Putting code in FLASH on the 68HC12 SBC

BACKGROUND:

The 68HC12 processor on the New Micros MNIS/L-0012 single-board-computer (SBC) contains internal FLASH-RAM memory that normally holds the MaxForth operating system and user programs. The FLASH memory is mapped in the $8000 - FFFF space.

External RAM is normally used for data storage and temporary working variables and maps to the $0000-7FFF space. Certain portions of this memory area are reserved for things like the CPU registers ($0000-00FF), Forth operating variables, and a section of EEPROM ($0D00-0DFF). There is a section of RAM on-chip at $0A0C-0BFF that can be used. If the command EXRAM is executed in the user code, the external RAM will be used in lieu of the internal RAM.

In my programs, I usually point my RAM variables at the (external ram) space $1000 and up to avoid conflicts with the rather choppy area at lower addresses.

Loading programs to FLASH is straightforward, if a trifle complicated. The procedures are described in the NMIS literature that comes with the SBC. Basically, they consist of two parts -- finding a starting place for the code in FLASH RAM and then modifying the code so that Forth knows to move each word to FLASH. The beginning portion of my flash code listing is appended to these notes.

SETUP:

First, make sure the FLASH programming jumper is installed on J7. If you hold the board with the serial jacks (the two 2X5 pin IDS connectors, J9 and J10) at the top, J7 will be just below these jacks, oriented horizontally. The shorting plug should be on the two right-most pins, next to the MAX662 IC.

Next, connect a pushbutton switch of some sort (jumper wires will do) to the RESET pins on J6. This jack is just below and slightly to the left of J7. This jack is a vertical array (3X2). The two pins you want are the lower 2 pins on the left row. Connecting these pins together causes a RESET of the CPU chip.

Finally, locate the PDLC0 pin on J8, the long, vertical jack on the left side of the card. This pin is right about in the middle of the jack on the right side and is labeled fairly clearly. It is just to the left of resistor R8. Attach a jumper wire to this pin.

You will need to be connected to the card with a terminal program with the capability of downloading files. One file you will have to have handy is

F12V50L.S19, the MaxForth 5.0 operating system S-files. Another, of course, is the program you want to load to FLASH.

If the SBC responds with the MaxForth 5.0 prompt when you power it up, then you can proceed immediately to loading a program. Otherwise, you may need to follow the directions below.

GAINING CONTROL & ERASING A PROGRAM IN FLASH:

If there is a FLASH program installed that auto runs, you will have to wrest control of the CPU away from it. This consists of a couple of steps: First, drop out of the operating program. Second, over-write any autostart commands. Third, erase and reprogram FORTH into the flash ram on the CPU. Fourth, load the new program.

I generally include a KILL command in my programs (CTRL-K) that changes a stored variable from FALSE to TRUE. Since my main word always includes some form of BEGIN - UNTIL loop to run the program endlessly, I fetch the stored flag variable just before the UNTIL command. Since I auto-run the program with the $A44A command, setting this flag TRUE allows the code to fall through to MaxForth when the KILL command is executed.

At this point, I over-write the $A44A autorun vector at $D00 with $FFFF. In most of my code, this is an option after the KILL command.

BUT -- YOU SHOULD CHECK THAT THE AUTORUN VECTOR HAS REALLY BEEN CHANGED OR YOU WILL HAVE PROBLEMS LATER.

The safe bet is to type

```
HEX
D00 10 DUMP
```

and make sure the first two words are $FFFF. If so, from now on, the card will restart in MaxForth instead of running the program in flash-ram.

To erase the FLASH memory, type FLASH. You should now see the following line on your terminal (you will have to change your baud rate to 9600 if it is not already set)

```
Erase (E) or Program (P)?
```

Type an E and the monitor code will erase the FLASH memory.

If for some reason you cannot recover to MaxForth in this way, you will need to overcome the code as follows. Attach the other end of the jumper on the

PDLC0 pin to the pin just to it's left, GND. Then push your RESET pushbutton to cause a RESET. You should now see the following line on your terminal

```
Erase (E) or Program (P)?
```

Type an E and this monitor code will erase the FLASH memory.

Now you need to reload the MaxForth S-file. Do the appropriate commands to download the file F12V50L.S19 to the board. Note that during this download, the board does not respond with OK<CR><LF> like MaxForth does. This may require some changes to your terminal program to pace the download. I use a 25 mSec delay at the end of each line and turn character echo off.

Another possibility -- if you can exit the driver program but cannot get to the FORTH prompt, try typing a CTRL-G and then short the reset pin to ground. The CTRL-G reset skips the autostart sequence and may allow recovery to FORTH.

