Introduktion

DBG12 är en monitor/debugger för MC68HCS12-baserade mikrodatorer. DBG12 ingår i en serie likartade monitor/debugger's för mikrodatorer. För närvarande finns följande varianter:

DBG11	För MC68HC11, speciellt mikrodatorn microlf MC11
DBG12	För MC68HCS12, speciellt mikrodatorn microlf MC12
DBG32	För MC68340, speciellt mikrodatorn microlf MC68

Enhetligheten hos de olika varianterna tjänar flera syften, de viktigaste är:

- Olika mikrodatorvarianter ger samma användargränssnitt, det går därför snabbt att introduceras till en ny mikroprocessor/controller då du en gång lärt dig använda DBG12.
- Källtextdebuggers (som exempelvis ETERM 7 och XCC) har inbyggt stöd för att kommunicera med DBGxx.

I resten av denna beskrivning används beteckningen DBG för att beskriva funktioner som är gemensamma för alla varianter. Processorspecifika avsnitt använder beteckningen DBG12.

Funktion

Då du startar DBG kommer systemet att initieras, dvs. systemet förbereds för att användas som monitor/debugger. Dessa förberedelser innebär bland annat att DBG initierar in- och ut-portar, seriekommunikation, relokerar mikrodatorns minne etc.

DBG identifierar därefter sig genom utskrift av *namn* och *version*.

Speciellt gäller för DBG12:

`Device: DxYYY`	Identifierar typ av MC68HCS12
`Revision Mask: YYYY`	Controllerns revision
`Registers: 0-YYY`	Adressintervall för interna register
`EEPROM: <storlek>`	Storleken av internt EEPROM, observera dock att detta inte används (inaktiveras) av DBG12
`RWM: <storlek>`	Storleken av internt RWM

Se även "Disposition av adressrum" nedan.

Följande kommandon accepteras av DBG

mm - modify memory	Ändra minnesinnehåll
dm - display memory	Visa minnesinnehåll
tr - trace instruction	Utför enstaka maskininstruktion
go - run program	Utför användarprogram
dasm - dissassemble memory	Dissassemblera minnesinnehåll
reg - display/modify registers	Visa/ändra processorns registerinnehåll
bp - breakpoints	Hantera brytpunkter i användarprogram
l - load from host	Överför program från värddator
help - display online help	Inbyggt hjälpsystem

Följande kommandon accepteras dessutom av DBG12:

fload - flash load	Programmera inbyggt FLASH-minne
feras - flash erase	Radera inbyggt FLASH-minne

Textkonventioner och notation

Tangenten “vagnretur”, dvs <Enter> på vissa tangentbord, <Return> på andra återges med sekvensen <CR> (“carriage return”).

I texten använd *stora hakar* för att ange en parameter som *kan* men inte *behöver* ges.
[*option*] anger att *option* kan utelämnas. Observera, stora hakar ska *inte* ges som kommando.

I texten används *vertical bar* (|) för att ange *alternativ*.
[-b|w|l] anger att *något av* -b, -w eller -l *kan*, men *behöver inte* anges.
b: byte (8 bitar)
w: word (16 bitar)
l:long (32 bitar)
Då detta alternativ utelämnas använder DBG *standardstorlek*.

Speciellt för DBG12:
- Alternativet 'l' används ej.
- *Standardstorlek* är 'b'

Indata kan under vissa omständigheter ges i något av talsystemen *binär*, *oktal*¸ *decimal* eller *hexadecimal* form. Du anger talsystem med ett prefix till det inmatade värdet:
%*värde* anger *binär* form
@*värde* anger *oktal* form

t*värde*
T*värde* anger *decimal* form

$*värde* ***eller***
värde anger *hexadecimal* form

Monitorkommandon

Modify memory
mm [size] address [value]

Med mm-kommandot kan minnesinnehåll visas och ändras. Kommandot har två former, där den första formen används för att modifiera ett enstaka element, den andra formen startar en *interaktiv mod*.

Former:

mm [-b|w|l] *address value*<CR>
Används för att ändra enstaka element på *address* till *value*. *size* kan, men behöver inte anges. Om *size* utelämnas används standardstorleken.

mm [-b|w|l] *address*<CR>
Formen används för att starta *interaktiv mod*. Nu skrivs *address* följt av innehållet ut till bildskärmen.

