Univerzitet u Nišu
Elektronski fakultet
Katedra za elektroniku

KONVERTOR KODOVA REALIZOVAN SA MIKROKONTROLEROM PIC16F877-04

(Seminarski rad iz predmeta MIKROPROCESORSKI SISTEMI)

Student Perica Stojanović 10091/T

Maj 2008.

SADRŽAJ

UVODNA REČ

Pred vama je seminarski rad koji je realizovan od kraja aprila do sredine maja 2008. godine. Prema predlogu mentora prof. dr Mileta Stojčeva projektovan je jednostavan konvertor kodova sa mikrokontrolerom PIC16F877-04. Prvobitna namera je bila da se upotrebi mikrokontroler PIC16F84A koji ima 13 I/O pinova, a njegovo "proširenje" bi se ostvarilo preko pomeračkih registara. Međutim, od tog rešenja se brzo odustalo. Izabran je, dakle, PIC16F877-04 koji može da radi na frekvenciji od 4 MHz i poseduje 33 I/O pinova.

Osvrt na najpoznatije kodove dat je u prvom poglavlju seminarskog rada.

Drugo poglavlje sadrži veoma kratak opis mikrokontrolera PIC16F877-04, s obzirom da je veoma dosta podataka o ovom mikrokontroleru izloženo u seminarskim radovima Mikroprocesori i Mikroprocesorski sistemi. Zbog toga sam smatrao da je nepotrebno vršiti ponavljanje.

U trećem poglavlju je ukratko opisan softverski UART terminal, koji služi za komunikaciju mikrokontrolera PIC16F877-04 i PC računara.

Postupak programiranja mikrokontrolera izložen je u četvrtom poglavlju. Kao alati su korišćeni ALLPIC i IC-Prog.

U petom i šestom poglavlju je predstavljena implementacija konvertora kodova, kao i način njegove upotrebe.

Na kraju je dat spisak korišćene reference.

P1. PREGLED NAJPOZNATIJIH KODOVA

Čovek i računar najčešće ostvaruju interakciju preko tastature i miša. Pri tome, svaki pritisak nekog tastera/dirke, ili klikom na taster miša, ima za posledicu generisanje određene informacije, kao što su alfanaumerički znaci (karakteri) ili upravljački znaci (tabulacija, CR, LF, Enter itd). Pritisak tastera se mora konvertovati u neki pogodan oblik. Ovo se obično izvodi tako što se svakom tasteru dodeli specifičan kod. Jedan od tih kodova je ASCII kod.

Informacija može takođe da se unese u računar i preko spoljnih senzora (kakvi su termometri, merači pritiska), prekidača, raznih davača i drugih uređaja. Svaka od ovih informacija se mora konvertovati na određeni način u binarni oblik sa ciljem da se istom korektno manipuliše od strane računara. Sa druge strane, čoveku je prirodnija manipulacija sa decimalnim brojevima, a digitalnim sistemima, kakvi su računari, manipulacija sa ekvivalentnim binarnim brojevima. To znači da, i pored toga što računari za izvođenje internih operacija koriste binarne brojeve, komunikacija sa spoljnim svetom se ostvaruje u decimalnom brojnom sistemu. Sa ciljem da se pojednostavi komunikacija, svaki decimalni broj se predstavlja jedinstvenom sekvencom binarnih cifara. Ovakav način prezentacije poznat je kao binarno kodiranje. S obzirom da postoji deset decimalnih cifara (0, 1, 2, ... 9), za njihovu prezentaciju u binarnoj formi su potrebne četiri binarne cifre. Sa četiri cifre moguće su 16 kombinacija od kojih se koriste samo deset.

Inače, binarni kodovi se dele na težinske (tj. ponderisane) i netežinske (neponderisane).

Pored nabrojanih razloga, postoje i drugi koji se odnose na neminovnost uvođenja kodiranja. Između ostalih, značajniji su oni koji se tiču šifriranja podataka, kao i detekcije i korekcije grešaka koje se javljaju tokom prenosa podataka. U ovoj glavi ćemo opisati standardne tehnike kodiranja i ukazati na neke od metoda za korekciju i detekciju grešaka.

P1.1. Težinski kodovi

Kod težinskog koda svakoj binarnoj cifri se dodeljuje težina t. Zbir težina čija je vrednost 1 ekvivalentna je decimalnom broju predstavljen četvorobitnom kombinacijom. Drugim rečima, ako su di (i = 0, . . .,3) cifarske vrednosti a ti predstavljaju odgovarajuće težine, tada je decimalni ekvivalent 4-bitnog binarnog broja dat kao

$$d_3 \cdot t_3 + d_2 \cdot t_2 + d_1 \cdot t_1 + d_0 \cdot t_0.$$

Za slučaj kada je $t_0 = 2^0 = 1$, $t_1 = 2^1 = 2$, $t_2 = 2^2 = 4$ i $t_3 = 2^3 = 8$ dobija se kod koji se naziva BCD (*Binary-Coded-Decimal*) kod. U odnosu na standardnu binarnu prezentaciju kod BCD kodiranja svaku decimalnu cifru kodiramo binarno. Na primer, decimalni broj 14 kod standardne binarne prezentacije oblika je 1110 a odgovarajućem BCD kodu ima oblik:

0010	0101
(2)	(5)

Obe cifre, 2 i 5, su kodirane binarno. U Tabeli ispod prikazana je forma nekoliko težinskih kodova.

Decimalni broj	8421	2421	4221	7421
0	0000	0000	0000	0000
1	0001	0001	0001	0001
2	0010	0110	0010	0010
3	0011	0101	0011	0011
4	0100	0100	1000	0100
5	0101	1011	1111	0101
6	0110	1010	1100	0110
7	0111	1001	1101	1000
8	1000	1000	1110	1001
9	1001	1111	1111	1010

Neki kodovi imaju osobinu da se devetični komplement kodne reči neke cifre (tj. 9-N ako je N cifra) dobija kao jedinični komplement njegove kodne prezentacije. Na primer, kod koda 4221 decimalnom broju 7 ekvivalentna kodna reč je 1101, devetični komplement broju 7 je 2 (=9-7) za koga odgovarajuća kodna reč je 0010 koja je jedinični komplement od 1101. Kodove koji imaju ovu osobinu zovemo autokomplementarnim kodovima. Kod 8421 je takođe autokomplementarni kod. BCD kod je najčešće korišćeni težinski kod. Proces sabiranja BCD brojeva je isti kao i kod binarnih brojeva sve dok je decimalna suma 9 ili manja.

Decimalni	BCD
5	0101
+4	+0100
9	**1001**

Za slučaj kada suma premašuje decimalni broj 9, rezultat se podešava dodavanjem decimalnog broja 6. Tako na primer, sabiranjem 9 i 4 dobijamo

Decimalni		BCD
9		1001
+4		+0100
13		**1101**
$+(6)_{bin}$		+0110
	0001	**0011**

Drugi primer je sabiranje 9 i 7. I pored toga što rezultat čine dva važeća BCD broja, suma nije korektna. Ona se koriguje dodavanjem 6. Ovo je potrebno izvesti kada se javi prenos na mestu MS bita BCD broja na naredni viši BCD broj.

Decimalni		BCD
9		1001
+7		+0111
13		**1 0000**
+(6)bin		+0110
	0001	**0110**

Inače, Nad BCD brojevima mogu se obavljati i druge aritmetičke operacije.

P1.2. Netežinski kodovi

Kod netežinskih kodova ne postoji specifična težina koja se pridružuje ciframa kao što je to bio slučaj sa težinskim kodovima.

Kod "višak 3"

Tipičan netežinski kod je kod "višak 3". Dobija se dodavanjem 3 decimalnom broju, a zatim se vrši konverzija rezultata u 4-bitni binarni broj. U Tabeli ispod prikazana je prezentacija decimalnih cifara u kodu "višak 3".

Kod "višak 3" je autokomplemenataran i koristan je kod aritmetičkih operacija. Na primer, analizirajmo slučaj sabiranja dve decimalne cifre čija je suma veća od 9. Ako se u konkretnom primeru koristi BCD kod neće se generisati bit prenosa. No, kada se koristi kod "višak 3" javiće se prenos ka narednoj cifri veće težine. Rezultantna suma se, nakon toga, podešava dodavanjem 3.

Decimalni broj	"višak 3"
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Ilustracije radi, sabiranjem 6 i 8 dobijamo:

Decimalni		"višak 3"
9		1100
+4		+0111
13	**0001**	**0011**
$+(3)_{bin}$	+0011	+0011
	0100	**0110**

Ako kod koda "višak 3" saberemo dve decimalne cifre čija je suma 9 ili manja, sumu treba da podesimo tako što od nje oduzmemo 3. Konkretnije, pogledajmo primer sabiranja 3 i 5:

Decimalni	"višak 3"
3	0110
+5	+1000
13	**1110**
$-(3)_{bin}$	-0011
	1011

Kod operacije oduzimanja u kodu "višak 3", razlika se podešava kada joj se doda 3. Na primer:

Decimalni		"višak 3"
17	0100	1010
-11	0100	0100
7	**0000**	**0110**
$+(3)_{bin}$		+0011
		1001

Primeri sabiranja i oduzimanja brojeva u kodu "višak 3"

1. Odrediti zbir $A = 18345$ i $B = 9567$ u kodu *visak 3*.

$A\alpha$	=		0100	1011	0110	0111	1000
$B\alpha$	=		0011	1100	1000	1001	1010
P'	=	0	1	0	1	1	0
C'_α	=		1000	0111	1111	0001	0010
$K\alpha$	=		1101	0011	1101	0011	0011
$C\alpha$	=		0101	1010	1100	0100	0101

Dobijeni rezultat sabiranja je broj 27912.

2. Odrediti zbir $A = 99001$ i $B = 999$ u kodu *visak 3*. Pri sabiranju se dobija prekoračenje (označeno sa ***) zbog prenosa na cifarskom mestu najveće težine ($p'_5 = 1$) u prvoj fazi sabiranja.

$A\alpha$	=		1100	1100	0011	0011	0100
$B\alpha$	=		0011	0011	1100	1100	1100
P'	=	***1	1	1	1	1	0
C'_α	=		0000	0000	0000	0000	0000

3. Odrediti razliku $A = 1275$ i $B = 452$ u kodu *visak 3*. Rezultat $C = A - B = 823$ se dobija primenom sabiranja u potpunom komplementu:

$B\alpha$	=		0011	0011	0111	1000	0101
$[-B\alpha]_{nk}$	=		1100	1100	1000	0111	1010
$+1$							0001
$[-B\alpha]_{pk}$	=		1100	1100	1000	0111	1011

$C = A + [B]_{pk}$							
$A\alpha$	=		0011	0100	0101	1010	1000
$[-B\alpha]_{pk}$	=		1100	1100	1000	0111	1011
P'	=	1	1	0	1	1	0
C'_α	=		0000	0000	1110	0010	0011
$K\alpha$	=		0011	0011	1101	0011	0011
$C\alpha$	=		0011	0011	1011	0101	0110

U skladu sa pravilima za sabiranje brojeva u potpunom komplementu pojava prenosa $p'_5 = 1$ ne označava prekoračenje.

Ciklični kod

Drugi netežinski kod koji za predstavu decimalnih brojeva koristi netežinske binarne cifre je ciklični kod. Ciklične kodove karakteriše osobina da se uzastopne kodne reči razlikuju samo u jednoj bit poziciji. U Tabeli ispod prikazan je jedan tipičan ciklični kod.

Decimalni broj	Ciklični
0	0000
1	0001
2	0011
3	0010
4	0110
5	0100
6	1100
7	1110
8	1010
9	1000

Drugi karakteristični tip cikličnog koda je reflektivni kod poznat pod imenom Grejov (*Gray*) kod, a prikazan je u Tabeli ispod. Uočimo da su, sa izuzetkom MS bit pozicije, sve kolone "reflektivne" (simetrične) u odnosu na srednju tačku. Kod MS bit pozicije, u gornjoj polovini imamo sve nule a u donjoj sve jedinice.

Funkcija koja vrši preslikavanje nije jedinstvena – tako da postoji više Grejovih kodova dužine n. Ovde je prikazana jedna od najčešće korišćenih funkcija.

Decimalni broj se konvertuje u Grejov kod najpre konverzijom u binarni. Binarni broj se konvertuje u Grejov kod formiranjem sume po modulu 2 između tekuće cifre (počevši od LS cifre) i susedne cifre veće težine. Na primer, ako je binarna prezentacija decimalnog broja data u obliku

$$b_3 b_2 b_1 b_0$$

,tada se odgovarajuća kodna reč iz Grejovog koda, $G_3G_2G_1G_0$, određuje kao:

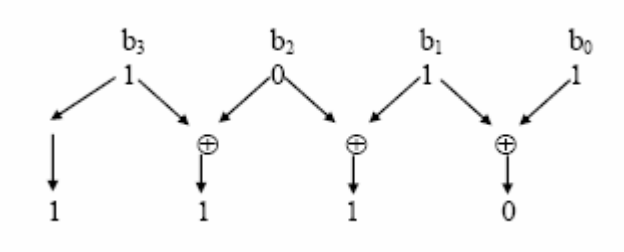
$$G_3 = b_3$$
$$G_2 = b_3 \oplus b_2$$
$$G_1 = b_2 \oplus b_1$$
$$G_0 = b_1 \oplus b_0$$

, gde simbol $\oplus$ ukazuje na ExOR operaciju, tj sabiranje po modulu 2.

Decimalni broj	Binarni	Grejov
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110

5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Ilustracije radi, analizirajmo konverziju decimalnog broja 11 u Grejov kod:



Grejova kodna reč za decimalan broj 11 je oblika:

G_3	G_2	G_1	G_0
1	1	1	0

Konverzija Grejove kodne reči u decimalni ekvivalent se izvodi sledećom sekvencom. Prvo se Grejov kod konvertuje u binarni a zatim se rezultujući binarni broj konvertuje u decimalni. Ilustracije radi, razmatrajmo konverziju broja 1110 iz Grejovog koda u decimalni.

$b_3 = G_3=1$	
$G_2= b_3 \oplus b_2=1=1 \oplus b_2$	,jer je $b_3 =1$
$b_2=0$	
$G_1= b_2 \oplus b_1=1=0 \oplus b_1$	,jer je $b_2 =0$
$b_1=1$	
$G_0= b_1 \oplus b_0=0=1 \oplus b_0$	,jer je $b_1 =1$
$b_0=1$	

Prema tome, binarni ekvivalent Grejove kodne reči 1110 je 1011 – što je ekvivalentno decimalnom broju 11.

P1.3. ASCII kod

ASCII je skraćenica od engleskih reči: **American Standard Code for Information Interchange**, što u prevodu znači: **američki standardni kod za razmenu informacija**. Izgovara se aski i predstavlja numeričku reprezentaciju karaktera. Ti karakteri mogu biti @ ?) = / & % $ a b c d ... G H I J K ...

Orginalna veličina podataka za predstavljanje ASCII karaktera je 7 bita. Iz toga sledi da je moguće kodovati 128 različita karaktera. Danas je u upotrebi prošireni ASCII kod koji se sastoji od 8 bita podataka, a sve češće se koristi unicode standard koji koristi više bajtova za kodovanje većeg broja simbola.

ASCII je zasnovan je na engleskom alfabetu i kontrolnim karakterima, te se stoga i sastoji iz dve grupe karaktera: printabilnih i kontrolnih karaktera.

Kontrolni karakteri

Naziv su dobili po ulozi kontrolisanja uređaja za prikaz informacija. Potreba se javila prilikom komunikacije dva uređaja da se na jednostavan način prenesu komande za formatiranje teksta i komande za kontrolu komunikacije. Korišćenje ovih karaktera u svom izvornom obliku sve više zastareva i manje se koristi.

Postoje 33 kontrolna karaktera i njihov spisak dat je u tabeli ispod. Opis karaktera je ostavljen na engleskom jeziku iz raloga ne postojanja adekvatnih standarizovanih izraza na srpskom jeziku za sve pojmove.

Bin.	Dec.	Hex.	Skraćenica	Značenje
000 0000	0	00	NUL	Null character
000 0001	1	01	SOH	Start of Header
000 0010	2	02	STX	Start of Text
000 0011	3	03	ETX	End of Text
000 0100	4	04	EOT	End of Transmission
000 0101	5	05	ENQ	Enquiry
000 0110	6	06	ACK	Acknowledgment
000 0111	7	07	BEL	Bell
000 1000	8	08	BS	Backspace
000 1001	9	09	HT	Horizontal Tab
000 1010	10	0A	LF	Line Feed
000 1011	11	0B	VT	Vertical Tab
000 1100	12	0C	FF	Form Feed
000 1101	13	0D	CR	Carriage Return
000 1110	14	0E	SO	Shift Out
000 1111	15	0F	SI	Shift In
001 0000	16	10	DLE	Data Link Escape
001 0001	17	11	DC1	Device Control 1 (ili XON)
001 0010	18	12	DC2	Device Control 2

001 0011	19	13	DC3	Device Control 3 (ili XOFF)
001 0100	20	14	DC4	Device Control 4
001 0101	21	15	NAK	Negative Acknowledgment
001 0110	22	16	SYN	Synchronus Idle
001 0111	23	17	ETB	End of Transmision Block
001 1000	24	18	CAN	Cancel
001 1001	25	19	EM	End of Medium
001 1010	26	1A	SUB	Substitute
001 1011	27	1B	ESC	Escape
001 1100	28	1C	FS	File separator
001 1101	29	1D	GS	Group separator
001 1110	30	1E	RS	Record separator
001 1111	31	1F	US	Unit Separator
111 1111	127	7F	DEL	Delete

Printabilni karakteri

Printabilni karakteri obuhvataju slova engleske abecede, cifre, znakove interpunkcije, prazninu između slova i nekoliko mešovitih simbola. Svi karakteri dati su u narednoj tabeli.

