Mikrokrmilnik ARM

Matjaž Vidmar, S53MV

1. Razvoj mikrokrmilnikov

Ceneni računalniki so postali dostopni širnim množicam z izumom mikroprocesorjev v sedemdesetih letih prejšnjega stoletja. Prvi 4-bitni mikroprocesorji so omogočili preproste računske strojčke, kalkulatorje, ki so znali bliskovito seštevati, odštevati, množiti in deliti. 4-bitni mikroprocesorji so večinoma vsebovali tovarniško vpisan program v bralni pomnilnik (ROM ali Read-Only Memory), torej uporabnik ni mogel strojčka programirati drugače.

Samo par let za 4-bitnimi mlinčki so se pojavili tudi prvi 8-bitni mikroprocesorji. Najbolj znana sta bila Intel I8080A in Motorola MC6800. 8-bitni mikroprocesorji so omogočili obdelavo besedil in elektronsko pošto, kot jo poznamo danes. Hkrati z 8-bitnimi mikroprocesorji so se pojavili tudi trajni pomnilniki za enkraten vpis (PROM ali one-time Programmable Read-Only Memory) oziroma takšni, ki se jih je dalo večkrat brisati in ponovno vpisati (EPROM ali Eraseable PROM), v katere je uporabnik lahko vpisal svoj program.

Razvoj je nato šel v dve smeri. Na eni strani poenostaviti in poceniti 8-bitne mikroprocesorje tako, da vse potrebne naloge: procesor, pomnilnike ROM in RAM (pisalno/bralni pomnilnik) ter vhodno/izhodne enote združimo na enem samem čipu. Takšni napravi pravimo mikrokrmilnik, najbolj znane so bile družine proizvajalca Intel I8048 in I8051. Na drugi strani izdelati še zmogljivejše mikroprocesorje: če je bil 16-bitni Intel I8086 le razširjena inačica 8-bitnega I8080, predstavlja 16-bitni Motorola MC68000 prelomnico, ki je napovedala izredno zmogljive domače računalnike in skorajšnje izumrtje "velikih" računalnikov (mainframe).

V osemdesetih letih prejšnjega stoletja smo mikroprocesorje in mikrokrmilnike programirali skoraj izključno v strojnem jeziku. Prevajalnik zbirnik (assembler) nam je pomagal le v toliko, da je samodejno izračunaval naslove pogojnih skokov na labele. Najbolj razširjen 8-bitni mikroprocesor vseh časov, Zilog Z80CPU, sicer zelo izpopolnjena inačica Intel I8080A, smo vsi poznali do podrobnosti kljub temu, da je jedro Zilog Z80CPU pravo sračje gnezdo. Vse ukaze njegovega strojnega jezika smo znali na pamet, saj drugačne izbire ni bilo!

V devetdesetih letih prejšnjega stoletja je tehnologija polprevodnikov toliko napredovala, da so postali dostopni trajni pomnilniki EEPROM

(Electrically Eraseable PROM) oziroma FLASH na istem čipu mikrokrmilnika. Takšni pomnilniki so omogočili električno brisanje programa in ponoven vpis s preprostimi programski orodji v čip mikrokrmilnika v cenenem plastičnem ohišju. Za razliko od predhodnikov PROM in EPROM, ki so zahtevali komplicirane in drage programatorje ter omogočali samo enkraten vpis (PROM) oziroma zamudno brisanje vsebine čipa v dragem keramičnem ohišju z okencem iz kremenovega stekla (EPROM) z UV svetlobo pred ponovnim programiranjem.

Veliki proizvajalci mikroprocesorjev so pri 8-bitnih mikrokrmilnikih zaspali in tu sta jih prehiteli dve majhni podjetji. Podjetje Atmel je začelo z dopolnjenimi inačicami I8051 in iz njih razvilo svoje 8-bitno jedro AVR. Podjetje Microchip je razvilo silno preprosto 8-bitno jedro PIC. 8-bitni mikrokrmilniki AVR in PIC so še danes zelo priljubljeni, saj so silno preprosti za uporabo. Vsa potrebna strojna in programska orodja za programiranje so preprosta za učenje in cenena oziroma zastonj.

S povečevanjem zmogljivosti pomnilnikov se je uveljavilo programiranje v višjih jezikih tudi za 8-bitne mlinčke, saj hitrost izvajanja programa v mikrokrmilniku pogosto ni omejujoč dejavnik. Napisati nekaj zahtevnejšega v strojnem jeziku za špartansko enostavno jedro PIC oziroma nepregledno sračje gnezdo jedra AVR sploh ni preprosto.

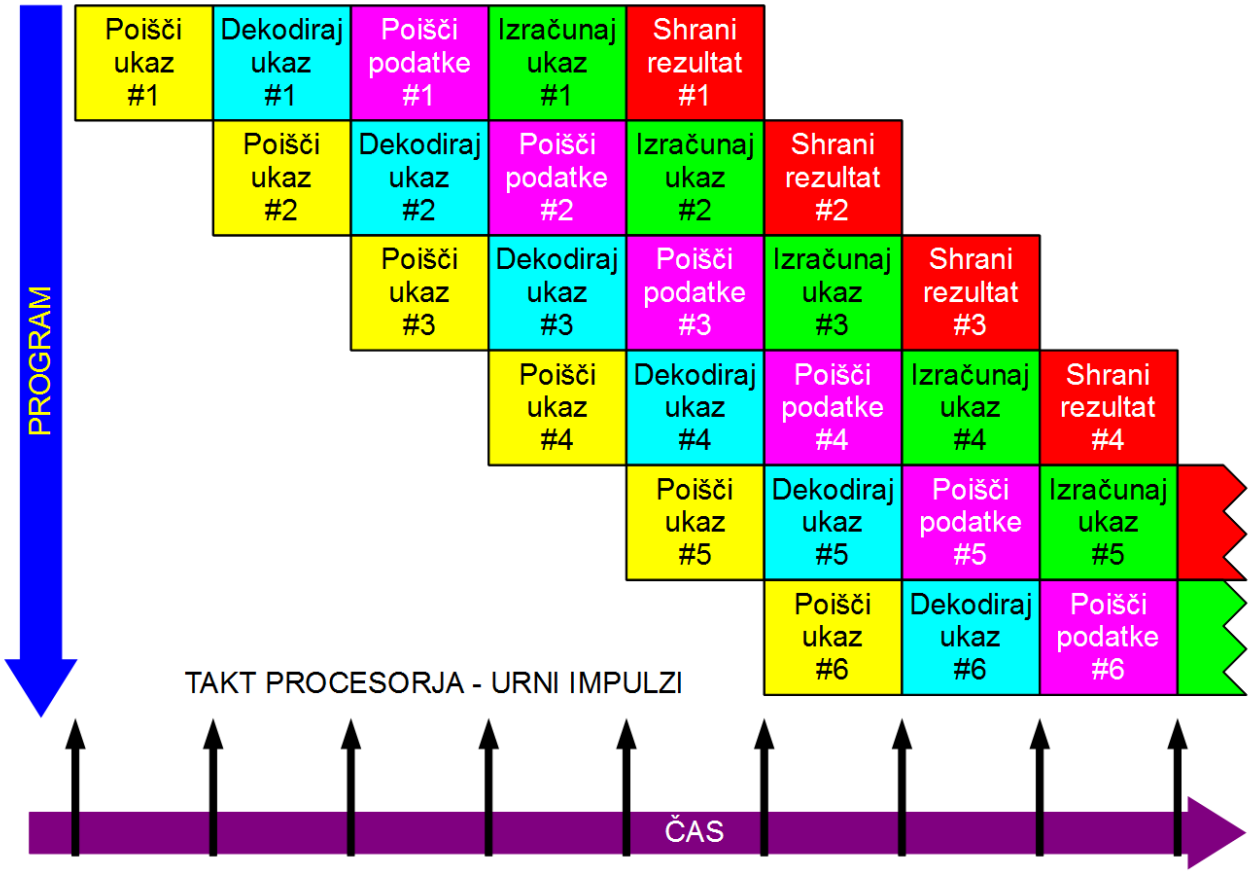
Svoj lonček so pristavili tudi pisci programskih orodij. Zbirnik predstavlja preslikavo 1:1, torej je iz prevedenega programa na sodišču nemogoče dokazati, v katerem zbirniku je bil napisan. Prevajalnik za višji jezik predstavlja preslikavo 1:mnogo, torej se iz prevedenega programa natančno vidi, kateri prevajalnik ga je proizvedel. Vsak heker zato objavi okrnjeno inačico svojega prevajalnika, plačljivo neokrnjeno inačico pa razširi na spletu v upanju, da se nekega dne pojavi v izdelku plačilno sposobnega kupca (avtomobilska industrija in podobno). V upanju na bajeslovni zaslužek nas zato vsak heker posiljuje s svojim prevajalnikom.

V osemdesetih letih prejšnjega stoletja sta vodilna proizvajalca mikroprocesorjev Intel in Motorola razširila svoje mikroprocesorje na 32-bitno vodilo z izdelki I80386 in MC68020. Razširila sta tudi nabor ukazov, na primer MC68020 pozna dvojne oklepaje na ravni strojnega jezika, torej zna izračunati f(g(x)) v enem samem ukazu. Izvajanje takšnega ukaza postane zamudno, napravo imenujemo tudi CISC (Complex Instruction Set Computer).

Manjši neodvisni proizvajalci so se 32-bitnih mikroprocesorjev lotili drugače. Če je cilj najvišja možna hitrost delovanja, je smiselno nabor ukazov poenostaviti do tem mere, da se izvedejo v enem samem ciklu ure procesorja. Mikroprocesor s takšnim naborom ukazov imenujemo tudi RISC

(Reduced Instruction Set Computer). Najbolj znana predstavnika 32-bitnih mikroprocesorjev RISC sta jedri ARM in MIPS.

Prav noben mikroprocesor, niti CISC niti RISC, v resnici ne more izvesti enega ukaza v enem samem ciklu ure. Vsi mikroprocesorji zato uporabljajo tehniko cevovoda (pipeline). Medtem ko se prvi ukaz dekodira, mikroprocesor že išče naslednji ukaz. Medtem ko prvi ukaz išče podatke, se drugi ukaz dekodira in mikroprocesor že išče tretji ukaz. Medtem ko prvi ukaz računa, drugi išče podatke, tretji se dekodira in mikroprocesor že išče četrti ukaz. Ko prvi ukaz shranjuje rezultat, drugi ukaz računa, tretji išče podatke, četrti se dekodira in mikroprocesor že išče peti ukaz, kot je prikazano na spodnji sliki:



Cevovod (pipeline) izvajanja ukazov

Cevovod lahko vsebuje različno število stopenj obdelave ukaza. Zagon cevovoda sicer zahteva ustrezno število urnih impulzov. V ustaljenem stanju vsak dodatni ukaz zahteva samo še en dodatni cikel ure mikroprocesorja, torej se ukazi vsaj navidez izvajajo v enem samem taktu jedra RISC, če seveda ni treba čakati na rezultat računanja prejšnjega ukaza (Microcomputer without Interlocked Pipeline Stages ali MIPS).

Če želimo prenašati po naftovodu bencin namesto dizelskega goriva, moramo najprej izprazniti celoten cevovod vseh ostankov prejšnjega goriva.

Povsem isti učinek ima skok v računalniškem programu. Celotne vsebina cevovoda izvajanja ukazov je takrat izgubljena. Cevovod moramo na novo napolniti z drugačnimi ukazi.

Izguba vsebine cevovoda pri programskih skokih, pogojnih skokih, razcepih programa, zankah ponavljanja, klicanju podprogramov in podobno zato izredno upočasnjujejo delovanje vseh sodobnih računalnikov. Strukturirano napisan računalniški program je sicer lepo pregleden za človeškega programerja, ampak silno neučinkovito izrablja razpoložljivo strojno opremo. Povrhu sodobni računalniki vsebujejo celo več kot en cevovod: cevovodu samega procesorja so zaporedno vezani še cevovodi predpomnilnikov in je izguba ob programskem skoku še toliko večja.

Programer v višjem jeziku na cevovode nima skoraj nobenega vpliva. Pri programiranju v zbirniku lahko v celoti izkoristimo možnosti reševanja vsebine cevovoda, ki nam jih ponujajo zmogljiva jedra ARM ali MIPS. Na primer, jedro ARM omogoča reševanje vsebine cevovoda tako, da je večina ukazov pogojno izvedljivih.

Poglejmo si preprost primer, napisan na tri različne načine v zbirniku ARM. Register R0 vsebuje število med 0 in 15 in to bi radi pretvorili v en šestnajstiški znak za izpis na zaslon. Šestnajstiške znake zapišemo s številkami 0-9 in črkami A-F. Številke 0-9 zapišemo z ASCII znaki 48-57, velike črke A-F pa z ASCII znaki 65-70. Programer, ki pozna možnosti jedra ARM, bo nalogo rešil brez skokov (brez izgube cevovoda) s samo tremi ukazi, kjer sta obe prištevanji pogojno izvedljivi:

```
        CMP     R0,#10          ;primerjaj R0<>10?
        ADDCC   R0,R0,#48       ;manjše prištej R0+48=>R0
        ADDCS   R0,R0,#55       ;enako ali večje prištej R0+55=>R0
```

Programer, ki zmogljivih ukazov jedra ARM ne pozna, bo nalogo rešil z enim pogojnim in enim brezpogojnim skokom, kot je to običajno pri 8-bitnem mlinčku. Skupaj torej 5 ukazov in v vsakem primeru izgubljen cevovod:

```
        CMP     R0,#10          ;primerjaj R0<>10?
        BCS     L1              ;pogojni skok enako ali večje na labelo L1
        ADD     R0,R0,#48       ;prištej R0+48=>R0
        B       L2              ;brezpogojni skok na labelo L2
L1      ADD     R0,R0,#55       ;prištej R0+55=>R0
L2
```

Končno, računalniški piflar bo vse skupaj napisal v višjem jeziku s strukturiranim stavkom oblike IF(pogoj)THEN(izračun1)ELSE(izračun2). Prevajalniki za višje jezike so običajno pisani splošno, torej se ne prilagajajo

ukazom niti naborom registrov posameznega mikroprocesorja. Prevajalniki za višje jezike zato skušajo rešiti nalogo s čim manjšim številom registrov na račun uporabe pomnilnika RAM. Prevod iz višjega jezika bi se v zbirniku ARM najverjetneje glasil takole:

```
    LDR     R0,[R3,#odmik1]     ;naloži podatek
    LDR     R1,[PC,#odmik2]     ;naloži vrednost 10
    CMP     R0,R1               ;primerjaj podatek<>10?
    BCS     L1                  ;pogojni skok na labelo L1
    LDR     R0,[R3,#odmik1]     ;naloži podatek
    LDR     R1,[PC,#odmik3]     ;naloži vrednost 48
    ADD     R2,R0,R1            ;prištej podatek+48=>podatek
    STR     R2,[R3,#odmik4]     ;shrani podatek
    B       L2                  ;brezpogojni skok na labelo L2
L1  LDR     R0,[R3,#odmik1]     ;naloži podatek
    LDR     R1,[PC,#odmik5]     ;naloži vrednost 55
    ADD     R2,R0,R1            ;prištej podatek+55=>podatek
    STR     R2,[R3,#odmik4]     ;shrani podatek
L2
```

Prevajalnik bo za isto nalogo porabil kar štiri registre, celih 13 ukazov in dodatno še 8 dostopov do pomnilnika za premikanje podatkov. Povsem jasno prevajalnik izgublja vsebino cevovoda procesorja, zelo verjetno tudi vsebino predpomnilnikov. Sem povedal vse?

Na prelomu tisočletja računska zmogljivost 8-bitnih jeder mikrokrmilnikov PIC in AVR za večino nalog še ni vprašljiva, pač pa postane ozko grlo zelo omejen naslovni prostor 8-bitnikov. Mikrokrmilniki s 16-bitnim jedrom se niso uveljavili. Vsi proizvajalci mikrokrmilnikov se na prelomu tisočletja preusmerijo na 32-bitna jedra ARM oziroma MIPS.

Žal so na prelomu tisočletja programska orodja mikrokrmilnikov zašla na stranpot. Programska orodja so postala strahotno komplicirana in nerodna za uporabo. Najgrozljivejši primer je sračje gnezdo "Eclipse", ki sploh ne dopušča celovitega dostopa do jedra mikrokrmilnika. Po nepotrebnem komplicirana in nerodna programska orodja so zagotovo kriva za delni neuspeh 32-bitnih mikrokrmilnikov, ki marsikje ne uspejo izriniti svojih 8-bitnih predhodnikov.

Marsikateri uporabnik mikrokrmilnikov zato še danes prisega na 8-bitne mlinčke PIC in AVR in njihova preprosta razvojna orodja. Seveda se 32-bitnih mikrokrmilnikov in jedra ARM lahko lotimo tudi drugače: na preprost, razumljiv in učinkovit način. Prav temu je namenjen ta sestavek!

2. Arhitektura ARM

Majhno angleško podjetje Acorn Computers je v začetku osemdesetih let prejšnjega stoletja uvidelo, da njihov domači računalnik "BBC Micro" z 8-bitnim mikroprocesorjem 6502 potrebuje nadgradnjo. Podjetje je zbralo pogum za načrtovanje lastnega RISC mikroprocesorja. Sophie Wilson si je zamislila nabor ukazov za povsem nov, silno preprost ampak zmogljiv 32-bitni mikroprocesor, ki so ga poimenovali Acorn RISC Machine ali na kratko ARM.

Acornov domači računalnik Archimedes je sicer v devetdesetih letih izgubil tekmo s PC računalniki. Procesorsko jedro ARM se je medtem razvijalo naprej. Od preloma tisočletja vse do danes so postala jedra družine ARM najbolj razširjeni mikroprocesorji in mikrokrmilniki na svetu!

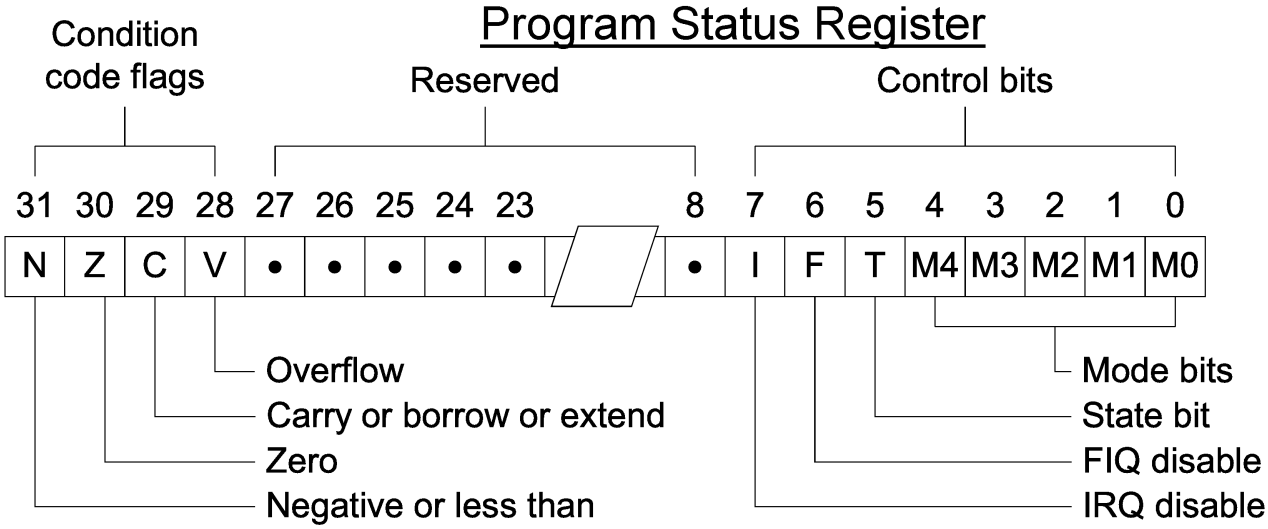
Modes						
	Privileged modes					
		Exception modes				
M=10000	M=11111	M=10011	M=10111	M=11011	M=10010	M=10001
User	**System**	**Supervisor**	**Abort**	**Undefined**	**Interrupt**	**Fast interrupt**
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

Registri jedra ARM

Od vseh najrazličnejših jeder ARM je danes najbolj razširjeno jedro ARM7TDMI-S, ki izvršuje nabor ukazov ARMv4T. Arhitektura jedra ARM, kot si jo je zamislila Sophie Wilson, vsebuje 16 skoraj enakovrednih 32-bitnih registrov poimenovanih R0 do R15. Med vsemi registri so možne vse računske operacije. R15 dodatno opravlja nalogo programskega števca ali PC (Program Counter). Dogovor velja, da register R14 običajno uporabljamo kot Link Register (LR), register R13 pa kot kazalec na sklad oziroma Stack Pointer (SP).

Jedro ARM pozna en sam, skupni 32-bitni naslovni prostor za program, podatke in vhodno/izhodne enote. Naslove šteje po bajtih, naslovni prostor torej obsega 4Gbyte. Preprosti mikroprocesorji ARM imajo eno samo vodilo. Skupni naslovni prostor še ne pomeni, da zmogljivejši procesor ARM ne more imeti ločenih predpomnilnikov in ločenih vodil za program in za podatke.