I *interaktiv mod* kan följande kommandon ges:

nytt värde<CR>	modifiera minnesinnehåll
-	ingen ändring, visa föregående
+	ingen ändring, visa nästa
.	('punkt') ingen ändring, avsluta

Display Memory
dm [size] address [count]

dm används för att visa minnesinnehåll. En *startadress*, dvs. adressen till den första minnespositionen, anges. Minnesinnehållet kan visas med formen *byte* (8 bitar), *word* (16 bitar) eller *long* (32 bitar). Minnesinnehållet visas *alltid* på *hexadecimal* form *och* med ASCII-representation.

Former:

dm [-b|w|l] *address* [*count*]<CR>

size-fältet *kan* men *behöver ej* anges. Om *size* utelämnas används standardstorleken.

address-fältet måste *alltid* anges. Adressen anges på *hexadecimal* form. Udda adress kan endast anges om *size*-fältet samtidigt är -b.

count-fältet *kan* men *behöver ej* anges. *count* ges på *decimal* form. Om *count*-fältet utelämnas kommer DBG att bestämma antalet visade ord beroende på *size*-fältet. Om *size* är -b , kommer 256 bytes att visas, om *size* är -w kommer 128 words att visas, om *size* är -l, kommer 64 long words att visas.

Anm: För att visa adresser (8000-BFFF) i bankat minne används 22-bitars adressfält enligt XX8000 t.o.m XXBFFF, där XX utgör någon sida (se *Disposition av adressrum* nedan). Om du utelämnar XX tolkas detta som den första tillgängliga minnesbanken (30 för MC12), dvs linjärt adressrum.

Trace instruction(s)
tr address [count]

tr-kommandot används för att utföra *enstaka* instruktioner. Detta kan jämföras med att sätta en brytpunkt *på varje* instruktion i programmet. Efter utfört tr-kommando skriver DBG *användarregistrens* innehåll till bildskärmen.

Former:

tr *address count*<CR>
Utför *count* (tolkas som decimal form) antal instruktioner, den första på *address* (tolkas som hexadecimal form).

tr *address* <CR>
Utför *en* instruktion med start på *address*.

tr<CR>
Utför instruktionen på den adress som anges av *användarregister PC*.

tr +<CR>
Aktiverar kortkommando för tr, ("HOT trace"). Då kortkommandot är aktivt räcker det med att trycka '.' (punkt) från DBG's prompter för att utföra kommandot tr<CR>.

tr -<CR>
Stänger av kortkommandot för tr.

Anm: För att stega i bankat minne (8000-BFFF) används 22-bitars adressfält enligt XX8000 t.o.m XXBFFF, där XX utgör någon sida (se *Disposition av adressrum* nedan). Om du utelämnar XX tolkas detta som den första tillgängliga minnesbanken (30 för MC12), dvs linjärt adressrum.

Execute program
go [-n] [address]

go-kommandot används för att starta ett tillämpningsprogram. Då DBG startar tillämpningsprogrammet kommer innehållet i *användarregistren* (se *Display/Modify Registers*) först att laddas till processorns register.

Former:

go *address*<CR>
Formen används för att starta ett tillämpningsprogram som börjar på *address*.

go<CR>
Samma som föregående men programmet startas från adress som anges av *användarregistret PC*.

go -n<CR>
Utför program *till nästa instruktion*. Speciellt används formen då man vill utföra en hel subrutin.

Dissassembly
dasm [address] [count]

Med dasm-kommandot kan minnesinnehåll *disassembleras*. Dvs. DBG läser innehållet på någon adress, *tolkar* detta som en *maskininstruktion* och skriver ut den *mnemonic* och de operander som anges av instruktionen.

Former:

dasm *address*<CR>
Disassemblera 10 instruktioner med start på *address*. Information om den sist disassemblerarade instruktionen sparas internt av DBG.

dasm *address instructions*<CR>
Med denna form ges också det antal instruktioner (*count*) som skall disassembleras. *address* tolkas av DBG som hexadecimalt, medan *count* tolkas som decimalt.

dasm<CR>
Den enklaste formen används för att fortsätta disassembleringen där den sist avslutades. 10 instruktioner disassembleras.

Display/Modify registers
reg [regname] [value]

reg-kommandot används för att *visa* eller *ändra* de registervärden som används vid utförande av tillämpningsprogram (*användarregister*). Dessa värden laddas i registren av go- respektive tr-kommandona. Vid brytpunkt sparas de aktuella registervärdena i den interna tabellen för *användarregister*.