Bin.	Dec.	Hex.	Simbol
010 0000	32	20	(RAZMAK)
010 0001	33	21	!
010 0010	34	22	"
010 0011	35	23	#
010 0100	36	24	$
010 0101	37	25	%
010 0110	38	26	&
010 0111	39	27	'
010 1000	40	28	(
010 1001	41	29	)
010 1010	42	2A	*
010 1011	43	2B	+
010 1100	44	2C	,
010 1101	45	2D	-
010 1110	46	2E	.
010 1111	47	2F	/

011 0000	48	30	0
011 0001	49	31	1
011 0010	50	32	2
011 0011	51	33	3
011 0100	52	34	4
011 0101	53	35	5
011 0110	54	36	6
011 0111	55	37	7
011 1000	56	38	8
011 1001	57	39	9
011 1010	58	3A	:
011 1011	59	3B	;
011 1100	60	3C	<
011 1101	61	3D	=
011 1110	62	3E	>
011 1111	63	3F	?
100 0000	64	40	@
100 0001	65	41	A
100 0010	66	42	B
100 0011	67	43	C
100 0100	68	44	D
100 0101	69	45	E
100 0110	70	46	F
100 0111	71	47	G
100 1000	72	48	H
100 1001	73	49	I
100 1010	74	4A	J
100 1011	75	4B	K
100 1100	76	4C	L
100 1101	77	4D	M
100 1110	78	4E	N
100 1111	79	4F	O
101 0000	80	50	P
101 0001	81	51	Q
101 0010	82	52	R
101 0011	83	53	S
101 0100	84	54	T
101 0101	85	55	U

101 0110	86	56	V
101 0111	87	57	W
101 1000	88	58	X
101 1001	89	59	Y
101 1010	90	5A	Z
101 1011	91	5B	[
101 1100	92	5C	\
101 1101	93	5D	]
101 1110	94	5E	^
101 1111	95	5F	_
110 0000	96	60	`
110 0001	97	61	a
110 0010	98	62	b
110 0011	99	63	c
110 0100	100	64	d
110 0101	101	65	e
110 0110	102	66	f
110 0111	103	67	g
110 1000	104	68	h
110 1001	105	69	i
110 1010	106	6A	j
110 1011	107	6B	k
110 1100	108	6C	l
110 1101	109	6D	m
110 1110	110	6E	n
110 1111	111	6F	o
111 0000	112	70	p
111 0001	113	71	q
111 0010	114	72	r
111 0011	115	73	s
111 0100	116	74	t
111 0101	117	75	u
111 0110	118	76	v
111 0111	119	77	w
111 1000	120	78	x
111 1001	121	79	y
111 1010	122	7A	z
111 1011	123	7B	{

111 1100	124	7C	|
111 1101	125	7D	}
111 1110	126	7E	~

Strukturne karakteristike ASCII-ja

Prilikom kreiranja ovog standarda vođeno je računa o strukturi samog rasporeda karaktera i kako se pogodnijom strukturom mogu ubrzati određene operacije. Ovde ćemo predstaviti neke od tih pogodnosti koje vam mogu zatrebati u svakodnevnom programiranju.

Decimalne cifre su u ovom standardu predstavljene sa prefiksom 011, ovo omogućuje jednostavno konvertovanje cifara iz BCD u ASCII i obrnuto.

Razlika imađu malog i velikog slova je u jednom bitu. Na ovaj način omogućena je brza konverziju iz malih u velika slova i/ili obrnuto. Ovo je vrlo bitna činjenica, jer se na taj način pospešuje pretraživanje koje je imuno na veličinu slova.

P2. MIKROKONTROLER PIC16F877-04

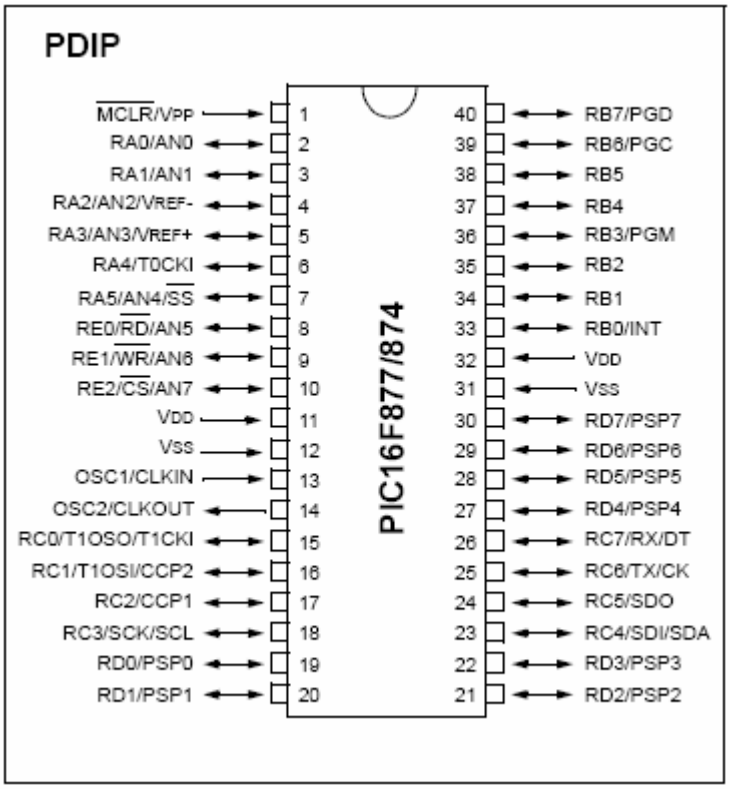
O ovom mikrokontroleru dati su široki opisi u seminarskim radovima iz predmeta Mikroprocesori i Mikroprocesorski sistemi – tako da ćemo se ovde osvrnuti isključivo na njegove najbitnije karakteristike. Za čitaoce koji žele da prodube znanja o mikrokontroleru PIC16F877, predlažemo da posete sajt *http://es.elfak.ni.ac.yu/students.html*.

PIC16F877 se na tržištu pojavio 1998. godine, a u međuvremenu se pojavila njegova naprednija verzija – PIC16F877A. Zasnovan je na RISC arhitekturi, a vreme izvršavanja svake instrukcije (izuzev instrukcija grananja) iznosi 4 taktna intervala. Tako na primer – ako je frekvencija oscilatora mikrokontrolera 4 MHz, instrukcija se izvršava za 1 μs. Ukoliko je frekvencija oscilatora 20 MHz, vreme izvršavanja instrukcije iznosi 250 ns.

Mikrokontroler PIC16F877 podržava tehniku prekida (eng. *interrupts*). Postoji ukupno 14 izvora prekida (spoljašnjih i unutrašnjih). Prekidi ne poseduju sopstveni interapt-vektor, već postoji jedinstvena adresa (0x0004) od koje se nastavlja izvršavanje programa kada se dogodi bilo koji prekid.

Postoji mogućnost da se izabere jedan od četiri tipa oscilatora, mogućnost upotrebe *Power-up* (PWRT) i *Oscillator Start-up* (OST) tajmera, kao i eventualno korišćenje *Power-on* (POR) i/ili *Brown-out* (BOR) reseta. Upotreba *Watchdog* tajmera (WDT) sprečava ulazak programa u «mrtve petlje», čime se povećava pouzdanost mikrokontrolera. Zaštitu kôda od neželjenog čitanja pruža opcija *code protection.*

Microchip-ovi PIC mikrokontroleri imaju prednost u odnosu na konkurentske mikrokontrolere relativno visokom strujom koju može propustiti kroz svaki I/O pin (25 mA). Isto tako, ova familija mikrokontrolera poseduje veoma širok opseg napona napajanja koji se proteže od 2,0 V do 5,5 V.

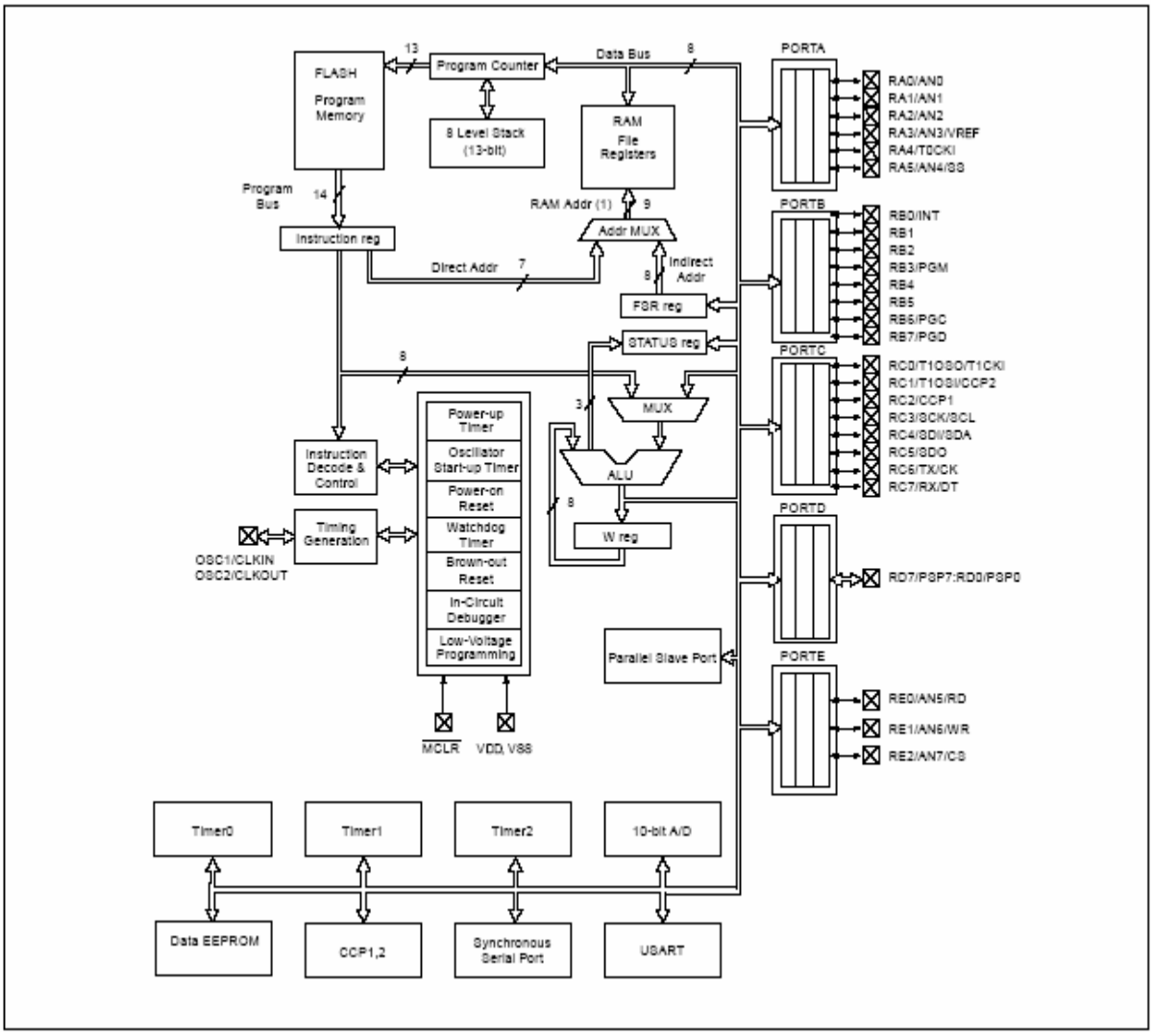


Slika 2-1. Raspored pinova

P2.1. Arhitektura mikrokontrolera

U arhitekturi mikrokontrolera izdvajaju se sledeći gradivni blokovi:

- Programska memorija (tipa *flash*) veličine 8 kword
- Operativna memorija (tipa RAM) – 368 bajtova
- Aritmetičko-logička jedinica (ALU)
- Akumulator (*Working Register*)
- Hardverski magacin sa 8 nivoa
- Memorija podataka (tipa EEPROM) – 256 bajtova
- Portovi
- Tajmeri
- ADC (A/D konvertor)
- CCP modul
- USART modul
- SSP modul itd.



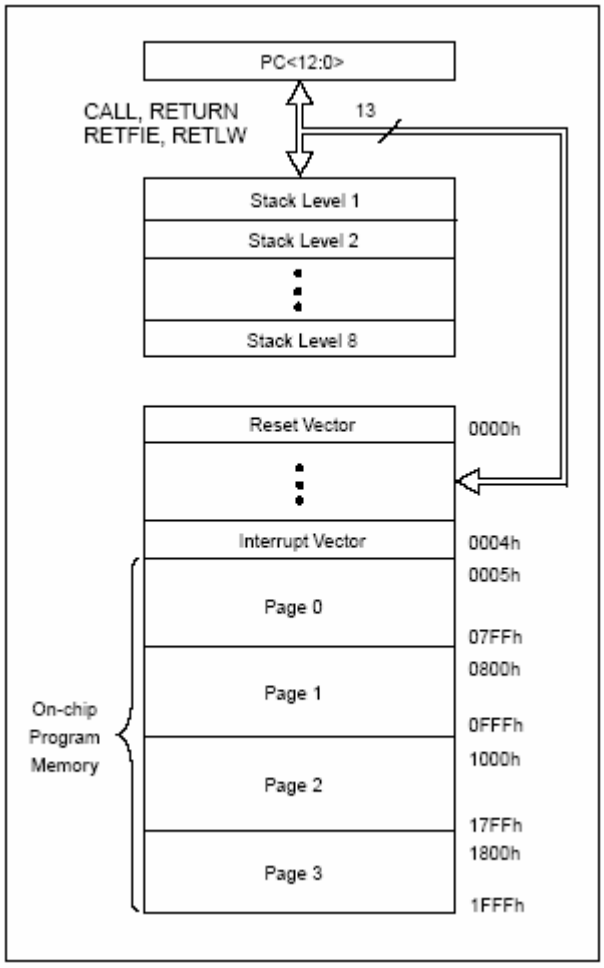
Slika 2-2. Arhitektura mikrokontrolera

P2.2. Memorija mikrokontrolera

Strukturu memorije kod PIC mikrokontrolera čine tri posebna bloka:

- Programska memorija
- Memorija podataka
- EEPROM memorija podataka.

Zasebno od navedenih memorijskih blokova postoji magacin (*stack*), koja se sastoji od osam 13-bitnih registara – što znači da se u njega može staviti osam različitih adresa; pokušaj unošenja devete izaziva brisanje prve itd. Prilikom izvršenja instrukcije CALL ili prilikom poziva prekida mikrokontrolera, adresa sledeće instrukcije se stavlja na magacin. Ponovno vraćanje starog sadržaja programskog brojača izaziva izvršenje instrukcije RETURN, RETFIE ili RETLW.

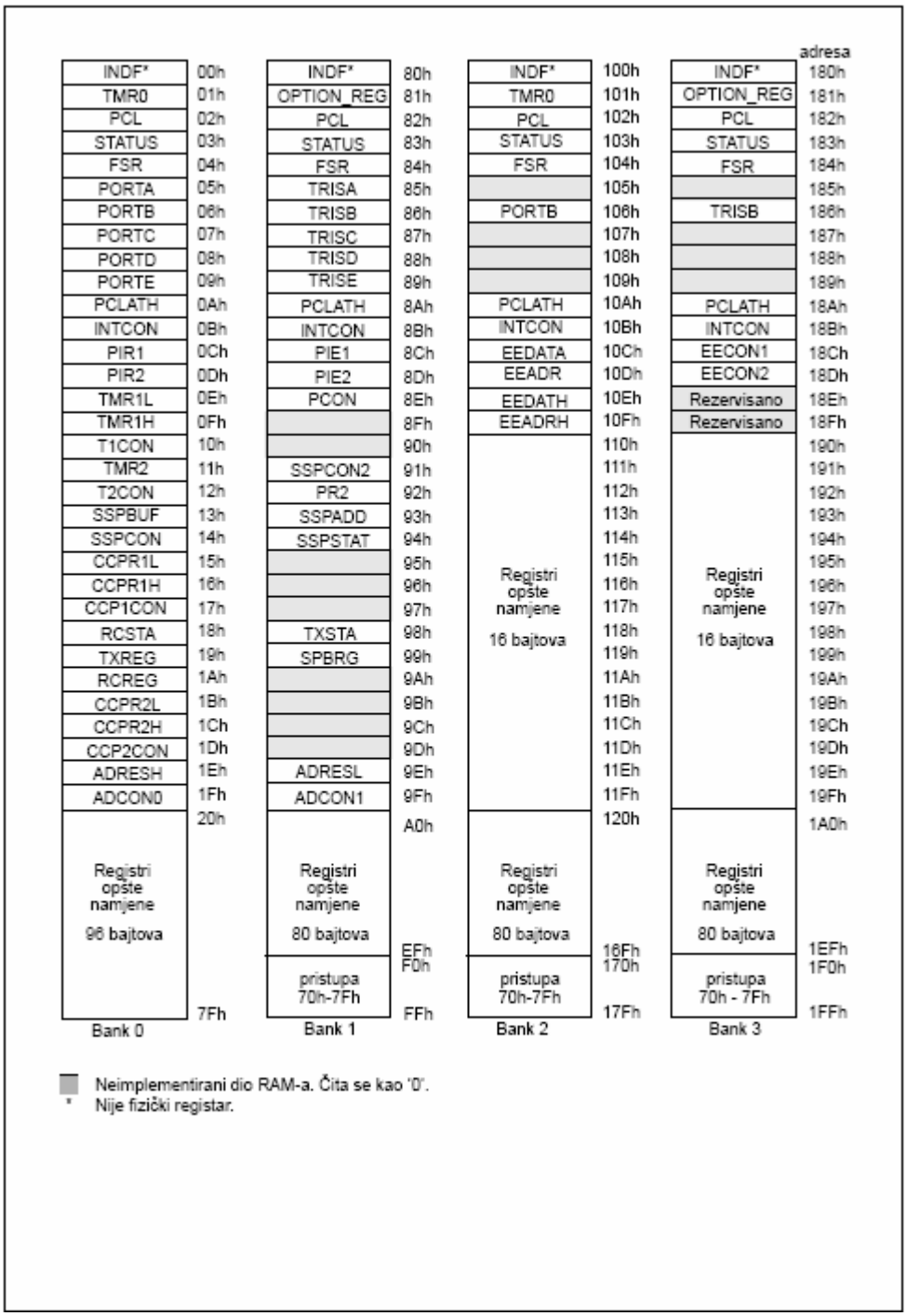


Slika 2-3. Mapa programske memorije

Organizacija programske memorije

Mikrokontroler PIC16F877 poseduje 13-bitni programski brojač (PC) koji je može da adresira memorijski prostor od 8k četrnaestobitnih programskih reči. Reset vektor je 0x0000 i od njega počinje

izvršavanje programa. Interapt vektor je 0x0004. Mapa programske memorije i magacin prikazani su dijagramom na *Slici 2-3*.



