

# Klase

Dragan Golubović, dipl.ing.

# u ARB51

Treći nastavak iz serije o objektno orijentisanom programskom interfejsu – ARB 8051 biće posvećen realno najinteresantnijem i najmoćnijem svojstvu ove biblioteke, a to je mogućnost formiranja sopstvenih objekata i sopstvene programske sintakse. Ljubitelji mikrokontrolera, a ujedno poznavaooci programskog jezika C++ ovom bibliotekom dobijaju gotovo neograničene mogućnosti da svoje najčešće korišćene strukture podataka i procedure vezane za njih grupišu u odgovarajuće klase i tako znatno unaprede svoju programersku produktivnost. Ovo područje primene biblioteke ARB predstavlja po našem mišljenju potencijalno njenu najperspektivniji domen gde druge alatke – assembleri i C kroskompajleri ne mogu da u potpunosti odgovore na sve naše zahteve.

**D**ok primeri predstavljeni u brojevima 3 i 4 časopisa "Mikroelektronika" ne zahtevaju neko naročito znanje iz viših programskih jezika (razumevanje nekih primera ne zahteva gotovo nikakvo znanje iz C/C++!), za praćenje narednog teksta podrazumeva se poznavanje osnovnih postulata objektno orijentisanog programiranja. Integrisano kolo 8255 je dobro poznato svima onima koji su imali potrebu da prošire broj ulazno-izlaznih portova mikrokontrolera 8051.

Naš prvi primer, prikazan u datoteci ppi.h, demonstrira mogućnost pristupa internim registrima ovog kola na način koji će poboljšati inače najranjiviju osobinu svakog sistemskog programera – produktivnost. Registri 8255 se po pravilu postavljaju u memorijsku mapu mikrokontrolera i pristupa im se tako što se najpre postavi adresa u neki registar – pokazivač (dptr,r0,r1) a zatim izvrši instrukcija transfera na relaciji spoljna memorija – akumulator. To praktično znači da se ovim registrima pristupa posredno, pomoću dve instrukcije. Formiranjem objekta ppi stvorimo priliku da sa programerskog stanovišta ovim registrima pristupamo kao registrima mikrokontrolera. Objekat tipa ppi sadrži samo jedan skriveni podatak (int index) kojim se definiše lokacija pojedinog registra u odnosu na adresu kola (npr. PPI\_ADR=0x4010). Konstruktorskom funkcijom ppi(int a) stvaramo 4 objekta tipa ppi i to na sledeći način:

```
ppi portA(0),portB(1),portC(2),control(3);
```

Podrazumeva se, naravno, da registru control inicijalno dodelimo takvu vrednost kojom će se ovo kolo konfigurirati u pogledu orijentacije portova A, B i C. Za programsku liniju oblika:

```
control="10000000";
```

"brine" operatorska funkcija operator=(char \*str), dok je za izraz portA=0x50; zaduena operatorska funkcija operator=(int podatak). Funkcija operator reg &() obavlja prisilnu konverziju objekta ppi u reg što nam omogućava da imamo izraz oblika:

```
a=portC;
```

Ovo sve zapravo znači da varijabli tipa ppi možemo direktno dodeljivati celobrojnu vrednost ali i akumulatoru dodeljivati sadržaj nekog od registara portA, portB ili portC. Zahvaljujući biblioteci ARB, posmatrano iz ugla programera, ovim registrima pristupamo kao da su integralni delovi mikrokontrolera. Naravno, ovu pogodnost obezbedimo u svim našim budućim programima uključivanjem zaglavlja "ppi.h"!

Pre nego pređemo na drugi primer, pozabavićemo se još jednim aspektom programske biblioteke ARB, koji do sada nismo pominjali. Reč je o pokazivačima (pointer). ARB, pored standardnih, poseduje i sopstveni skup pokazivačkih varijabli. U skladu sa arhitekturom mikrokontrolera možemo deklarirati pokazivače na interni RAM, eksterni RAM, pokazivače na programsku memoriju, kao i pokazivače tipa direct i flag.