Som registernamn "regname" används (speciellt för DBG12):

regname	initialt värde	Beskrivning
A	0	Ackumulator A
B	0	Ackumulator B
D		Ack A (MSB) tillsammans med ack B (LSB)
X	0	Index register X
Y	0	Index register Y
CC	D0	Flaggregister "Condition Codes"
PC	1000	Programräknare
S	3C80	Stackpekare

Former:

reg<CR>
används för att *visa* samtliga registerinnehåll. DBG genererar en utskrift till bildskärmen.

reg *regname* <CR>
används för att visa enstaka registerinnehåll. *regname* måste vara ett giltigt registernamn. Små eller stora bokstäver kan användas.

reg *regname value*<CR>
används för att *ändra* registerinnehåll. *regname* måste vara ett giltigt registernamn. Små eller stora bokstäver kan användas.
value anger det värde som ska placeras i registret. Det kan anges på *binär* form (med prefix %), på *oktal* form (med prefix @), på *decimal* form (med prefix t) eller på *hexadecimal* form (utan prefix).

Breakpoints
bp [number] [action] [address]

bp-kommandot används för att *visa*, *sätta ut* och *ta bort* brytpunkter i tillämpningsprogrammet. En brytpunkt är antingen *aktiv* eller *inaktiv*. Om brytpunkten är aktiv, kommer DBG att avbryta utförandet av tillämpningsprogrammet då processorn *skall utföra* den instruktion som ersatts med brytpunkt. Brytpunkter sätts lämpligen i *början* av subrutiner som ska testas. Genom att därefter utföra programmet *instruktionsvis* (*se "Utför program instruktionsvis*RefTrace") kan programflödet enkelt följas och kontrolleras.

Brytpunkterna placeras (internt) av DBG i en *brytpunktstabell*,
Maximalt 10 brytpunkter åt gången, kan hanteras av DBG12.
Brytpunkterna numreras ("namnges") 0 t.o.m.9.

Former:

bp<CR>
Hela brytpunktstabellen skrivs till skärmen. För varje brytpunkt skrivs, om den används för tillfället, om den är aktiv och dess adress.

bp *number* set *address*<CR>
Används för att sätta ut en ny brytpunkt. *number* skall vara brytpunktens nummer och tolkas av DBG som ett decimalt tal. *address* är brytpunktens adress. Om en annan brytpunkt tidigare definierats med samma nummer modifieras denna brytpunkt. Brytpunkten är aktiv.

bp *number* dis<CR>
Inaktiverar brytpunkt *number* om denna är aktiv. Brytpunkten behålls i tabellen men orsakar inget programavbrott.

bp *number* en<CR>
Aktiverar brytpunkt *number* om denna tidigare var inaktiv.

bp *number* rem<CR>
Tar bort brytpunkt *number* ur brytpunktstabellen.

bp clear<CR>
Tar bort **samtliga** brytpunkter ur brytpunktstabellen.

Anm: Brytpunkter kan inte användas i FLASH-minne.

Load from host
l

l-kommandot används då man vill överföra kod/data till måldatorns primärminne. DBG accepterar endast Motorola S1,S2 och S3 format.

Anmärkning
Om Du använder ETERM eller XCC behöver du inte skriva l-kommandot till DBG, detta utförs då du ger ladd-kommando till ETERM (XCC).

Display Help-menu
help

help-kommandot används i två olika former:

help<CR>
ger en översikt av *tillgängliga* kommandon.

help *command*<CR>
där *command* är ett av de angivna tillgängliga kommandona.
Ger utförligare hjälp om detta kommando

FLASH Load
fload

fload används i stället för det ordinarie l-kommandot då MC68HCS12 interna flash-minne ska programmeras. Kommandot medger endast programmering av det bankade minnet (se *Disposition av adressrum* nedan). Detta innebär alltså att endast minnesbankar i adressrummet XX8000-XXBFFF kan programmeras. Undantag är bankar 3E och 3F som upptas av DBG12 (dessa kan inte programmeras). Speciellt gäller att adressintervallet 008000-00BFFF tolkas som 308000-30BFFF. S-poster för adressintervall utanför bankat minne ignoreras.

Kommandot

fload<CR>

ges vid DBG12's prompter.
DBG12 svarar
Loading Flash
DBG12 väntar nudärefter startas nedladdning från värddatorn på samma sätt som vid det ordinarie l-kommandot.

Anm: Kontrollera alltid att minnesbanken är raderad innan programmeringen.

FLASH Erase
feras bank | all

Används för att radera samtliga minnesbankar, eller en enstaka minnesbank i MC68HCS12's interna flash-minne. Undantag är bankar 3E och 3F som upptas av DBG12.