Poleg 16 računskih registrov vsebuje jedro ARM še Program Status Register (PSR). Najpogosteje uporabljamo gornje štiri bite N (Negative ali minus), Z (Zero ali nič), C (Carry ali prenos) in V (oVerflow) kot zastavice računskih operacij. Z biti I in F izključimo oziroma vključimo prekinitve IRQ in FIQ. Bit T označuje izvrševanje okrnjenega nabora 16-bitnih ukazov Thumb.

Spodnjih pet bitov M določa stanje izjem (exceptions). Najuporabnejši izjemi sta prekinitev IRQ in hitra prekinitev FIQ. Jedro ARM se po resetu zbudi v načinu Supervisor. Mali mikrokrmilnik bomo sicer večino časa poganjali v načinu Supervisor. Ostali načini User, System, Abort in Undefined so uporabni edino v kompliciranih računalnikih in pripadajočih operacijskih sistemih. V malem mikrokrmilniku lahko pomenijo tako hudo programsko napako, da je edini smiselni izhod reset.

Skok v izjemo mora vedno omogočati povratek nazaj v staro stanje.

Trenutno stanje procesorja oziroma Current PSR (CPSR) se shrani v dodatni register Saved PSR (SPSR), izjema pa dobi nov CPSR. Poleg shranjevanja PSR dobi vsaka izjema vsaj dva nova (banked) registra R13 (SP) in R14 (LR). Hitra prekinitev FIQ dobi kar sedem novih (banked) registrov R8-R14, da ne izgubljamo dragocenega časa z reševanjem vsebine računskih registrov v sklad.

3. Ukazi ARM

Večina mikroprocesorjev uporablja pogojne skoke za razcep izvajanja programa. 32-bitni ukazi ARM uvajajo pomembno novost: skoraj vsi ukazi so pogojno izvedljivi! Dober programer pogojno izvedljive ukaze uporabi tako, da prepreči izgubo vsebine cevovoda in pohitri izvajanje programa. Od 32 bitov kode ukaza so najvišji štirje biti 28, 29, 30 in 31 namenjeni pogoju:

Ukazi ARM

	31 30 29 28	27 26 25	24 23 22 21	20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Data processing immediate shift	cond [1]	0 0 0	opcode	S	Rn	Rd	shift amount	shift 0	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x			0	x x x x
Data processing register shift [2]	cond [1]	0 0 0	opcode	S	Rn	Rd	Rs	0 shift 1	Rm
Miscellaneous instructions: See Figure A3-4	cond [1]	0 0 0	1 0 x x	0	x x x x x x x x x x x x			0 x x 1	x x x x
Multiplies: See Figure A3-3 Extra load/stores: See Figure A3-5	cond [1]	0 0 0	x x x x	x	x x x x x x x x x x x x			1 x x 1	x x x x
Data processing immediate [2]	cond [1]	0 0 1	opcode	S	Rn	Rd	rotate	immediate	
Undefined instruction	cond [1]	0 0 1	1 0	x	0 0	x x x x x x x x x x x x			x x x x
Move immediate to status register	cond [1]	0 0 1	1 0	R	1 0	Mask	SBO	rotate	immediate
Load/store immediate offset	cond [1]	0 1 0	P U	B	W L	Rn	Rd	immediate	
Load/store register offset	cond [1]	0 1 1	P U	B	W L	Rn	Rd	shift amount	shift 0 Rm
Media instructions [4]: See Figure A3-2	cond [1]	0 1 1	x x x x	x	x x x x x x x x x x x x				1 x x x x
Architecturally undefined	cond [1]	0 1 1	1 1 1 1	1	x x x x x x x x x x x x			1 1 1 1	x x x x
Load/store multiple	cond [1]	1 0 0	P U	S	W L	Rn	register list		
Branch and branch with link	cond [1]	1 0 1	L	24-bit offset					
Coprocessor load/store and double register transfers	cond [3]	1 1 0	P U	N	W L	Rn	CRd	cp_num	8-bit offset
Coprocessor data processing	cond [3]	1 1 1 0	opcode1			CRn	CRd	cp_num	opcode2 0 CRm
Coprocessor register transfers	cond [3]	1 1 1 0	opcode1	L		CRn	Rd	cp_num	opcode2 1 CRm
Software interrupt	cond [1]	1 1 1 1	swi number						
Unconditional instructions: See Figure A3-6	1 1 1 1	x x x x	x x x x	x	x x x x x x x x x x x x				x x x x

Pogoj se nanaša na zastavice N (Negative), Z (Zero), C (Carry) in V (oVerflow) iz trenutnega registra stanja CPSR. Zastavice postavljajo računske operacije. Za razliko od ostalih mikroprocesorjev ARM tudi tu uvaja novost: pri računskih operacijah lahko z bitom S (bit 20 v kodi ukaza) izbiramo, ali določena računska operacija sploh postavlja zastavice N, Z, C in O ali jih pusti nespremenjene. Strojno kodo procesorja ARM hitro prepoznamo po temu, da je večji del ukazov zapisan v šestnajstiški obliki kot 0xE(nekaj), ker pogoj 1110 pomeni brezpogojno izvajanje (AL ali ALways).

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
1110	AL	Always (unconditional)	-
1111	-	See *Condition code 0b1111*	-

Pogoji ARM

Arhitektura RISC zahteva, da računski ukazi delujejo samo z registri. Procesor RISC ima zato ločene ukaze za prenos podatkov med registri in pomnilnikom. Ukaze jedra ARM zato v grobem delimo na računske ukaze in na ukaze za premikanje podatkov med registri procesorja in zunanjim pomnilnikom. Čeprav lahko programske skoke izvedemo s spreminjanem vsebine programskega števca R15 (PC), so programskim skokom namenjeni še dodatni ukazi ARM, ki so znatno udobnejši za uporabo.

32-bitna koda ukaza ARM omogoča, da v računskem ukazu navedemo ločeno prvi podatek, drugi podatek in rezultat računske operacije. V zbirniku ARM računski ukaz zapišemo v obliki:

(ukaz) (rezultat),(podatek1),(podatek2)

Pri tem sta rezultat in prvi podatek obvezno registra ARM. Drugi podatek je v obliki "ARM shifter operand". To je lahko preprosto register ARM,

lahko je 8-bitna konstanta, lahko je 8-bitna konstanta ali vsebina registra, zamaknjena za določeno število bitov, ki jih določimo s še eno konstanto oziroma z vsebino še enega (četrtega) registra. Arhitektura ARM torej ne pozna posebnega ukaza za zamik (shift) oziroma zasuk (rotate) vsebine registra, pač pa so zamiki in zasuki kar del drugih računskih izrazov. V praksi to pomeni, da en sam ARM ukaz nadomešča najmanj dva ukaza običajnih mikroprocesorjev!

Preproste aritmetične operacije so vedno 32-bitne in dopuščajo, da sta podatka in rezultat v istem registru. Nekaj zgledov:

```
ADD     R1,R2,R3        ;ADD pomeni R2+R3=>R1
SUB     R1,R2,#10       ;SUBtract pomeni R2-10=>R1
RSB     R1,R2,#33       ;Reverse SuBtract pomeni 33-R2=>R1
ADC     R0,R0,R1        ;ADd with Carry pomeni R0+R1+C=>R0
SBC     R0,R1,R2        ;SuBtract with Carry pomeni R1-R2-C=>R0
RSC     R0,R1,R2        ;Reverse Subtract w Carry R2-R1-C=>R0

ADD     R0,R0,R1,LSL#8      ;pomeni R0+(R1*2^8)=>R0
ADD     R3,R3,R6,ASR#7      ;pomeni R3+(R6*2^-7)=>R3
ADD     R0,R0,R1,LSL R2     ;pomeni R0+(R1*2^R2)=>R0
```

POZOR! Večina mikroprocesorjev uporablja zastavico Carry oziroma prenos naprej pri seštevanju tudi kot zastavico Borrow oziroma izposodi pri odštevanju. V arhitekturi ARM je pomen izposojanja v bitu C za prenos obrnjen: C=1 ni izposoje, C=0 imamo izposojo!

Arhitektura ARM pozna naslednje zamike oziroma zasuke:

```
LSL  #n    pomeni Logical Shift Left za n mest, z desne vstavlja ničle
LSR  #n    pomeni Logical Shift Right za n mest, z leve vstavlja ničle
ASR  #n    pomeni Arithmetic Shift Right za n, z leve ponavlja bit predznaka
ROR  #n    pomeni ROtate Right za n mest, zasuk v desno
RRX        pomeni Rotate Right with eXtend, zasuk v desno skozi C, vedno
```

samo za eno mesto, zato se število zasukov ne navaja!

Aritmetičnim operacijam so zelo podobne logične operacije:

```
AND     R1,R2,R3        ;AND pomeni R2(AND)R3=>R1
BIC     R1,R2,R3        ;BIt Clear pomeni R2(AND)R3=>R1
ORR     R0,R0,#0x0F     ;OR pomeni R0(OR)0x0F=>R0
EOR     R3,R4,R5        ;ExOR pomeni R4(EXOR)R5=>R3
```

Vsi računski izrazi so pogojno izvedljivi. Primeri:

```
ANDEQ   R1,R2,R3        ;R2(AND)R3=>R1 samo v primeru Z=1
```

```
SUBCS   R1,R2,#10       ;R2-10=>R1 samo v primeru C=1
RSBPL   R1,R2,#33       ;33-R2=>R1 samo v primeru M=0
```

Vsi računski izrazi lahko postavljajo zastavice NZCV s pripono S:

```
ADDS    R3,R3,R3        ;ADD Set pomeni R3+R3=>R3 in NZCV
SUBS    R7,R7,#1        ;SUBtract Set pomeni R7-1=>R7 in NZCV
EORS    R7,R6,R5        ;ExOR Set ali R6(EXOR)R5=>R7 in NZCV
```

V nekaterih primerih nas sam rezultat računanja ne zanima, pač pa nas zanimajo samo zastavice NZCV. Taki ukazi potrebujejo samo dva podatka in ne potrebujejo pripone S za postavljanje zastavic, saj je to samoumevno:

```
CMP     R1,R2       ;CoMPare zavrže rezultat R1-R2 a postavi NZCV
CMN     R1,R2       ;CoMpare Negated zavrže R1+R2 postavi NZCV
TST     R0,R3       ;TeST zavrže R0(AND)R3 a postavi NZCV
TEQ     R0,R6       ;Test EQuivalence NZCV in zavrže R0(EXOR)R6
```

Končno, preprosti premiki podatkov med registri sodijo v družino računskih operacij in imajo samo en podatek in en rezultat. Primeri:

```
MOVNE   R4,R0       ;MOVe pomeni R0=>R4 samo v primeru Z=0
MOV     R3,#123     ;MOVe pomeni 123=>R3
MOVS    R2,R7       ;MOVe Set pomeni R7=>R2 in postavi NZCV
MVN     R0,R0       ;MoVe Not pomeni R0=>R0
MOV     R5,R5,LSL#5 ;pomeni zamik v levo R5*2^5=>R5
```

Od zahtevnejših računskih operacij pozna osnovna arhitektura ARMv4T samo celoštevilsko množenje. Deljenja niti zahtevnejšega računanja ARMv4T ne pozna. Za zahtevnejše operacije moram pripadajoče podprograme napisati sami oziroma uporabiti zmogljivejše jedro ARM, ki lahko vsebuje tudi matematični koprocesor s plavajočo vejico.

Osnovna arhitektura ARM pozna tri različne ukaze za 32-bitno množenje: MUL, UMULL in SMULL. MUL zmnoži dve 32-bitni števili in proizvede samo spodnjih 32 bitov rezultata. Spodnji 32 biti so popolnoma enaki pri nepredznačenem kot pri predznačenem računanju. Celoten rezultat množenja 32 bitov z 32 biti proizvede 64-biten rezultat in tu sta potrebna dva različna ukaza za nepredznačeno in za predznačeno množenje:

```
MUL     R1,R2,R3        ;MULtiply R2*R3=>R1
UMULL   R7,R8,R9,R10    ;Unsigned MULtiply Long R9*R10=>R8,R7
SMULL   R4,R3,R2,R1     ;Signed MULtiply Long R2*R1=>R3,R4
```

Ukazi za množenje delujejo izključno z registri. Shifter operand tu ni

dovoljen. Za razliko od preprostih računskih operacij morajo biti vsi navedeni registri v ukazu množenja različni med sabo! 64-bitni rezultat se vpiše v dva registra. Spodnjih 32 bitov zmnožka gre v prvi navedeni register, gornjih 32 bitov zmnožka pa v drugi register v seznamu.

Ukazi za množenje so eni redkih v arhitekturi ARM, ki za svoje izvajanje porabijo več taktov ure procesorja. Jedro ARM7TDMI-S vsebuje samo strojni množilnik 8 bitov X 32 bitov. Množenje 32 bitov X 32 bitov torej zahteva štiri takte ure za štiri množenja 8X32 in še en takt ure za seštevek končnega rezultata. Skupaj torej pet taktov za ukaze MUL, UMULL oziroma SMULL.

Pri številski obdelavi signalov (DSP) pogosto potrebujemo množenje in prištevanje k skupnemu rezultatu. V ta namen pozna ARM ukaze vrste MuLtiply and Accumulate (MLA). Tudi tu zadošča en sam ukaz za spodnjih 32 bitov rezultata in dva različna ukaza za celoten nepredznačen oziroma predznačen 64-bitni rezultat:

```
MLA     R1,R2,R3       ;MuLtiply Accumulate (R2*R3)+R1=>R1
UMLAL   R7,R8,R9,R10   ;nepredznačeno (R9*R10)+R8,R7=>R8,R7
SMLAL   R4,R3,R2,R1    ;predznačeno (R2*R1)+R3,R4=>R3,R4
```

Arhitektura RISC procesorja zahteva ločene ukaze za premikanje podatkov med registri procesorja in pomnilnikom. Arhitektura ARM uporablja v ta namen dva zelo zmogljiva ukaza LDR (LoaD Register) in STR (STore Register), ki omogočata številne različne načine naslavljanja pomnilnika. Oba ukaza LDR in STR sta lahko pogojno izvedljiva. Premiki podatkov med registri ARM in pomnilnikom podatkov ne preračunavajo, zato v nobenem primeru ne postavljajo zastavic NZCV.

Prvotni ARM je omogočal samo prenos 8-bitnih podatkov B (Byte) in 32-bitnih podatkov (brez pripone). Pozneje so dodali še 16-bitni podatek H (Half-word) in možnost nalaganja predznačene vsebine SB (Signed Byte) in SH (Signed Half-word). Nalaganje iz pomnilnika vedno vpliva na vseh 32 bitov ciljnega registra, ne glede na širino podatkov. V preostali del registra (gornji biti) se naložijo ničle (B ali H) ali pa predznak (SB ali SH), to je najvišji bit 8-bitnega ali 16-bitnega podatka:

```
LDR     R0,[R1]    ;naloži 32 bitov (R1)=>R0
LDRNE   R0,[R1]    ;naloži 32 bitov (R1)=>R0 samo v primeru Z=0
LDRB    R2,[R3]    ;naloži 8 bitov (R3)=>R2, ničle v bite 8-31
LDRH    R4,[R5]    ;naloži 16 bitov (R5)=>R4, ničle v bite 16-31
LDRSB   R6,[R7]    ;naloži 8 bitov (R7)=>R6, predznak v bite 8-31
LDRSH   R8,[R9]    ;naloži 16 bitov (R9)=>R8, predznak v bite 16-31
```

Shranjevanje podatkov iz registrov v pomnilnik je preprostejše. V

pomnilnik se zapiše točno toliko bitov, kolikor je širina prenosa:

```
STR     R0,[R1]    ;shrani vseh 32 bitov R0=>(R1)
STRB    R2,[R3]    ;shrani spodnjih 8 bitov R2=>(R3)
STRBCC  R2,[R3]    ;shrani spodnjih 8 bitov R2=>(R3) samo pri C=0
STRH    R4,[R5]    ;shrani spodnjih 16 bitov R4=>(R5)
```

Oba ukaza LDR in STR omogočata tudi odmik od naslovnega registra. Odmik ima lahko oba predznaka, je lahko preprosta konstanta ali pa kompliciran shifter operand. Predznačeni odmik lahko ne spreminja naslovnega registra, lahko se naslovnemu registru prišteje pred prenosom podatkov (pre-indexed) in lahko se naslovnemu registru prišteje po prenosu podatkov (post-indexed). Vseh možnosti je res veliko, naj tu omenim le nekaj preprostih zgledov za 32-bitni prenos, saj delujejo B, SB, H in SH podobno:

```
LDR     R0,[R1,#16]     ;pomeni (R1+16)=>R0
STR     R2,[R3,R4]      ;pomeni R2=>(R3+R4)
LDR     R1,[SP],#4      ;najprej (R13)=>R1, potem R13+4=>R13
STR     R2,[SP,#-4]!    ;najprej R13-4=>R13, potem R2=>(R13)
```

Tu je slovnica zbirnika ARM čudna. Popravljanje naslova pred prenosom (pre-indexed) je označeno s klicajem za oglatim oklepajem, ki vsebuje naslov in odmik. Popravljanje naslova po izvršenem prenosu (post-indexed) je označeno z odmikom za oglatim oklepajem, ki vsebuje samo naslov.

Še večje razlike so v strojni kodi ARM. Prvotna 32-bitni prenos in 8-bitni prenos B dopuščata konstanto odmika vse do +/-4095. Prenosi vrste SB, H in SH so bili dodani v nabor veljavnih ukazov kasneje, imajo povsem drugačno strojno kodo in dopuščajo konstanto odmika samo +/-255. Prenosi glede na programski števec PC (R15) imajo še drugačen strojni zapis in večinoma omogočajo takojšnje odmike v območju +/-4095. Prav te zadnje uporablja zbirnik ARM pri prevajanju naših psevdo-ukazov in zato ni vseeno, kam postavimo LTORG.

Nalaganje več registrov iz pomnilnika opravi ukaz LDM (LoaD Multiple register), shranjevanje več registrov v pomnilnik pa ukaz STM (STore Multiple register). Ker oba ukaza LDM in STM uporabljamo zelo poredko, ju tukaj ne bom opisoval v podrobnosti.

Zelo pomemben ukaz prenosa med registri in pomnilnikom je SWP (32-bitni SWaP) oziroma SWPB (8-bitni SWaP). SWP najprej prečita vsebino pomnilnika, jo vpiše v prvi register in takoj nato vpiše vsebino drugega registra na isto mesto v pomnilniku. Ukaz je namenoma nedeljiv (atomarni), torej ga uporabljamo kot semafor v večopravilnem sistemu. 32-bitna primera:

```
        SWP     R1,R2,[R5]      ;najprej (R5)=>R1, potem R2=>(R5)
        SWP     R0,R0,[R7]      ;zamenjaj vsebini R0 in (R7) med sabo
```

V večopravilni sistem hitro zaidemo tudi na malem mikrokrmilniku, kjer se istočasno izvajajo nek glavni program, prekinitve IRQ in prekinitve FIQ. Ko je štafetna palica (blok pomnilnika) prosta, jo hranimo v pomnilniku. Nedeljiv, atomarni ukaz SWP pri tem zagotavlja, da bo štafetno palico (blok pomnilnika) dobilo natančno eno opravilo. Vsa ostala opravila morajo na štafetno palico (blok pomnilnika) počakati.

V arhitekturi ARM je programski števec PC preprosto R15 oziroma eden od računskih registrov. Razcep izvajanja programa oziroma programski skok izvedemo tako, da vsebino R15 spremenimo z enim od računskih ukazov. Absolutni skok naredimo tako, da v PC vpišemo novo vsebino. Relativni skok naredimo tako, da vsebini PC prištejemo ali odštejemo določeno vrednost.

Računska ukaza ADD oziroma SUB torej omogočata relativne skoke. Žal je takojšnji odmik v ukazih ADD oziroma SUB omejen na 8-bitno vrednost ali na 255 oziroma -255. Arhitektura ARM zato za relativne skoke uvaja ukaz B (Branch), ki ima rezerviranih kar 24 bitov za takojšnji odmik. Teh 24 bitov se štetih po 32-bitnih ukazih preslika v programsko področje velikosti 64Mbyte oziroma skok v razponu od -32Mbyte (nazaj) do +32Mbyte (naprej).

Preračunavanje odmika pri relativnem skoku je duhamorno opravilo. Glavna naloga zbirnikov vseh znanih mikroprocesorjev je samodejno preračunavanje odmikov glede na labele v programu:

```
        ...             ;primer (brezpogojnega) skoka
        ...
        B       L1      ;tu skočimo na labelo L1, zbirnik samodejno izračuna odmik
        ...
L1      ...             ;tu se izvajanje nadaljuje
```

Povsem jasno je ukaz B (Branch) lahko pogojno izvedljiv kot vsi ostali ukazi ARM. Na večini drugih mikroprocesorjev je ukaz B (Branch) sploh edini, ki je vedno lahko pogojno izvedljiv. Ko se (pogojni) ukaz B izvede, nujno povzroči izgubo vsebine cevovoda na vseh znanih procesorjih.

```
        ...             ;primer pogojnega skoka
        ...
        BEQ     L1      ;pogojno skočimo na labelo L1 samo v primeru Z=1
        ...             ;tu nadaljujemo brez skoka v primeru Z=0
        ...
L1      ...             ;tu se izvajanje nadaljuje v primeru Z=1
```

Poleg skoka oziroma pogojnega razcepa programa pogosto uporabljamo klic podprogramov. Vsi mikroprocesorji imajo v ta namen posebne ukaze CALL oziroma BSR (Branch to SubRoutine) in podobno. Poleg skoka mora takšen ukaz omogočiti tudi povratek nazaj na natančno isto mesto v glavnem programu, od koder smo klicali podprogram.