ARB 8051 podržava sledeće izraze:

```
pointer<xram>x(dptr);//dptr->ekst. RAM
pointer<iram>y(r0); // r0->interni RAM
pointer<code>z(pc);// pc->prog. memoriju
pointer<direct>pDirect;
pointer<flag>pFlag;
```

Pored ovoga, ovim varijablama je dodeljen odgovarajući skup operatora ('\*', '+', '-', '[]', '=',), u saglasju sa setom

## Datoteka PPI.H

```
const int PPI_ADR=0x4010;
class ppi {
int index;
public:
ppi(int a){index=a;}
ppi operator=(int podatak);
ppi operator=(char *str);
operator reg
&(){dptr=PPI_ADR+index;return *dptr;}
};
ppi ppi::operator=(int podatak)
{
dptr=PPI_ADR+index;
a=podatak;
*dptr=a;
return *this;
}
ppi ppi::operator=(char *str)
{
dptr=PPI_ADR+index;
a=str;
*dptr=a;
return *this;
}
```

## Datoteka MOTOR.H

```
#include <stdio.h>
#include "arb8051.h"
const int ADR_BRZINA=0x40;
const int ADR_PREKIDAC=0x90;
class motor {
pointer<direct> BrzinaMotora;
pointer<flag> Prekidac;
public:
static int BrojMotora;
motor(void);
motor(int a);
è motor(){--BrojMotora;}
void CitajBrzinu(void){}
direct & Brzina(void){return
*BrzinaMotora;}
void Start(void){*Prekidac=visok;}
void Stop(void){*Prekidac=nizak;}
friend void Obrada(void);
};
int motor::BrojMotora=0;
motor::motor(void)
{
BrzinaMotora=ADR_BRZINA+BrojMotora;
Prekidac=ADR_PREKIDAC+BrojMotora;
++BrojMotora;
}
motor::motor(int a)
{
BrzinaMotora=ADR_BRZINA+a;
Prekidac=ADR_PREKIDAC+a;
++BrojMotora;
}
void Obrada(void)
{
for(int i=0;i<motor::BrojMotora;i++)
{
motor * pMotor=new motor(i);
pMotor->CitajBrzinu();
delete pMotor;
}
return ;
}
```

instrukcija mikrokontrolera. Posebno æemo rasvetliti poslednja dva sluèaja. Varijablama 'pDirect' i 'pFlag' nije pridru en nijedan registar koji obavlja pokazivaèku funkciju za razliku od predhodnih sluèajeva. Promenljivim ovog tipa mo emo dodeljivati celobrojnu vrednost (pDirect=10), uveæavati ih ili umanjivati (++pflag,—pDirect), dodeljivati ulogu indeksne promenljive (pDirect[5] ili (pDirect+10)) i naravno, ovim varijablama dodeljivati ulogu pokazivaèa (\*pFlag=visok). U pojedinim situacijama, kao u našem sledeæem primeru, upotreba pokazivaèa je po eljna, štaviše neophodna. Pretpostavimo sledeæi sluèaj: potrebno je upravljati radom jednog ili više motora. Mikrokontroler vrši ukljuèivanje i iskljuèivanje motora, obavlja oèitavanje njegove (njihove) brzine i eventualno, na displeju prikazuje