Kommandot kan endast ges på formen:

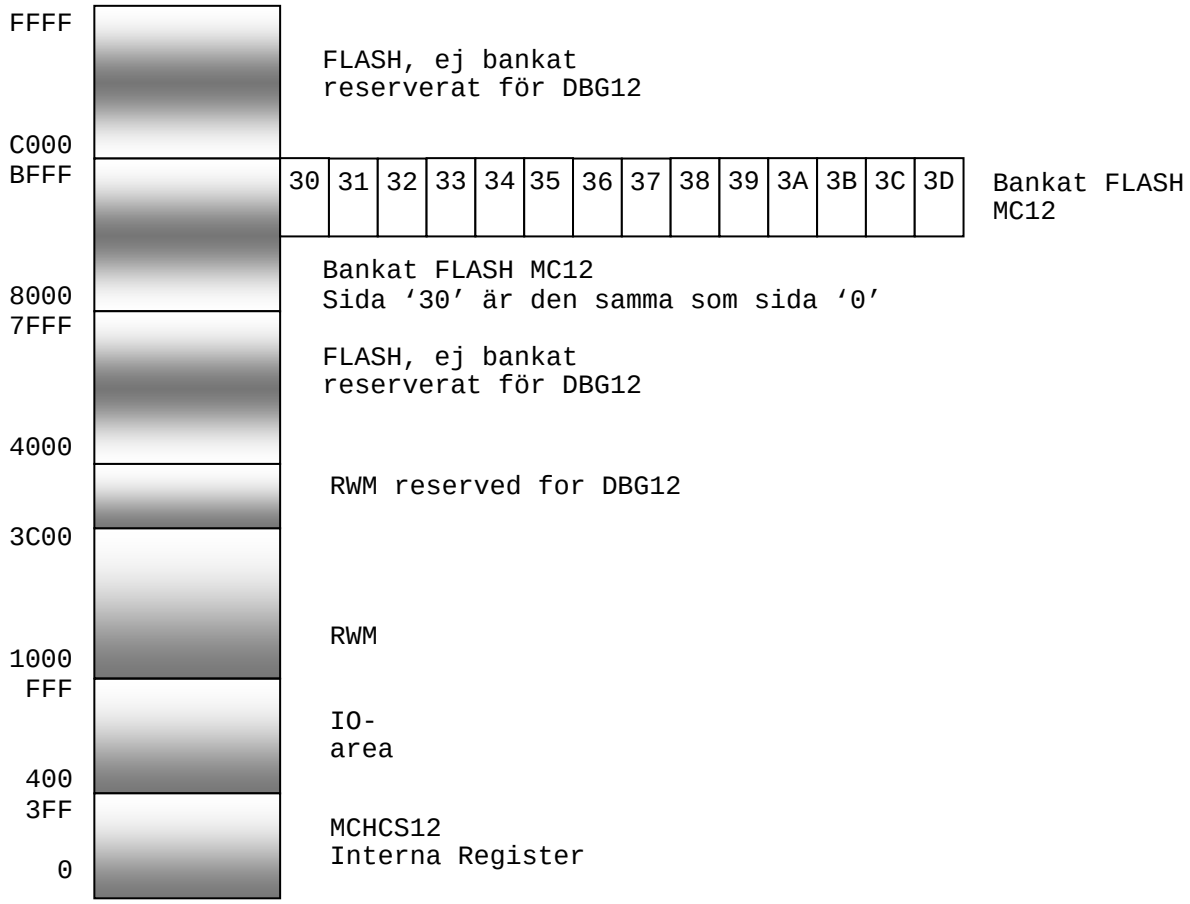
feras *bank*<CR>

bank anger en giltig minnesbank (30-3D, se även *Disposition av adressrum* nedan), *eller*

feras all<CR>
raderar samtliga minnesbankar (30-3D).

Disposition av adressrum

DBG12 disponerar adressrummet hos MC68HCS12 på följande sätt:



DBG12 använder internt RWM 3D00-3FFF. Applikationsprogram kan använda intervallet 1000 t.o.m. 3CFF.

DBG12 använder "naturlig" översättning av adresser i det bankade minnet. Detta innebär att de 6 mest signifikanta bitarna i adressen används direkt för att initiera PAGE-registret. Undantag är bank 30, som adresseras om du bara använder 16-bitars adresser (8000-BFFF).

MC12 ger möjlighet att använda maximalt 224 kByte expanderat minne.

Inbyggda hjälprutiner

DBG12 tillhandahåller några inbyggda hjälprutiner som kan vara praktiska i olika sammanhang. Rutinerna används vanligen från assemblerprogram och utnyttjar en enkel tabell placerad omedelbart efter DBG12's startpunkt.