Razlika med skokom in klicem podprograma je v temu, da si mora klic poleg skoka zapomniti povratno pot. Večina mikroprocesorjev v tem primeru shrani naslov povratka na sklad. V arhitekturi ARM je povratek iz podprograma rešen na preprostejši in predvsem hitrejši način. Procesor ARM si povratni naslov preprosto zapomni v R14, ki ga imenujemo tudi Link Register ali LR. Pripadajoči ukaz se imenuje BL (Branch and Link):

```
        ...             ;glavni program s primerom klica podprograma
        ...
        BL    L1        ;Branch and Link pomeni PC=>LR in nato skok na L1
        ...             ;sem se vrnemo iz podprograma
        ...

L1      ...             ;klicani podprogram
        ...
        MOV PC,LR       ;tu se vrnemo nazaj (enakovreden ukaz je BX LR)
```

Kot ostali ukazi je tudi BL v arhitekturi ARM lahko pogojno izvedljiv. Povsem jasno, če podprogram v svoji notranjosti kliče še druge pod-podprograme, bo treba LR nekam shraniti, običajno kar na sklad:

```
L1      STR  LR,[SP,#-4]!     ;v podprogramu L1 shranimo LR v sklad
        ...
        BL    L2              ;tu kličemo pod-podprogram na labeli L2
        ...
        LDR  PC,[SP],#4       ;povratni naslov naložimo iz sklada v PC
```

Seveda nas arhitektura ARM nikjer ne sili, da bi morali uporabljati sklad. V gornjem primeru bi LR lahko shranili tudi drugam, mogoče brez zamudnega dostopa do pomnilnika? Če sklad uporabljamo na mikroprocesorju ARM, ga običajno uporabljamo tako, kot pri drugih mikroprocesorjih: sklad raste navzdol, podatke shranjujemo na sklad s STR pre-indexed navzdol in dobimo nazaj z LDR post-indexed navzgor.

4. Izjeme ARM

Poleg običajnega izvajanja programa mikroprocesor doživi tudi izjemne dogodke. Ob vklopu napajanja je nujen reset za pravilen začetek delovanja. Zunanji izvori lahko prožijo prekinitve na različnih ravneh. Lahko se zgodi, da mikroprocesor zahteva ukaz ali podatek iz pomnilnika, ki (še) ni razpoložljiv (Abort). Pri računanju lahko pride do napake, na primer zahteva po deljenju z nič. Mikroprocesorju lahko posredujemo (napačen) ukaz, ki ga mikroprocesor ne zna dekodirati (Undefined). Končno marsikateri mikroprocesor dopušča možnost, da izjemo proži kar glavni program z ukazom SWI (SoftWare Interrupt) ali podobnim.

Izvirni ARM vključno z jedrom ARM7TDMI-S ima razmeroma preproste izjeme in dopušča le dve ravni prekinitev IRQ in FIQ:

Izjeme ARM

Address	Exception	Mode on entry	I state on entry	F state on entry
0x00000000	Reset	Supervisor	Disabled	Disabled
0x00000004	Undefined instruction	Undefined	I	F
0x00000008	Software interrupt	Supervisor	Disabled	F
0x0000000C	Abort (Prefetch)	Abort	I	F
0x00000010	Abort (Data)	Abort	I	F
0x00000014	Reserved	Reserved	-	-
0x00000018	IRQ	IRQ	Disabled	F
0x0000001C	FIQ	FIQ	Disabled	Disabled

Ob izjemnem dogodku se izvajanje trenutnega programa prekine in mikroprocesor izvede klic podprograma na pripadajočem naslovu v seznamu izjem. Preprost seznam izjem izvirnega ARM vsebuje le osem naslovov, ki se nahajajo na začetku naslovnega prostora. V vsak naslov vpišemo ukaz B (brezpogojni Branch), ki bo skočil na pripadajočo obdelavo izjeme. Izjema FIQ je namenoma postavljena na zadnje mesto, da ni treba izgubljati časa z ukazom B, pač pa kar nadaljujemo z obdelavo hitre prekinitve.

Ker je izjema nepredvidljiv dogodek, ki lahko prekine izvajanje nekega drugega programa kjerkoli, ne moremo prav nič predpostavljati o vsebini Link

Registra R14 (LR) niti o vsebini Program Status Registra (Current PSR). Vsaka izjema zato dobi svoj nov R14 in svoj nov PSR, da tja takoj shrani naslov za povratek. Stari R14 in stari PSR prekinjenega programa se medtem shranita, da se njuno vsebino povrne ob zaključku obdelave izjeme.

Izvorna arhitektura ARM ne dopušča neporavnanega dostopa (unaligned access) do pomnilnika. To pomeni, da se morajo 16-bitni podatki vedno nahajati na sodem naslovu: Podobno se morajo 32-bitni ukazi nahajati na naslovu, ki je deljiv s štiri. Ker vnaprej ne vemo, kako velike podatke potiska na sklad nek program: 8-bitne, 16-bitne ali 32-bitne, tudi ne moremo zagotoviti, da je kazalec na sklad R13 ali SP (Stack Pointer) poravnan na naslov, ki je deljiv s štiri.

Vsaka izjema ARM zato dobi tudi svoj nov, povsem neodvisen kazalec na sklad R13. Stari R13 prekinjenega programa se medtem shrani, da se njegovo vsebino povrne ob zaključku obdelave izjeme. Če izjema uporablja sklad, moramo pred klicem izjeme zagotoviti, da smo pripadajoči R13 izjeme pravilno nastavili in pravilno poravnali glede na predvideno uporabo v izjemi.

Posebnost je hitra prekinitev FIQ. Ta dobi kar sedem novih registrov. Poleg LR (R14) in SP (R13) še pet novih računskih registrov R8-R12. Na ta način ne izgubljamo dragocenega časa z reševanjem vsebine računskih registrov prekinjenega glavnega programa v sklad ali kam drugam v pomnilnik. Po pravilnem izhodu iz prekinitve FIQ jasno dobimo nazaj vseh sedem shranjenih R8-R14 in shranjeni (Saved) PSR.

Izjemi Undefined in Abort mogoče zna rešiti velik operacijski sistem. V malem mikrokrmilniku, kjer se izvaja samo naš program, predstavljata Undefined oziroma Abort tako hudo programsko napako, da je edini možni izhod reset mikrokrmilnika.

Izjeme, ki jih v mikrokrmilnikih pogosto potrebujemo, so prekinitve. Izvirna arhitektura ARM razpolaga z dvema prekinitvama IRQ in FIQ. Obe prekinitvi sta po resetu izklopljeni. Vključimo ju s pripadajočima bitoma I in F v CPSR šele takrat, ko smo nastavili prekinitve v vhodno/izhodnih enotah in pripadajoče registre v jedru ARM: dodatni SP za IRQ in še kakšen dodaten register več za FIQ. Primer vklopa FIQ:

```
        MRS     R0,CPSR         ;premakni vsebino CPSR=>R0
        BIC     R0,R0,#0x40     ;F=0 vključi prekinitve FIQ
        MSR     CPSR_c,R0       ;premakni vsebino R0=>CPSR spodnji
byte
```

Do dodatnih registrov prekinitev dostopamo tako, da v glavnem programu spremenimo način M v CPSR iz Supervisor v IRQ oziroma FIQ.

Takoj po nastavitvi dodatnih (banked) registrov in obvezno pred vklopom prekinitev vrnemo M nazaj v Supervisor!

Ob prekinitvi jedro ARM takoj prepreči nadaljnje prekinitve iste ali nižje ravni. Prekinitev IRQ torej izključi nadaljnje prekinitve IRQ. Prekinitev FIQ takoj izključi nadaljnje prekinitve FIQ in IRQ, ker so slednje na nižji ravni.

Prekinitve se samodejno ponovno vključijo ob predpisanem povratku iz izjeme. Različne izjeme zahtevajo v arhitekturi ARM različne ukaze za povratek. Arhitektura ARM predpisuje naslednji ukaz za povratek iz prekinitve:

```
SUBS    PC,LR,#4        ;povratek iz prekinitve IRQ oziroma FIQ
```

Predpisani ukaz naredi tri stvari:
1) zamenja posebne (banked) registre z registri glavnega programa,
2) v CPSR vpiše shranjeno vrednost glavnega programa iz SPSR in
3) iz LR izračuna PC za povratek v glavni program.

Kmalu po prelomu tisočletja je arhitektura ARM postala žrtev lastnega tržnega uspeha. Silno pametni tržniki so najprej zahtevali varčevanje. Zmogljivemu naboru 32-bitnih ukazov ARM so v jedru ARM7TDMI-S dodali šlampast nabor 16-bitnih ukazov Thumb. Thumb naj bi privarčeval nekaj programskega pomnilnika za visoko ceno izgube polovice registrov, izgube pogojnega izvajanja ukazov in izgube zmogljivega shifter operanda. V ARM7TDMI-S lahko teče Thumb samo kot glavni program, vse izjeme pa se izvajajo v 32-bitnem načinu ARM.

Tržniki so nadalje ugotovili, da dve ravni prekinitev IRQ in FIQ ne zadoščata niti v preprostem mikrokrmilniku. Motorola MC680xx ima sedem ravni prekinitev, MIPS pet ravni. ARM se je preusmeril v novo procesorsko jedro CORTEX, ki pozna samo še ukaze Thumb in omogoča večje število ravni prekinitev. Varčevanje je pometlo z dodatnimi (banked) registri izjem, torej reševanje na sklad upočasnjuje prekinitve. Nabor ukazov so silno zakomplicirali, da bi mogoče dosegli učinkovitost manjkajočih 32-bitnih ukazov ARM?

Povedano po domače, CORTEX izgleda kot bolna starka Motorola MC68000 na invalidskem vozičku. Tržniki so torej uspeli zasukati zgodovino za več kot tri desetletja nazaj. Ta prava Motorola medtem ni držala križem rok. Na pogorišču MC68000 je zrasel ColdFire, ki izvaja zmogljivi nabor CISC ukazov MC68000 s hitrostjo sodobnega RISC. Tam, kjer je zmogljivost procesorja še kako pomembna, na primer v domači omrežni opremi: usmerjevalniki, WLAN in podobno, je zmogljivejši MIPS izrinil ARM...

5. Zbirnik ARM

Isti računalnik lahko programiramo za isto nalogo na nešteto možnih načinov. Na eni strani imamo skrajneže, ki zapišejo enice in ničle, da iz njih sestavijo računalniški program. Na drugi strani imamo skrajneže, ki narišejo oblačke in puščice, da bi sestavili računalniški program. Učinek dela enih in drugih skrajnežev je običajno majhen, saj niti premajhna niti prevelika stopnja abstrakcije programskega jezika nista učinkoviti.

Na kakšen način bomo zapisali računalniški program, je zagotovo odvisno od naloge, ki jo moramo rešiti. Če moramo številsko integrirati tirnico plovila na poti iz Zemlje na Mars, potrebujemo višji jezik FORTRAN ali kaj podobnega z najdaljšo možno mantiso. Če se ukvarjamo s številsko obdelavo signalov in skušamo napisati učinkovit FFT, bo treba natančno preučiti, kakšne računske ukaze točno zmore naš računalnik na ravni strojnega jezika.

Če pišemo gonilnik za vmesnik za Ethernet, bo večina našega truda posvečena preučevanju vmesnika in je procesor le stranski pripomoček. Če moramo v naš program vgraditi kompliciran izdelek nekoga drugega, na primer IP sklad, se bo treba prilagoditi izdelku drugih. Končno, če računamo na zaslužek z napakami, okvarami in vzdrževanjem, bomo uporabili prevajalnik Crash++.

Izbira programskega jezika je odvisna tudi od arhitekture procesorja. Če je naš procesor sračje gnezdo z nepreglednim naborom ukazov v strojnem jeziku, bo programiranje v višjem jeziku bolj učinkovito. Če je arhitektura procesorja kristalno čista kot izvorni ARM, programiranje v zbirniku zagotovo pripelje prej do boljšega rezultata kot učenje višjega jezika.

Kaj je to zbirnik? Zbirnik (assembler) je programski jezik, ki omogoča natančno preslikavo 1:1 v strojno kodo določenega procesorja. Vsi zgledi v tem sestavku so zaradi preglednosti napisani v zbirniku ARM, kjer strojne ukaze zbirnik nadomešča s človeku razumljivejšim zapisom, ampak vedno v točni preslikavi 1:1.

Kljub raznolikosti arhitektur in naborov ukazov različnih mikroprocesorjev so si zbirniki med sabo precej podobni. V vsako vrstico zbirnika vpišemo en strojni ukaz našega mikroprocesorja. Ukaz je vedno zamaknjen glede na začetek vrstice. Karkoli je na začetku vrstice, zbirnik smatra za labelo. Z labelo si označimo položaj v programu, kamor bomo napeljali skok ali klic podprograma. Izjemoma nekateri zbirniki (GNU) uporabljajo drugačen zapis label, ki ni pogojen s položajem v vrstici.

Vsak zbirnik dopušča naše komentarje, ki nimajo nobenega učinka na

delovanje zbirnika. Komentar je vse, kar v določeni vrstici sledi znaku ";" (podpičje). Podpičje s komentarjem je lahko tudi na začetku vrstice, na primer:

```
;(komentar1)
            (ukaz-račun1)   A,B,C       ;(komentar2)
(labela1)   (ukaz-račun2)   D,E         ;(komentar3)
            (ukaz-račun3)   G,H,J       ;(komentar4)
            (ukaz-skok)     (labela1)   ;(komentar5)
            (ukaz-račun4)   F,K         ;(komentar6)
```

Osnovna naloga zbirnika je prevajanje ukazov iz naše človeške oblike v strojno kodo izbranega procesorja. Dodatno vsak zbirnik izračuna položaje label v strojni kodi. Izračunane naslove label vstavi v pripadajoče skoke oziroma klice podprogramov.

Slovnica ukazov se od procesorja do procesorja razlikuje. Vsi zbirniki navajajo najprej vrsto ukaza. Za samo vrsto ukaza po presledku sledijo argumenti, ločeni z vejicami. Računski ukazi v zbirniku ARM vsebujejo najprej rezultat in za njim podatke. V zbirniku MC680xx je ravno obratno!

Vsi zbirniki poznajo ukaz EQU. Ta se nikamor ne prevede, pač pa vrednosti labele priredi neko vrednost, ki je drugačna od naslova v programu. Zbirnik ARM z ukazom EQU prireja labelam ne samo konstante, pač pa vrednosti računskih izrazov, ki lahko vsebujejo prej določene labele:

```
KVARC   EQU 14745600        ;frekvenca kristala (Hz)
CLK     EQU KVARC*4         ;frekvenca jedra ARM (PLL)
PCLK    EQU CLK/2           ;frekvenca vhodno/izhodnih enot (APBDIV)
BAUD    EQU PCLK/153600     ;modulo deljenja UART za 9600bps*16
N1US    EQU CLK/3000000     ;konstanta čakanja T1US za HD44780
```

Računanje v stavkih EQU opravi zbirnik. Ti stavki se nikamor ne prevajajo v strojno kodo ciljnega procesorja. Pač pa zbirnik vpiše izračunane vrednosti label povsod tam, kjer se naš program nanje sklicuje.

Takojšnje številske vrednosti (oznaka "#" oziroma "lojtrca") lahko zapišemo v zbirniku ARM na različne načine. Vsi štirje spodnji ukazi MOV se prevedejo v popolnoma enako strojno kodo, čeprav so zapisani na različne načine:

```
MOV     R0,#65              ;zapis v desetiškem sistemu
MOV     R0,#0x41            ;zapis v šestnajstiškem sistemu
MOV     R0,#0b01000001      ;zapis v dvojiškem sistemu
MOV     R0,#"A"             ;zapis kot ASCII znak
```

Pri vseh mikroprocesorjih RISC naletimo na hudo težavo: kako naložiti v register takojšnjo številsko vrednost, ki ima enako število bitov, kot je dolžina ukaza? Na primer, kako v 32-bitni procesor RISC naložiti 32-bitno število? Po definiciji RISC predstavlja vsak ukaz natančno ena beseda. Če potrebujemo 32 bitov za število, nam ostane nič bitov za samo kodo ukaza?

Nalaganje 32-bitne konstante rešuje zbirnik ARM s psevdo-ukazi LDR in LTORG. Ta posebni LDR se od običajnega LDR razlikuje v enačaju "=" pred konstanto namesto lojtrc "#". Primer uporabe psevdo-ukazov:

```
LDR     R0,=0xE01FC0C4      ;psevdo-ukaz LDR
...
...
B       (nekam)             ;brezpogojni skok brez povratka
LTORG                       ;neizvedljiv položaj v programu!
```

Z ukazom LTORG si rezerviramo neizvedljiv položaj v pomnilniku (na primer za brezpogojnim skokom brez povratka), kamor bo zbirnik shranil našo konstanto in izračunal njen odmik od psevdo-ukaza. LTORG mora biti v programu zadosti blizu psevdo-ukazu LDR, da ga lahko naslavljamo kot [PC,#odmik]. Psevdo-ukaze bo zbirnik ARM nato prevedel v izvedljive ukaze:

```
LDR     R0,[PC,#odmik]      ;izvedljiv ukaz
...
...
B       (nekam)             ;brezpogojni skok
DCD     0xE01FC0C4          ;neizvedljiv položaj v programu!
```

Seveda lahko konstante naložimo v neizvedljiv del programskega pomnilnika tudi sami z ukazi zbirnika DCB (8bit), DCW (16bit) in DCD (32bit). Primeri:

```
DCB     0xEE                ;naložimo en byte
DCB     97,13,10            ;naložimo tri zaporedne byte
DCB     "K"                 ;naložimo en byte
DCB     0xA1,0xB2,0xC3      ;naložimo tri zaporedne byte
DCB     "abcdefghijk"       ;naložimo 11 zaporednih byte
ALIGN                       ;zagotovimo poravnan 32-bitni naslov
DCW     0xA1F5              ;naložimo 16-bitov (Half-word)
ALIGN                       ;zagotovimo poravnan 32-bitni naslov
DCD     0xAACC0123          ;naložimo 32-bitov (Word)
```

Ko smo z ukazi DCB in DCW nalagali 8-bitne in 16-bitne vrednosti, smo povsem zanemarili zahtevo ARM, da morajo biti naslovi poravnani, torej 16-

bitne vrednosti na sodih naslovih in 32-bitne vrednosti oziroma ukazi ARM na naslovih, deljivih s štiri. Tu nam pomaga ukaz ALIGN. ALIGN brez argumenta vstavi toliko ničelnih byte, da je naslov za njim poravnan na mejo 4 byte.

Argument ukaza ALIGN je lahko tudi večji od 4. 32-bitni mikrokrmilniki različnih proizvajalcev imajo pomnilnik FLASH organiziran v večjo širini, tudi 128 bitov (16 byte) oziroma 4 ARM ukaze vzporedno. Predvidljivo obnašanje programskega predpomnilnika bo zagotovil ukaz ALIGN v neizvedljivem odseku pred časovno zahtevnim odsekom programa:

```
ALIGN     16     ;zapolnimo z ničelnimi byte do mnogokratnika 16 byte
```

Po gornjem ukazu lahko položaj programa v pomnilniku FLASH še natančno nastavimo z ukazom SPACE. SPACE bo preprosto vstavil toliko ničelnih byte, kot mu navedemo v argumentu:

```
SPACE     12     ;vstavimo 12 praznih (ničelnih) byte
```

V marsikateri program želimo vgraditi tudi večje količine podatkov, kjer bi bila uporaba DCB, DCW ali DCD silno zamudna. V teh primerih bo vstavil zbirnik obsežne podatke iz zunanjega zapisa (zvočni zapis, bitna slika in podobno) v naš program z ukazom INCBIN. Primer uporabe v elektronskem zvoncu:

```
;obvezno neizvedljivi del programa!
L1    INCBIN    DINDON.WAV    ;zvočni zapis "din-don"
      ALIGN                   ;poravnava naslova
```

Labeli L1 je v gornjem primeru dodeljen začetni naslov vstavljenega zvočnega zapisa DINDON.WAV. Ker imajo zapisi .WAV poljubno dolžino byte, moramo na koncu obvezno dodati ALIGN, če sledijo 32-bitni podatki oziroma izvedljiv program.