Slika 2-4. Registarska mapa mikrokontrolera

Organizacija memorije podataka

Ovaj tip memorije je organizovan u više celina, tzv. banki (eng. *banks*) – a inače se sastoji od registara opšte namene (*General Purpose Registers*) i registara specijalne funkcije (*Special Function*

Registers). U jednom od specijalnih registara, tzv. STATUS registru postoje dva bita RP1 i RP0 koji služe za biranje željene banke podataka.

Mapa registara procesora PIC16F877 ilustrovana je na *Slici 2-4*. Nekoliko specijalnih su usko povezani sa funkcionisanjem CPU. Ostali registri su povezani za periferne module i služe njihovom upravljanju i kontroli statusa.

Interni EEPROM za podatke

Ova memorija sadrži 256 bajtova. Ukoliko je potrebno neke podatke sačuvati i po ukidanju napajanja mikrokontrolera, treba ih prethodno zapisati u interni EEPROM.

P2.3. I/O portovi

Ulazno/izlazni portovi mikrokontrolera predstavljaju njegovu vezu sa okruženjem. PIC16F877 poseduje pet portova i oni su označeni slovima od A do E. Inače, nejednake su širine:

- Port A (6-pinski),
- Portovi B,C i D (8-pinski),
- Port E (3-pinski).

Pojedini pinovi I/O porta su povezani sa nekom perifernom funkcijom mikrokontrolera. Jedino ukoliko jeo dgovarajuća periferna jedinica neupotrebljena, moguće je njen pin koristiti kao I/O liniju opšte namene. Konfiguracija smera portova vrši se upisom konfiguracionog bajta u pripadajući TRIS registar po pravilu da nula čini pin izlaznim, a jedinica ulaznim. Npr. sa TRISA = 111111 definišemo sve pinove Porta A kao ulazne; dok sa TRISC = 11110000 više pinove Porta C (C7-C4), a niže pinove (C3-C0) kao izlazne.

Svaki port poseduje odgovarajući registar (PORTX) preko kojeg se programski pristupa I/O pinovima. Upis u neki od tih registara iniciraće upis u leč tog porta, a njegovo čitanje rezultiraće čitanjem logičkih stanja direktno sa pinova. Sve instrukcije upisivanja su tzv. *read-modify-write* instrukcije. To znači da se pri upisu u port prvo očitaju stanja pinova, izvrši modifikacija, a potom ispravljena vrednost smesti u leč porta.

Inače, nema velike razlike u električnoj konstrukciji navedenih pet portova. Port B se od ostalih razlikuje interesantnom opcijom koju nude četiri njegova viša bita. Ukoliko se setuje bit RBIE u registru INTCON, svaka promena stanja na ovim pinovima izazvaće prekid kod mikrokontrolera.

P2.4. Tajmeri

Ovi moduli se koriste za merenje vremena i brojanje eksternih događaja. U arhitekturu mikrokontrolera su ugrađena tri tajmerska modula (TMR 0,1,2). Svaki od njih ima svoje specifičnosti.

Tajmer0 je jednostavni 8-bitni brojač koji generiše interapt pri prelasku sa 0xFF na 0x00 (*overflow*). Poseduju ga svi niži PICmicro™ procesori i ovde je zadržana kompatibilnost s njima. Izvor takta za Tajmer0 može biti bilo interni sistemski sat (Fosc/4), bilo spoljašnji generator takta spojen na pin RA4/T0CKI. Moguće je podesiti da se brojač inkrementira na svaku rastuću ili opadajuću ivicu spoljašnjeg signala.

U kombinaciji sa Tajmerom0 može da se koristi programabilni preskaler (delilac frekvencije) sa odnosima deljenja od 1:2 do 1:256. Ukoliko deljenje nije potrebno, preskaler se dodeljuje *watchdog* tajmeru.

Kada je preskaler u upotrebi maksimalna frekvencija eksternog izvora iznosi 50 MHz, što je veće od maksimalne frekvencije samog mikrokontrolera.

Tajmer1 je 16-bitni i osposobljen je da radi kao brojač/merač vremena. Poseduje tri izvora takta: sistemski sat (Fosc/4), spoljašnji takt ili spoljašnji kristal.

Brojač eksternih događaja može da se sinhronizuje sa internim oscilatorom, a postoji i asinhroni način rada koji omogućuje da se brojač inkrementira i u *sleep* režimu. Preskaliranje je upotrebljivo sa vrednostima deljenja 1:1, 1:2, 1:4 i 1:8.

Tajmer 2 je 8-bitni tajmer sa programabilnim pre skalerom i postskalerom. Može da bude tajmer/brojač opšte namene. Međutim, potreban je CCP modulu prilikom generisanja PWM signala, te modulu za sinhronu serijsku komunikaciju (SSP) kao *Baud Rate* generator. U kooperaciji sa Tajmerom2 koristi se registar PR2 (*Period Register*). Kada se vrednost brojača izjednači sa vrednošću upisanom u registar PR2, generiše se odgovarajući interapt.

P2.5. Periferijski komunikacioni moduli

Mikrokontroler PIC16F877 poseduje tri korisna periferijska komunikaciona modula u cilju razmene podataka sa ostalim komoponentama (memorijama, drugim mikrokontrolerima itd.)

Prvi od njih je SSP modul (*Synchronous Serial Port*), koji služi za komunikaciju sa serijskim EEPROM-ima, pomeračkim registrima, displej-drajverima itd. Ovaj modul može da radi u jednom od sleeća dva režima:

- *Serial Peripheral Interface* (SPI)
- *Inter-Integrated Circuit* (I2C).

Drugi serijski komunikacioni modul je USART (*Universal Synchronous Asynchronous Receiver Transmiter*). On uglavnom služi za povezivanje sa personalnim računarom, mada to nije jedina njegova mogućnost primene.

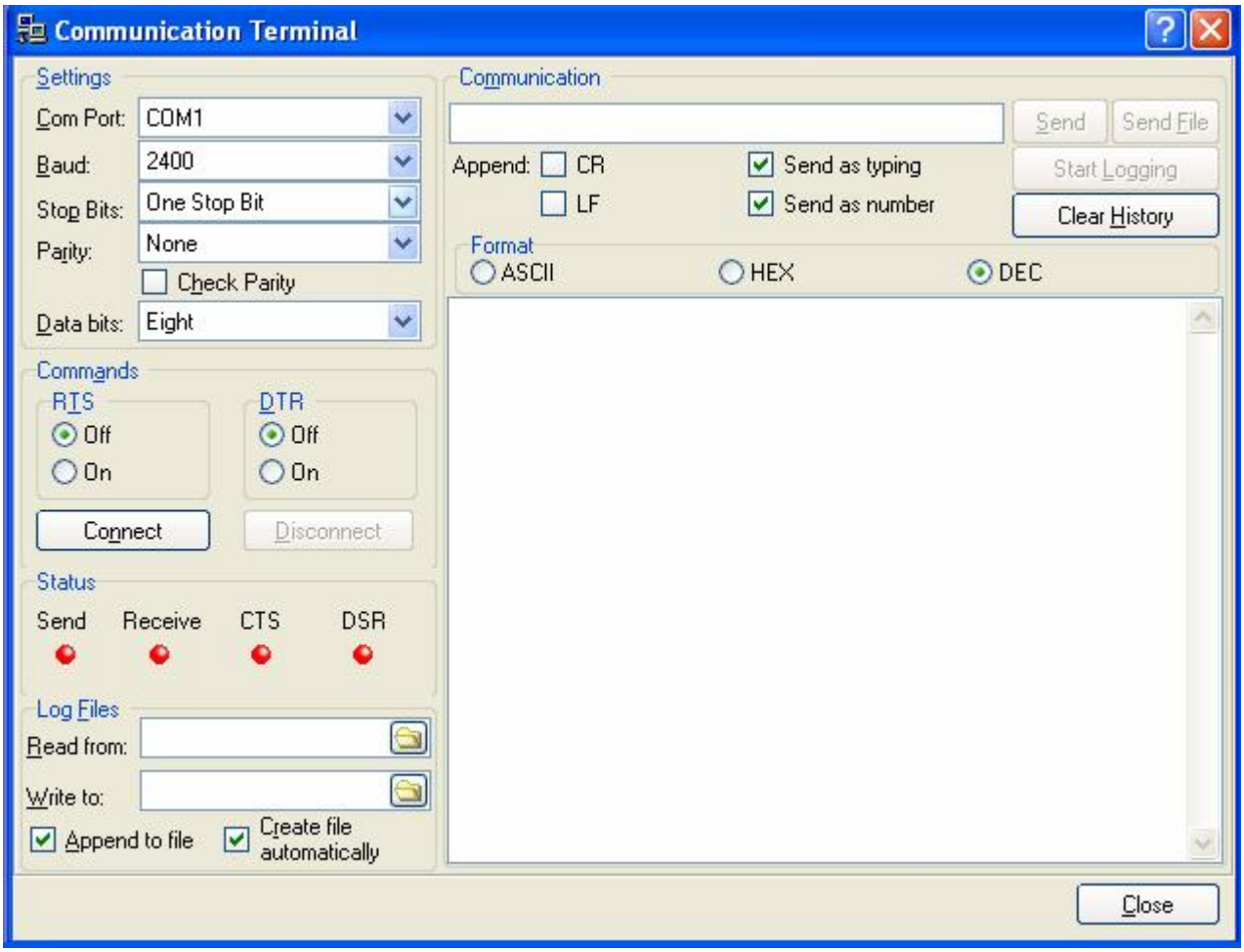
USART modul se konfiguriše u neki od sledeća tri režima rada:

- Asinhroni rad (*full duplex*)
- Sinhroni *master* rad (*half duplex*)
- Sinhroni *slave* rad (*half duplex*).

Osim serijskih, postoji i jedan paralelni komunikacioni modul. U pitanju je modul PSP (*Parallel Slave Port*). Njegov zadatak je da PIC16F877 direktno poveže na 8-bitnu magistralu podataka drugog mikroprocesora.

P3. SOFTVERSKI UART TERMINAL

Na *Slici 3.1* ilustrovan je prozor softverskog UART terminala za komunikaciju sa mikrokontrolerom PIC16F877. Pomenuti terminal je deo softverskog alata mikroC, koji je u ovom radu korišćen za pisanje programa i kompajliranje.



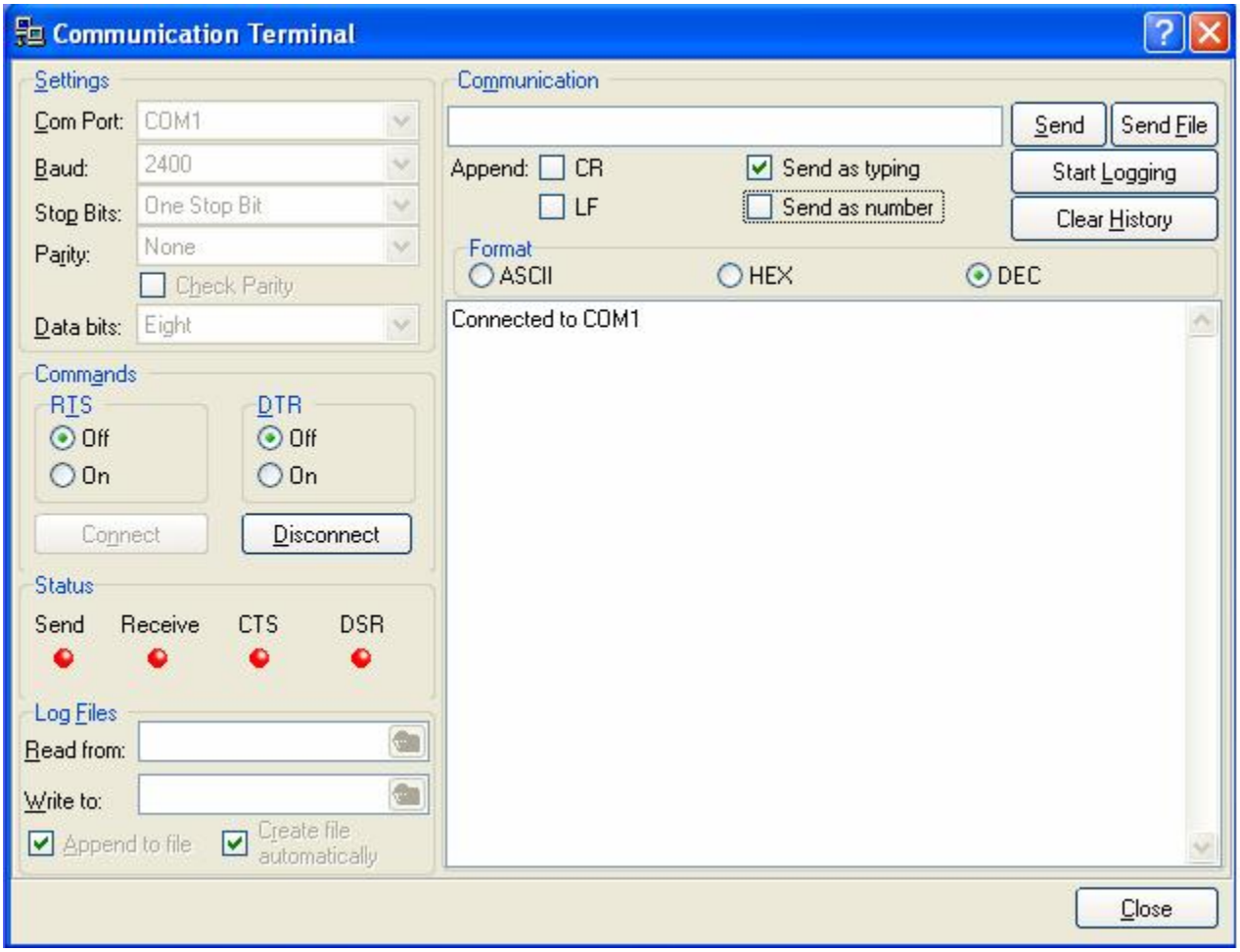
Slika 3.1. Izgled prozora kod UART komunikacionog terminala

Sada ćemo navesti najbitnije parametre UART terminala koje je moguće podesiti:

- Serijski port (u padajućem meniju treba izabrati željeni serijski port, a kod standardnih računara u pitanju su COM1 i COM2)
- Brzina prenosa (u našem slučaju, za frekvenciju kvarca od 4MHz treba izabrati 2400 bauda)
- Broj stop bitova (1, 1.5 i 2)
- Izbor parnosti (bez, parna, neparna itd.)
- Broj bitova koji čine karakter (5-8)
- Uključivanje/isključivanje *handshaking* signala (RTS i DTR)
- Izbor formata podatka koji se šalje (ASCII, heksadecimalni ili dekadni)
- Način slanja (odmah po kucanju *Send as typing* ili kada se klikne na opciju *Send*).

Pored navedenih, UART terminal ima niz dodatnih mogućnosti kojima se nećemo baviti, jer se ne tiču projekta koji će biti prezentovan u narednom poglavlju.

Kada se podese parametri, terminal se startuje klikom/pritiskom na komandno dugme **Connect**. Posle toga, terminal je spreman da prihvati komande (*Slika 3.2.*). Ukoliko se želi završetak komunikacije, treba kliknuti na komandno dugme **Disconnect**.



Slika 3.2. Izgled prozora kod UART komunikacionog terminala kada je spreman za slanje/prijem

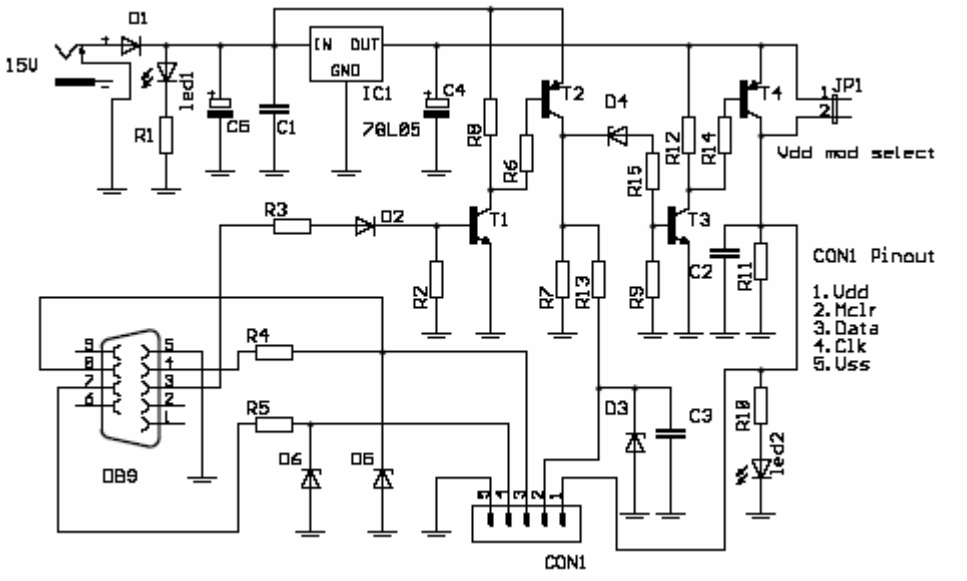
P4. ALLPIC I IC-PROG

Za upisivanje heksadecimalnog koda (dobijen iz mikroC kompajlera) u mikrokontroler, neophodno je posedovati odgovarajući programator – koji se sastoji iz *bootstrap loadera* (hardverski deo) i softvera (za programiranje *firmware*-a).

