Differences 3.3 to 3.5

Motorola has decided to discontinue the 68HC11A8, which is the heart of our F68HC11 V3.3. We have until September of this year ('99) to place final orders for the V3.3. After those orders are filled, we will not have a source for F68HC11 V3.3 anymore, and therefore, will not be able to supply them to you.

The F68HC11 3.5 will continue to be available into the foreseeable future. The F68HC11 V3.5 is based on the 68HC11E9. There are some differences between the two parts although they are on the whole very similar. Both parts are in 52 pin PLCC's and are interchangeable in the same socket. Existing software may, or may not, work, however. The purpose of this letter is to identify the differences.

Also, we would like to offer our assistance in making your product transition from V3.3 to V3.5, or possible a newer processor or updated design. In many cases the V3.5 will be a drop in replacement, however, we have long been encouraging our customers to move to the V3.5, and we know some have resisted, or not been able to, for software reasons. We've given this situation some thought. We may be able to help you convert your software from V3.3 dependencies to V3.5 compatibility, even if your source code or programmer is long gone. We may be able to "decompile" old code in many cases and rebuild source.

If conversion is not possible, which will be only in the rarest cases, updating your product might be desirable, anyway. Again, old code can be decompiled (although it is always easier to start from the original source and engineering notes) and move to newer processor. Power, speed, and features may be improved in the process. Please consider letting us help you with your engineering needs.

The differences between the V3.3 and the V3.5 over all are minor. They fall into four groups: Ports, Registers, Memory Map and Operating System.

Ports

An added feature of the E9 (our V3.5) over the A8 (our V3.3) is the bidirectionality of the Port A, line 3. Unfortunately the part powers up with this pin as an input, where it used to be an output. The original A8 part's Port A had three inputs (PA0-2), four outputs (PA3-6), and one bidirectional line (PA7). Three Input Captures and five Output Compares corresponded to the inputs and outputs respectively. In the A8 part, PA3 is an output only, and had output compare abilities only. In the E9 part, probably to get a balance between Input Capture and Output Compare functions, PA3 became programmable as a bidirectional line with both the Output Compare abilities it had previously, and the Input Capture feature added. All this would have been fine, since the old functionality was still there, with one exception. If the default state of PA3 had been an output, like the previous part, the added features would have not been noticed. However, typical software written for the A8 part would not have to take any action to change PA3 to an output, since it wasn't possible to change it before. In the E9 part, plugged into an A8 socket, the software must initialize the PACTL register to establish PA3 as an output. Otherwise, the software will write to PA3 (still an input), but nothing will come out.

Registers

The next compatibility issues to be discussed are the register changes. As mentioned the PACTL register has an additional bit, Bit3, to control the direction of PA3. Since this bit was unused previously, there should be little effect. However, any writes to the PACTL in the existing software for other purposes might accidentally turn PA3 back into an input.

A similar situation exists in the same register in the next bit, PACTL Bit2. In the A8 version, it was unused.
In the E9 version it chooses which of the available two functions are associated with PA3, Input Capture, or Output Compare. Like the direction bit, the default state is opposite in the two parts. The power up

state of the E9 makes PACTL PA3 an input (as previously noted) and assigns the Input Capture function to it. So, existing software that uses the Output Compare function will not work unless the PACTL is reprogrammed on power up, and all other writes to the PACTL in the program might need to be modified to avoid accidentally changing the I4/O5 bit.

The TCTL2 Register also had bits added to support the Input Compare 4 feature in the E9. These are the top two bits in the register. Since these bits were unused in the A8, the existing software will most likely be unaffected by this change.

Finally, an added register to the E9, called the BPROT Register is present, where there was an empty location before. Since these bits were unused in the A8, the existing software will most likely be unaffected by this change. The operating system of V3.5 manages the settings in this register on power up to be compatible with the V3.3 part.