V zbirniku se pogosto zgodi, da na več mestih oziroma večkrat uporabljamo isto zaporedje ukazov. V časovno kritičnem delu programa preprosto ponavljamo iste ukaze, saj bi zanka ponavljanja s pogojnim skokom porušila vse cevovode. V takšnih primerih nam zelo pomaga ukaz MACRO. Makro moramo v programu obvezno definirati pred njegovo uporabo:

```
      MACRO                   ;začnemo definicijo makroja
      SODO                    ;navedemo ime makroja "SODO"
      LDR       R0,[R1],#4    ;prvi ukaz makroja naloži 32-bitno število
      BIC       R0,R0,#0x01   ;drugi ukaz makroja naredi število sodo
      STR       R0,[R2],#4    ;tretji ukaz makroja shrani sodo število
      MEND                    ;konec definicije makroja
```

Definicijo makroja lahko postavimo kamorkoli, v izvedljivi kot tudi v neizvedljivi del programa, ampak obvezno pred njegovo uporabo. Zbirnik ne bo na to mesto vpisal prav ničesar v prevedeni program! Pač pa bo zbirnik prepisal vsebino makroja povsod tja, kjer se na makro sklicujemo. Primer programa:

```
SODO            ;makro ponovimo 4-krat, obdelamo prvo število
SODO            ;obdelamo drugo število v seznamu
SODO            ;obdelamo tretje število v seznamu
SODO            ;obdelamo četrto število v seznamu
```

POZOR! Ime našega makroja mora biti različno od vseh znanih ukazov zbirnika ARM. Gornji zapis bo zbirnik ARM prevedel v:

```
LDR     R0,[R1],#4      ;prva uporaba makroja
BIC     R0,R0,#0x01
STR     R0,[R2],#4
LDR     R0,[R1],#4      ;druga uporaba makroja
BIC     R0,R0,#0x01
STR     R0,[R2],#4
LDR     R0,[R1],#4      ;tretja uporaba makroja
BIC     R0,R0,#0x01
STR     R0,[R2],#4
LDR     R0,[R1],#4      ;četrta uporaba makroja
BIC     R0,R0,#0x01
STR     R0,[R2],#4
```

Končno, vsak zbirnik zahteva za svoje delovanje neko čarobno formulo na začetku, pred prvim veljavnim ukazom našega programa. Kaj ta čarobna formula točno pomeni in kaj natančno počne, ne zna odgovoriti niti ciganka s svojo kristalno kroglo. Zbirnik ARM preverjeno deluje z naslednjo čarobno formulo:

```
AREA    RESET, CODE, READONLY, ALIGN=9
ENTRY                   ;začetek delovanja zbirnika
CODE32                  ;izberemo nabor ukazov ARM (ne Thumb!)
```

Kar je pricurljalo mimo ciganke, argument ALIGN=9 predpisuje največjo vrednost argumentov ukazov ALIGN in SPACE v eksponentni obliki 2^9=512. Konec delovanja večine zbirnikov in tudi zbirnika ARM označuje ukaz:

```
END                     ;konec delovanja zbirnika
```

Ta ukaz sledi zadnjemu veljavnemu ukazu našega programa.

Zbirnik ARM je kompliciran program. Tudi najpreprostejši prevod v strojni zapis zahteva uporabo najmanj treh programov: ARMASM, ARMLINK in FROMELF. Če uporabljamo licenčni inačici programov ARMASM in ARMLINK, ju skupaj s pripadajočim ključem "license.dat" namestimo v mapo c:\flexlm v Windows 98, XP ali 7.

Naš program uredimo s čim enostavnejšim urejevalnikom besedila, na primer z beležnico Windows Notepad. Da se prevajalnikom ARM ne bo zmešalo, v našem programu prav nikjer ne uporabljamo "jugo" znakov, niti v komentarjih! Torej strogo samo ANSI znaki, zato priporočam preklop tipkovnice na ameriško. (Zgledi iz tega sestavka preverjeno ne delajo zaradi "jugo" črk v komentarjih.) Naše čitljivo besedilo preimenujemo končnico v ".s", ker takšno končnico predpisuje zbirnik ARM za izvorni program.

Naš izvorni program, imenovan na primer "a.s", nato prevedemo z naslednjim zaporedjem ukazov:

```
c:\flexlm\armasm a.s
c:\flexlm\armlink --ro_base 0x00 a.o -o a.axf
c:\flexlm\fromelf --i32 -o a.hex a.axf
```

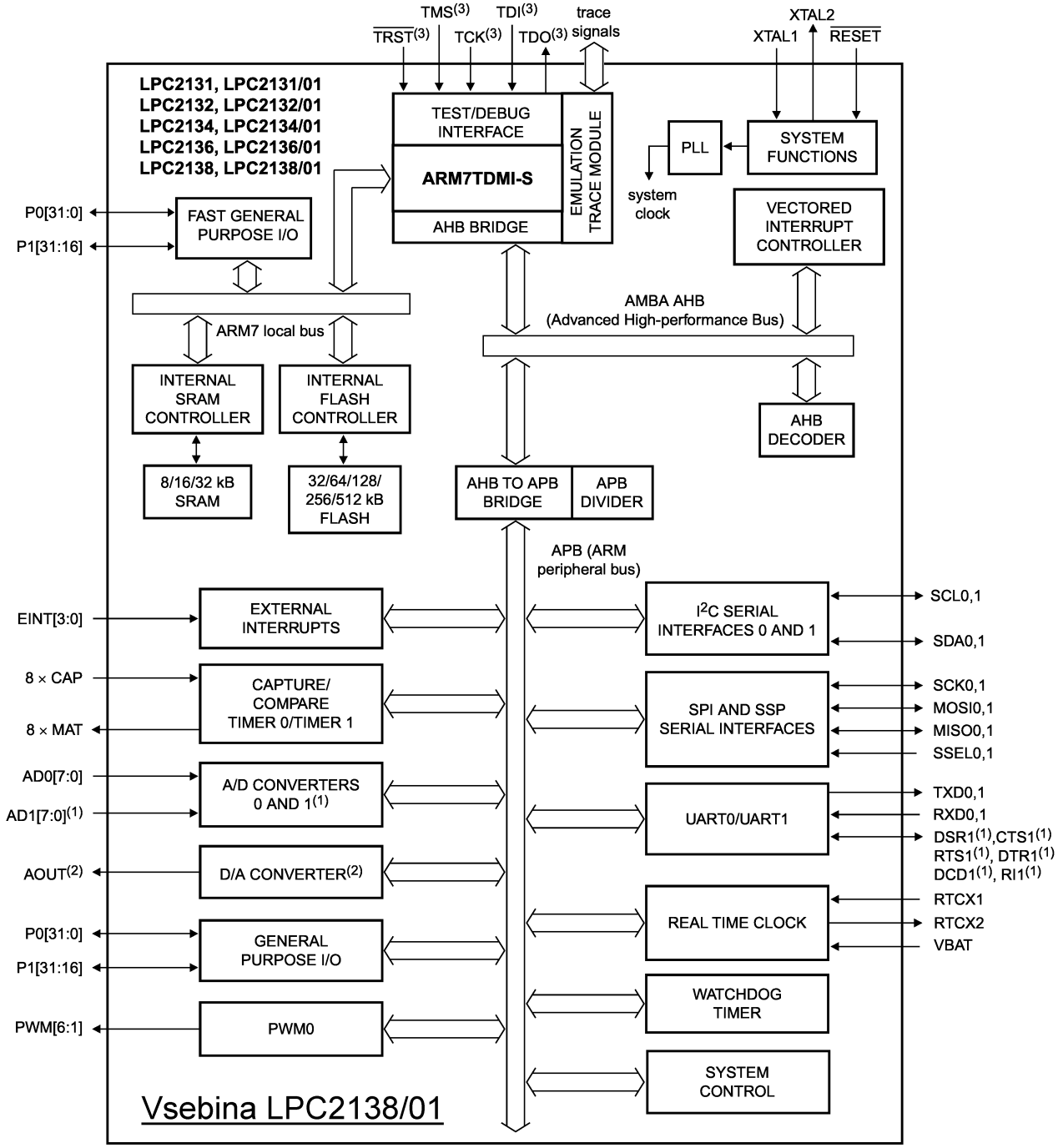
Prvi ukaz požene res pravi zbirnik ARMASM, ki iz našega programa "a.s" naredi prevod "a.o". Če smo kjerkoli v našem programu uporabljali ukaz INCBIN, moramo prevajalniku ponuditi tudi vse zapise, na katere se INCBIN sklicuje. Vse izvorne datoteke vključno z našim programom se morajo nahajati v isti mapi.

ARMLINK združi enega (v predstavljenem primeru) ali več prevodov iz istega oziroma različnih programskih jezikov, v dokončno, skupno strojno kodo "a.axf" v obliki ELF. Za razliko od drugih zbirnikov, zbirnik ARM ne pozna ukaza ORG. Naslov, kam naj se naš izvorni program dokončno prevede, navedemo v ukazni vrstici ARMLINK.

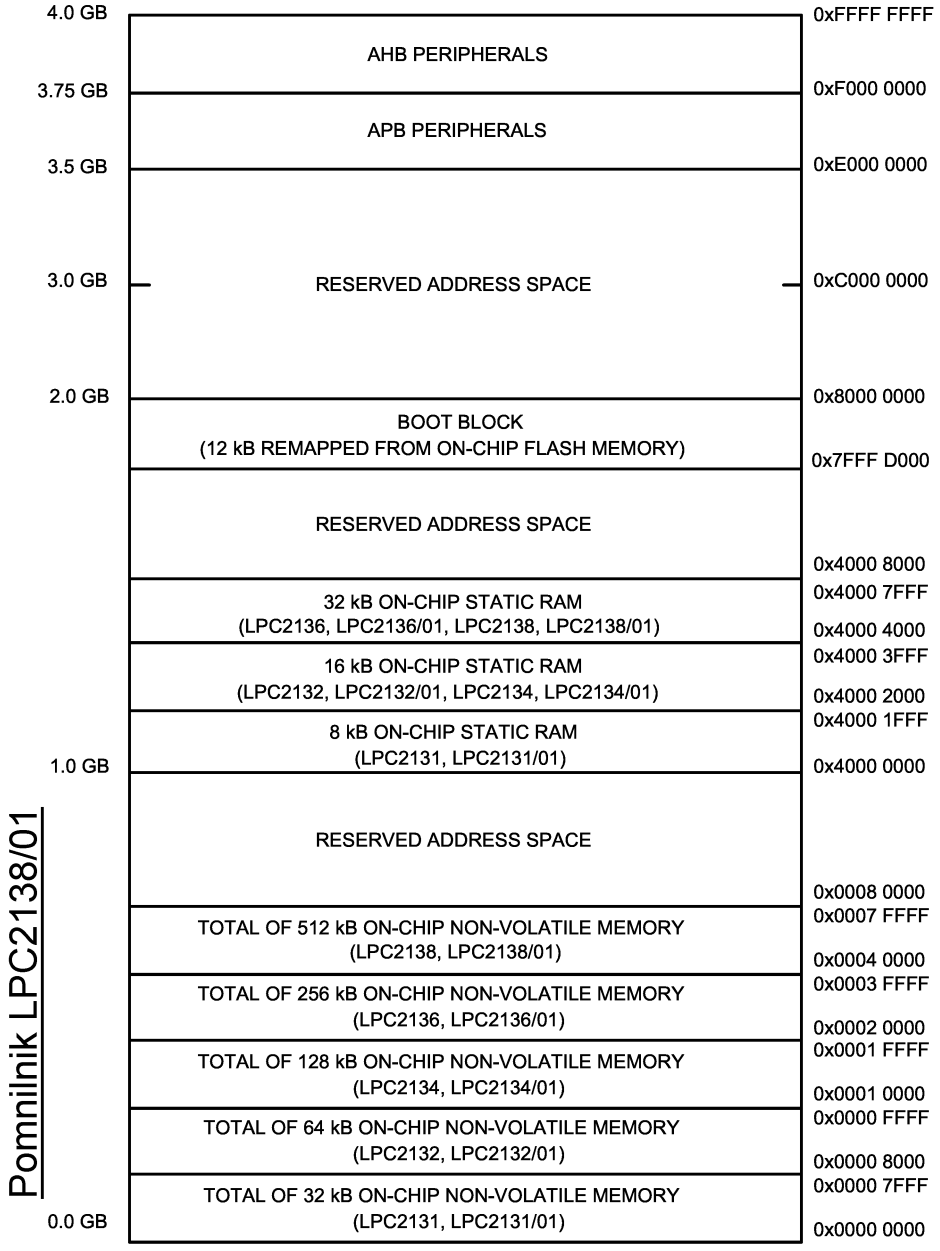
Dokumenti ARM predpisujejo zapis strojne kode v obliki ELF. Ker sicer učinkovite oblike ELF ne razume noben programator, si na koncu pomagamo s FROMELF. Slednji prevede "a.axf" v obliko Intel-HEX v zapisu "a.hex", kar pozna večina orodij za vpis pomnilnikov FLASH.

6. Mikrokrmilnik LPC2138/01

Samostojen 32-bitni mikroprocesor je silno neroden za uporabo: do pomnilnikov in vhodno/izhodnih enot je treba napeljati nepregleden šop žic. Dosti bolj preprosta je uporaba mikrokrmilnika, ki v istem čipu združuje mikroprocesor, različne pomnilnike in vhodno/izhodne enote. Večina mikrokrmilnikov sploh nima zunanjega vodila mikroprocesorja, pač pa le zunanje priključke vhodno/izhodnih enot. Preprost mikrokrmilnik z 32-bitnim jedrom ARM, brez zunanjega vodila, zelo primeren za začetnike, je LPC2138/01 proizvajalca NXP (oddelek polprevodnikov Philips):



Nikakor ne se ustrašiti načrta vsebine LPC2138/01! Sodobni mikrokrmilniki vsebujejo toliko različnih vhodno/izhodnih enot, da jih je popolnoma nemogoče vse izkoristiti v isti napravi. Ohišje čipa sploh nima zadosti zunanjih priključkov, da bi vse enote lahko hkrati počele kaj koristnega. Če smo v neki napravi povezali komaj eno tretjino zunanjih priključkov določenega mikrokrmilnika, se v današnjih časih že lahko pohvalimo, da smo izbrani mikrokrmilnik zelo dobro izkoristili!



Naslovni prostor mikroprocesorja ARM velikosti 4Gbyte je razdeljen med vse različne pomnilnike in vhodno/izhodne enote mikrokrmilnika LPC2138/01. Na začetek naslovnega prostora je nameščen pomnilnik FLASH (Non-Volatile Memory). Proizvajalec čipa zagotavlja, da vgrajeni pomnilnik FLASH (EEPROM) zadrži svojo vsebino brez napajanja za najmanj 20 let.

LPC2138/01 vsebuje pomnilnik FLASH velikosti 512kbyte.

Omejitvi pomnilnika FLASH sta hitrost vpisa in število brisanj/vpisov. Pred vpisom je treba pomnilnik FLASH najprej pobrisati. Proizvajalec obljublja, da vgrajeni FLASH zdrži 100000 (sto tisoč) ciklov brisanja in ponovnega vpisovanja. Poleg tega sta brisanje in vpis v FLASH najmanj tisočkrat počasnejša od periode ure mikroprocesorja. Pomnilnik FLASH torej ni primeren za hranjenje spremenljivk! V pomnilnik FLASH običajno vpišemo naš program s primernim zunanjim programskim orodjem.

Spremenljivkam je namenjen pomnilnik RAM (Random Access Memory). Statični RAM lahko zelo hitro beremo oziroma vanj pišemo, v samo enem ciklu ure mikrokrmilnika. Statični RAM je silno preprost za uporabo. Število ciklov vpisovanja ni omejeno. Žal je ob izgubi napajanja vsebina statičnega RAM izgubljena!

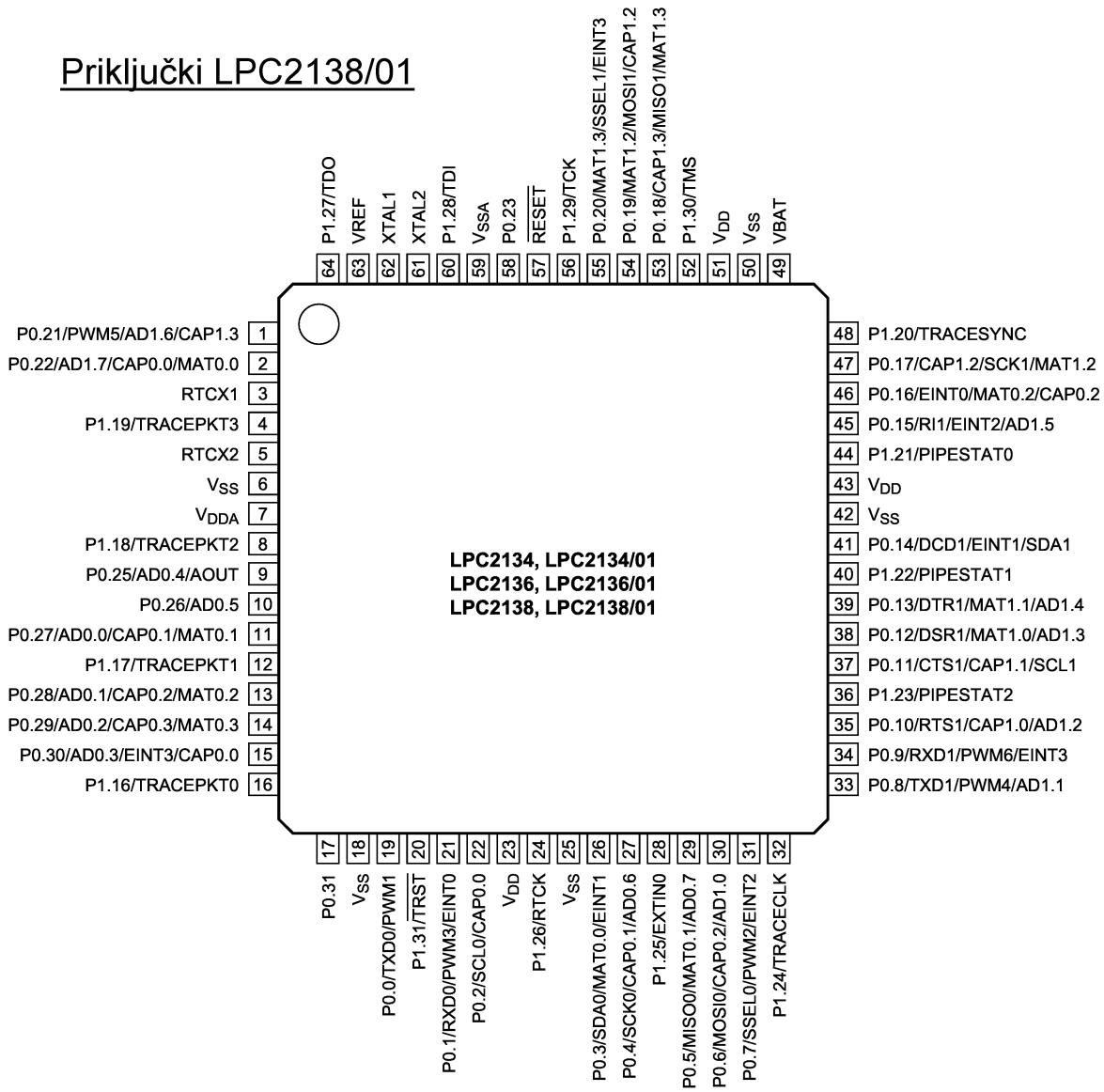
Statični RAM mikrokrmilnika LPC2138/01 je izvedljiv (executable) za razliko od 8-bitnih mikrokrmilnikov. Vanj lahko vpišemo program in celo tja prestavimo seznam izjem (MEMMAP). V statičnem RAM se program izvaja hitreje kot v FLASH, a ga je treba ob vsakem vklopu napajanja ponovno vpisati oziroma prepisati iz FLASH. LPC2138/01 vsebuje statični RAM velikosti 32kbyte, žal razmeroma malo glede na zmogljivost procesorja ARM!

Mikrokrmilniki vključno z LPC2138/01 ne vsebujejo dinamičnega RAM, ki hrani dosti več podatkov od statičnega RAM, a je bolj zahteven za uporabo in počasnejši. Tehnologija dinamičnih pomnilnikov namreč ni združljiva s tehnologijo vezij mikrokrmilnika. Nekateri mikrokrmilniki imajo tudi zunanje vodilo in celo krmilnik za dinamične RAM, kamor lahko priključimo dodatne zunanje pomnilnike.

Najrazličnejše vhodno/izhodne enote (peripherals) mikrokrmilnika občuti procesorsko jedro ARM kot svojevrsten pomnilnik, ki ga imenujemo tudi registri vhodno/izhodnih enot. Teh registrov vhodno/izhodnih enot nikakor ne smemo zamešati z registri procesorja! Vsak register vhodno/izhodnih enot je preprosto pomnilniška beseda dolžine 8 bitov, 16 bitov ali 32 bitov na določenem naslovu. V nekatere registre vhodno/izhodnih enot lahko procesor samo vpisuje, vsebino nekaterih registrov lahko samo bere kot pomnilnik ROM, nekatere registre lahko bere in vpisuje kot pomnilnik RAM.