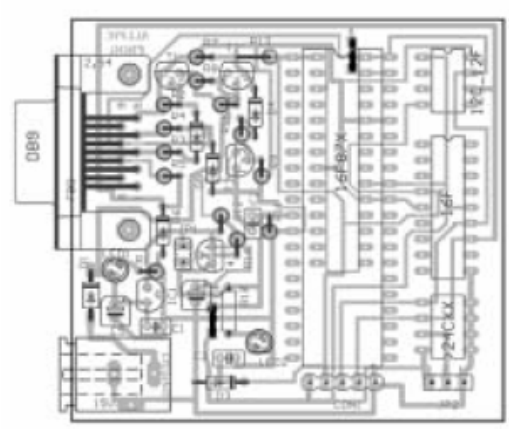
Uloga *bootstrap loader*-a je da transformiše naponske nivoe koji su aktuelni na portu na nivoe standardne logike (TTL), dok je uloga softvera da upiše heksadecimalni kôd u programsku (najčešće: EPROM ili *flash*) memoriju mikrokontrolera.

Za potrebe ovog rada korišćeni su *bootstrap loader* **ALLPIC** i softver **IC-Prog**.

Električna šema ALLPIC *bootstrap loader*-a prikazana je na *Slici 4.1*, dok su izgled pločice i gotov uređaj – na *Slikama 4.2 i 4.3*, respektivno.



Slika 4.1. Električna šema ALLPIC bootstrap loader-a



Slika 4.2. Izgled štampane pločice ALLPIC-a

Slika 4.3. Izgled uređaja ALLPIC

ALLPIC je namenjen programiranju većine serijskih PICmikrokontrolera. Pomenućemo samo neke od najpoznatijih: PIC16F8x, PIC16F62x, PIC16F87x, PIC12C50x, PIC12F6xx itd. Hardver je zasnovan na već oprobanom i veoma popularnom serijskom programatoru *Pony Prog*.

Sa šeme veza (*Slika 4.1*.) vidi se jednostavnost, kao i kompaktnost uređaja. Uređaj se povezuje sa PC računarom putem RS232 serijskog porta, koji standardno postoji na većini današnjih PC računara. Za rad programatora potreban je i slabiji izvor napona napajanja oko 15V. Dioda Dl na ulazu za napajanje je predviđena kao zaštita od pogrešnog priključenja polariteta izvora za napajanje. Izvor može biti i naizmenični napon, ukoliko vam je takav pri ruci.

Regulator napona 78L05 obezbeđuje 5 V napon potreban za napajanje mikrokontrolera, dok se Vpp napon (oko 13V) ostvaruje pomoću Zener diode D3. Ostatak kola čine tranzistori NPN i PNP, univerzal nog tipa, kao i propratne Zener diode D5 i D6 i otpornici R4 i R5 koji obezbeđuju TTL logičke nivoe (5V) potrebne za programiranje mikrokontrolera.

Razlika u odnosu na postojeće programatore koji se mogu sresti na Internetu (*Pony prog*, *JDM*) je upravo u delu kola koga čine T3, T4 i JPl sa pratećim elementima.

Kod starih modela programatora (*PonyProg, JDM,...)* javlja se problem ukoliko je potrebno reprogramirati PIC koji je prethodno programiran sa uključenim INT/RC i uključenim internim MCLR, u *FUSES* podešavanjima. U tom slučaju *PonyProg* programator ne može ni da obriše PIC, jer nije ispoštovana procedura preporučena od strane proizvođača mikrokontrolera, koja se odnosi na to da se Vdd (+5 V) napon sme pojaviti tek nakon uspostavljanja MCLR (+13V) programskog napona. Jedino je sa ovakvim sistemom moguće da PIC uđe u programski mod i da se ponovno reprogramira. Sve navedene osobine poseduje ALLPIC.

Međutim stari modeli PIC mikrokontrolera koji nemaju interni oscilator zahtevaju obrnut proces od gore navedenog tj. prvo Vdd napon pa tek nakon toga MCLR napon i za takve PIC kontrolere je ubačena podrška u vidu džampera JPl kojim se bira mod rada *Vdd mode select* .

Indikator LED1 nas informiše o prisustvu napona napajanja programatora i on treba da bude u granicama 14-20V, s tim da treba računati da pri 20V napajanju grejanje stabilizatora može biti primetno. Indikator LED2 indicira prisutan napon napajanja (Vdd +5V) na čipu koji se programira. Kratkospajač JPl ima funkciju: ukoliko je spojen, Vdd (+5 V) napon će stalno biti prisutan što će se i vide ti upaljenom LED2. Ovaj mod je za sve Microchip PIC mikrokontrolere koji nemaju interni oscilator (16F84, 16F87x,) kao i eeprome.

Za sve nove mikrokontrolere potrebno je skinuti JPl i u tom modu Vdd (+5V) se pojavljuje tek po uspostavljanju Vpp (13V) programskog napona na početku programiranja. Ovakav način rada je potreban za programiranje sledećih mikrokontrolera 16F627, 16F628, 12F629, 12F675...

Kratkospajač JP2 služi za izbor Write protect moda za eeprome 24CXX. Poseban konektor koji je označen na šemi veza kao CON1 koristi se za eksterni priključak za incircuit programiranje (ICSP).

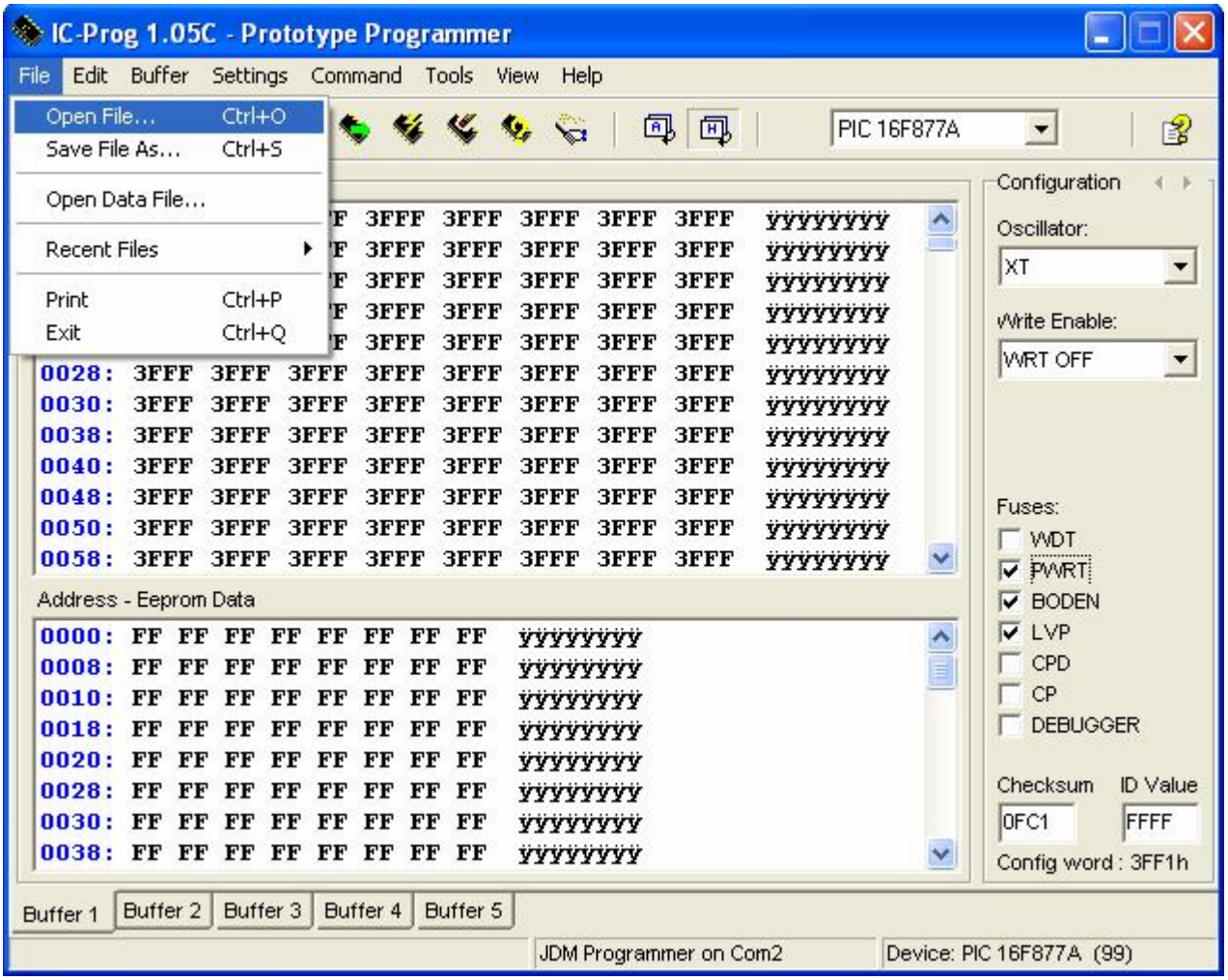
Inače, ALLPIC nije predviđen za 16C5x seriju i još neke specifične ***Microchip*** mikrokontrolere.

IC-Prog

Softver potreban za rad ovog programatora možete skinuti (*download*-ovati) sa Internet sajta adresihttp://www.ic-prog.com/. Izgled programa prikazan je na *Slici 4.4*. Program se može koristiti pod Windows operativnim sistemima počev od Win95 do WinXP verzije, uz neznatna prepodešavanja. Mi ćemo na primeru programiranja PIC16F877 mikrokontrolera prikazati podešavanja i postupak programira nja ovog mikrokontrolera pod operativnim sistemom Windows 98.

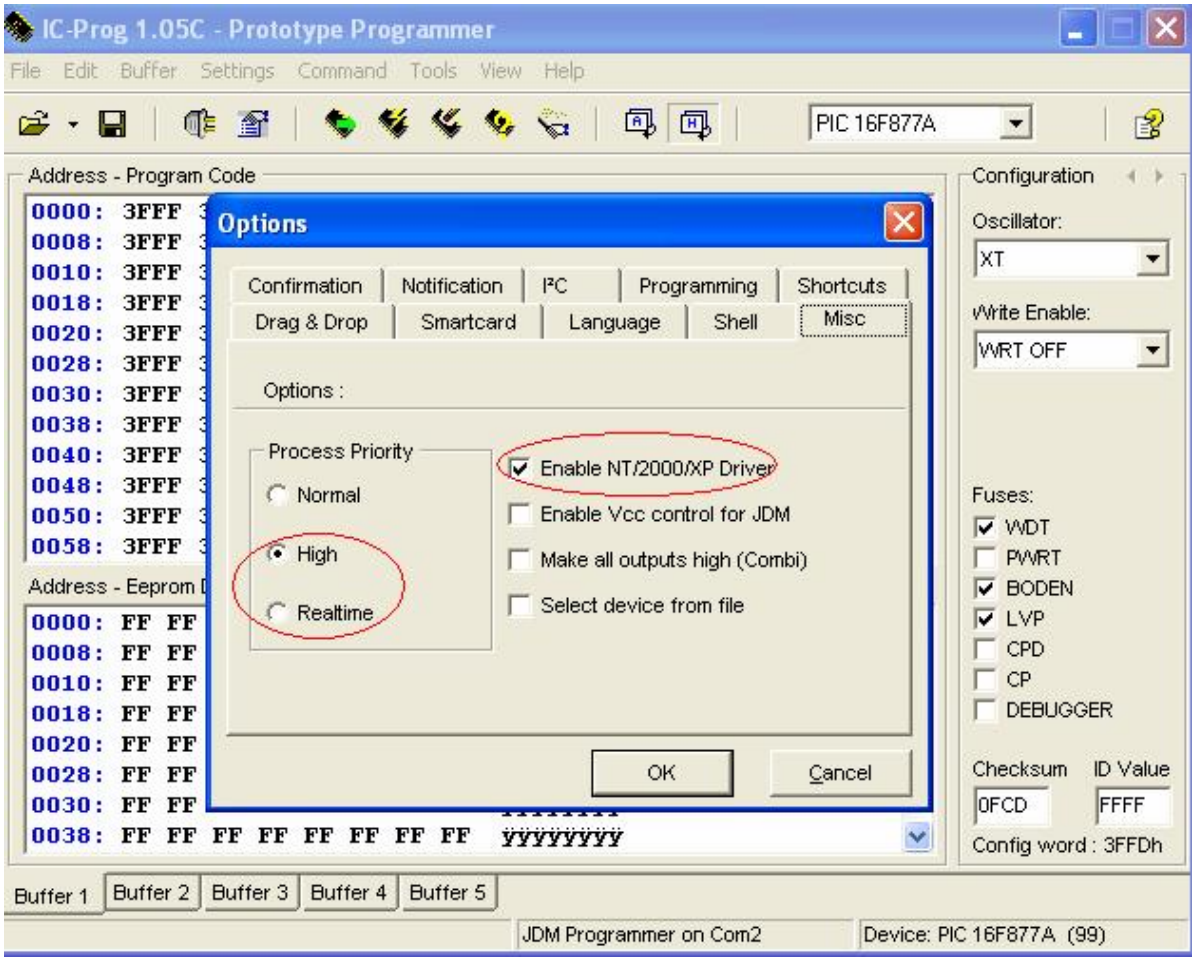
Prvo što je potrebno uraditi je da skinemo JPl. Ovim smo odabrali potrebnu sekvencu Vdd i Vpp napona. Sada možemo postaviti PIC16F877 u za njega namenjeno podnožje (40 pina) i zatim uključujemo napon napajanja. Sada prelazimo na podešavanje softvera i vršimo inicijalna podešavanja koja je potrebno uraditi samo pri prvom startovanju programa na našem računaru. Pritiskom na taster F3 ili

klikom na *Settings*; *Hardware* dobijami prozor kao na *Slici 4.5*. Potrebno je sada izvršiti podešavanja kao što je prikazano na slici, s tim da u zavisnosti od COM porta na koji je priključen hardver programatora na vašem računaru odaberete odgovarajući COMl ili COM2. Potvrdite sa OK. Time su inicijalna podešavanja završena.



Slika 4.4. Izgled osnovnog prozora softvera IC-Prog

Zatim u polju *Settings;Device;Microchip PIC* odaberite tip mikrokontrolera kojeg želite da isprogramirate, to je u našem primeru PIC16F877. Sada je na red došlo učitavanje HEX fajla namenjenog za programiranje u mikrokontroler. (*File ; Open file*). Na kraju ukoliko *Fuses* podešavanja nisu implementirana u sklopu učitanog HEX fajla, ručno podesite Fuses koji se nalaze u desnom delu prozora *Ic Prog* programa kao i tip oscilatora. Klikom na Command;Program All programiranje počinje. Program će vas izveštavati u toku programiranja šta trenutno radi i na kraju programiranja o uspešnosti programiranja. Upotreba programa pod WindowsXP (WinNT) operativnim sistemima je takođe moguća, ali je potrebno neznatno prepodesiti IcProg softver. Prvo je potrebno sa *IcProg* sajta (www.ic-prog.com) skinuti drajver za Win2000/NT pod nazivom *icprog.sys*. Ovaj fajl se raspakuje u isti direktorijum gde se nalazi i sam IcProg i zatim je potrebno startovati Ic Prog. Nakon toga u Settings ; *Options ; Misc* aktivirajte polje ;*Enable NT/2000/XP Driver*;. Na ovaj način omogućava se Direct I/O programiranje i pod XP operativnim sistemom. Takođe je poželjno podesiti u istom polju i Procces priority na High ili Real time.Sada možemo pristupiti konkretnom programiranju čipa, i pod Windows XP operativnim sistemom, kao što je to prethodno objašnjeno u primeru pod Windows 98 sistemom.



