

OEMmodule 486

486SX 100MHz PC/AT
Single Component
Computer



OEMmodule 486 Single Component Computer (SCC) verovatno predstavlja najbrži i najjeftiniji put da se u aplikaciju uključi jedan PC/AT računar. Kombinacijom hardvera, softvera u jenom čipu obezbeđuje se pouzdaniji rad ali i brži izlazak proizvoda na tržište. Mala potrošnja i veličina manja od kreditne kartice čine ovaj proizvod idealnim za aplikacije koje zahtevaju intenzivna izračunavanja uz timski rad i PC kompatibilnost. Unutar modula se nalaze i BIOS i DOS, čiju licencu kupujete zajedno sa modulom. Ovo je od izuzetne važnosti jer su u cenu modula uračunate i dve licence.

Program se razvija na računaru i onda učitava u modul. Za razvoj se mogu koristiti svi DOS programski alati prisutni kod standardnih PC računara.

Zahvaljujući integrisanom rešenju nije potrebno razvijati više odvojenih modula čiji kontakti usled industrijskih uslova rada (prašina, vibracije..) mogu vremenom otkazati. OEMmodule486 je potpuno kompatibilan sa PC/AT industrijskim standardom ISA čime se omogućuje priključenje velikog izbora hardvera od hiljada firmi koje razvijaju svoje uređaje na ISA arhitekturi.

OEMmodule 486 sadrži sve standardne osobine matične ploče:

- 100MHz CPU
- 2Mb internog DRAM-a. Sistem se može proširiti sa spoljnim RAM-om do 64 Mb-ta.
- 2 serijska i jedan paralelni port
- Flopi i IDE disk kontroler za dva uređaja (npr. HDD+CD)
- Kontroler AT tastature
- AT kompatibilan BIOS
- Caldera DR-DOS učitani u internoj memoriji.
- Korisnički FLASH disk od 2 Mb-ta

Pored ovih klasičnih sadržaja svakog računara postoje i delovi koji se koriste u korisničkim aplikacijama kao što su zaštita od grešaka u programu i softverska promena brzine izvršavanja programa za potrebe štednje energije.

OEMmodule486 koristi jedno napajanje od 5 V i troši oko 2.5 W pri radu na 100MHz-a.

Dimenzije modula su 56X76X12 mm sa 240 izvoda na rastojanju od 0.04 inča za površinsku montažu.

Za više informacija o OEMmodule486 obratite se proizvođaču na:

ZF Microsystems,inc
1052 Elwell Court
Palo Alto, CA 94303
USA
www.zfmicro.com

Real - Time sistemi



cMCX11

Real Time operativni sistem u programskom jeziku C

Asembler je brz, C je elegantan. Kombinovanjem ova dva programska jezika na pravi način mogu se dobiti aplikacije kvalitetnije nego korištenjem svakog ponaosob. U real-time operativnim sistemima jezgo ili kernel se obično pravi u assembleru, a ostali deo koda u C-u kako bi se iskoristile prednosti i assemblera i C-a.

Stevan Tošić, dipl. ing.

Programski jezik C je odavno zauzeo svoje mesto u embedid aplikacijama. Prednosti C-a naročito dolaze do izražaja pri izradi složenijih programa. Istina je da programiranje u assembleru maksimalno koristi resurse mikrokontrolera, ali je na mestu pitanje kako ćete nekom drugom objasniti šta ste (ili kako) nešto isprogramirali. U timskom radu ova činjenica može biti vrlo otežavajuća okolnost. Ovdje je reč o izvesno transformaciji RTOS MCX11 koji je napisan u assembleru (IAS11 gdje slovo I označava firmu ImageCraft), u programski jezik C (ImageCraftov kompajler ICC11). Dakle, novi RTOS se zove cMCX11, gde prefiks c simbolizira jezik C. Ideja je bila (i ostvarena) da se iskoristi postojeći MCX11 koji je napisan u assembleru, odnosno da se uradi »omotač« oko assemblerskog kernela kako bi se taskovi mogli programirati u C-u. Reč omotač je uvedena kako bi sugerisala da se na izvestan način sakriva assemblerski kod. Konkretno, dati omotač je u stvari datoteka MCX_LIB.C u kojoj su definisane funkcije u jeziku C, a koje sada zamenjuju kod napisan u assembleru. Drugim rečima, pišemo kod u C-u, a u izvršava se u stvari »sakriveni« assemblerski kod.

Poređenje

Pogledajmo neke delove datoteke MCX_LIB.C. Funkcija .wait. koja se u assembleru poziva sa:

```
ldab #SEMA_x  
swi  
FCB .wait.
```

u C-u se poziva sa:

```
mcxWait(SEMA_x);
```

Dakle, tri linije programskog koda u assembleru se zamenjuju sa jednom naredbom u jeziku C. Dalje se neće navoditi primeri za ostale funkcije (sveukupno postoji 15 funkcija). Namera je bila da se kratko uporede ova dva načina. Radi pojašnjenja kako se koristi cMCX11, pogledajmo delove unutar omotača. Gornja funkcija (uzgred vredi napomenuti i fundamentalna) .wait. je realizovana (datoteka MCX_LIB.C) na sledeći način:

```
void mcxWait(int semaid)
{
asm(» ldab 3,x  »);
asm(» swi  »);
asm(» fcb1  »);
}.
```

Asemblerke funkcije su napisane u C.