LPC2138/01 se od prvotnega LPC2138 razlikuje prav v manjših izboljšavah in dopolnitvah vhodno/izhodnih enot. Njegova najpreprostejša vhodno/izhodna enota je (Fast) GPIO. Če jo programiramo kot vhodno enoto, pripadajoči register procesor lahko samo bere kot ROM, prebrani podatek pa ustreza signalom, ki jih pripeljemo od zunaj. Če jo programiramo kot izhodno enoto, pripadajoči register procesor samo vpisuje, vpisana vsebina se

posreduje zunanjemu svetu na pripadajočih priključkih.



Mikrokrmilnik LPC2138/01 omogoča celo dva različna GPIO na istih zunanjih priključkih: Fast General Purpose I/O in (Legacy) General Purpose I/O. Fast GPIO je spojen neposredno na vodilo procesorja, Legacy GPIO pa na počasnejše vodilo vhodno/izhodnih enot. LPC2138/01 ima skupno 47 takšnih priključkov: P0.0-23, P0.25-31 in P1.16-31, pod pogojem, da jih ne uporabljamo za kaj drugega.

Zahtevnejši vmesniki LPC2138/01 vključujejo komunikacijo, časovnike in analogne pretvornike. Komunikacija vključuje dva neodvisna UART (Universal Asynchronous Receiver Transmitter enakovredno COM portu na PC računalniku), dva neodvisna SPI (Serial Peripheral Interface bus) in dva vmesnika I2C (Inter-Integrated Circuit bus). POZOR! Vmesniki I2C zahtevajo izhode vrste open-drain na P0.2, P0.3, P0.11 in P0.14, zato je delovanje teh izhodov tudi v načinih GPIO (Fast in Legacy) okrnjeno na pull-down!

Časovniki vključujejo dva neodvisna števca TIMER0 in TIMER1, pulzno-širinski modulator PWM0 in uro s koledarjem RTC (Real-Time Clock). RTC lahko deluje tudi na neodvisno (baterijsko) napajanje V_{BAT} in lasten kristal. Analogni vmesniki vključujejo dva neodvisna 10-bitna A/D pretvornika, vsak z več vhodi in en 10-bitni D/A pretvornik.

Poleg vhodno/izhodnih enot se preko podobnih registrov nastavljajo tudi številne sistemske funkcije celotnega mikrokrmilnika LPC2138/01: fazno-sklenjena zanka za takt (PLL), predpomnilnik za pospeševanje FLASH (MAM), zunanje prekinitve, upravljanje vseh prekinitev z Vectored Interrupt Controller (VIC), kužapazi (Watchdog), izbira takta vhodno/izhodnih enot (APBDIV), vklop posameznih vhodno/izhodnih enot (PCONP) ter ne nazadnje, katera vhodno/izhodna enota bo sploh dosegljiva preko določenega zunanjega priključka mikrokrmilnika (PINSEL). Na primer, nogica številka 9 ohišja mikrokrmilnika LPC2138/01 je lahko GPIO P0.25, lahko je vhod A/D pretvornika AD0.4 ali pa izhod D/A pretvornika AOUT.

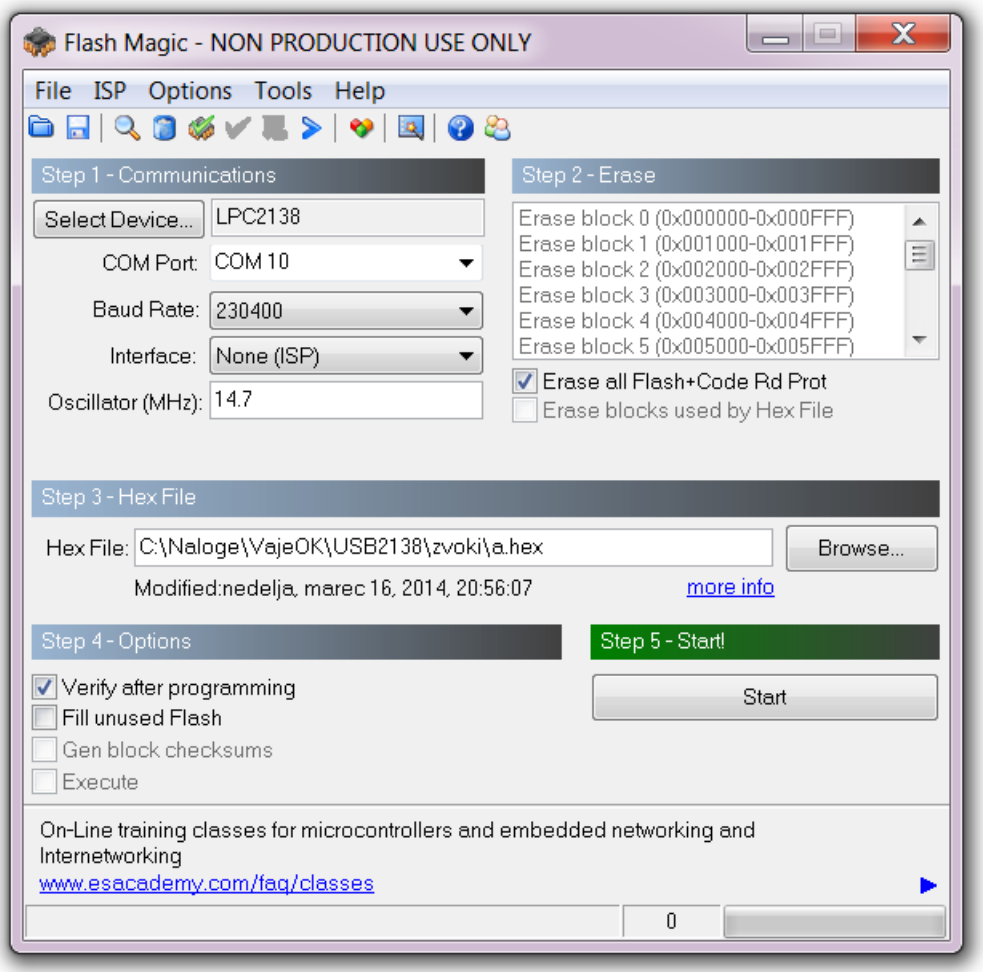
Za uporabnika stvari niso tako komplicirane. Reset ob vklopu postavi večino vhodno/izhodnih enot in sistemskih funkcij LPC2138/01 v znano stanje. Vsi zunanji priključki so takrat GPIO vhodi. Vse sistemske funkcije se postavijo tako, da mikrokrmilnik oživi ne glede na frekvenco zunanjega takta. Vse prekinitve so na začetku izključene. Naš uporabniški program seveda požene le tiste vhodno/izhodne enote, ki jih v resnici potrebuje. Neuporabljenim vhodno/izhodnim enotam se da celo izklopiti napajanje, da privarčujemo pri porabi.

Sodobni mikrokrmilniki vsebujejo tudi tovarniško vpisan program imenovan Bootloader, ki omogoča programiranje pomnilnika FLASH ISP (In-System Programming oziroma čip vgrajen v ciljno vezje) oziroma razhroščevanje. V mikrokrmilniku LPC2138/01 je Bootloader tovarniško vpisan v gornjih 12kbyte pomnilnika FLASH. Za uporabniški program ostane na voljo 500kbyte pomnilnika FLASH.

Do programa Bootloader lahko dostopamo preko vmesnikov UART0 oziroma JTAG. Dostop do Bootloaderja izbiramo z vezavo priključkov P0.14 (UART0), P0.31 (JTAG), P1.20 (razhroščevalnik TRACE) in P0.26 (JTAG) ob resetu mikrokrmilnika LPC2138/01.

Bootloader se sicer vedno zažene ob vklopu in preveri seznam vseh osem izjem ARM, vključno z neuporabljeno izjemo na naslovu 0x00000014. Mikrokrmilniki družine LPC2xxx uporabljajo to mesto v pomnilniku kot kontrolno vsoto (checksum) izjem. Samo v primeru, da se ta kontrolna vsota ujema, Bootloader prepusti izvajanje uporabniškemu programu v FLASH.

Bootloader ISP je najbolj preprosto uporabljati na priključku UART0. Komunikacijski protokol za programiranje FLASH je enostaven in je objavljen v dokumentaciji mikrokrmilnika LPC2138/01. Poleg tega nam daje proizvajalec NXP na voljo sicer počasen, a brezplačen program FlashMagic za programiranje pomnilnika FLASH vseh njegovih izdelkov.



Kar žal ni nikjer objavljeno, sam postopek brisanja in vpisa FLASH pomnilnika. Proizvajalec NXP si je preko uporabe Bootloader ISP omogočil, da posodobi tehnologijo FLASH pomnilnikov ter spremeni postopek brisanja in vpisa, pri tem pa uporabniku ni treba menjati ničesar v orodjih za programiranje. Za razliko od mikrokrmilnikov Microchip PIC, kjer programiranje novejše različice istega mikrokrmilnika po starejšem postopku ali obratno povzroči uničenje mikrokrmilnika!

Program v mikrokrmilniku LPC2138/01 se da zavarovati pred nepooblaščenim kopiranjem tako, da se onesposobi oziroma izključi Bootloader z vpisom ključne vsebine v FLASH na naslov 0x000001FC. Ker mikrokrmilnik na ta način lahko uničimo, brez Bootloaderja je mrtev, moramo biti zelo previdni, kaj vpišemo na ta naslov. Ena možnost zavarovanja je ukaz ALIGN 512 takoj po seznamu izjem oziroma po krajšem podprogramu FIQ.

Bootloader lahko prikličemo tudi iz našega uporabniškega programa, ko želimo kaj vpisati v FLASH. Bootloader podpira uporabniške ukaze IAP (In-Application Programming) za brisanje in vpis FLASH. Celoten Bootloader je napisan s 16-bitnimi ukazi Thumb, torej glavobol za nas uporabnike!

Podporo IAP prikličemo na lihem naslovu 0x7FFFFFF1 z ukazom BX (register). Lihi naslov preklopi jedro ARM7TDMI-S iz 32-bitnih ukazov ARM v 16-bitne ukaze Thumb. Resnični naslov programa v Thumb je seveda 0x7FFFFFF0. Bootloader sprejme naš ukaz od tam, kamor kaže R0 in nam odgovori tja, kamor kaže R1:

```
LDR     R0,=IAPCMD          ;naslov v RAM (R0) za ukaz IAP
...                         ;tu zapišemo ukaz IAP
...
LDR     R1,=IAPRES          ;naslov v RAM (R1) za odgovor IAP
LDR     R2,=0x7FFFFFF1      ;IAP naslov (Thumb) v R2
MOV     LR,PC               ;cevovod povzroči ADD LR,PC,#8
BX      R2                  ;tu kličemo IAP program (Thumb)
LDR     R1,=IAPRES          ;naslov v RAM (R1) odgovora IAP
...                         ;tu preberemo odgovor IAP (R1)
...
```

Bootloader sicer pozna celo vrsto ukazov, ki so natančno opisani v "User Manual" pripadajočega mikrokrmilnika. Pred uporabo IAP je obvezno izključiti vse prekinitve IRQ in FIQ ter upoštevati, da IAP uporablja zadnjih 32byte vgrajenega pomnilnika RAM za svoje notranje delovanje!

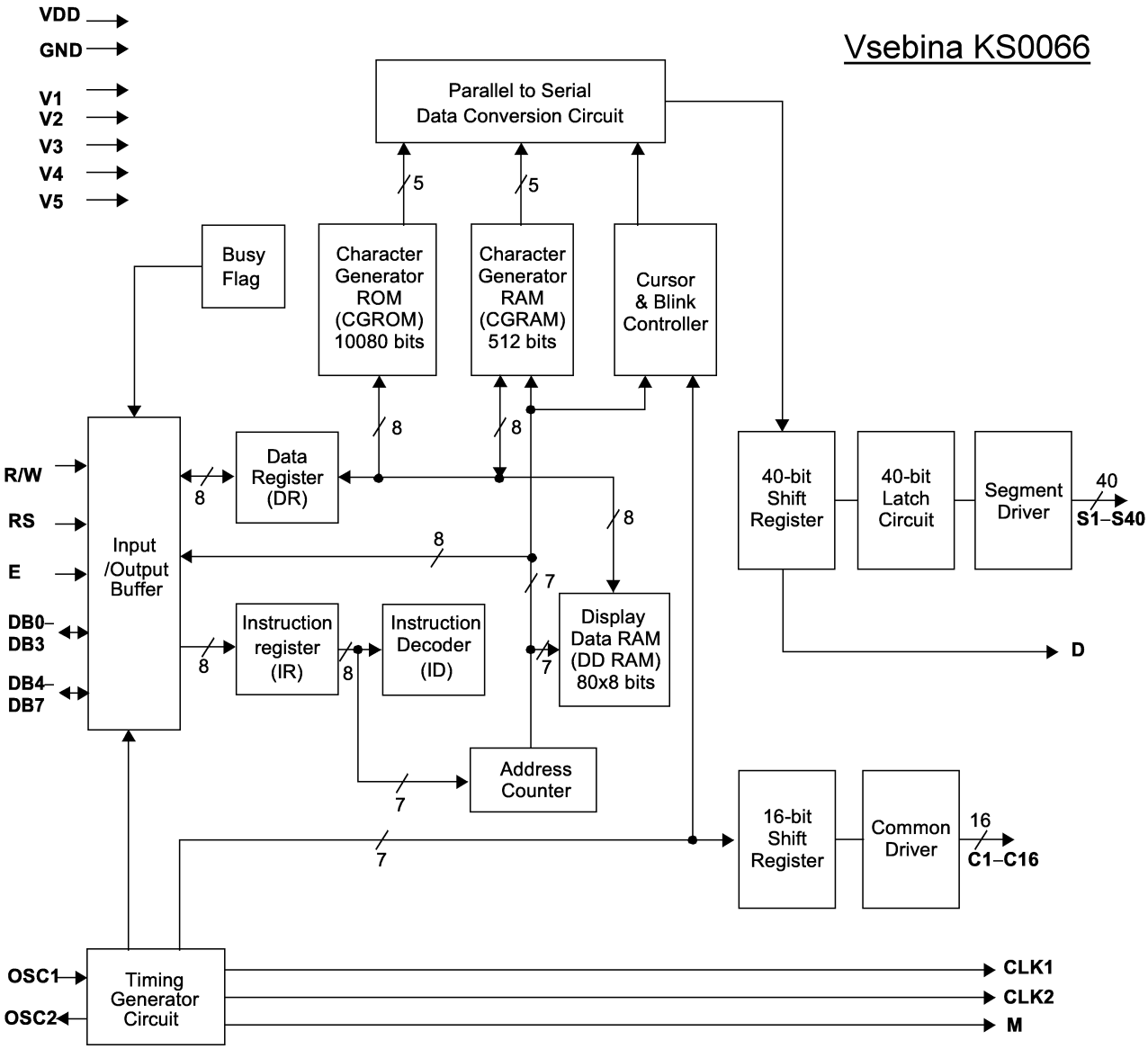
Mikrokrmilnik LPC2138/01 potrebuje eno samo zunanje napajanje V_{DD}=+3.3V. Napajanje analognega dela V_{DDA} in V_{SSA} je ločeno od napajanja številskega dela V_{DD} in V_{SS}. Kljub ločenemu napajanju in skrbni izvedbi tiskanega vezja sta spodnja dva bita A/D pretvornikov šum, če med pretvorbo mikrokrmilnik melje še kaj drugega. Napajanje notranjosti mikrokrmilnika je sicer 1.8V preko vgrajenega regulatorja. Poleg zunanjega priključka za reset vsebuje LPC2138/01 tudi notranji "brown-out reset", ki se proži pri 2.6V.

Mikrokrmilnik LPC2138/01 je CMOS vezje in njegova poraba je v glavnem odvisna od frekvence takta: do 25MHz neposredno iz kristalnega oscilatorja oziroma do 60MHz z uporabo vgrajenega PLL. Skupna poraba na 3.3V se običajno giblje med 15mA in 60mA glede na takt in število vključenih vhodno/izhodnih enot.

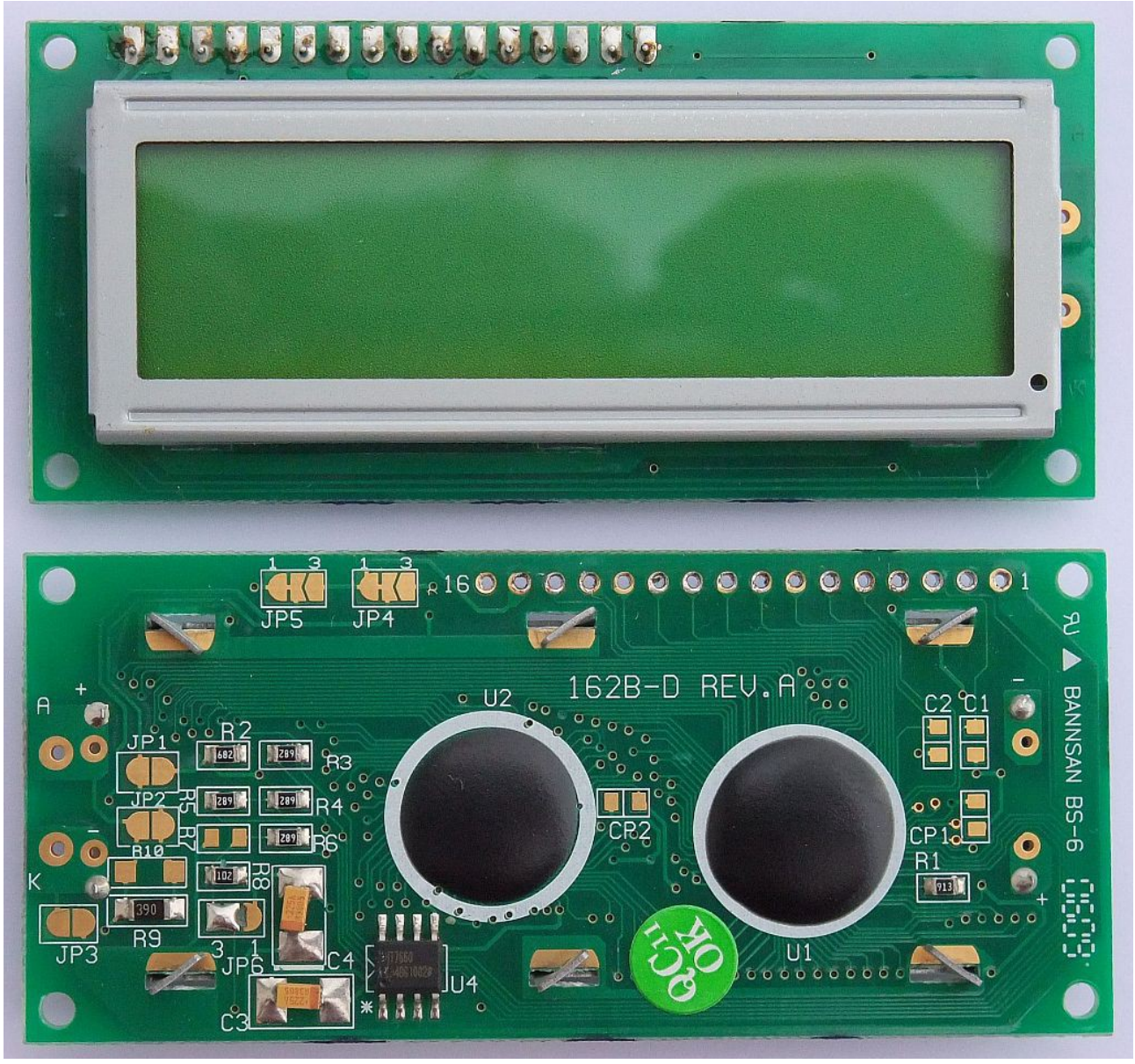
7. Alfanumerični LCD s HD44780

Mikrokrmilnik pogosto potrebuje mali prikazovalnik, kaj se v mikrokrmilniku dogaja, kako je mikrokrmilnik nastavljen oziroma kaj je mikrokrmilnik izmeril. Na tržišču obstaja nepregledna množica različnih prikazovalnikov LCD, TFT oziroma OLED. Prikazovalnike po nalogi delimo na grafične in alfanumerične. Tehnološko jih delimo na multipleksirane in nemultipleksirane. Prikazovalnike dobimo brez vsakršne krmilne elektronike, samo s pomikalnimi registri oziroma z vgrajenim krmilnikom.