Slika 4.5. Uključivanje drajvera ic-prog.sys

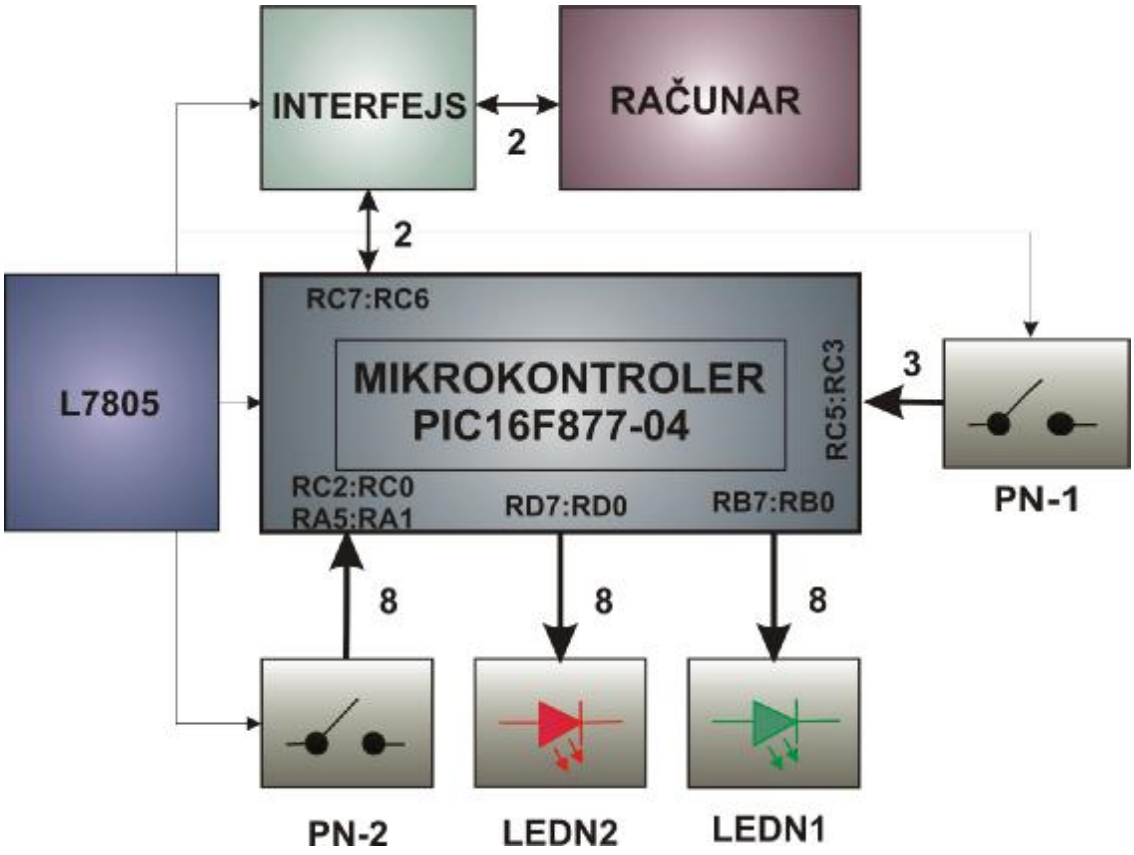
Kako je IcProg program koji podržava veći broj mikrokontrolera, kao i više tipova programatora, u menijima postoji veliki broj čipova. Vama je sa ovim programatorom na raspolaganju meni Microchip PIC i I2C Eeprom;24CXX.

IcProg je kompatibilan sa mnogim *bootstrap loader*-ima, kao što su *JDM Programmer*, *TAFE Programmer*, *TAIT Programmer*, *Conquest Programmer*, *ProPIC 2 Programmer* itd.

P5. REALIZACIJA KONVERTORA KODOVA

Na *Slici 5.1.* je prikazana blok šema, a na *Slikama 5.2.* i *5.3* date su električne šeme konvertora kodova relizovanog sa mikrokontrolerom PIC16F877-04 (radi na frekvenciji 4MHz). Omogućena je serijska veza mikrokontrolera sa PC računarom, a za konverziju naponskih nivoa (RS232C-TTL i obrnuto) zaduženo je interfejsno IC MAX232. Logika u kolu (PIC i MAX232) se napaja sa naponom 5V, koji se obezbeđuje preko stabilizator napona L7805 (u kućištu TO-220).

Tasterom T1 se resetuje mikrokontroler.

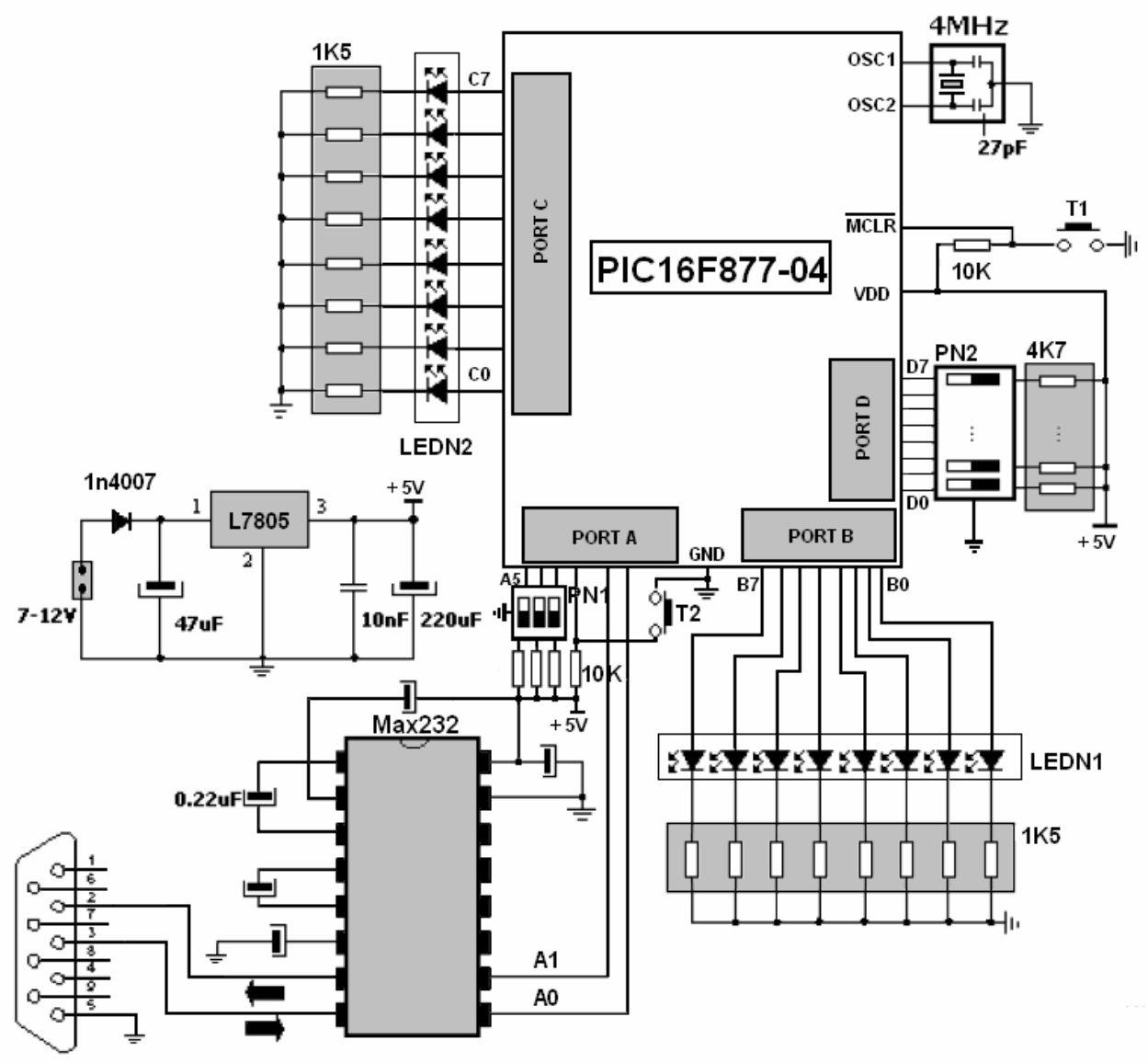


Slika 5.1. Izgled blok-šeme konvertora

Spisak delova za konvertor:

R.br.	DEO	KOL.	*SOCKET*
1	**Adapter 220V~/DC9V- (nestabilisani)**	1	
2	**Regulator napona L7805**	2	TO-220
3	**Mikrokontroler PIC16F877-04**	1	DIP-40
4	**Interfejs integrisano kolo MAX232**	1	DIP-16
5	**Dioda 1n4007**	1	*through hole*
6	**Ugljenoslojni standarni otpornik 10K**	5	*through hole*
7	**Niz otpornika 8x1K5**	2	*through hole*

8	**LED RED sa malom potrošnjom 3mm**	8	*through hole*
9	**LED YELLOW sa malom potrošnjom 3mm**	8	*through hole*
10	**Blok standardni kondenzator 220nF**	6	*through hole*
11	**Blok standardni kondenzator 10nF**	1	*through hole*
12	**Elektrolitski kondenzator 47uF 25V**	1	*through hole*
13	**Elektrolitski kondenzator 220uF 16V**	1	*through hole*
14	**Keramički disk kondenzator 27pF**	2	*through hole*
15	**Kvarc 4MHz**	1	*through hole*
16	**Prekidač za PCB, 3-položajni**	11	*through hole*
17	**4-pinski taster za PCB**	2	*through hole*
18	**DIP-40, *low-cost***	1	
19	**DIP-16, *low-cost***	1	
20	**DC konektor za PCB, 2mm**	1	
21	**DB9F za PCB**	1	



Slika 5.2. Električna šema konvertora

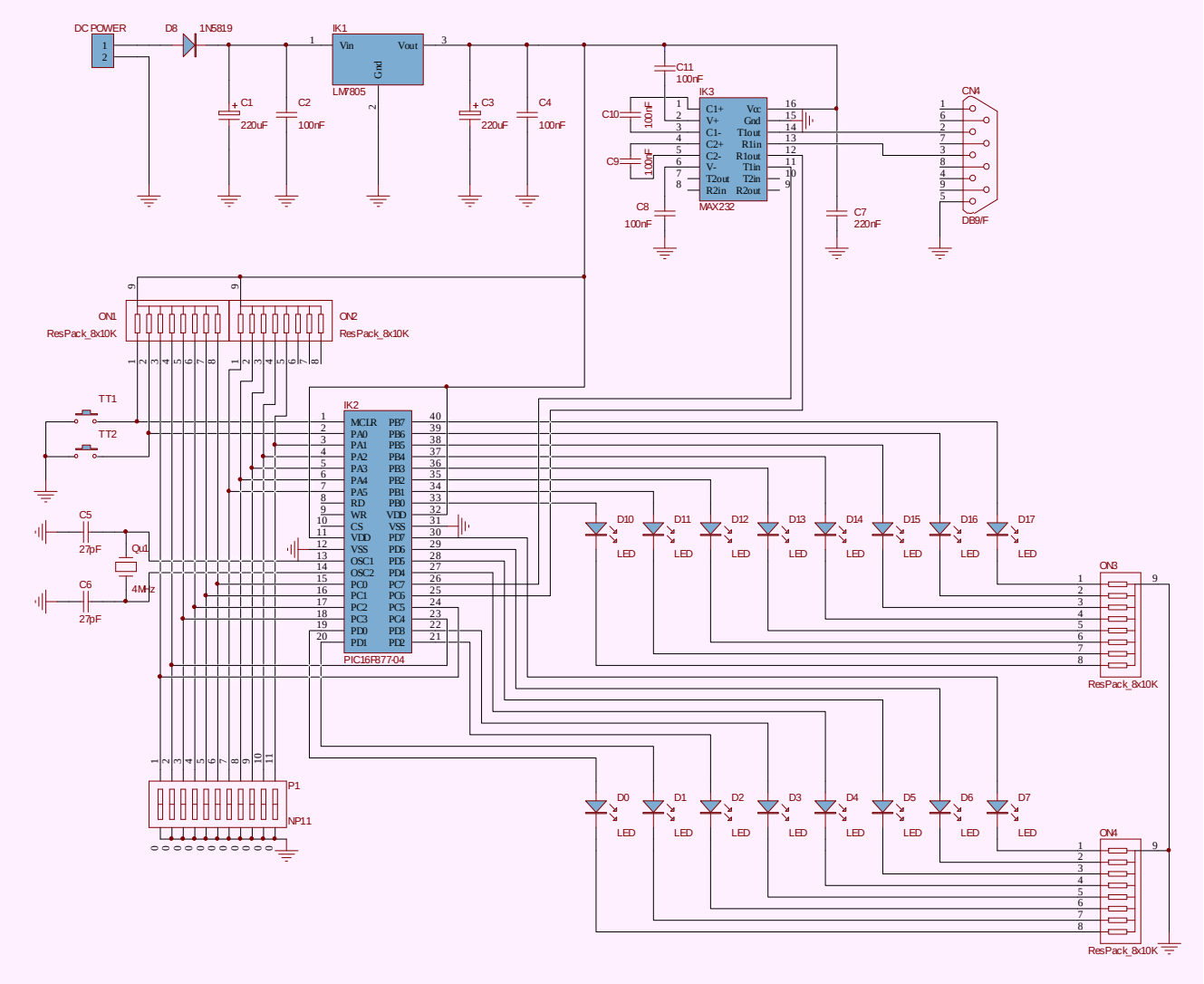
Na *Slici 5.1*. mogu da se uoče dve grupe prekidača – PN1 (8) i PN2 (3). Preko PN2 (vezani su za Port B) se zadaje 8-bitna sekvenca koju treba transformisati, dok se preko PN1 (vezani su za pinove RC5-RC3) zadaje tip transformacije – što je ilustrovano u Tabeli ispod.

Kombinacija (RC5-RC3)	**Zadata 8-bitna reč**	**Transformisana 8-bitna reč**
000	binarni kod	"višak 3"
001	binarni kod	Grejov kod
010	binarni kod (PC)	"višak 3"
011	binarni kod (PC)	Grejov kod

Prve dve kombinacije se ostvaruje na samoj ploči gde se nalazi mikrokontroler sa potrebnim interfejsima. Poslednje dve kombinacije se ostvaruju uz pomoć UART terminala, o kojem je bilo reči u trećem poglavlju. Izabrali smo potpuni Grejov kod sa 4 binarne cifre i ovde ćemo priložiti Tabelu transformacije, koja je već bila izložena u prvom poglavlju.

Binarni	**Grejov**
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Na *Slici 5.1*. mogu se, dalje, uočiti dve grupe LED – LEDN1 (8) i LEDN2 (8). Grupa LEDN1 (vezane za Port B) indicira netransformisana 8-bitna reč (odgovara stanjima na PN2), dok se preko LEDN2 (vezani su za Port C) indicira transformisana 8-bitna reč. To ćemo sada ilustrovati primerom: ako se izabere transformacija binarnog u kod "višak 3" i na prekidačima PN2 zada kombinacija 00000000, neće svetleti nijedna LED iz grupe LEDN1 – dok će u grupi LEDN2 svetleti dve odgovarajuće diode (00000011) kada se pritisne taster TT2.



Slika 5.3. Električna šema konvertora (detaljna)

Napomena:

Tasterom T2 vrši se realizacija transformacije za prvih šest kombinacija iz tabele. Dakle, najpre se zada stanje na prekidačima PN2, pa onda pritisne taster T2. Tek tada će se na LEDN1 i LEDN2 prikazati odgovarajuća stanja.

Inače, o transformacijama je detaljnije bilo reči u prvom poglavlju, pa možete pogledati tabele koje su tamo prikazane.

Studenti sami mogu da realizuju još četiri kombinacije u okviru laboratorijske vežbe.

Programski kod u mikroC-u je maksimalno uprošćen kako bi bio jasan početnicima.

Kada je prekidač u središnjem položaju, zadata je logička '0'. Kada je prekidač u krajnjem položaju, zadaje se logička '1'.

Pritiskom na taster TT2 ostvaruje se transformacija kodova na ploči.

P4.1. Program u mikroC-u

```
unsigned short i, j, k, l, m, n =0;

void main()     {

// DEFINISANJE PORTOVA
  TRISB = 0x00;
  PORTB = 0x00;
  TRISD = 0x00;
  PORTD = 0x00;
  TRISC = 0xBF;
  TRISA = 0xFF;
  PORTA = 0x00;

  ADCON1 = 6;
// Inicijalizacija USART modula
  Usart_Init(2400);

  do {

    if (PORTC.F4==0) {

    n.F7=PORTC.F2;
    n.F6=PORTC.F1;
    n.F5=PORTC.F0;
    n.F4=PORTA.F5;
    n.F3=PORTA.F4;
    n.F2=PORTA.F3;
    n.F1=PORTA.F2;
    n.F0=PORTA.F1;
    j = (n & 0xF0) >> 4;
    k = n & 0x0F;
    PORTB = n;
    PORTD = 0x00;
// KONVERZIJA "VISAK 3"
    if ((PORTC.F3==0)&&(k<=9)&&(j<=9)&&(PORTA.F0==0)) {

    l = (j + 3) << 4;
    m = k + 3;
    PORTD = l+m;
```

```
    }

// KONVERZIJA U GREJOV KOD
    if ((PORTC.F3==1)&&(PORTA.F0==0)) {

    PORTD.F7=J.F3;
    PORTD.F6=((J.F3&&!(J.F2))||(!(J.F3)&&J.F2));
    PORTD.F5=((J.F2&&!(J.F1))||(!(J.F2)&&J.F1));
    PORTD.F4=((J.F1&&!(J.F0))||(!(J.F1)&&J.F0));
    PORTD.F3=K.F3;
    PORTD.F2=((K.F3&&!(K.F2))||(!(K.F3)&&K.F2));
    PORTD.F1=((K.F2&&!(K.F1))||(!(K.F2)&&K.F1));
    PORTD.F0=((K.F1&&!(K.F0))||(!(K.F1)&&K.F0));

    }
    }
    if (PORTC.F4==1) {
// PODATAK SE PRIMA I VRACA KA RACUNARU
    if (Usart_Data_Ready()) {
      i = Usart_Read();
      Usart_Write(i);
    }

    j = (i & 0xF0) >> 4;
    k = i & 0x0F;
    PORTB = i;
    PORTD = 0x00;
// KONVERZIJA U "VISAK 3"
    if ((PORTC.F3==0)&&(k<=9)&&(j<=9)) {

    l = (j + 3) << 4;
    m = k + 3;
    PORTD = l+m;

    }
// KONVERZIJA U GREJOV KOD
    if (PORTC.F3==1) {

    PORTD.F7=J.F3;
    PORTD.F6=((J.F3&&!(J.F2))||(!(J.F3)&&J.F2));
    PORTD.F5=((J.F2&&!(J.F1))||(!(J.F2)&&J.F1));
    PORTD.F4=((J.F1&&!(J.F0))||(!(J.F1)&&J.F0));
```

```
   PORTD.F3=K.F3;
   PORTD.F2=((K.F3&&!(K.F2))||(!(K.F3)&&K.F2));
   PORTD.F1=((K.F2&&!(K.F1))||(!(K.F2)&&K.F1));
   PORTD.F0=((K.F1&&!(K.F0))||(!(K.F1)&&K.F0));

   }
   }


  } while (1);
}//~!
```