Sometimes it is possible to enter the FLASH reprogramming code and then do a CTRL-G reset escape (50-50 chance) to avoid having to erase FLASH. This requires installing a jumper on the PDLC-0 pin and ground (just across from this pin on the 72-pin connector on the SBC itself). Set the terminal to 9600 baud. After the jumper is installed, do a reset (short the reset pin on the BDM connector to ground) and you should see the FLASH reprogramming line. Remove the jumper from PDLC-0. Type CTRL-G and short the reset pin again. You may now see the FORTH prompt (and you may need to change baud rates to see it).

ERASING FLASH EPROM EXTERNALLY

If the directions above are not followed precisely (and, sometimes, even if they are), it is possible to lock up the CPU and lose control. This typically happens when the driver program is changed but the autostart flag is left in EEPROM. The hints here are intended to help but, of course, the solution depends upon the problem.

A Background Debug Monitor is a useful device/program for fixing problems related to autostart vectors and such. We use a unit from Axiom Manufacturing in Garland TX (AX-BDM12) but almost any BDM setup for the 68HC12 CPU will probably work. There is a BDM connector on the CPU card -- obviously, you will have to open TAPS and remove the electronics unit to get access to this connector.

When we are developing code -- with the inevitable code lock-ups that entails -- we remove the CPU card from the electronics cage and work with it alone on the bench. We use a 12V DC supply to provide board power (J1 pin 1 = +, pin 2 = ground). We have made an adaptor to plug onto J6 that runs to a DB-9 serial connector. Pin 1 is SI (serial data in) and pin 2 is SO (serial data out). Pins 3-4

are ground. SI, SO, and GND are the only pins required for a simple serial interface to a PC.

I created a text file called EEPROM.S19; it contains the following lines:

```
S014000046696C653A20656570726F6D2E61736D0A10
S1050FFEFFFFEF
S9030000FC
```

This file can be used with a BDM to download over the EEPROM at address $0D00 to kill any autostart vectors if the procedures above do not work. There is generally a reset control on the BDM that lets the BDM program interrupt the CPU and view and alter memory contents.

One thing to note -- the BDM connector on the NMIS/L-912 card is a 6-pin connector. The BDM connector on my BDM is an 8-pin connector. The two right-hand pins are not used but I had to bend the pins of the adjacent connector over to fit the BDM cable connector on the card.

DOWN-LOADING A NEW PROGRAM:

This assumes you have accomplished the steps above and have a clean flash-ram into which you want to load a new operating program.

I assume the code has been modified to put all code in FLASH and point variables to the appropriate RAM (internal or external). Download the program in the usual way. Ignore the NOT UNIQUE responses from re-definition of CONSTANT, VARIABLE, etc.

When the code is loaded, you MUST install an auto-run link of some sort or the code will not be usable. If you intend to put variables or data in the external RAM, make sure that EXRAM is executed in your code somewhere. I generally have an INIT word that does all the port setups, etc. This is a good spot for EXRAM.

At the end of your code, you can include some lines to cause an autorun vector to be placed in EEPROM. My top-level word that runs my programs is called MAIN. Thus, I follow the code with

```
hex

a44a d00 ee!                ( set auto-run flag in EEPROM
' main cfa d02 ee!          ( put execution address in EEPROM

decimal
```

which causes MAIN to run when the SBC starts up. (Note that my terminal program has a CAPS filter that lets me write code in lower case if I wish; code going to the FORTH interpreter must be upper case to be recognized.) The $A44A vector allows me to exit to Forth from the program ($A55A does not), which is handy for reloading the program and, occasionally, for trouble-shooting the program by inspecting the variables manually.

The following is a sample listing from actual operating code that sets up Forth to place code in FLASH RAM. The code itself is identical to some code I originally loaded into EXTERNAL RAM for testing.

```
cold

hex

( Begin by finding a starting point in FLASH for the code

: flblank
    f800 f800 8000 do
      ff I 10 over + swap do
         I c@ and loop
      ff = if drop I leave then
      100 +loop
    dup f800 = if cr ." flash is full" cr
    else dup cr u. cr then
;

flblank fdp !
forget flblank
( ------------------------------------------------------------
( Re-define some words to put code in FLASH, data in EXRAM
( The command to move a word to FLASH is FLWORD
( Begin by re-defining ';' and ',' to include FLWORD

: ; [compile] ; flword ; immediate flword
: , [compile] , flword ; immediate flword

( Now re-define CONSTANT def'ns to point to FLASH

: constant  constant flword ;

: 2constant 2constant flword ;

: create    here constant ;


: bvariable create 1 allot ;      ( Create a byte variable

: variable create 2 allot ;       ( Create a word variable

: 2variable create 4 allot ;      ( Create a dbl-word variable

exram

1000 dp !
```

Note that I've left blank lines after the re-definitions of constant, etc. This helps my terminal program (BitCom) to pace the download slowly enough that the program can respond with a NOT UNIQUE response without stepping on the following line input. Your terminal program may differ.