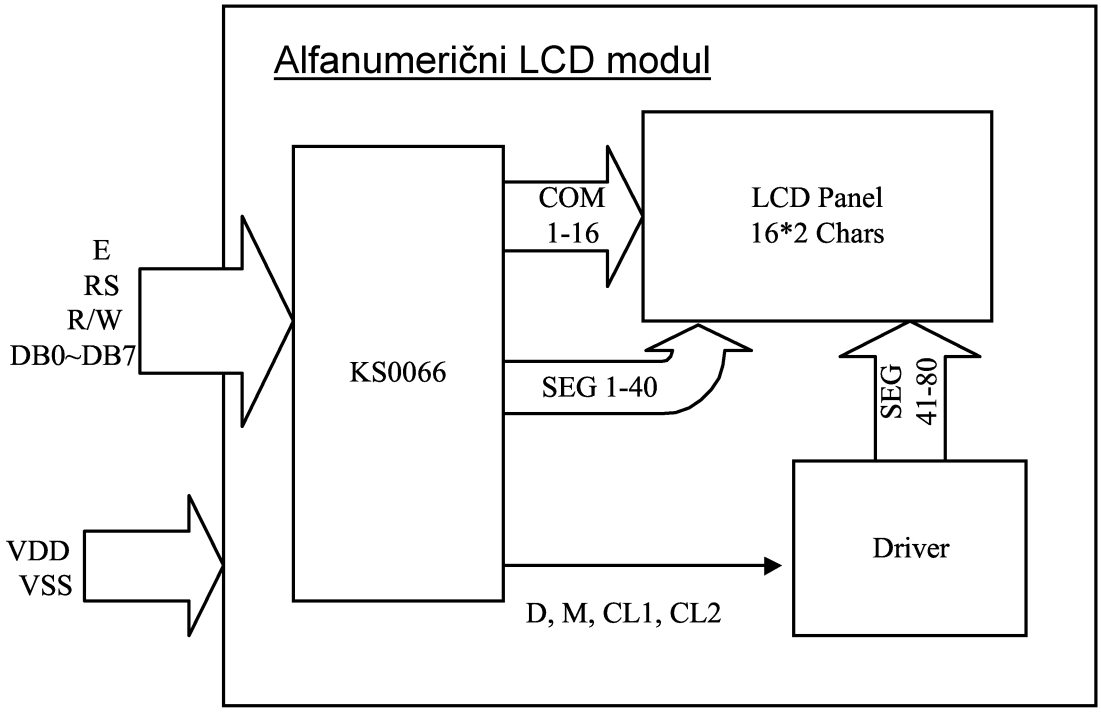
Kaj od navedenega je primerno za začetnika? Vsekakor prikazovalnik z vgrajenim krmilnikom, da je uporaba enostavnejša. Grafični prikazovalnik je zmogljivejši od alfanumeričnega, a zahteva dosti več programiranja. Odločitev je jasna: mali alfanumerični LCD s krmilnikom Hitachi HD44780 oziroma Samsungovim ponaredkom KS0066, ki kraljujeta že četrt stoletja!



HD44780 oziroma KS0066 krmili multipleksiran LCD, ki prikazuje eno ali dve vrstici alfanumeričnih znakov, skupno največ do 80 znakov. Do 8 znakov v vsaki vrstici lahko HD44780 oziroma KS0066 krmili sam, za daljše vrstice pa potrebuje razširitvene čipe (segment driver) za dodatne stolpce. Najnovejši inačici HD44780U oziroma KS0066U delujeta v napetostnem razponu od 2.7V do 5.5V.



Cenen LCD modul ima goli čip HD44780/KS0066 kar bondiran na tiskano vezje in zalit s kapljo črne smole. Povsem enako je vgrajen eden ali več razširitvenih čipov za daljše vrstice. Krmilni priključki modula: vzporedno 8-bitno vodilo, kontrolni signali vodila in napajanje so standardizirani, so oštevilčeni od 1 do 14 in so na vseh znanih modulih vezani enako. Najlažje določimo priključek 1, ki je masa GND oziroma V_{SS} in je vezan na široko vezico na tiskanem vezju. Napajanje modula V_{DD} je +5V ali +3.3V.



No.	Symbol	Function
1	VSS	Ground (0V)
2	VDD	Supply Voltage for Logic (+5V or +3.3V)
3	VO	Contrast Adjustment
4	RS	Data/Instruction Select
5	R/W	Read/Write Select
6	E	Enable Signal
7	DB0	Data Bus
8	DB1	Data Bus
9	DB2	Data Bus
10	DB3	Data Bus
11	DB4	Data Bus
12	DB5	Data Bus
13	DB6	Data Bus
14	DB7	Data Bus
15	LED_A	LED Power Supply + (5V)
16	LED_K	LED Power Supply - (5V)

Sam LCD se krmili z napetostjo, ki jo pritisnemo med V_{DD}(+) in V_O(-). Pri starejših LCD gre ta napetost tudi preko 10V. Sodobni LCD dosežejo dober kontrast že s 5V ali manj. Moduli z napajanjem +3.3V pogosto vsebujejo čip podvojevalnik napetosti ICL7660, da nam ni treba pripeljati negativne napetosti V_O od zunaj. ICL7660 lahko izključimo z mostički na tiskanem vezju

modula, ko ga ne potrebujemo pri napajanju +5V.

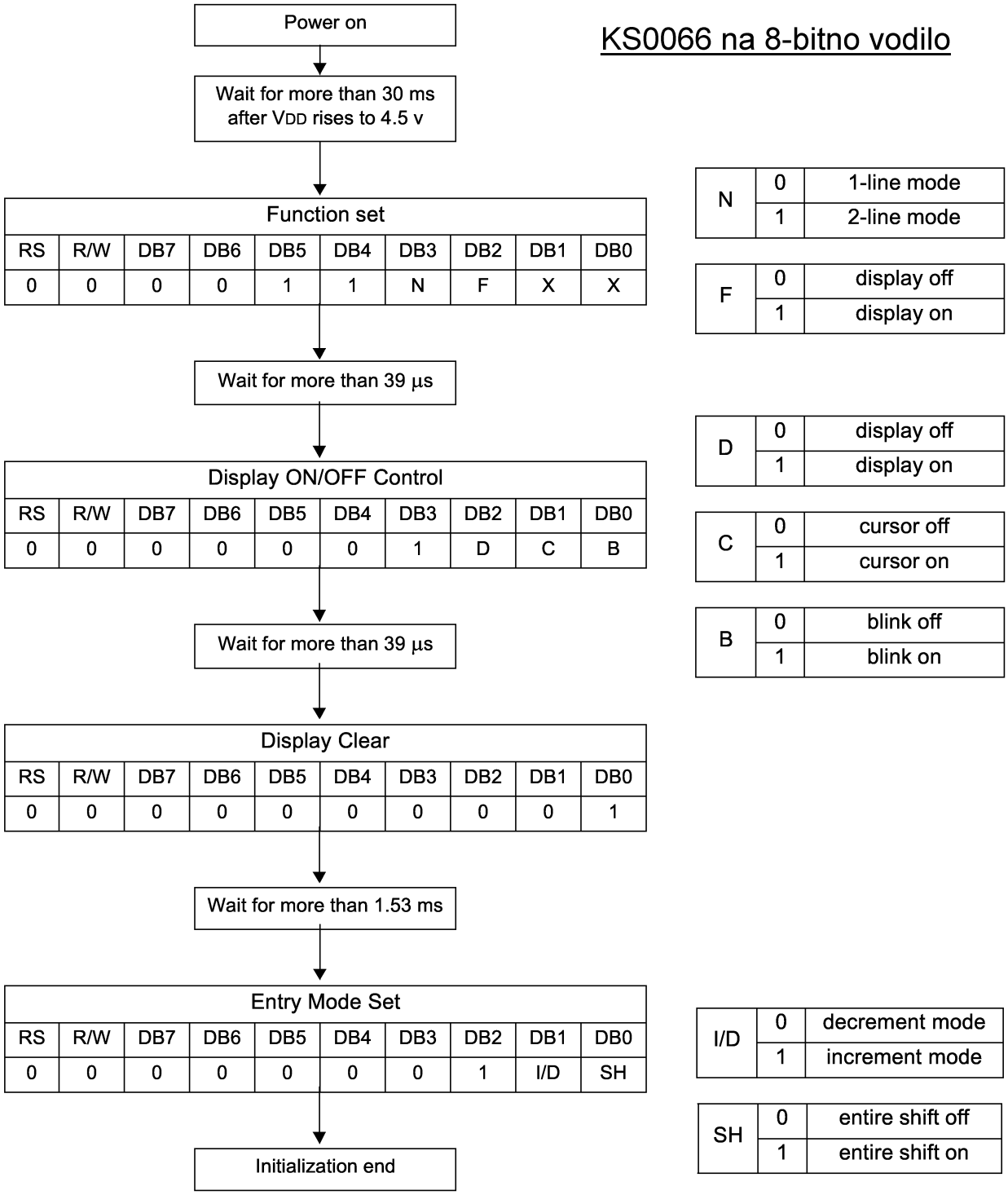
Priključka za osvetlitev LCD (backlight) 15 in 16 nista standardizirana. Njun položaj je lahko kjerkoli na modulu, tudi pred priključkom 1! Starejši moduli so imeli EL folijo, ki je zahtevala izmenično krmiljenje 100V frekvence okoli 400Hz. Vsi novejši moduli imajo svetleče diode. Na tiskanem vezju modula imamo največkrat mostičke, s katerimi lahko priključimo LED osvetlitev vzporedno napajanju logike modula ter izbiramo zaporedni upor, ki omejuje tok skozi svetleče diode.

Instruction	Instruction Code										Description	Execution time (fosc= 270 kHz)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM and set DDRAM address to "00H" from AC	1.53 ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.53 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction and enable the shift of entire display.	39 μs
Display ON/ OFF Control	0	0	0	0	0	0	1	D	C	B	Set display(D), cursor(C), and blinking of cursor(B) on/off control bit.	39 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift control bit, and the direction, without changing of DDRAM data.	39 μs
Function Set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-bit/4-bit), numbers of display line (N: 2-line/1-line) and, display font type (F:5×11dots/5×8 dots)	39 μs
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address counter.	39 μs
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter.	39 μs
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0 μs
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43 μs
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43 μs

Ukazi HD44780 oziroma KS0066

Vodilo med mikrokrmilnikom in LCD krmilnikom HD44780 oziroma

KS0066 vsebuje štiri ali osem podatkovnih vodov ter tri kontrolne vode E (Enable), R/W /Read/Write) in RS (Register Select). Neaktiven E je na logični ničli. Kratek impulz (približno 1µs) na logično enico na vhodu E označuje en cikel vpisa oziroma branja. Pri tem pomeni R/W=0 vpis v HD44780 in R/W=1 branje iz HD44780. RS=0 pomeni vpis ukaza oziroma branje stanja HD44780, RS=1 pa prenos podatkov med mikrokrmilnikom in HD44780.



KS0066 na 8-bitno vodilo

N	0	1-line mode
	1	2-line mode

F	0	display off
	1	display on

D	0	display off
	1	display on

C	0	cursor off
	1	cursor on

B	0	blink off
	1	blink on

I/D	0	decrement mode
	1	increment mode

SH	0	entire shift off
	1	entire shift on

Ob vklopu napajanja moramo posredovati LCD krmilniku HD44780 oziroma KS0066 vrsto ukazov, s katerimi nastavimo širino vodila (8 bitov),

število vrstic LCD modula (2 vrstici), način vnosa znakov na zaslon, prikaz utripajoče značke (če jo želimo) in seveda brisanje celotne vsebine zaslona. Če je karkoli narobe v povezavi oziroma programu mikrokrmilnika, HD44780 oziroma KS0066 brez teh ukazov ostane v načinu ena vrstica, torej nariše gornjo vrstico temno in spodnjo vrstico prazno.

Po nastavitvah ob vklopu je uporaba LCD krmilnika HD44780 oziroma KS0066 silno preprosta. Prikazovalniku preprosto pošiljamo ASCII znake z "Write Data to DDRAM". ASCII znaki v razponu od 32 do 127 so standardni. Znaki med 128 in 255 se lahko od enega krmilnika do drugega razlikujejo, ker obstaja več različic CGROM. LCD krmilnik samodejno izpisuje znake enega za drugim na zaslon.

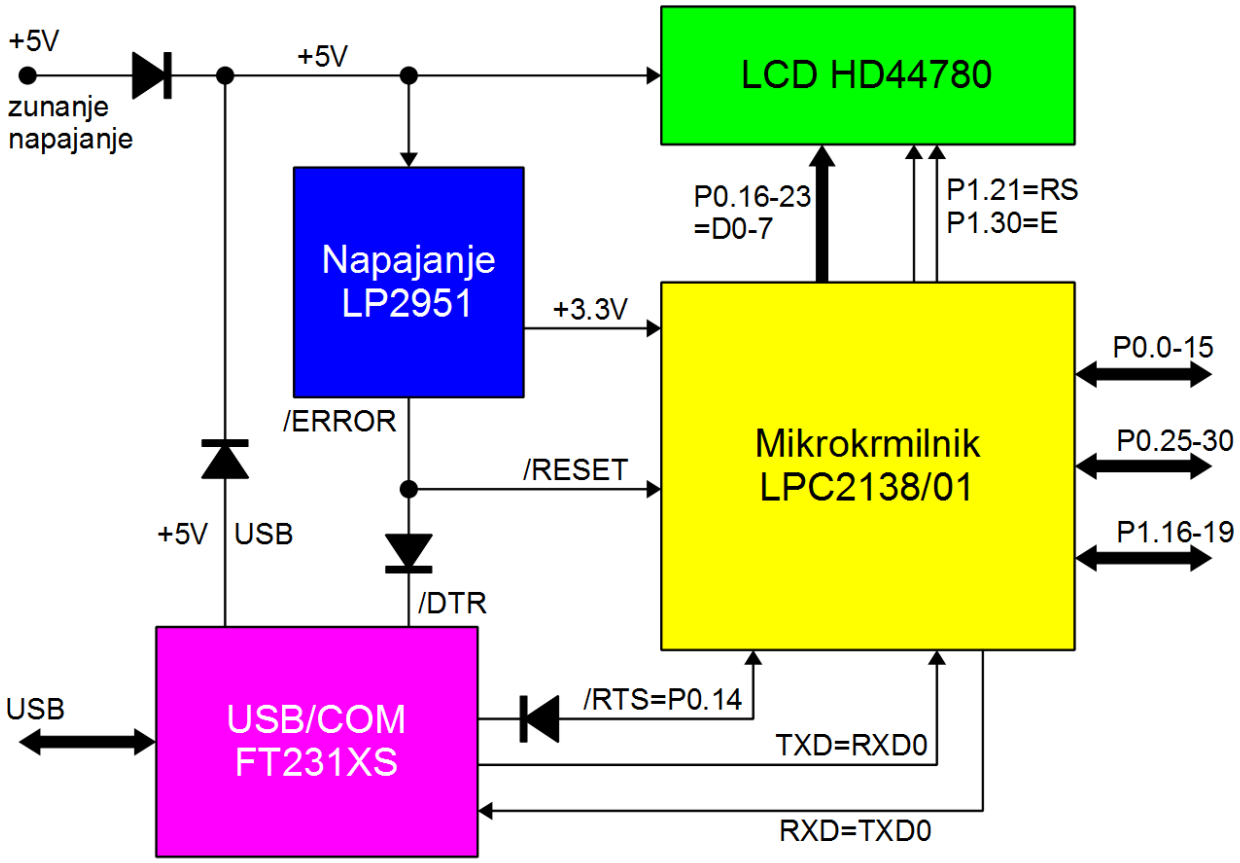
LCD krmilnik HD44780 oziroma KS0066 ne pozna ASCII kontrolnih znakov, pač pa razume vseh 256 kombinacij kot natisljiv znak. Znake od 0-7 lahko kot uporabniki nastavimo sami z "Write Data to CGRAM". Skupno torej lahko nastavimo 8 poljubnih znakov, na primer "jugo" črke, ikonice, razne vzorčke, simbole itd. V večini primerov potrebujemo le še ukaza "Set DDRAM Address" oziroma "Set CGRAM Address", da določimo, kam vpisujemo podatke.

Častitljiva starost HD44780 in njegovih številnih ponaredkov pomeni, da je takšen LCD krmilnik za najmanj dva velikostna razreda počasnejši od sodobnih 32-bitnih mikrokrmilnikov. LCD moduli s krmilniki družine HD44780 so danes še vedno izredno razširjeni, da je sploh težko najti drugačen alfanumeričen LCD. Mikrokrmilnik ARM bo torej treba primerno upočasniti za delo z LCD modulom. Običajno to ni ovira pri uporabi, saj časovno zahtevna opravila mikrokrmilnikov itak poganjamo v ozadju na prekinitvah.

8. Vezava mikrokrmilnika LPC2138/01

Po tolikšnem hvalisanju bo treba zvezati mikrokrmilnik ARM v nekaj uporabnega. Večina mikrokrmilnikov na razvojnih ploščah žal zaključi svojo življenjsko pot kot "Blinky.c", kar zagotovo ni naš cilj! Želimo samostojno enoto z mikrokrmilnikom in vso potrebno podporo na majhnem tiskanem vezju. Enota naj omogoča neposredno vgradnjo v večjo napravo. Enota naj omogoča programiranje s standardnim PC računalnikom, po možnosti prenosnikom, brez kakršnihkoli dodatnih pripomočkov.

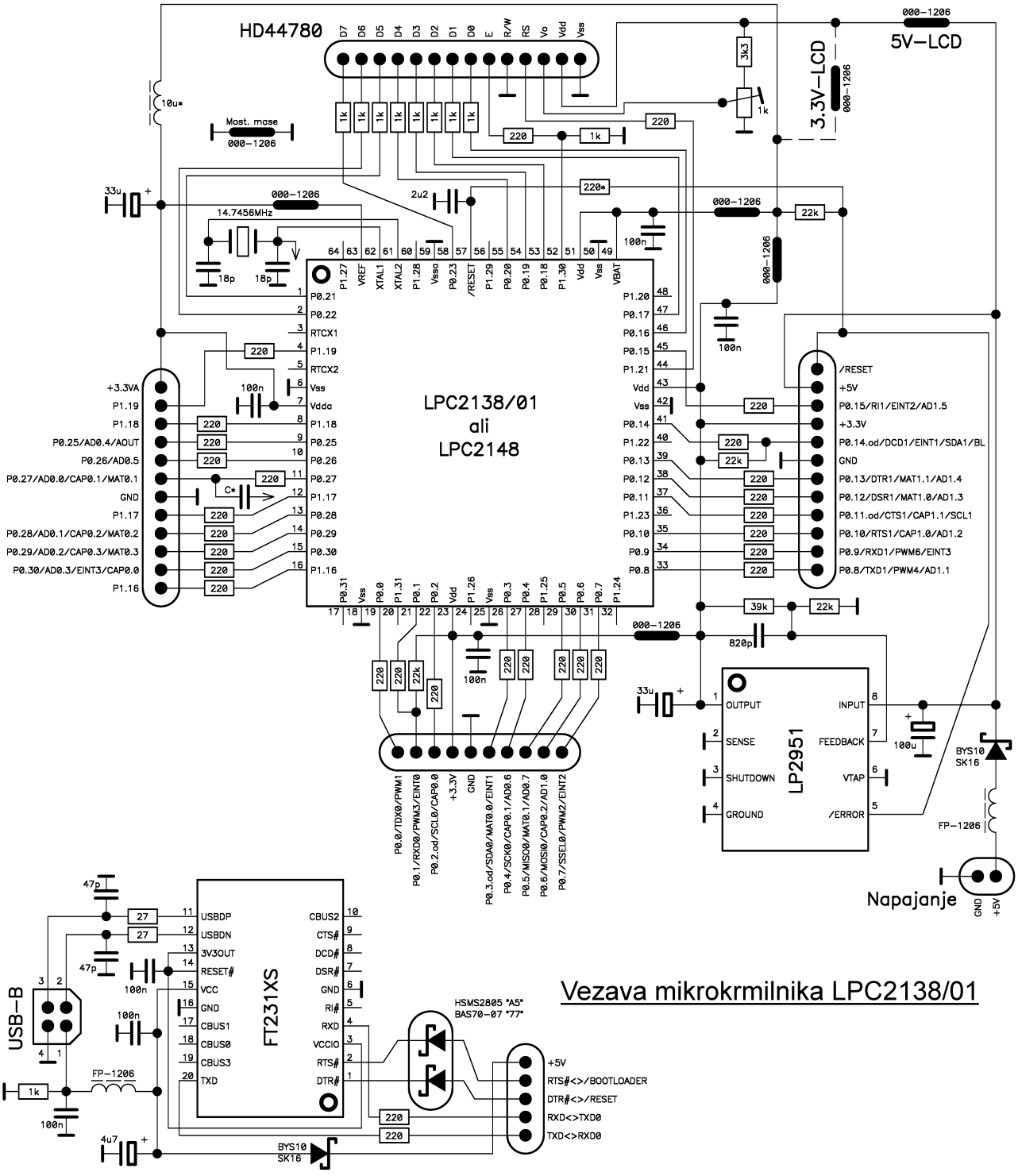
Predlagana vezava mikrokrmilnika vsebuje poleg LPC2318/01 še priključek za standardni LCD modul s HD44780. Napajalnik z LP2951 poskrbi za dodaten, zanesljiv "brown-out reset" preko signala /ERROR. USB COM port z vezjem FT231XS omogoča programiranje enote s katerimkoli PC računalnikom. Poleg programiranja lahko USB priključek uporabimo tudi za napajanje mikrokrmilnika in celotne naprave oziroma za krmiljenje naprave in zajem podatkov preko PC računalnika.



Vezava mikrokrmilnika LPC2138/01

Kako bomo torej vezali 47 vhodno/izhodnih priključkov LPC2138/01? Takoj se odrečemo popolnoma neuporabnim funkcijam, kot je razhroščevalnik

TRACE. Celo JTAG se danes opušča. Vse sodobne mikrokrmilnike ARM in MIPS programiramo preko vmesnika UART in vgrajenega Bootloaderja. Na priključkih P1.16-31 nam torej ostane samo še GPIO.