P4.2. Program u asembleru

```
;  ASM code generated by mikroVirtualMachine for PIC - V. 6.2.1.0
;  Info: http://www.mikroelektronika.co.yu

; ADDRESS   OPCODE      ASM
; ------------------------------------------
$0000 $283D             GOTO  _main
$0004 $     _Usart_Data_Ready:
$0004 $3000             MOVLW 0
$0005 $1303             BCF   STATUS, RP1
$0006 $1283             BCF   STATUS, RP0
$0007 $1A8C             BTFSC PIR1, 5
$0008 $3001             MOVLW 1
$0009 $00F1             MOVWF STACK_1
$000A $0871             MOVF  STACK_1, 0
$000B $00F0             MOVWF STACK_0
$000C $0008             RETURN
$000D $     _Usart_Read:
$000D $1303             BCF   STATUS, RP1
$000E $1283             BCF   STATUS, RP0
$000F $081A             MOVF  RCREG, 0
$0010 $00A6             MOVWF Usart_Read_tmp_L0
$0011 $1C98             BTFSS RCSTA, 1
$0012 $2815             GOTO  L_Usart_Read_2
$0013 $1218             BCF   RCSTA, 4
$0014 $1618             BSF   RCSTA, 4
$0015 $     L_Usart_Read_2:
$0015 $0826             MOVF  Usart_Read_tmp_L0, 0
$0016 $00F0             MOVWF STACK_0
$0017 $0008             RETURN
$0018 $     _Usart_Write:
$0018 $     L_Usart_Write_3:
$0018 $3000             MOVLW 0
$0019 $1303             BCF   STATUS, RP1
$001A $1683             BSF   STATUS, RP0
$001B $1898             BTFSC TXSTA, 1
$001C $3001             MOVLW 1
```

```
$001D $00F1             MOVWF STACK_1
$001E $0871             MOVF  STACK_1, 0
$001F $3A00             XORLW 0
$0020 $1D03             BTFSS STATUS, Z
$0021 $2824             GOTO  L_Usart_Write_4
$0022 $0000             NOP
$0023 $2818             GOTO  L_Usart_Write_3
$0024 $     L_Usart_Write_4:
$0024 $1283             BCF   STATUS, RP0
$0025 $0826             MOVF  FARG_Usart_Write+0, 0
$0026 $0099             MOVWF TXREG
$0027 $0008             RETURN
$0028 $     GlobalIniKONVERTOR:
$0028 $3000             MOVLW 0
$0029 $1303             BCF   STATUS, RP1
$002A $1283             BCF   STATUS, RP0
$002B $00A0             MOVWF _n+0
$002C $0008             RETURN
$002D $     _Usart_Init:
$002D $1303             BCF   STATUS, RP1
$002E $1683             BSF   STATUS, RP0
$002F $1698             BSF   TXSTA, 5
$0030 $3090             MOVLW 144
$0031 $1283             BCF   STATUS, RP0
$0032 $0098             MOVWF RCSTA
$0033 $1683             BSF   STATUS, RP0
$0034 $1787             BSF   TRISC, 7
$0035 $1307             BCF   TRISC, 6
$0036 $     L_Usart_Init_0:
$0036 $1283             BCF   STATUS, RP0
$0037 $1E8C             BTFSS PIR1, 5
$0038 $283C             GOTO  L_Usart_Init_1
$0039 $081A             MOVF  RCREG, 0
$003A $00AA             MOVWF Usart_Init_tmp_L0
$003B $2836             GOTO  L_Usart_Init_0
$003C $     L_Usart_Init_1:
$003C $0008             RETURN
$003D $     _main:
;KONVERTOR.c,3 ::             void main()      {
;KONVERTOR.c,6 ::             TRISB = 0x00;
$003D $2028             CALL  GlobalIniKONVERTOR, 1
$003E $1683             BSF   STATUS, RP0
$003F $0186             CLRF  TRISB, 1
;KONVERTOR.c,7 ::             PORTB = 0x00;
$0040 $1283             BCF   STATUS, RP0
$0041 $0186             CLRF  PORTB, 1
;KONVERTOR.c,8 ::             TRISD = 0x00;
$0042 $1683             BSF   STATUS, RP0
$0043 $0188             CLRF  TRISD, 1
;KONVERTOR.c,9 ::             PORTD = 0x00;
$0044 $1283             BCF   STATUS, RP0
$0045 $0188             CLRF  PORTD, 1
;KONVERTOR.c,10 ::            TRISC = 0xBF;
$0046 $30BF             MOVLW 191
$0047 $1683             BSF   STATUS, RP0
$0048 $0087             MOVWF TRISC
;KONVERTOR.c,11 ::            TRISA = 0xFF;
$0049 $30FF             MOVLW 255
$004A $0085             MOVWF TRISA
```

```
;KONVERTOR.c,12 ::           PORTA = 0x00;
$004B $1283             BCF   STATUS, RP0
$004C $0185             CLRF  PORTA, 1
;KONVERTOR.c,14 ::           ADCON1 = 6;
$004D $3006             MOVLW 6
$004E $1683             BSF   STATUS, RP0
$004F $009F             MOVWF ADCON1
;KONVERTOR.c,16 ::           Usart_Init(2400);
$0050 $3067             MOVLW 103
$0051 $0099             MOVWF SPBRG
$0052 $1518             BSF   TXSTA, BRGH
$0053 $202D             CALL  _Usart_Init
;KONVERTOR.c,18 ::           do {
$0054 $     L_main_0:
;KONVERTOR.c,20 ::           if (PORTC.F4==0) {
$0054 $3000             MOVLW 0
$0055 $1A07             BTFSC PORTC, 4
$0056 $3001             MOVLW 1
$0057 $00F1             MOVWF STACK_1
$0058 $0871             MOVF  STACK_1, 0
$0059 $3A00             XORLW 0
$005A $1D03             BTFSS STATUS, Z
$005B $29B1             GOTO  L_main_2
;KONVERTOR.c,22 ::           n.F7=PORTC.F2;
$005C $3000             MOVLW 0
$005D $1907             BTFSC PORTC, 2
$005E $3001             MOVLW 1
$005F $00F1             MOVWF STACK_1
$0060 $3000             MOVLW 0
$0061 $1871             BTFSC STACK_1, 0
$0062 $3080             MOVLW 128
$0063 $0620             XORWF _n, 0
$0064 $3980             ANDLW 128
$0065 $06A0             XORWF _n, 1
;KONVERTOR.c,23 ::           n.F6=PORTC.F1;
$0066 $3000             MOVLW 0
$0067 $1887             BTFSC PORTC, 1
$0068 $3001             MOVLW 1
$0069 $00F1             MOVWF STACK_1
$006A $3000             MOVLW 0
$006B $1871             BTFSC STACK_1, 0
$006C $3040             MOVLW 64
$006D $0620             XORWF _n, 0
$006E $3940             ANDLW 64
$006F $06A0             XORWF _n, 1
;KONVERTOR.c,24 ::           n.F5=PORTC.F0;
$0070 $3001             MOVLW 1
$0071 $0507             ANDWF PORTC, 0
$0072 $00F0             MOVWF STACK_0
$0073 $3000             MOVLW 0
$0074 $1870             BTFSC STACK_0, 0
$0075 $3020             MOVLW 32
$0076 $0620             XORWF _n, 0
$0077 $3920             ANDLW 32
$0078 $06A0             XORWF _n, 1
;KONVERTOR.c,25 ::           n.F4=PORTA.F5;
$0079 $3000             MOVLW 0
$007A $1A85             BTFSC PORTA, 5
$007B $3001             MOVLW 1
```

```
$007C $00F1             MOVWF STACK_1
$007D $3000             MOVLW 0
$007E $1871             BTFSC STACK_1, 0
$007F $3010             MOVLW 16
$0080 $0620             XORWF _n, 0
$0081 $3910             ANDLW 16
$0082 $06A0             XORWF _n, 1
;KONVERTOR.c,26 ::             n.F3=PORTA.F4;
$0083 $3000             MOVLW 0
$0084 $1A05             BTFSC PORTA, 4
$0085 $3001             MOVLW 1
$0086 $00F1             MOVWF STACK_1
$0087 $3000             MOVLW 0
$0088 $1871             BTFSC STACK_1, 0
$0089 $3008             MOVLW 8
$008A $0620             XORWF _n, 0
$008B $3908             ANDLW 8
$008C $06A0             XORWF _n, 1
;KONVERTOR.c,27 ::             n.F2=PORTA.F3;
$008D $3000             MOVLW 0
$008E $1985             BTFSC PORTA, 3
$008F $3001             MOVLW 1
$0090 $00F1             MOVWF STACK_1
$0091 $3000             MOVLW 0
$0092 $1871             BTFSC STACK_1, 0
$0093 $3004             MOVLW 4
$0094 $0620             XORWF _n, 0
$0095 $3904             ANDLW 4
$0096 $06A0             XORWF _n, 1
;KONVERTOR.c,28 ::             n.F1=PORTA.F2;
$0097 $3000             MOVLW 0
$0098 $1905             BTFSC PORTA, 2
$0099 $3001             MOVLW 1
$009A $00F1             MOVWF STACK_1
$009B $3000             MOVLW 0
$009C $1871             BTFSC STACK_1, 0
$009D $3002             MOVLW 2
$009E $0620             XORWF _n, 0
$009F $3902             ANDLW 2
$00A0 $06A0             XORWF _n, 1
;KONVERTOR.c,29 ::             n.F0=PORTA.F1;
$00A1 $3000             MOVLW 0
$00A2 $1885             BTFSC PORTA, 1
$00A3 $3001             MOVLW 1
$00A4 $00F1             MOVWF STACK_1
$00A5 $3000             MOVLW 0
$00A6 $1871             BTFSC STACK_1, 0
$00A7 $3001             MOVLW 1
$00A8 $0620             XORWF _n, 0
$00A9 $3901             ANDLW 1
$00AA $06A0             XORWF _n, 1
;KONVERTOR.c,30 ::             j = (n & 0xF0) >> 4;
$00AB $30F0             MOVLW 240
$00AC $0520             ANDWF _n, 0
$00AD $00A1             MOVWF _j
$00AE $0CA1             RRF   _j, 1
$00AF $13A1             BCF   _j, 7
$00B0 $0CA1             RRF   _j, 1
$00B1 $13A1             BCF   _j, 7
```

```
$00B2 $0CA1             RRF   _j, 1
$00B3 $13A1             BCF   _j, 7
$00B4 $0CA1             RRF   _j, 1
$00B5 $13A1             BCF   _j, 7
;KONVERTOR.c,31 ::            k = n & 0x0F;
$00B6 $300F             MOVLW 15
$00B7 $0520             ANDWF _n, 0
$00B8 $00A2             MOVWF _k
;KONVERTOR.c,32 ::            PORTB = n;
$00B9 $0820             MOVF  _n, 0
$00BA $0086             MOVWF PORTB
;KONVERTOR.c,33 ::            PORTD = 0x00;
$00BB $0188             CLRF  PORTD, 1
;KONVERTOR.c,35 ::            if ((PORTC.F3==0)&&(k<=9)&&(j<=9)&&(PORTA.F0==0)) {
$00BC $3000             MOVLW 0
$00BD $1987             BTFSC PORTC, 3
$00BE $3001             MOVLW 1
$00BF $00F1             MOVWF STACK_1
$00C0 $0871             MOVF  STACK_1, 0
$00C1 $3A00             XORLW 0
$00C2 $1D03             BTFSS STATUS, Z
$00C3 $28EA             GOTO  L_main_5
$00C4 $0822             MOVF  _k, 0
$00C5 $3C09             SUBLW 9
$00C6 $1C03             BTFSS STATUS, C
$00C7 $28EA             GOTO  L_main_5
$00C8 $0821             MOVF  _j, 0
$00C9 $3C09             SUBLW 9
$00CA $1C03             BTFSS STATUS, C
$00CB $28EA             GOTO  L_main_5
$00CC $3001             MOVLW 1
$00CD $0505             ANDWF PORTA, 0
$00CE $00F1             MOVWF STACK_1
$00CF $0871             MOVF  STACK_1, 0
$00D0 $3A00             XORLW 0
$00D1 $1D03             BTFSS STATUS, Z
$00D2 $28EA             GOTO  L_main_5
$00D3 $     L84_ex_L_main_5:
;KONVERTOR.c,37 ::            l = (j + 3) << 4;
$00D3 $3003             MOVLW 3
$00D4 $0721             ADDWF _j, 0
$00D5 $00F2             MOVWF STACK_2
$00D6 $0872             MOVF  STACK_2, 0
$00D7 $00F1             MOVWF STACK_1
$00D8 $0DF1             RLF   STACK_1, 1
$00D9 $1071             BCF   STACK_1, 0
$00DA $0DF1             RLF   STACK_1, 1
$00DB $1071             BCF   STACK_1, 0
$00DC $0DF1             RLF   STACK_1, 1
$00DD $1071             BCF   STACK_1, 0
$00DE $0DF1             RLF   STACK_1, 1
$00DF $1071             BCF   STACK_1, 0
$00E0 $0871             MOVF  STACK_1, 0
$00E1 $00A3             MOVWF _l
;KONVERTOR.c,38 ::            m = k + 3;
$00E2 $3003             MOVLW 3
$00E3 $0722             ADDWF _k, 0
$00E4 $00F0             MOVWF STACK_0
$00E5 $0870             MOVF  STACK_0, 0
```

```
$00E6 $00A4             MOVWF _m
;KONVERTOR.c,39 ::            PORTD = l+m;
$00E7 $0870             MOVF  STACK_0, 0
$00E8 $0771             ADDWF STACK_1, 0
$00E9 $0088             MOVWF PORTD
;KONVERTOR.c,41 ::            }
$00EA $     L_main_5:
;KONVERTOR.c,44 ::            if ((PORTC.F3==1)&&(PORTA.F0==0)) {
$00EA $3000             MOVLW 0
$00EB $1987             BTFSC PORTC, 3
$00EC $3001             MOVLW 1
$00ED $00F1             MOVWF STACK_1
$00EE $0871             MOVF  STACK_1, 0
$00EF $3A01             XORLW 1
$00F0 $1D03             BTFSS STATUS, Z
$00F1 $29B1             GOTO  L_main_8
$00F2 $3001             MOVLW 1
$00F3 $0505             ANDWF PORTA, 0
$00F4 $00F1             MOVWF STACK_1
$00F5 $0871             MOVF  STACK_1, 0
$00F6 $3A00             XORLW 0
$00F7 $1D03             BTFSS STATUS, Z
$00F8 $29B1             GOTO  L_main_8
$00F9 $     L110_ex_L_main_8:
;KONVERTOR.c,46 ::            PORTD.F7=J.F3;
$00F9 $3000             MOVLW 0
$00FA $19A1             BTFSC _j, 3
$00FB $3001             MOVLW 1
$00FC $00F1             MOVWF STACK_1
$00FD $3000             MOVLW 0
$00FE $1871             BTFSC STACK_1, 0
$00FF $3080             MOVLW 128
$0100 $0608             XORWF PORTD, 0
$0101 $3980             ANDLW 128
$0102 $0688             XORWF PORTD, 1
;KONVERTOR.c,47 ::            PORTD.F6=((J.F3&&!(J.F2))||(!(J.F3)&&J.F2));
$0103 $3000             MOVLW 0
$0104 $19A1             BTFSC _j, 3
$0105 $3001             MOVLW 1
$0106 $00F1             MOVWF STACK_1
$0107 $0871             MOVF  STACK_1, 0
$0108 $1903             BTFSC STATUS, Z
$0109 $290C             GOTO  L129_ex_L_main_14
$010A $1D21             BTFSS _j, 2
$010B $2917             GOTO  L_main_14
$010C $     L129_ex_L_main_14:
$010C $19A1             BTFSC _j, 3
$010D $2915             GOTO  L143_ex_L_main_14
$010E $3000             MOVLW 0
$010F $1921             BTFSC _j, 2
$0110 $3001             MOVLW 1
$0111 $00F1             MOVWF STACK_1
$0112 $0871             MOVF  STACK_1, 0
$0113 $1D03             BTFSS STATUS, Z
$0114 $2917             GOTO  L_main_14
$0115 $     L143_ex_L_main_14:
$0115 $01F0             CLRF  STACK_0, 1
$0116 $2919             GOTO  L_main_13
$0117 $     L_main_14:
```

```
$0117 $3001             MOVLW 1
$0118 $00F0             MOVWF STACK_0
$0119 $     L_main_13:
$0119 $3000             MOVLW 0
$011A $1870             BTFSC STACK_0, 0
$011B $3040             MOVLW 64
$011C $0608             XORWF PORTD, 0
$011D $3940             ANDLW 64
$011E $0688             XORWF PORTD, 1
;KONVERTOR.c,48 ::            PORTD.F5=((J.F2&&!(J.F1))||(!(J.F2)&&J.F1));
$011F $3000             MOVLW 0
$0120 $1921             BTFSC _j, 2
$0121 $3001             MOVLW 1
$0122 $00F1             MOVWF STACK_1
$0123 $0871             MOVF  STACK_1, 0
$0124 $1903             BTFSC STATUS, Z
$0125 $2928             GOTO  L164_ex_L_main_20
$0126 $1CA1             BTFSS _j, 1
$0127 $2933             GOTO  L_main_20
$0128 $     L164_ex_L_main_20:
$0128 $1921             BTFSC _j, 2
$0129 $2931             GOTO  L178_ex_L_main_20
$012A $3000             MOVLW 0
$012B $18A1             BTFSC _j, 1
$012C $3001             MOVLW 1
$012D $00F1             MOVWF STACK_1
$012E $0871             MOVF  STACK_1, 0
$012F $1D03             BTFSS STATUS, Z
$0130 $2933             GOTO  L_main_20
$0131 $     L178_ex_L_main_20:
$0131 $01F0             CLRF  STACK_0, 1
$0132 $2935             GOTO  L_main_19
$0133 $     L_main_20:
$0133 $3001             MOVLW 1
$0134 $00F0             MOVWF STACK_0
$0135 $     L_main_19:
$0135 $3000             MOVLW 0
$0136 $1870             BTFSC STACK_0, 0
$0137 $3020             MOVLW 32
$0138 $0608             XORWF PORTD, 0
$0139 $3920             ANDLW 32
$013A $0688             XORWF PORTD, 1
;KONVERTOR.c,49 ::            PORTD.F4=((J.F1&&!(J.F0))||(!(J.F1)&&J.F0));
$013B $3000             MOVLW 0
$013C $18A1             BTFSC _j, 1
$013D $3001             MOVLW 1
$013E $00F1             MOVWF STACK_1
$013F $0871             MOVF  STACK_1, 0
$0140 $1903             BTFSC STATUS, Z
$0141 $2944             GOTO  L199_ex_L_main_26
$0142 $1C21             BTFSS _j, 0
$0143 $294D             GOTO  L_main_26
$0144 $     L199_ex_L_main_26:
$0144 $18A1             BTFSC _j, 1
$0145 $294B             GOTO  L211_ex_L_main_26
$0146 $3001             MOVLW 1
$0147 $0521             ANDWF _j, 0
$0148 $00F0             MOVWF STACK_0
$0149 $1D03             BTFSS STATUS, Z
```

```
$014A $294D             GOTO  L_main_26
$014B $     L211_ex_L_main_26:
$014B $01F0             CLRF  STACK_0, 1
$014C $294F             GOTO  L_main_25
$014D $     L_main_26:
$014D $3001             MOVLW 1
$014E $00F0             MOVWF STACK_0
$014F $     L_main_25:
$014F $3000             MOVLW 0
$0150 $1870             BTFSC STACK_0, 0
$0151 $3010             MOVLW 16
$0152 $0608             XORWF PORTD, 0
$0153 $3910             ANDLW 16
$0154 $0688             XORWF PORTD, 1
;KONVERTOR.c,50 ::            PORTD.F3=K.F3;
$0155 $3000             MOVLW 0
$0156 $19A2             BTFSC _k, 3
$0157 $3001             MOVLW 1
$0158 $00F1             MOVWF STACK_1
$0159 $3000             MOVLW 0
$015A $1871             BTFSC STACK_1, 0
$015B $3008             MOVLW 8
$015C $0608             XORWF PORTD, 0
$015D $3908             ANDLW 8
$015E $0688             XORWF PORTD, 1
;KONVERTOR.c,51 ::            PORTD.F2=((K.F3&&!(K.F2))||(!(K.F3)&&K.F2));
$015F $3000             MOVLW 0
$0160 $19A2             BTFSC _k, 3
$0161 $3001             MOVLW 1
$0162 $00F1             MOVWF STACK_1
$0163 $0871             MOVF  STACK_1, 0
$0164 $1903             BTFSC STATUS, Z
$0165 $2968             GOTO  L237_ex_L_main_32
$0166 $1D22             BTFSS _k, 2
$0167 $2973             GOTO  L_main_32
$0168 $     L237_ex_L_main_32:
$0168 $19A2             BTFSC _k, 3
$0169 $2971             GOTO  L251_ex_L_main_32
$016A $3000             MOVLW 0
$016B $1922             BTFSC _k, 2
$016C $3001             MOVLW 1
$016D $00F1             MOVWF STACK_1
$016E $0871             MOVF  STACK_1, 0
$016F $1D03             BTFSS STATUS, Z
$0170 $2973             GOTO  L_main_32
$0171 $     L251_ex_L_main_32:
$0171 $01F0             CLRF  STACK_0, 1
$0172 $2975             GOTO  L_main_31
$0173 $     L_main_32:
$0173 $3001             MOVLW 1
$0174 $00F0             MOVWF STACK_0
$0175 $     L_main_31:
$0175 $3000             MOVLW 0
$0176 $1870             BTFSC STACK_0, 0
$0177 $3004             MOVLW 4
$0178 $0608             XORWF PORTD, 0
$0179 $3904             ANDLW 4
$017A $0688             XORWF PORTD, 1
;KONVERTOR.c,52 ::            PORTD.F1=((K.F2&&!(K.F1))||(!(K.F2)&&K.F1));
```

```
$017B $3000             MOVLW 0
$017C $1922             BTFSC _k, 2
$017D $3001             MOVLW 1
$017E $00F1             MOVWF STACK_1
$017F $0871             MOVF  STACK_1, 0
$0180 $1903             BTFSC STATUS, Z
$0181 $2984             GOTO  L272_ex_L_main_38
$0182 $1CA2             BTFSS _k, 1
$0183 $298F             GOTO  L_main_38
$0184 $     L272_ex_L_main_38:
$0184 $1922             BTFSC _k, 2
$0185 $298D             GOTO  L286_ex_L_main_38
$0186 $3000             MOVLW 0
$0187 $18A2             BTFSC _k, 1
$0188 $3001             MOVLW 1
$0189 $00F1             MOVWF STACK_1
$018A $0871             MOVF  STACK_1, 0
$018B $1D03             BTFSS STATUS, Z
$018C $298F             GOTO  L_main_38
$018D $     L286_ex_L_main_38:
$018D $01F0             CLRF  STACK_0, 1
$018E $2991             GOTO  L_main_37
$018F $     L_main_38:
$018F $3001             MOVLW 1
$0190 $00F0             MOVWF STACK_0
$0191 $     L_main_37:
$0191 $3000             MOVLW 0
$0192 $1870             BTFSC STACK_0, 0
$0193 $3002             MOVLW 2
$0194 $0608             XORWF PORTD, 0
$0195 $3902             ANDLW 2
$0196 $0688             XORWF PORTD, 1
;KONVERTOR.c,53 ::             PORTD.F0=((K.F1&&!(K.F0))||(!(K.F1)&&K.F0));
$0197 $3000             MOVLW 0
$0198 $18A2             BTFSC _k, 1
$0199 $3001             MOVLW 1
$019A $00F1             MOVWF STACK_1
$019B $0871             MOVF  STACK_1, 0
$019C $1903             BTFSC STATUS, Z
$019D $29A0             GOTO  L307_ex_L_main_44
$019E $1C22             BTFSS _k, 0
$019F $29A9             GOTO  L_main_44
$01A0 $     L307_ex_L_main_44:
$01A0 $18A2             BTFSC _k, 1
$01A1 $29A7             GOTO  L319_ex_L_main_44
$01A2 $3001             MOVLW 1
$01A3 $0522             ANDWF _k, 0
$01A4 $00F0             MOVWF STACK_0
$01A5 $1D03             BTFSS STATUS, Z
$01A6 $29A9             GOTO  L_main_44
$01A7 $     L319_ex_L_main_44:
$01A7 $01F0             CLRF  STACK_0, 1
$01A8 $29AB             GOTO  L_main_43
$01A9 $     L_main_44:
$01A9 $3001             MOVLW 1
$01AA $00F0             MOVWF STACK_0
$01AB $     L_main_43:
$01AB $3000             MOVLW 0
$01AC $1870             BTFSC STACK_0, 0
```

```
$01AD $3001             MOVLW 1
$01AE $0608             XORWF PORTD, 0
$01AF $3901             ANDLW 1
$01B0 $0688             XORWF PORTD, 1
;KONVERTOR.c,55 ::           }
$01B1 $     L_main_8:
;KONVERTOR.c,56 ::           }
$01B1 $     L_main_2:
;KONVERTOR.c,57 ::           if (PORTC.F4==1) {
$01B1 $3000             MOVLW 0
$01B2 $1A07             BTFSC PORTC, 4
$01B3 $3001             MOVLW 1
$01B4 $00F1             MOVWF STACK_1
$01B5 $0871             MOVF  STACK_1, 0
$01B6 $3A01             XORLW 1
$01B7 $1D03             BTFSS STATUS, Z
$01B8 $2ABB             GOTO  L_main_45
;KONVERTOR.c,60 ::           if (Usart_Data_Ready()) {
$01B9 $2004             CALL  _Usart_Data_Ready
$01BA $0870             MOVF  STACK_0, 0
$01BB $1903             BTFSC STATUS, Z
$01BC $29C3             GOTO  L_main_46
;KONVERTOR.c,61 ::           i = Usart_Read();
$01BD $200D             CALL  _Usart_Read
$01BE $0870             MOVF  STACK_0, 0
$01BF $00A5             MOVWF _i
;KONVERTOR.c,62 ::           Usart_Write(i);
$01C0 $0870             MOVF  STACK_0, 0
$01C1 $00A6             MOVWF FARG_Usart_Write+0
$01C2 $2018             CALL  _Usart_Write
;KONVERTOR.c,63 ::           }
$01C3 $     L_main_46:
;KONVERTOR.c,65 ::           j = (i & 0xF0) >> 4;
$01C3 $30F0             MOVLW 240
$01C4 $0525             ANDWF _i, 0
$01C5 $00A1             MOVWF _j
$01C6 $0CA1             RRF   _j, 1
$01C7 $13A1             BCF   _j, 7
$01C8 $0CA1             RRF   _j, 1
$01C9 $13A1             BCF   _j, 7
$01CA $0CA1             RRF   _j, 1
$01CB $13A1             BCF   _j, 7
$01CC $0CA1             RRF   _j, 1
$01CD $13A1             BCF   _j, 7
;KONVERTOR.c,66 ::           k = i & 0x0F;
$01CE $300F             MOVLW 15
$01CF $0525             ANDWF _i, 0
$01D0 $00A2             MOVWF _k
;KONVERTOR.c,67 ::           PORTB = i;
$01D1 $0825             MOVF  _i, 0
$01D2 $0086             MOVWF PORTB
;KONVERTOR.c,68 ::           PORTD = 0x00;
$01D3 $0188             CLRF  PORTD, 1
;KONVERTOR.c,70 ::           if ((PORTC.F3==0)&&(k<=9)&&(j<=9)) {
$01D4 $3000             MOVLW 0
$01D5 $1987             BTFSC PORTC, 3
$01D6 $3001             MOVLW 1
$01D7 $00F1             MOVWF STACK_1
$01D8 $0871             MOVF  STACK_1, 0
```

```
$01D9 $3A00             XORLW 0
$01DA $1D03             BTFSS STATUS, Z
$01DB $29FB             GOTO  L_main_49
$01DC $0822             MOVF  _k, 0
$01DD $3C09             SUBLW 9
$01DE $1C03             BTFSS STATUS, C
$01DF $29FB             GOTO  L_main_49
$01E0 $0821             MOVF  _j, 0
$01E1 $3C09             SUBLW 9
$01E2 $1C03             BTFSS STATUS, C
$01E3 $29FB             GOTO  L_main_49
$01E4 $     L366_ex_L_main_49:
;KONVERTOR.c,72 ::             l = (j + 3) << 4;
$01E4 $3003             MOVLW 3
$01E5 $0721             ADDWF _j, 0
$01E6 $00F2             MOVWF STACK_2
$01E7 $0872             MOVF  STACK_2, 0
$01E8 $00F1             MOVWF STACK_1
$01E9 $0DF1             RLF   STACK_1, 1
$01EA $1071             BCF   STACK_1, 0
$01EB $0DF1             RLF   STACK_1, 1
$01EC $1071             BCF   STACK_1, 0
$01ED $0DF1             RLF   STACK_1, 1
$01EE $1071             BCF   STACK_1, 0
$01EF $0DF1             RLF   STACK_1, 1
$01F0 $1071             BCF   STACK_1, 0
$01F1 $0871             MOVF  STACK_1, 0
$01F2 $00A3             MOVWF _l
;KONVERTOR.c,73 ::             m = k + 3;
$01F3 $3003             MOVLW 3
$01F4 $0722             ADDWF _k, 0
$01F5 $00F0             MOVWF STACK_0
$01F6 $0870             MOVF  STACK_0, 0
$01F7 $00A4             MOVWF _m
;KONVERTOR.c,74 ::             PORTD = l+m;
$01F8 $0870             MOVF  STACK_0, 0
$01F9 $0771             ADDWF STACK_1, 0
$01FA $0088             MOVWF PORTD
;KONVERTOR.c,76 ::             }
$01FB $     L_main_49:
;KONVERTOR.c,78 ::             if (PORTC.F3==1) {
$01FB $3000             MOVLW 0
$01FC $1987             BTFSC PORTC, 3
$01FD $3001             MOVLW 1
$01FE $00F1             MOVWF STACK_1
$01FF $0871             MOVF  STACK_1, 0
$0200 $3A01             XORLW 1
$0201 $1D03             BTFSS STATUS, Z
$0202 $2ABB             GOTO  L_main_50
;KONVERTOR.c,80 ::             PORTD.F7=J.F3;
$0203 $3000             MOVLW 0
$0204 $19A1             BTFSC _j, 3
$0205 $3001             MOVLW 1
$0206 $00F1             MOVWF STACK_1
$0207 $3000             MOVLW 0
$0208 $1871             BTFSC STACK_1, 0
$0209 $3080             MOVLW 128
$020A $0608             XORWF PORTD, 0
$020B $3980             ANDLW 128
```

```
$020C $0688             XORWF PORTD, 1
;KONVERTOR.c,81 ::            PORTD.F6=((J.F3&&!(J.F2))||(!(J.F3)&&J.F2));
$020D $3000             MOVLW 0
$020E $19A1             BTFSC _j, 3
$020F $3001             MOVLW 1
$0210 $00F1             MOVWF STACK_1
$0211 $0871             MOVF  STACK_1, 0
$0212 $1903             BTFSC STATUS, Z
$0213 $2A16             GOTO  L400_ex_L_main_56
$0214 $1D21             BTFSS _j, 2
$0215 $2A21             GOTO  L_main_56
$0216 $     L400_ex_L_main_56:
$0216 $19A1             BTFSC _j, 3
$0217 $2A1F             GOTO  L414_ex_L_main_56
$0218 $3000             MOVLW 0
$0219 $1921             BTFSC _j, 2
$021A $3001             MOVLW 1
$021B $00F1             MOVWF STACK_1
$021C $0871             MOVF  STACK_1, 0
$021D $1D03             BTFSS STATUS, Z
$021E $2A21             GOTO  L_main_56
$021F $     L414_ex_L_main_56:
$021F $01F0             CLRF  STACK_0, 1
$0220 $2A23             GOTO  L_main_55
$0221 $     L_main_56:
$0221 $3001             MOVLW 1
$0222 $00F0             MOVWF STACK_0
$0223 $     L_main_55:
$0223 $3000             MOVLW 0
$0224 $1870             BTFSC STACK_0, 0
$0225 $3040             MOVLW 64
$0226 $0608             XORWF PORTD, 0
$0227 $3940             ANDLW 64
$0228 $0688             XORWF PORTD, 1
;KONVERTOR.c,82 ::            PORTD.F5=((J.F2&&!(J.F1))||(!(J.F2)&&J.F1));
$0229 $3000             MOVLW 0
$022A $1921             BTFSC _j, 2
$022B $3001             MOVLW 1
$022C $00F1             MOVWF STACK_1
$022D $0871             MOVF  STACK_1, 0
$022E $1903             BTFSC STATUS, Z
$022F $2A32             GOTO  L435_ex_L_main_62
$0230 $1CA1             BTFSS _j, 1
$0231 $2A3D             GOTO  L_main_62
$0232 $     L435_ex_L_main_62:
$0232 $1921             BTFSC _j, 2
$0233 $2A3B             GOTO  L449_ex_L_main_62
$0234 $3000             MOVLW 0
$0235 $18A1             BTFSC _j, 1
$0236 $3001             MOVLW 1
$0237 $00F1             MOVWF STACK_1
$0238 $0871             MOVF  STACK_1, 0
$0239 $1D03             BTFSS STATUS, Z
$023A $2A3D             GOTO  L_main_62
$023B $     L449_ex_L_main_62:
$023B $01F0             CLRF  STACK_0, 1
$023C $2A3F             GOTO  L_main_61
$023D $     L_main_62:
$023D $3001             MOVLW 1
```