Beteckning	Adress	Beskrivning
start	C000	Återstart (RESET) av DBG12
tstchar	C003	Kontrollera om tecken finns i SCI0
outc	C006	Skriv tecken till SCI0 via buffert
gvb	C009	Lokalisera avbrottsvektorer
svb	C00C	Sätt basadress för avbrottsvektorer
restart	C00F	Återstarta DBG12 från prompter
prstring	C012	Skriv textsträng via SCI0
outc_dir	C015	Skriv tecken direkt till SCI0

tstchar, adress C003

(Test if character) kontrollera om tecken finns i SCI.
Rutinen använder samma SCI som DBG (SCI0). Om ett tecken finns tillgängligt returneras detta i B-registret, annars returneras 0.
Endast B-registret påverkas

EXEMPEL:
Följande rutin returnerar nästa tecken från SCI0

```
...
inchar:
      JSR   $C003      använd DBG12
      TSTB             finns tecken ?
      BEQ   inchar     om inte, försök igen
      RTS              returnera med tecken i ack B
```

outchar, adress C006

(Output character) skriv tecken till SCI0
Inget register påverkas.

EXEMPEL:
Följande sekvens skriver ASCII tecknet 'A' till SCI0

```
...
      LDAB  #'A'
      JSR   $C006
...
```

gvb, adress C009

(Get Vector Base) returnerar basadressen till minnesarean för avbrottsvektorer. Se Avbrottshantering i DBG12 nedan.
Returvärde: Basadress i D-registret.
Inga andra register påverkas.

svb, adress C00C

(Set Vector Base) sätter en ny basadress till minnesarean för avbrottsvektorer. Se Avbrottshantering i DBG12 nedan.
Parameter: Ny basadress i register D.
Inga andra register påverkas.

restart, adress C00F
Då ett applikationsprogram avslutas kan ‘restart’ anropas för att ge återstart vid DBG12’s prompter.

prstring, adress C012
Skriv textsträng, avslutad med tecken 0, till SCI0. Före anropet måste register X laddas med adressen till textsträngens första tecken

EXEMPEL:
Följande sekvens skriver textsträngen ”Ny Text” till SCI0, därefter återstartas DBG12 från prompter.

```
...
      LDX   #Text
     JSR    $C012
     JMP    $C00F

Text  FCS   "Ny Text"
      FCB   0
...
```

Avbrottshantering i DBG12

DBG använder endast ett fåtal avbrottsvektorer, samtliga vektorer initieras dock med en standardhanterare. Detta betyder att om ett oväntat avbrott inträffar kommer DBG att fånga detta och skriva ett meddelande till terminalen.

Samtliga avbrottsvektorer har omdirigerats till RWM. Basadressen till avbrottsvektorerna finns kan ändras av ett tillämpningsprogram (svb) men kan också användas för att ändra en enstaka avbrottsvektor. Detta sker enkelt med de inbyggda hjälprutinerna i DBG.

EXEMPEL:
Tillämpningsprogrammet använder en enstaka IRQ-vektor (FFF0) och vill omdirigera denna till en egen avbrottshanterare:

```
      JSR   $C009
* adress till avbrottsvektorer nu i D
      XGDX  till X-registret
      LDD #($FFF0-$FF8C)
* offset till basadressen för HCS12 avbrottsvektorer
* eftersom denna offset alltid är mindre än 256 bytes
* är det säkert att använda ...
      ABX
* adress till DBG12's avbrottshanterare för IRQ-vektor
* FFF0 finns nu i X-registret
      LDD   #my_FFF0_handler
* applikationsprogrammets avbrottshanterare
      STD   ,X
* till avbrottsvektor i RWM
...
```

EXEMPEL
Tillämpningsprogrammet tillhandahåller en egen vektorarea för samtliga avbrottsvektorer (se Anm nedan).

```
my_exception_table:
      FDB   my_exception_vector_1
      FDB   my_exception_vector_2
...
```

observera att samtliga vektorer måste tillhandahållas och dessutom i rätt ordning.

```
      LDD   #my_exception_table
      JSR   $C00C
```

nu hanteras samtliga avbrott av tillämpningsprogrammet.

Anm:
DBG12 använder vektoravbrott SWI för "trace" och "brytpunkter". Under *vissa* omständigheter kan du därför få problem om du omdirigerar dessa vektorer och försöker testa ditt program med användning av "trace" eller om du försöker sätta brytpunkter.