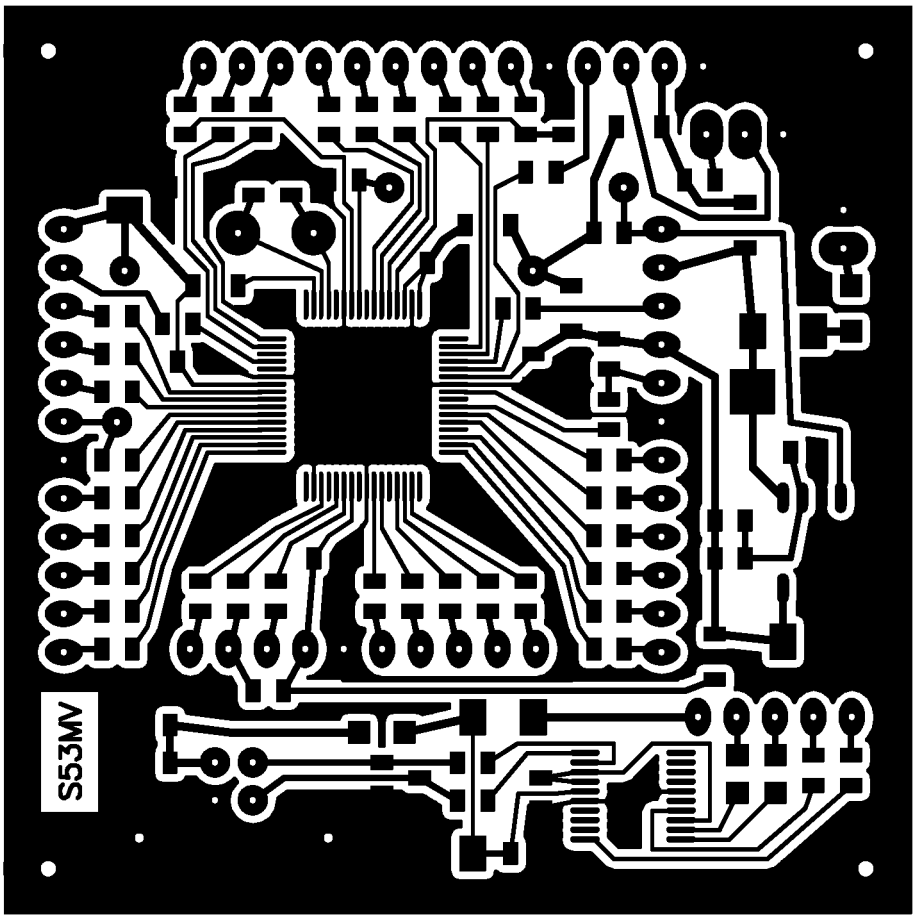


Vezava mikrokrmilnika LPC2138/01

Vse zahtevnejše vhodno/izhodne enote so dosegljive na priključkih P0.0-31. Podatkovno vodilo HD44780 uporablja P0.16-23, torej žrtvujemo enega od dveh SPI. Signal R/W za HD44780 je vezan na maso, saj običajno v HD44780 samo vpisujemo. Signala RS in E krmilimo s P1.21 in P1.30.

P1.16-19 so namenjeni štirim tipkam na maso, saj imajo vsi P1.16-31 vgrajen pull-up, ko jih programiramo kot vhode. P0.0-15 in P0.25-30 so napeljani na priključne letvice. Nekateri priključki so namenoma izpuščeni: P0.31, P1.20 in P1.26 so namenoma nepovezani, da vgrajeni pull-up zagotavlja pravilno stanje ob resetu LPC2138/01.

Na priključnih letvicah so poleg signalov vhodno/izhodnih enot napeljana napajanja +3.3V in +5V, signal /RESET in seveda masa. Vsi signali vhodno/izhodnih enot imajo vgrajene zaporedne dušilne upore. Dušilni upori 220Ω oziroma 1kΩ opravljajo najmanj tri pomembne naloge:
(1) dušijo zvonjenje priključnih vodov,
(2) omejujejo sevanje radijskih motenj in
(3) omejujejo tok kratkega stika, da ne uničimo mikrokrmilnika!

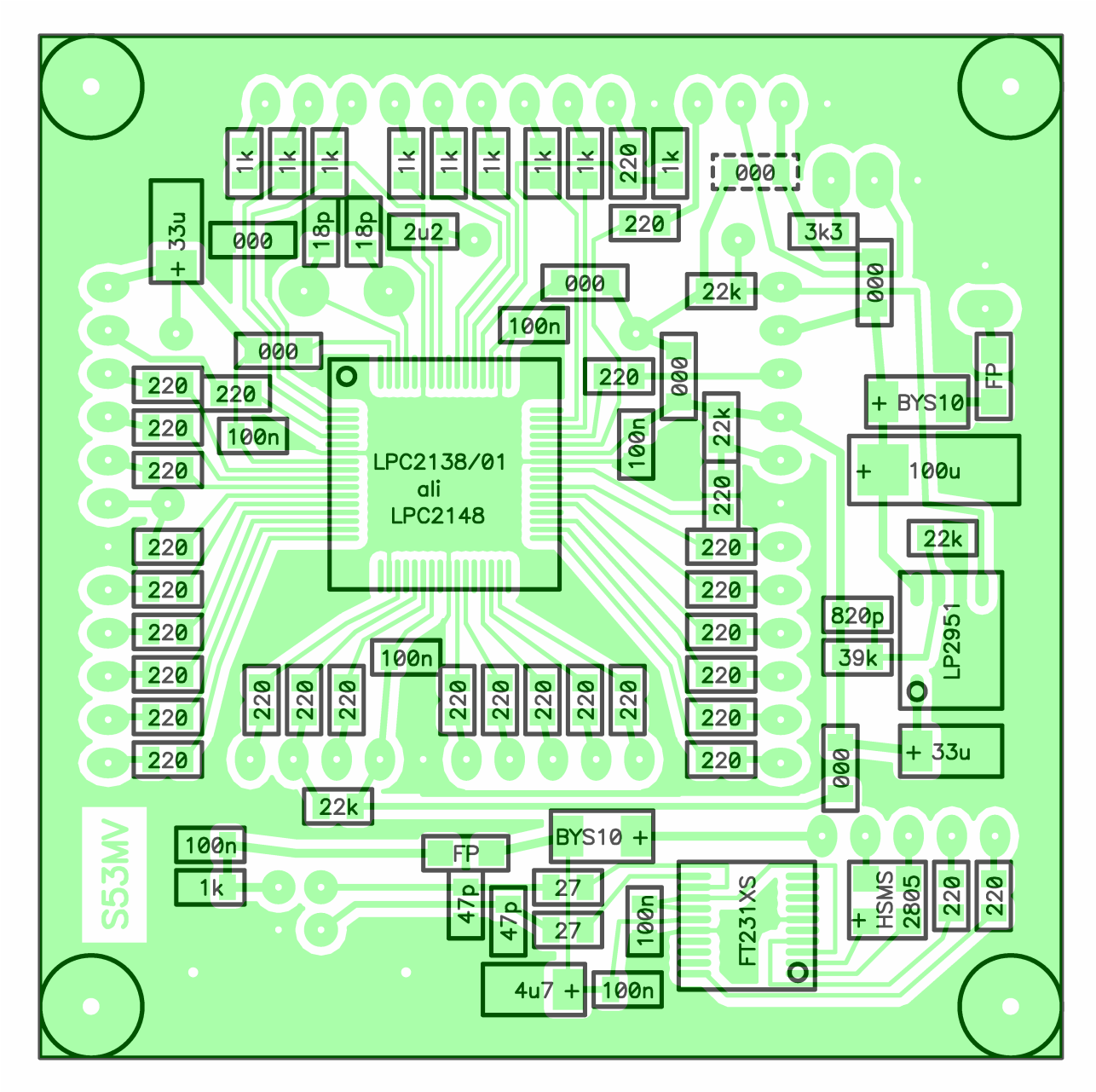


LPC2138/01 in pripadajoča vezja vgradimo na enostransko tiskano vezje z izmerami 60mm X 60mm. Enostransko tiskano vezje zahteva šest mostičkov SMD velikosti 1206: štirje na napajanju +3.3V, eden za dodatno povezavo mase in eden za izbiro napajanja LCD modula +5V ali +3.3V. Kot mostički za preskok povezav so uporabljeni tudi številni drugi SMD gradniki.

Poleg priključnih letvic je na tiskano vezje vgrajena tudi vtičnica USB-B.

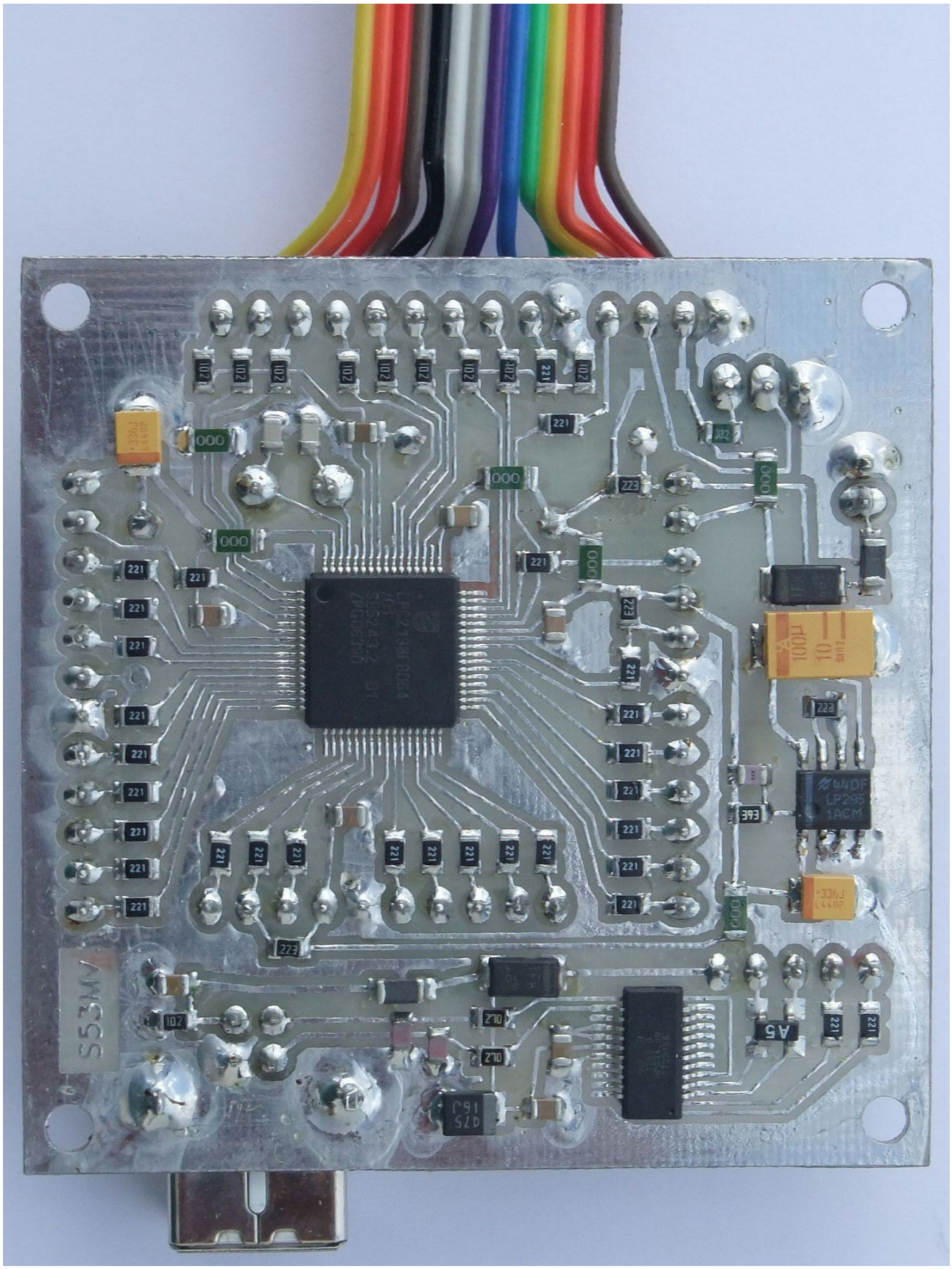
USB COM port FT231XS proizvajalca FTDI je izbran zaradi odlične podpore z gonilniki na vseh znanih operacijskih sistemih. Žal je programska podpora USB vmesnikov, vgrajenih neposredno v mikrokrmilnik, na primer v LPC2148 (enakovreden LPC2138/01 z dodatkom USB), izredno slaba!

Večina gradnikov enote mikrokrmilnika je SMD in je vgrajena na spodnji strani tiskanega vezja. Uspešno spajkanje SMD gradnikov zahteva kakovostno pocinjeno tiskano vezje pred dodajanjem gradnikov!

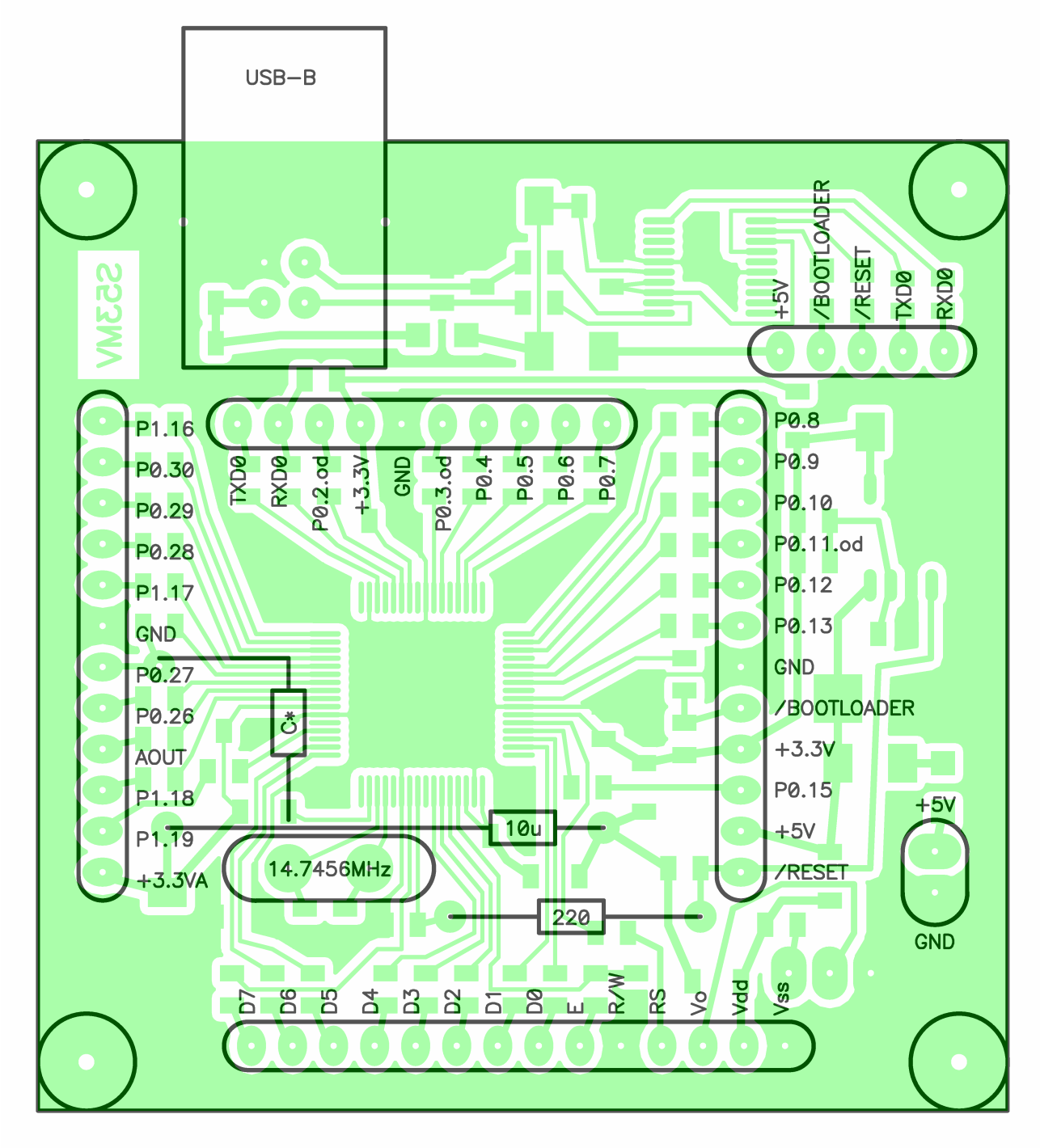


Gradnjo enote mikrokrmilnika obvezno začnemo s spajkanjem najzahtevnejšega gradnika, to je LPC2138/01 v ohišju QFP (poštna znamka) s priključki v rastru 0.5mm. LPC2138/01 natančno nastavimo na svoje mesto pod mikroskopom in najprej zacinimo samo dve vogalni nogici. Šele ko smo s položajem poštne znamke zadovoljni, nadaljujemo s spajkanjem ostalih

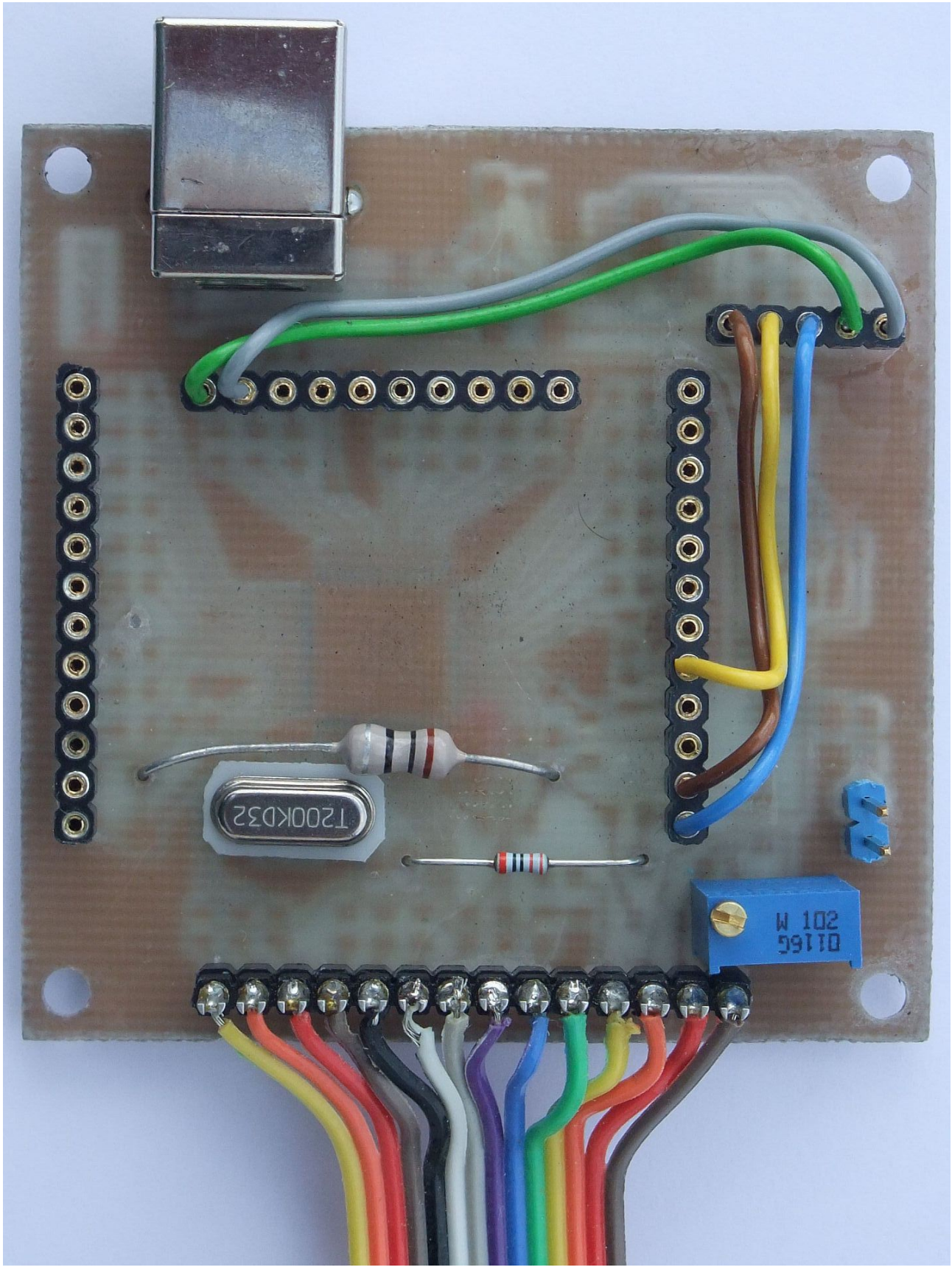
priključkov. Spajkamo z obilico redkega fluksa (stearin ipd) in z najmanjšo možno količino spajke. Kratke stike rešujemo prav s skrbno izbranim fluksom, ki povečuje površinsko napetost staljene spajke.