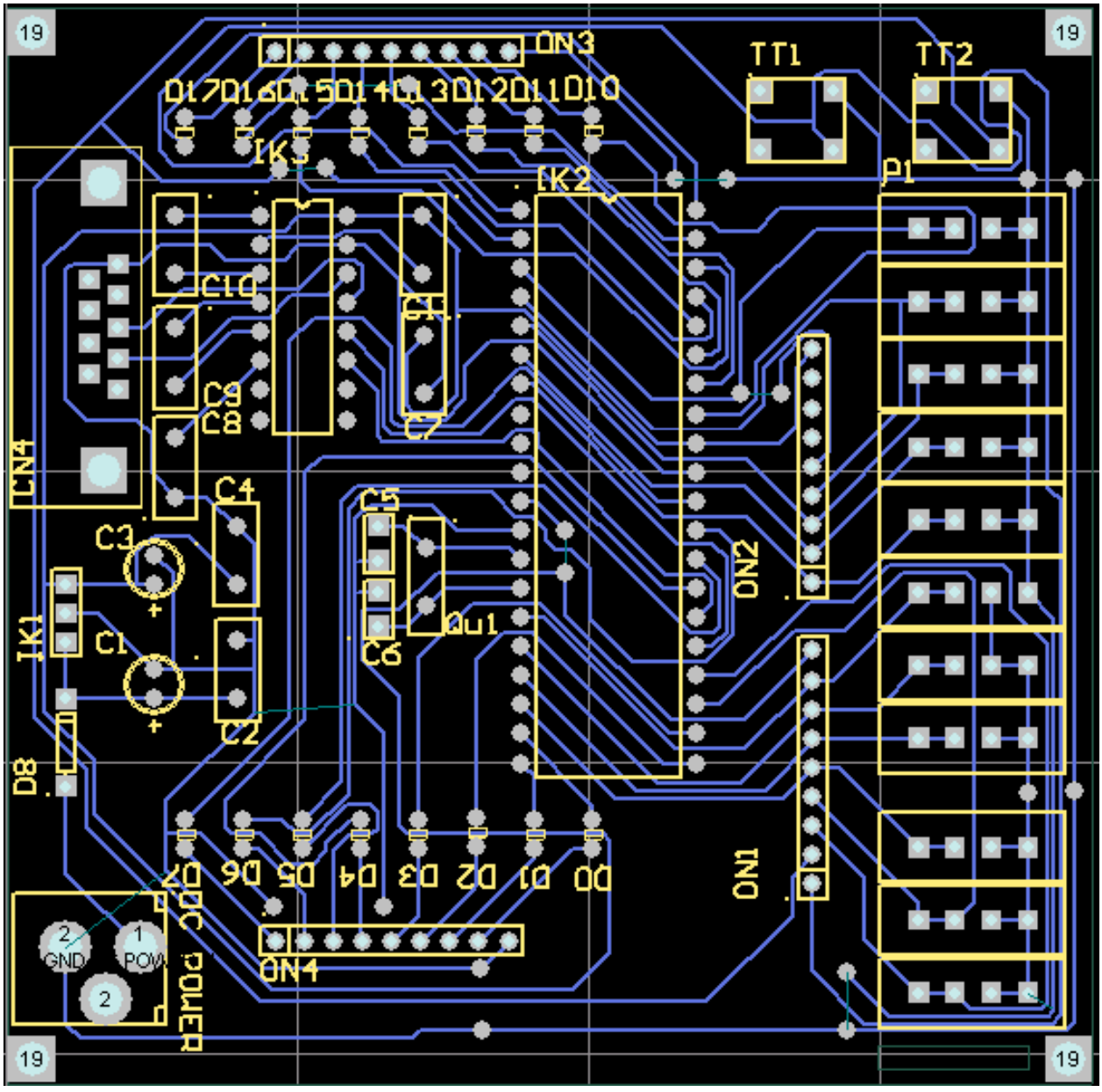
```
$023E $00F0             MOVWF STACK_0
$023F $     L_main_61:
$023F $3000             MOVLW 0
$0240 $1870             BTFSC STACK_0, 0
$0241 $3020             MOVLW 32
$0242 $0608             XORWF PORTD, 0
$0243 $3920             ANDLW 32
$0244 $0688             XORWF PORTD, 1
;KONVERTOR.c,83 ::           PORTD.F4=((J.F1&&!(J.F0))||(!(J.F1)&&J.F0));
$0245 $3000             MOVLW 0
$0246 $18A1             BTFSC _j, 1
$0247 $3001             MOVLW 1
$0248 $00F1             MOVWF STACK_1
$0249 $0871             MOVF  STACK_1, 0
$024A $1903             BTFSC STATUS, Z
$024B $2A4E             GOTO  L470_ex_L_main_68
$024C $1C21             BTFSS _j, 0
$024D $2A57             GOTO  L_main_68
$024E $     L470_ex_L_main_68:
$024E $18A1             BTFSC _j, 1
$024F $2A55             GOTO  L482_ex_L_main_68
$0250 $3001             MOVLW 1
$0251 $0521             ANDWF _j, 0
$0252 $00F0             MOVWF STACK_0
$0253 $1D03             BTFSS STATUS, Z
$0254 $2A57             GOTO  L_main_68
$0255 $     L482_ex_L_main_68:
$0255 $01F0             CLRF  STACK_0, 1
$0256 $2A59             GOTO  L_main_67
$0257 $     L_main_68:
$0257 $3001             MOVLW 1
$0258 $00F0             MOVWF STACK_0
$0259 $     L_main_67:
$0259 $3000             MOVLW 0
$025A $1870             BTFSC STACK_0, 0
$025B $3010             MOVLW 16
$025C $0608             XORWF PORTD, 0
$025D $3910             ANDLW 16
$025E $0688             XORWF PORTD, 1
;KONVERTOR.c,84 ::           PORTD.F3=K.F3;
$025F $3000             MOVLW 0
$0260 $19A2             BTFSC _k, 3
$0261 $3001             MOVLW 1
$0262 $00F1             MOVWF STACK_1
$0263 $3000             MOVLW 0
$0264 $1871             BTFSC STACK_1, 0
$0265 $3008             MOVLW 8
$0266 $0608             XORWF PORTD, 0
$0267 $3908             ANDLW 8
$0268 $0688             XORWF PORTD, 1
;KONVERTOR.c,85 ::           PORTD.F2=((K.F3&&!(K.F2))||(!(K.F3)&&K.F2));
$0269 $3000             MOVLW 0
$026A $19A2             BTFSC _k, 3
$026B $3001             MOVLW 1
$026C $00F1             MOVWF STACK_1
$026D $0871             MOVF  STACK_1, 0
$026E $1903             BTFSC STATUS, Z
$026F $2A72             GOTO  L508_ex_L_main_74
$0270 $1D22             BTFSS _k, 2
```

```
$0271 $2A7D             GOTO  L_main_74
$0272 $     L508_ex_L_main_74:
$0272 $19A2             BTFSC _k, 3
$0273 $2A7B             GOTO  L522_ex_L_main_74
$0274 $3000             MOVLW 0
$0275 $1922             BTFSC _k, 2
$0276 $3001             MOVLW 1
$0277 $00F1             MOVWF STACK_1
$0278 $0871             MOVF  STACK_1, 0
$0279 $1D03             BTFSS STATUS, Z
$027A $2A7D             GOTO  L_main_74
$027B $     L522_ex_L_main_74:
$027B $01F0             CLRF  STACK_0, 1
$027C $2A7F             GOTO  L_main_73
$027D $     L_main_74:
$027D $3001             MOVLW 1
$027E $00F0             MOVWF STACK_0
$027F $     L_main_73:
$027F $3000             MOVLW 0
$0280 $1870             BTFSC STACK_0, 0
$0281 $3004             MOVLW 4
$0282 $0608             XORWF PORTD, 0
$0283 $3904             ANDLW 4
$0284 $0688             XORWF PORTD, 1
;KONVERTOR.c,86 ::            PORTD.F1=((K.F2&&!(K.F1))||(!(K.F2)&&K.F1));
$0285 $3000             MOVLW 0
$0286 $1922             BTFSC _k, 2
$0287 $3001             MOVLW 1
$0288 $00F1             MOVWF STACK_1
$0289 $0871             MOVF  STACK_1, 0
$028A $1903             BTFSC STATUS, Z
$028B $2A8E             GOTO  L543_ex_L_main_80
$028C $1CA2             BTFSS _k, 1
$028D $2A99             GOTO  L_main_80
$028E $     L543_ex_L_main_80:
$028E $1922             BTFSC _k, 2
$028F $2A97             GOTO  L557_ex_L_main_80
$0290 $3000             MOVLW 0
$0291 $18A2             BTFSC _k, 1
$0292 $3001             MOVLW 1
$0293 $00F1             MOVWF STACK_1
$0294 $0871             MOVF  STACK_1, 0
$0295 $1D03             BTFSS STATUS, Z
$0296 $2A99             GOTO  L_main_80
$0297 $     L557_ex_L_main_80:
$0297 $01F0             CLRF  STACK_0, 1
$0298 $2A9B             GOTO  L_main_79
$0299 $     L_main_80:
$0299 $3001             MOVLW 1
$029A $00F0             MOVWF STACK_0
$029B $     L_main_79:
$029B $3000             MOVLW 0
$029C $1870             BTFSC STACK_0, 0
$029D $3002             MOVLW 2
$029E $0608             XORWF PORTD, 0
$029F $3902             ANDLW 2
$02A0 $0688             XORWF PORTD, 1
;KONVERTOR.c,87 ::            PORTD.F0=((K.F1&&!(K.F0))||(!(K.F1)&&K.F0));
$02A1 $3000             MOVLW 0
```

```
$02A2 $18A2             BTFSC _k, 1
$02A3 $3001             MOVLW 1
$02A4 $00F1             MOVWF STACK_1
$02A5 $0871             MOVF  STACK_1, 0
$02A6 $1903             BTFSC STATUS, Z
$02A7 $2AAA             GOTO  L578_ex_L_main_86
$02A8 $1C22             BTFSS _k, 0
$02A9 $2AB3             GOTO  L_main_86
$02AA $     L578_ex_L_main_86:
$02AA $18A2             BTFSC _k, 1
$02AB $2AB1             GOTO  L590_ex_L_main_86
$02AC $3001             MOVLW 1
$02AD $0522             ANDWF _k, 0
$02AE $00F0             MOVWF STACK_0
$02AF $1D03             BTFSS STATUS, Z
$02B0 $2AB3             GOTO  L_main_86
$02B1 $     L590_ex_L_main_86:
$02B1 $01F0             CLRF  STACK_0, 1
$02B2 $2AB5             GOTO  L_main_85
$02B3 $     L_main_86:
$02B3 $3001             MOVLW 1
$02B4 $00F0             MOVWF STACK_0
$02B5 $     L_main_85:
$02B5 $3000             MOVLW 0
$02B6 $1870             BTFSC STACK_0, 0
$02B7 $3001             MOVLW 1
$02B8 $0608             XORWF PORTD, 0
$02B9 $3901             ANDLW 1
$02BA $0688             XORWF PORTD, 1
;KONVERTOR.c,89 ::           }
$02BB $     L_main_50:
;KONVERTOR.c,90 ::           }
$02BB $     L_main_45:
;KONVERTOR.c,93 ::           } while (1);
$02BB $2854             GOTO  L_main_0
;KONVERTOR.c,94 ::           }//~!
$02BC $2ABC             GOTO  $
```

