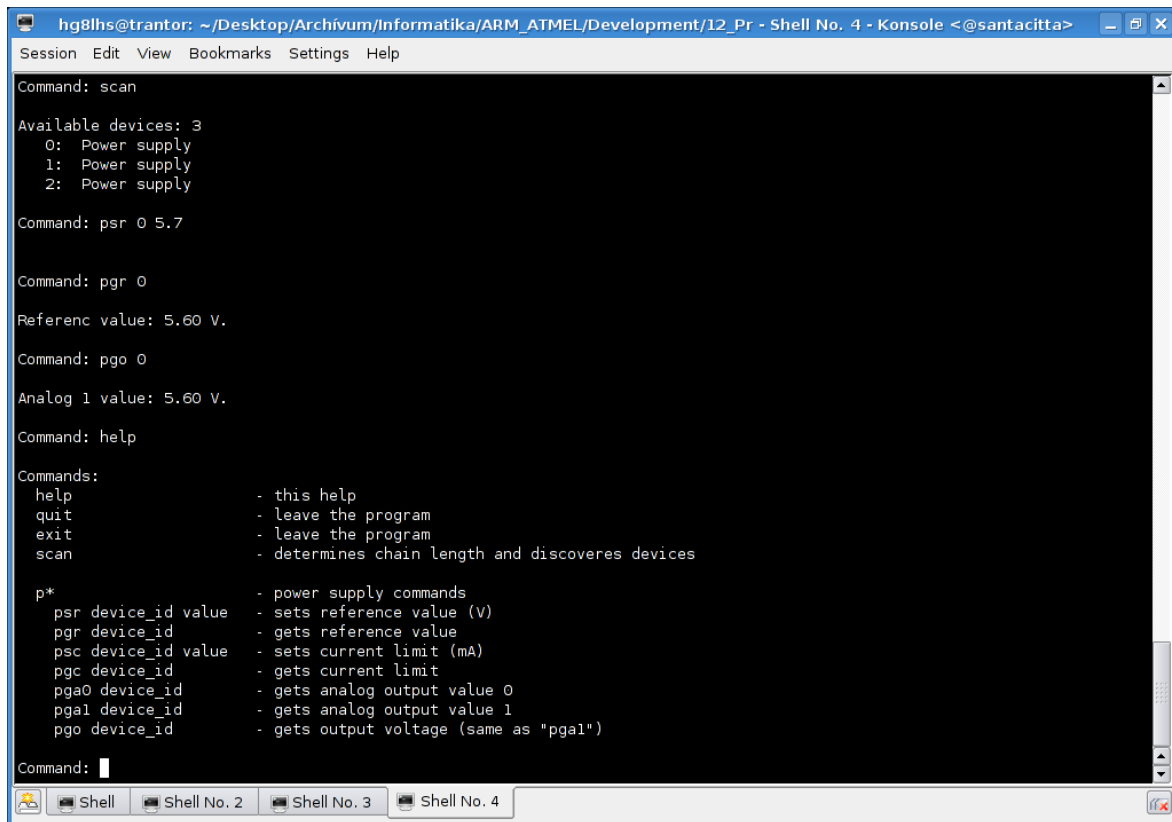
Available devices: 3
0: Power supply
1: Power supply
2: Power supply

Command: 
```

A felhasználó a prompthoz gépelheti be a parancsait. A beolvasáshoz „readline”-t használok, „history”-t engedélyezve, vagyis használható minden szerkesztő billentyűkombináció, amit Linux alatt megszoktam: visszafelé gomb (még ki nem adott parancs szerkesztése), felfelé gomb (előző parancsok megtekintése), CTRL-r (keresés a begépelte parancsok között), stb.

6. Felhasználói interfészek

A használható parancsok listája folyamatosan bővül, jelenleg a következők használhatók (a „help” listázza valamennyit):



```
hg8lhs@trantor: ~/Desktop/Archívum/Informatika/ARM_ATMEL/Development/12_Pr - Shell No. 4 - Konsole <@santacitta>
Session Edit View Bookmarks Settings Help

Command: scan

Available devices: 3
0: Power supply
1: Power supply
2: Power supply

Command: psr 0 5.7

Command: pgr 0

Referenc value: 5.60 V.

Command: pgo 0

Analog 1 value: 5.60 V.

Command: help

Commands:
help          - this help
quit          - leave the program
exit          - leave the program
scan          - determines chain length and discovers devices

p*
psr device_id value - sets reference value (V)
pgr device_id       - gets reference value
psc device_id value - sets current limit (mA)
pgc device_id       - gets current limit
pga0 device_id      - gets analog output value 0
pgal device_id      - gets analog output value 1
pgo device_id       - gets output voltage (same as "pgal")

Command: 
```

Parancsok

help	segítség kérése
quit	program elhagyása, kilépés
exit	program elhagyása, kilépés
scan	hálózat felderítése, ezt automatikusan elvégzi a program induláskor
p*	a tápegységeknek kiadható parancsok
psr eszköz érték	referencia értékének beállítása (Volt-ban)
pgr eszköz	referencia értékének lekérdezése
psc eszköz érték	áramhatár beállítása (mA-ben)
pgc eszköz	áramhatár lekérdezése
pga0 eszköz	belső analóg 0 érték lekérdezése
pgal eszköz	külső analóg 1 érték lekérdezése
pgo eszköz	kimeneti feszültség lekérdezése (ugyanaz, mint „pgal”)

Néhány példa:

```
Command: scan

Available devices: 3
0: Power supply
1: Power supply
2: Power supply

Command: psr 0 5.7
```

```

Command: pgr 0
Reference value: 5.60 V.
Command: pgo 0
Analog 1 value: 5.60 V.
Command: psc 1 400

Command: pgc
Current limit: 393 mA.