Jednostavnost

Za pisanje programa nije potrebno poznavati unutrašnjost datoteke

MCX_LIB.C. Jedino je bitno da se odgovarajući parametri pri pozivu pravilno proslede po tipu i redosledu. Poređenja radi, navodi se program sa taskovim iz prošlog broja zajedno sa novim programom koji je napisan u C-u (Listing 1. i 2.). Ostaje nam da ukratko rezimiramo šta se dobija primenom jezika C (sa strane gledišta programera) – kraći, jednostavniji, čitljiviji, modularniji kod koji je lakši za održavanje. Uvek ostaje mogućnost da se kritični delovi koda napišu u assembleru. Ovo ne znači da oni koji ne poznaju C ne mogu pratiti analizu MCX11. Njima preostaje samo da izvrše inverznu operaciju, odnosno da ono što bude napisano u C-u prevedu u assembler na osnovu datoteke MCX_LIB.C.

Napomena: Za realizaciju cMCX11 je korišten ImageCraftov kompajler ICC11. Autori ICC11: Richard Man i Christina Willrich.

Ideja za MCX11: Brian Dombrowski.

Autor cMCX11: Grant Beattie.

web:

<http://ourworld.compuserve.com/homepages/grantb>.

LISTING 1. Programski jezik C

```
#define TSK1 1 /* Taskovi */
#define TSK2 2
#define TSK3 3
#define SELF 0

#define Sem1 1 /* Semafori */
#define Sem2 2
#define Sem3 3

int main(void)
{
asm(" jmp 0x9000 "); /* Prvo se pokrece Kernel Start-up */
return 0;
}

void Task1(void)
{
/* Ovde dolazi kod za: podesavanje IC1,IC2, SPI komunikacije */
/* i porta G (izlazni) */
mcxExecute(TSK2); /* Omoguci tasks 2 */
mcxDelay(0,0,2000/49); /* 2 sekunde kasni */
PORTG|=0x01; /* PG0 - ON */
mcxWait(Sem1); /* Cekaj */
PORTG&=~0x01; /* PG0 - ON */
mcxExecute(TSK3); /* Omoguci task 3 */
mcxTimer(TSK3,Sem3,1500/49); /* Task 3,Semafor 3, 1.5 sek */
mcxTerminate(SELF); /* Kraj */
}

void Task2(void) /* PG0 */
{
static char i;
mcxWait(Sem2); /* Cekaj na semaforu 2 */
for(i=0;i<5;i++){
mcxDelay(0,0,667/49); /* 2/3 sek. kasni */
PORTG|=0x02; /* PG0 - ON */
mcxDelay(0,0,333/49); /* 1/3 sek kasni */
PORTG&=~0x02; /* PG0 - OFF */
prikaz(i);
}
mcxTerminate(SELF); /* Kraj */
}

void Task3(void) /* PG1 */
{
mcxWait(Sem3); /* Cekaj na semaforu 3 */
PORTG|=0x04; /* PG2 - ON */
mcxDelay(0,0,3000/49); /* 3 sek kasni */
PORTG&=~0x04; /* PG2 -OFF */
mcxTerminate(SELF); /* Kraj */
}
}
```

LISTING 1. Assembler

```
tsk1 <inicijalizacija SPI komunikacije,
inicijalizacija IC1 i IC2 i def.
porta G kao izlaznog>

ldaa #T2 ; pokreni T2
swi
FCB .execute.

clra ;
clrb ;č ekaj 2 sek.
ldx #2000/49
swi
FCB .delay.

ldaa PORTG ; pali PG0
oraa #S01
staa PORTG

ldab #SEMA_1 ; č ekaj IC1
swi
FCB .wait.

ldaa PORTG ; gasi PG0
anda #SFE
staa PORTG

ldaa #T3 ; pokreni task 3
swi
FCB .execute.

ldaa #T3 ; postavi tajmer
ldab #SEMA_3 ; za task 4
ldx #1500/49 ; 1.5 sek.
swi
FCB .timer.

clra ; kraj taska 2
swi
FCB .terminate.

tsk2 ldab #SEMA_2 ; cekaj IC2
swi
FCB .wait.

LOOP clr BROJ ; brojac na 0
clra ; 2/3 sek.
clrb
ldx #667/49
swi
FCB .delay.

ldaa PORTG ; pali PG1
oraa #S02
staa PORTG

clra ; 1/3 sek.
clrb
ldx #333/49
swi
FCB .delay.

ldaa PORTG ; gasi PG1
anda #SFD
staa PORTG

ldd BROJ ; inkrementiraj
add #1 ; BROJ
std BROJ
jsr PRIKAZ ; prikazi BROJ

ldd #5 ; izvršavava se
cpd BROJ ; 5 taktova

bgt LOOP

clra ; kraj taska2
swi
FCB .terminate.

tsk3 ldab #SEMA_3 ; č ekaj signal
swi ; tajmera, vidi
FCB .wait. ; task 2

ldaa PORTG ; pali PG2
oraa #S04
staa PORTG

clra ; 3 sek.
clrb
ldx #3000/49
swi
FCB .delay.

ldaa PORTG ; gasi PG2
anda #SFB
staa PORTG

clra
swi
FCB .terminate. ; kraj taska 3
```