P6. LABORATORIJSKA VEŽBA

Najpre ćemo na *Slici 6.1.* prikazati izgled štampane ploče sa raspored komponenti na njoj.



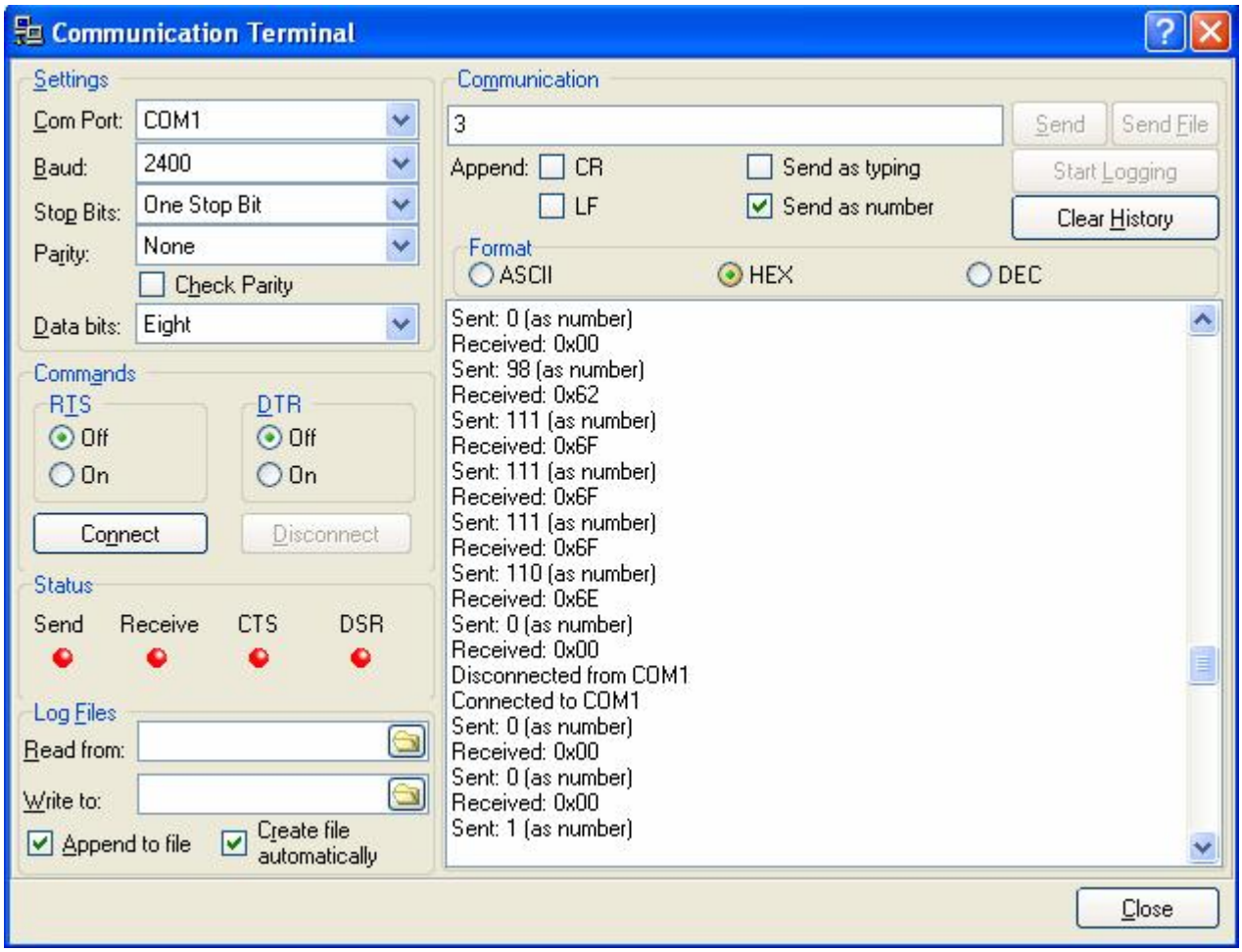
Slici 6.1. Štampana ploča sa komponentama konvertora

Zadatak studenta se sastoji u tome da provere rad realizovanog konvertora za kombinacije date u tabeli ispod.

Kombinacija (RC5-RC3)	Zadata 8-bitna reč	Transformisana 8-bitna reč
000	binarni kod	“višak 3”
001	binarni kod	Grejov kod
010	binarni kod (PC)	“višak 3”
011	binarni kod (PC)	Grejov kod

Nakon provere rada konvertora za četiri kombinacije, studenti treba da daju predlog programskog koda u mikroC-u za bar još dve transformacije. Na primer, transformacije iz «višak 3» u binarni kod, Grejovog koda u binarni itd.

Na Slici 6.2. data su podešavanja u programu USART terminal za izvođenje laboratotijske vežbe.



Slici 6.2. Izgled prozora UART terminala sa podešenim parametrima

P7. REFERENCE

[1] ***PIC16F87X***, *Datasheet, Microchip Tecnology Inc,* *http://www.microchip.com*

[2] ***Razvojni sistem za PIC16F877A***, *B. Dimitrijević, S. Aleksić, I. Antić, Elektronski fakultet - Niš, (seminarski rad),* *http://es.elfak.ni.ac.yu/students.html*

[3] ***Serijska komunikacija računara i USART modula mikrokontrolera PIC16F877A***, *Dejan Lazić, Ivan Novaković, Elektronski fakultet - Niš, (seminarski rad),* *http://es.elfak.ni.ac.yu/students.html*

[4] ***Impulsna i digitalna elektronika***, *S. Lubura, Elektrotehnički fakultet - Istočno Sarajevo,* *http://www.etf.unssa.rs.ba/~slubura/Impulsna_i_digitalna_elektronika*

[5] ***Binarno kodirani dekadni brojevi***, *Pririodno-matematički fakultet - Beograd,* *http://www.matf.bg.ac.yu/~nenad/uor/6.decimal.pdf*

[6] ***ASCII tabela***, *http://www.otpornik.com*

[7] ***Softver mikroC***, *Mikroelektronika - Beograd,* *http://mikroe.com*

[8] ***IC-Prog, Version 1.05***, *http://www.ic-prog.com*

P8. CV

Prezime:	**Stojanović**
Ime:	**Perica**
Adresa stanovanja:	**Predejane, opština Leskovac**
***E-mail* adresa:**	
Bračno stanje:	**neoženjen**
Datum rođenja:	**08.11.1980.**
Nacionalnost:	**Srbin**
Obrazovanje:	**Tehnička škola, Vladičin han**
Diploma:	**IV stepen**
Vozačka dozvola:	**B kategorija**
Poznavanje jezika	**Ruski (dobro)**
Poznavanje operativnih sistema i programskih paketa	**WINDOWS, MS OFFICE, PHOTOSHOP, EMBEDDED C, EAGLE, PROTEL**
Posebna interesovanja	**Čitanje beletristike, programiranje**
Sposobnosti	**Upornost, preduzimljivost, kreativnost**
Napomene	**Nosilac Vukove diplome u osnovnoj i srednjoj školi.**

ZAVRŠNA REČ

U prethodnim redovima je opisan način praktične realizacije i korišćenje jednostavnog konvertora kodova realizovanog pomoću mikrokontrolera PIC16F877-04.

Prvo poglavlje seminarskog rada se bavi najpoznatije kodovima koji se koriste u računarskoj tehnici: težinski (binarni, BCD – 8421, 2421, itd.) i netežinski (Grejov, «višak tri»). Na kraju poglavlja je ukratko opisan ASCII kod.

Kratak opis mikrokontrolera PIC16F877-04 izložen je u drugom poglavlju rada. Prvo je predstavljena arhitektura mikrokontrolera, zatim organizacija memorije, zatim portovi i tajmeri, da bi na kraju bili predstavljeni periferijski komunikacioni moduli kontrolera.

Za komunikaciju mikrokontrolera i konvertora kodova koristi se softverski UART terminal, čije su karakteristike i parametri opisani u trećem poglavlju.

U četvrtom poglavlju rada je izložen postupak programiranja mikrokontrolera PIC16F877-04. Članak, uz određene korekcije u tekstu, je preuzet iz časopisa Info Elektronika broj 53. Za programiranje mikrokontrolera je, naime, potreban hardverski (*bootstrap loader AllPIC*) i softverski alat (*IC-Prog*). Najbitnije je da se u softverskom alatu izvrše pravilna podešavanja parametara.

Električna šema sa opisom parametara konvertora kodova je predstavljena u petom poglavlju rada, dok je u šestom poglavlju predstavljena laboratorijska vežba sa zadatkom za studente.

Na samom kraju rada dat je spisak referenci, kao i CV autora seminarskog rada.