Command: help
Commands:
  help          - this help
  quit          - leave the program
  exit          - leave the program
  scan          - determines chain length and discovers devices

  p*
    psr device_id value - sets reference value (V)
    pgr device_id       - gets reference value
    psc device_id value - sets current limit (mA)
    pgc device_id       - gets current limit
    pga0 device_id      - gets analog output value 0
    pga1 device_id      - gets analog output value 1
    pgo device_id       - gets output voltage (same as "pga1")

Command:

```

6.2. Grafikus

A program egyszerű, jól áttekinthető módon mutatja a tápegység állapotát. Jól látszik ez a képernyőképen is:

Reference value (V)	Stored reference	Current limit (mA)	Stored current limit	Analog 0 (V)	Output voltage
3.5	3.40 V	500	482 mA	3.60 V	3.30 V
5.0	4.90 V	150	142 mA	5.00 V	4.90 V
9.0	8.90 V	100	89 mA	8.90 V	8.90 V

Update Exit

Az állapot lekérdezése mellett bizonyos paraméterek be is állíthatók. Ahhoz, hogy az új értékek kifejtsék hatásukat a tápegységben, a felhasználónak meg kell nyomnia az „Update” gombot. Ez egyúttal az állapotinformációkat is frissíti.

Az „Exit” gomb kilépésre szolgál.

A grafikus felhasználói interfész lefordításához és használatához *GTK+* függvénykönyvtár szükséges.

Összefoglalás, végkövetkeztetés

Elérkeztünk a dolgozat utolsó fejezetéhez. Visszatekintve megállapíthatjuk, hogy igazam volt, mikor azt írtam a bevezetőben, hogy az ARM mikrovezérlők kiválóan alkalmasak laboratóriumi műszerek építésére. Ahhoz, hogy megírjuk programunkat, nem kell feltétlenül „bitszinten” ismerni a mikrovezérlőt, de szükséges bizonyos mértékű háttérismeret megléte.

A processzor megismerése külön tudomány, erre sajnos nem fordíthattunk elég időt, de az Olvasó beláthatja, mégis van értelme némi energiát szánni az ARM mag tanulmányozására. Már csak azért is, mert néhány típus MPU-val, memóriavédelmi egységgel felszerelve kerül forgalomba, vagyis operációs rendszer írható rájuk. Az ARM9-es mag pedig futtatja a Linuxot (de azt kifejezetten ARM9-re kell fordítani, például az általunk előkészített arm-elf-gcc-vel).

Kezdeti lépésnek tekinthető az áramkör „megálmodása”, az ötlet megfogalmazása, a specifikációkészítés és a paraméterek kiszámítása. Az áramkör elvi rajzának és nyomtatott áramköri lapjának elkészítése már tervezői – kicsit nagyképűen azt is mondhatnám, hogy mérnöki – feladat. Az alkatrészek beültetése az áramkörgyártás legkevésbé kreatív, mégis izgalmas terület. Kinek nem ver hevesebben a szíve, amikor kezébe veszi a 64 lábú tokot, vagy a tranzisztort, ami alig nagyobb, mint egy mákszem?

Nem kerülhetjük meg a szoftverrel kapcsolatos teendőket sem: fordítót kellett fordítani, feltöltőprogramot kellett letölteni. És amikor mindez sikerült, következtek az ismeretlen felderítésének bátortalan lépései. Jelen dolgozatban siker koronázta erőfeszítésünket, és semmi sem indokolja, hogy ennek másként kellene lennie a jövőben.

A perifériák programozása már közelebb áll a digitális technikát kevésbé ismerőkhöz is, hiszen egy PWM vagy egy soros port semmiképpen sem tekinthető „ördögtől való” dolognak.

Ahogy a kisgyerekek az építőkockákból várat építenek, úgy építettük fel mi is berendezésünk szoftverét az előre gondosan elkészített függvényekből.

A felhasználó felületek kialakításakor a felhasználók igényeihez kellett igazodni.

De bizonyára az Olvasó is kialakította saját véleményét, talán még javaslatai, továbbfejlesztéssel kapcsolatos észrevételei is vannak a témával kapcsolatban. Ha ezeket meg kívánja osztani velem, írjon bátran a

Fuszenecker Róbert <hg8lhs@gmail.com>

e-mail címre.

Nem tagadom, örömmel veszek minden kezdeményezést vagy segítséget, ami arra irányul, hogy akár ebből a tápegységből, akár más eszközökből egyre több lássa meg a napvilágot.

Felhasznált irodalom

- ARM7TDMI Technical Reference Manual (Rev 3), **ARM Limited**, 2001.
- AT91 ARM Thumb-based Microcontrollers, **ATMEL Corp.**, 2006. november 22.
- **Fuszenecker Róbert**: Mérő- és vezérlőberendezés megvalósítása ARM alapú mikrovezérlővel és Linux-szal, 2007

Felhasznált szoftverek

- Ubuntu Linux 6.06 (Dapper Drake) Linux kernel 2.6.15-27-k7
- OpenOffice 2.2.1
- VIM – VI IMproved version 6.4.6
- The GIMP 2.2.11
- aspell (International Ispell Version 3.1.20 (but really Aspell 0.60.4))
- binutils 2.17 (using BFD version 2.17)
- gcc 4.0.3
- openocd – Open On-Chip Debugger (2007-05-30 17:45 CEST)
- Eagle 4.16r2 for Linux, Light Edition