trenutnu brzinu motora. Za naš krajnje pojednostavljen primer, prihvatiaemo da se podatak o brzini sprema u lokaciju direktno adresiranu privatnom promenljivom 'BrzinaMotora', a da se startovanje ( i stopiranje rada ) motora obavlja preko bita direktno adresiranog varijablom 'Prekidac'. Dakle, imamo jednu strukturu podataka koja se sastoji od jednog objekta 'direct' i jednog tipa 'flag'. Buduæi da je moguæe postojanje više objekata tipa 'motor', treba obezbediti da se podaci novih objekata ne preklapaju sa podacima starih, pa se stoga uvodi celobrojna javna varijabla static int BrojMotora koja èuva informaciju o tome koliko je objekata tipa 'motor' do tog momenta formirano. Oèitavanje brzine se vrši u pridru enoj funkciji 'CitajBrzinu()' (koja je u našem sluèaju prazna) i koja u principu zavisi od hardverske konfiguracije ureðaja. Stvaranje objekta je prepušteno konstruktorima motor() i motor(int a). U prvom sluèaju struktura podataka se skladišti u skladu sa vrednošæu varijable 'BrojMotora', a u drugom se pozicija strukture odreðuje eksplicitno preko celobrojnog parametra 'a'. "Prijateljska" funkcija 'Obrada()', u for petlji, objedinjuje izvršenja svih funkcija 'CitajBrzinu()' nezavisno koliko je objekata tipa 'motor' do tog momenta formirano.

Stvaranjem klase 'motor', sebi donosimo silna olakšanja u postupku programiranja. Više ne moramo da vodimo raèuna o tome gde je bit-prekidaè za pojedini motor, niti u kojoj smo lokaciji smestili podatak o trenutnoj brzini nekog motora.

Zakljuèak: biblioteka ARB nudi **novi pogled na programiranje mikrokontrolera**. Sistematski rad na stvaranju novih biblioteka funkcija i klasa doprinosi spajanju dva sveta: assemblera i viših programskih jezika. Pomisao da u sistemski orijentisanom okru enju posedujemo objekte kao što su na primer celi i decimalni brojevi, real-time objekti i procedure, PLC varijable èine inspirativnim sve buduæe napore usmerene ka stvaranju komfornijeg i produktivnijeg programiranja za mikrokontrolere.

### Kontakt:

Za sva pitanja u vezi teksta obratite se autoru na tel. 026 / 223-120 svakog radnog dana između 18 i 19 časova.



Charles Calvert

## DELPHI punom snagom

Pre svega ova knjiga predstavlja relativno potpun i kvalitetan vodiè za programera bez velikog iskustva u pravljenju kompletnih projekata. Naravno, svako razvojno okru enje ima isti cilj, ostaje utisak da je ovaj paket idealan za poèetnike, male timove i individualce. Knjiga u potpunosti prati i objašnjava razvoj pojedinih delova koji uèestvuju u jednom uobièajenom projektu. Treba naglasiti da ovaj paket predstavlja dobar izbor za one korisnike u èijem radu postoji potreba za izradom kvalitetnog korisnièkog interfejsa, da se na jednostavan naèin dodaje "motor" odnosno program koji æe nešto konkretno da radi. Dalje, u poslednjih nekoliko poglavlja dobro je objašnjeno OOP ili objektno orijentisano programiranje. Ova tema je, naravno, veoma opširna i ne mo e se opisati na malo prostora, ali ono što ovde proèitate je više nego dovoljno za kvalitetan start i dalji samostalni napredak. Navedimo i nekoliko zamerki. Knjiga je napisana za prvu verziju Delphi-ja, a sada je veæ izdata i èetvrta verzija, zbog toga pojedina objašnjenja koja su data u knjizi æe biti nepotpuna ili èak i netaèna. Naravno, ovo æe te primetiti jedino ako jako pa ljivo èitate knjigu ili ukoliko vam trebaju neki specifièni delovi alat koji se u novijim verzijama programa prebaèeni u neki drugi deo menija. Ipak ostaje utisak da je ova knjiga pravi izbor za uèenje kako samog jezika tako i alata. Naravno, ni druge knjige, koje su izdate na našem jeziku, nije loše imati na polici, ali stièe se utisak da "Delphi programiranje punom snagom" predstavlja najpotpunije rešenje. **B.I.**

Izdavaè:

CET Computer Equipment and Trade Skadarska 45  
11000 Beograd  
tel: 011 3243-043, 3237-246  
fax: 011 3235-139