Po spajkanju nepotrebni stearin stalimo s segrevanjem celotnega tiskanega vezja in popivnamo s cunjico ali papirnato brisačo. Nato nadaljujemo s spajkanjem FT231XS, LP2951 in ostalih manj zahtevnih SMD gradnikov. Za spajkanje SMD uporov in kondenzatorjev zadošča že fluks v spajkalni žici, torej dodajanje in zamudno odstranjevanje stearina ne bo potrebno.



Ko smo spajkanje vseh SMD gradnikov uspešno zaključili in preverili pod mikroskopom, se lotimo spajkanja spajkanja maloštevilnih gradnikov na gornji strani tiskanega vezja:



Na gornji strani tiskanega vezja imamo še nekaj stopenj svobode glede na namen izdelanega mikrokrmilnika. Predlagani kristal za 14.7456MHz je preverjeno najboljša izbira za autobaud, z njim FlashMagic doseže najvišjo hitrost pri programiranju 230.4kbit/s. Če bomo z mikrokrmilnikom šteli

mikrosekunde, je smiselna izbira kristal za okroglo frekvenco 20MHz kot na gornji sliki, a z njim FlashMagic ne doseže več kot 38.4kbit/s. Končno, na priključek XTAL1 vezja LPC2138/01 lahko pripeljemo signal zunanjega kristalnega oscilatorja, na primer visoko-stabilen TCXO za frekvencmeter, preko primernega sklopnega kondenzatorja C*.

USB COM port FT231XS je popolnoma neodvisen od mikrokrmilnika in ima lastno priključno letvico s štirimi signali in napajanjem. Uporabo USB torej določajo žični mostički med letvico FT231XS in letvicami LPC2138/01. Namesto na UART0 lahko FT231XS vežemo tudi na UART1 mikrokrmilnika LPC2138/01 ali pa ga uporabimo povsem neodvisno.

Med razvojem programske opreme želimo celovito upravljati LPC2138/01 s FlashMagic in takrat vgradimo vseh pet žičnih mostičkov kot na gornji sliki. Ko smo razvoj programske opreme dokončali, odstranimo mostička za reset in Bootloader, da po nesreči ne prikličemo Bootloaderja in ne povozimo programa v FLASH. Končno, če naš izdelek predvideva uporabo P0.0 in P0.1 v drugačne namene, odstranimo tudi mostička RXD in TXD.

USB COM port FT231XS je načrtovan tako, da se njegovi priključki obnašajo kot odprte sponke, ko čip nima napajanja. Upor 1kΩ preko USB napajanja +5V poskrbi, da je brez USB kabla napajalna napetost FT231XS zares nič. Izhodi in vhodi čipa FT231XS brez napajanja tedaj ne motijo delovanja LPC2138/01.

Zaporedno z izhodoma /DTR in /RTS vezja FT231XS sta vezani dve schottky diodi HSMS2805 oziroma BAS70-07. Dioda na izhodu /DTR omogoča zakasnitev signala /RESET s kondenzatorjem 2.2µF in hkrati signal /RESET tudi iz izhoda /ERROR (odprti kolektor) napajalnika LP2951. Dioda na izhodu /RTS omogoča zagon Bootloaderja na priključku P0.14 LPC2138/01 in hkrati preprečuje kratek stik, ko P0.14 uporabljamo kot izhod open-drain. Za pravilen zagon uporabniškega programa v vsakem primeru poskrbi pull-up upor 22kΩ na priključku P0.14.

9. Uporaba mikrokrmilnika LPC2138/01

Pri praktični uporabi mikrokrmilnika bomo zagotovo potrebovali še kakšen podatek, ki ne more biti napisan v tem kratkem sestavku. Pri 8-bitnih mikrokrmilnikih (primer Microchip PIC) so bile stvari preproste: vse informacije smo dobili iz istega izvora, vključno s prevajalniki in programskimi orodji, običajno neposredno od proizvajalca mikrokrmilnika.

Pri 32-bitnih mikrokrmilnikih grejo stvari drugače. Procesorsko jedro načrtuje prvo podjetje, ki svojo intelektulano lastnino (IP) jedra licencira drugemu podjetju, proizvajalcu mikrokrmilnika. Slednji opremi procesorsko jedro z najrazličnejšimi vhodno/izhodnimi enotami. Prevajalnike razvija tretje, neodvisno podjetje. Končno četrto, spet neodvisno podjetje sestavi programsko orodje. Torej moramo črpati informacije iz najmanj štirih neodvisnih naslovov!

Procesorsko jedro ARM razvija podjetje ARM. Odgovore na vprašanja v zvezi s procesorskim jedrom LPC2138/01 dobimo v dveh dokumentih:
(1) Arhitektura: "ARM Architecture Reference Manual" in
(2) Jedro: "ARM7TDMI-S Technical Reference Manual".

Prvi dokument se ukvarja z naborom ukazov, drugi dokument pa se ukvarja z izvedbo jedra procesorja. Mikrokrmilnik LPC2138/01 vsebuje jedro ARM7TDMI-S, ki izvršuje nabor ukazov ARMv4T. Oznaka jedra in oznaka nabora ukazov torej nista ena in ista stvar! Isti nabor ukazov lahko izvršuje večje število različnih jeder, ki se med sabo lahko razlikujejo po številu taktov ure za določen ukaz oziroma po dolžini cevovoda, kar zelo zakomplicira računanje s programskim števcem PC oziroma R15!

Proizvajalec NXP mikrokrmilnika LPC2138/01 odgovarja edino za hardversko izvedbo procesorskega jedra. Odgovore o napajalni napetosti, frekvenci ure mikrokrmilnika, časovnih zakasnitvah oziroma vzdržljivosti pomnilnika FLASH dobimo v dokumentu "Data Sheet". Dosti pomembnejši in obsežnejši dokument za določen mikrokrmilnik proizvajalca NXP je "User Manual", ki podrobno opisuje vse vhodno/izhodne enote in sistemske funkcije ter njihovo programiranje. Končno vsak proizvajalec redno objavlja tudi "Errata", to je seznam ugotovljenih napak posameznih inačic čipa ter nasvetov za izogibanje napakam (workaround).

Natančen opis zbirnika ARMASM dobimo v "RealView Compilation Tools Assembler Guide". Isto podjetje je izdelalo tudi ARMC, ARMLINK in FROMELF, ki so prav tako opisani v pripadajočih dokumentih. Težje je dobiti dobre priročnike za zastonjkarske prevajalnike GNU, čeprav omogočajo marsikaj zanimivega, na primer "in-line" labele.

Končno potrebujemo programsko orodje, da naš program vpišemo v pomnilnik FLASH mikrokrmilnika. Orodje FlashMagic omogoča programiranje FLASH preko UART0, orodje H-JTAG pa preko kabla JTAG iz LPT porta PC računalnika. Oba sta preprosta, vgrajeni "help" zadošča. Priporočam uporabo starejšega FlashMagic, ker so novejše zastonjkarske inačice počasnejše!

Komplicirana orodja, kot so Keil, IAR, Eclipse in podobna, vsebujejo prevajalnike in programator. Pogosto zahtevajo zelo drago zunanjo USB škatlico. So silno zamudna za uporabo, zahtevajo študij debelih priročnikov oziroma obiskovanje tečajev. Kar je najhujše od vsega, isti program, napisan v eni od inačic takšnega medgalaktičnega orodja, na naslednji inačici istega orodja ne dela več!

V tem sestavku sem skušal opisati, kako se takšnim neučinkovitim orodjem in pripadajočim nevšečnostim izogniti. Še vedno ostane nekaj tisoč strani različnih priročnikov, ki jih ne morem stlačiti v ta kratki sestavek. Priročnike torej uporabljamo kot priročnike, se jih ne učimo na pamet, pač pa v njih poiščemo tisto, kar trenutno potrebujemo.

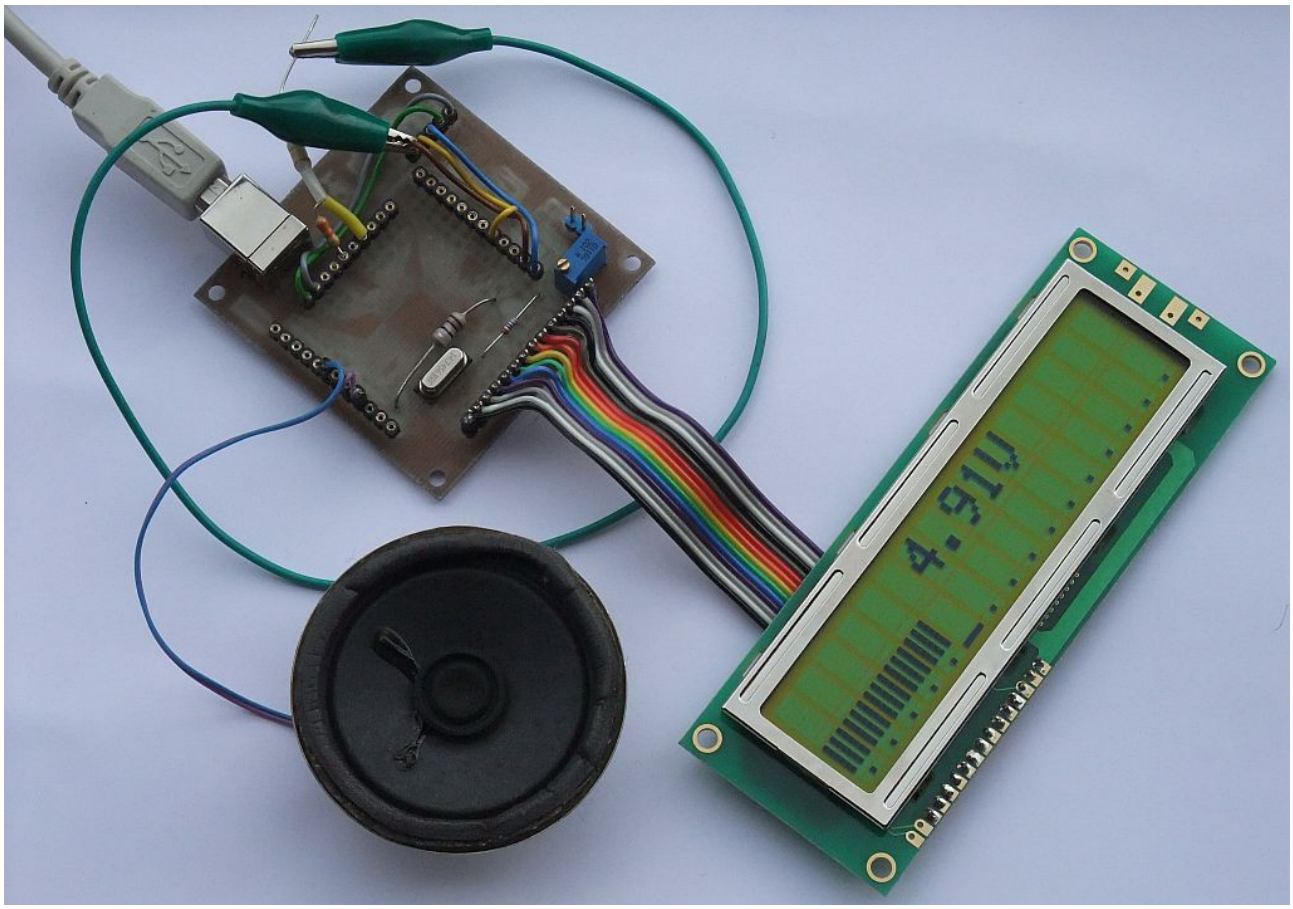
Niti izbira mikrokrmilnika ni preprosta. Proizvajalci vključno z NXP danes naravnost tekmujejo med sabo, kdo bo izdelal več različnih, med sabo nezdružljivih CORTEXov. Prve LPC2138 so izdelali že leta 2004 in kmalu odpravili nekaj začetniških pomanjkljivosti v LPC2138/01. Uspeh jedra ARM7TDMI-S se kaže v temu, da priročnike za LPC2138/01 pridno posodabljajo: najnovejši "User Manual" ima letnico 2012! Družini mikrokrmilnikov LPC213x in LPC214x (enak razpored priključkov) izgledata uspešnici, ki ju bomo verjetno srečevali še vrsto let.

Kako se lotimo pisanja programa za mikrokrmilnik družine LPC2xxx? Najprej moramo natančno določiti naloge mikrokrmilnika in preveriti, da so izvedljive z razpoložljivimi vhodno/izhodnimi enotami. LPC2138/01 na primer nima vmesnika za Ethernet in njegov pomnilnik je razmeroma majhen za internetni protokol. V tem primeru je smiselno uporabiti mikrokrmilnik z vgrajenim Ethernet MAC in večjim pomnilnikom iz družine LPC23xx.

Pred pisanjem programa si je smiselno ogledati kakšen dober zgled. Za opisano vezavo mikrokrmilnika sem pripravil dva zgleda: Test2138 in Voltmeter. Oba programa vsebujeta zglede, kako programirati posamezne funkcije mikrokrmilnika, kako sploh pisati program in kako ga dokumentirati s komentarji. Prvi preizkus komaj izdelanega mikrokrmilnika je seveda vzpostavitev povezave vgrajenega Bootloaderja s FlashMagic!

Test2138 je preprost program, ki izpisuje vse razpoložljive znake v prvo vrstico LCD in riše rastoči stolpec v drugo vrstico. Vse razpoložljive znake

hkrati pošilja na oddajnik UART0. Krmili tudi D/A pretvornik, da na izhodu P0.25/AOUT lahko pomerimo naraščajočo napetost. Vse ostale priključke P0.x programira kot izhode in izmenično preklaplja, P1.16-19 pa programira kot vhode z vgrajenim pull-up. Test2138 torej omogoča, da s preprostimi orodji preverimo delovanje celotne vezave mikrokrmilnika.



Voltmeter je primer zahtevnejšega programiranja. Uporablja zahtevnejše funkcije zbirnika ARM, vstavlja zunanje datoteke, uporablja prekinitve FIQ, časovnik TIMER0, UART0, A/D pretvornik in D/A pretvornik. Voltmeter najprej povpreči izmerjeno napetost, ki jo pripeljemo preko primernega uporovnega delilnika na P0.4/AD0.6. Rezultat izpiše številsko v prvi vrstici LCD in kot rastoči stolpec v drugi vrstici LCD. Številski rezultat pošilja tudi na oddajnik UART0.

Končno, program "Voltmeter" številski rezultat pretvori v govor tako, da uporablja vnaprej pripravljene zapise izgovorjenih številk. Vzorce zvočnih zapisov pošilja v taktu prekinitev 8kHz na D/A pretvornik, ki je že sam po sebi sposoben krmiliti slušalke oziroma manjši zvočnik. Končni rezultat je uporabna naprava, govoreči voltmeter, ki v praksi pride zelo prav takrat, ko morajo biti naše oči obrnjene drugam!

Kaj moramo obvezno storiti v našem programu, da bo mikrokrmilnik oživel? Prva naloga je nastaviti kazalec za sklad (Stack Pointer), ker bomo

zelo verjetno sklad tudi uporabljali. Če bo naš program uporabljal prekinitve, bo treba nastaviti tudi dodatne kazalce na sklad, to je dodatne (banked) R13: SP_IRQ oziroma SP_FIQ. Vse te nastavitve opravimo na registrih jedra ARM, torej moramo tu obvladati arhitekturo jedra!

Vse ostale nastavitve opravimo na registrih vhodno/izhodnih enot. V nadaljevanju se zato sklicujem na imena registrov, kot jih navaja "User Manual" LPC2138/01. Druga pomembna in obvezna naloga je izbira priključkov vhodno/izhodnih enot v registrih PINSEL. Legacy oziroma Fast GPIO izbiramo v registru SCS. Nato vključimo izhode z IODIR oziroma FIODIR, sicer iz mikrokrmilnika ne bo pricurljalo ven prav nič, saj so po resetu vsi priključki vhodi! Opisano zadošča, da izhodi lahko migajo, torej naš mikrokrmilnik pokaže znake življenja.

Dodatne sistemske nastavitve so popolnoma neobvezne, nam pa omogočajo bolje izkoristiti mikrokrmilnik. Večina sistemskih nastavitev je ob zagonu mikrokrmilnika izključena oziroma nastavljena na najbolj počasno delovanje. Na prvem mestu je tu MAM (Memory Accelerator Module). MAM je predpomnilnik med glavnim pomnilnikom FLASH in jedrom ARM.

V LPC2138/01 je 60MHz jedro ARM trikrat hitrejše od 20MHz pomnilnika FLASH. Po drugi strani FLASH omogoča vzporedno branje štirih ukazov. Primerno nastavljen predpomnilnik MAM tedaj zagotavlja, da izvajanje ukazov v jedru ARM ni prav nič ovirano s počasnim FLASH vse dokler ne izgubimo vsebine cevovoda s skokom ali razcepom v programu.

LPC2138/01 vsebuje fazno-sklenjeno zanko (Phase-Locked Loop) za množenje frekvence ure. Faktor množenja frekvence kristalnega oscilatorja najprej nastavimo v registrih PLL. Nato počakamo, da se PLL uklene. Končno preklopimo takt mikrokrmilnika iz neposrednega izhoda kristalnega oscilatorja na izhod PLL. Ob uporabi kristala za 14.7456MHz faktor množenja običajno nastavimo na 4, da jedro ARM teče z uro 58.9824MHz.

Za vhodno/izhodne enote običajno ni nujno, da so tako hitre kot jedro ARM. LPC2138/01 privzeto krmili vhodno/izhodne enote (PCLK) s četrtino frekvence ure jedra ARM (CCLK). Frekvenco ure vhodno/izhodnih enot lahko zvišamo na polovico ali celo enako frekvenci jedra z nastavitvijo v registru APBDIV. Višja frekvenca ure CMOS vezij pomeni tudi višjo porabo. Neuporabljene enote lahko tedaj odklopimo v registru PCONP.

Nastavitev nekaterih vhodno/izhodnih enot v LPC2138/01 je zelo preprosta. A/D pretvornik poženemo z enim samim vpisom registra ADCR in moramo le še počakati na zastavico v ADGDR, da je pretvorba končana. Žal A/D pretvornikom iz LPC2138/01 manjka vzorčevalno vezje (sample-and-hold) na vhodu. D/A pretvornik vključi že nastavitev PINSEL1.

Oba UARTa v LPC2138/01 sta popolnoma enaka 16C550, torej standardnemu COM portu PC računalnika. Tu moramo nastaviti hitrost (baudrate), število podatkovnih bitov, pariteto in število stop bitov. UART vrste 16C550 vsebuje tudi dva vmesna pomnilnika FIFO (First-In-First-Out) dolžine 16 byte v sprejemniku in v oddajniku, ki nam znatno olajšata pisanje programa.

Hitrejše delovanje UARTa zahteva uporabo prekinitev. Izvor prekinitev moramo najprej vključiti v izbrani vhodno/izhodni enoti. Nato moramo izvor prekinitev vključiti v VIC (Vectored Interrupt Controller) in izbrati, kakšno vrsto prekinitve želimo, IRQ ali FIQ. Končno je treba prekinitve IRQ oziroma FIQ sprostiti v CPSR. V samem prekinitvenem podprogramu moramo vsakokrat resetirati izvor prekinitev, sicer se ne bomo nikoli več vrnili v glavni program!

Večina mikrokrmilnikov vključno z LPC2138/01 vsebuje številne izvore prekinitev, ki jih v resnici ne bomo nikoli potrebovali. Prekinitve so zahtevne za programiranja, vnašajo večopravilnost in z njo možnost zelo zahrbtnih napak v programu. Prekinitve zato uporabljamo edino tam, kjer jih v resnici potrebujemo zaradi časovne stiske.

Ostane nam pisanje našega programa, da bomo rešili našo nalogo. Izpis števila v desetiški obliki zahteva deljenje z ostankom, česar ARM na ravni strojnega jezika ne zna. Tako je tudi prav, da mi sami izbiramo, kaj storiti v primeru deljenja z nič! Deljenje je hkrati osnovna računska operacija, ki jo potrebujemo v algoritmu za kvadratni koren.

Kvadratni koren nadalje potrebujemo za izračun kotnih funkcij polovičnih kotov, kar je osnova učinkovitega algoritma CORDIC za računanje kotnih funkcij in pripadajočih obratnih funkcij v dvojiškem svetu. V dvojiškem svetu je logaritem z osnovo dva silno preprost, potrebujemo le eno množenje (kvadriranje) za vsako dvojiško mesto za dvojiško vejico. Na neučinkovito Taylorjevo vrsto lahko v dvojiškem svetu mirno pozabimo...

Čemu potem višji programski jeziki? Dokler razpolagamo s kristalno čistim jedrom ARM, kot nam ga je podarila Sophie Wilson, je programiranje v zbirniku ARM čisti užitek!

Linux+Mac  linuxmac.zip

Ostalo  ostalo.zip

Tiskana vezja  pcb.zip

Podatkovni listi  podatki.zip

Zgledi v zbirniku  programi.zip

Windows zbirnik  winzbirnik.zip

Windows pekač  winpekac.zip

* * * * *