Memory map

The largest change between the two parts was the increase in memory resource. The A8 had only 256 bytes of internal RAM, while the E9 has twice as much, 512 bytes. Generally more wouldn't seem to be a problem. Assuming lower external memory is not used at all, the existing software will probably work fine without much notice. If the lower external memory is used for RAM, the new RAM will over map the external RAM. The existing software will probably work well, as long as there is not a battery-backup issue, where internal is backed and external is not, or vice versa. The area of potential conflict is restricted to the additional RAM at locations 0100-01FF.

Another potential problem with the added RAM might be found in use of free space. Since twice the RAM was available on the E9 part over the A8 part, the operating system spread some of the movable software features around. The TIB was enlarged and moved, and PAD was moved with it. The stacks are also moved and expanded. An additional floating-point stack was added, and the Dictionary Point was moved to the end of all other structures, so it did not have to be specifically moved outside internal RAM to make a program of size.

Conversion of existing software to work with the E9 should take these address changes into account. If the program lives well in the Max-FORTH paradigm, many of these uses will be transparent. References to PAD, TIB, DP, etc., are vectored through the user area, which remained in the same place.

Early examples for the A8 often moved TIB and the DP to external RAM during development. The addresses suggested were 100 in TIB and 200 in DP. The DP will do fine at 200, but the TIB AT 100 may interfere with the descending data stack in the E9 V3.5 system.

More difficulty may exist in dealing with the ROM changes. The ROM size increased by 50% from the A8 (V3.3) to E9 (V3.5). In the original A8, only 8K ROM was available. Max-Forth was fit in the space available. When the new E9 came out, the additional 4K of memory was not optional - there are no 8K E9 available. So the additional of memory was used to add floating point code, patches to the existing software, and additional dictionary heads, which were left orphaned in the first version. Great pains were taken to keep the entry points between the two software versions compatible. For instance the address of DUP in V3.3 is the same address as DUP in the new V3.5. Code written for the V3.3 will execute without modification on the V3.5 version, and run. Whether it operates correctly depends on the PA3 and register usage described above. However, a great deal of compatibility was gained by retaining entry points.

The mostly likely problem to arise is the loss of the D000 4K memory block. Examples for the A8 V3.3 system often used the target program area of C000. As long as programs made at C000 are less than 4K in length, they will not encounter a conflict. However, if any program uses program EPROM (EEPROM, ROM etc.) in the D000 area, the program will not be seen, for being overshadowed by the expanded internal ROM. The same would be true for any peripherals mapped in the D000 block. If programs are small, they may be relocated elsewhere in memory. This could be accomplished as simply as doing a recompilation of the source. If programs are large, for instance, occupying all of the memory from 8000 to DFFF, some other technique might have to be used. Without hardware changes, the only hope to make existing software fit into the smaller space might be a rewrite of the code to reduce its size. However, there are techniques which can reduce program size. For instance, headerless code compilation can often reduce code size by up to half. Again, if we can be of assistance, we are available by contract to work with you on these changes.

Operating System Changes

A few things were changed in the V3.5 Operating System as well. Most of these changes should improve performance, or correct a few minor errors in the kernel. The biggest change which would be likely noticed between the two versions, would be in increase in speed in the math routines. The division routines were considerably improved in speed, and made smaller as well. Unless this increased speed causes time dependencies to arise, no changes should be required to existing software.

Most of the improvement occurred in the words:

CODE-SUB	Moved to Assembler Vocabulary
DMIN	Fixed, Erroneously did DMAX
2ROT	Fixed, Erroneously off target
EEC!	Eliminated dependence on TOC5
UM/MOD	Better division
0	Fixed, Speed
(LOOP)	Better looping
(+LOOP)	" "
/MOD	Better division
MOD	" "
/	" "
M/	" "
;CODE	Fix concerning <BUILDS DOES>
FIND	Reduced dependency on Y register
INDEX	Fixed
THRU	Fixed
FILL	Fixed
STATE	Improved
[	made immediate
]	matches [
FORTH83	