APPENDIX C
D-BUG12 STARTUP CODE

The D-Bug12 startup code is located in the EPROMs, U7 and U9A, in the address range $FD80 to $FDFF, as shown in Table 3-5.

To customize this startup code, it is necessary to reprogram the EPROMs. For more information, refer to Appendix E, Customizing the EPROMs.

The following D-Bug12 startup code is distilled from the source listing for clarity. To assemble the startup code for programming into the EPROMs, the .DEFINEs must be included ahead of the code listed below. These are available on the Internet at http://www.mot.com/m68hc12.

```
                        opt   lis               ; assembler directive to turn
                                                ; listing on

0A00        MonRAMStart equ   $0A00
0200        MonRAMSize  equ   $0200

0800        RAM_START   equ   $0800

0400        RAMSize     equ   $0400

0C00        STACKTOP    equ   RAM_START+RAMSize ; stack at top of int RAM

1000        EE_START    equ   $1000             ; 4K EEPROM located here out
                                                ; of reset(in expanded modes)
FD80                    org   $fd80

            ;*****************************************************************
            ; INITIALIZATION
            ;
            ; Initialization code for the M68HC12A4EVB D-Bug12 monitor program
            ;*****************************************************************

FD80        CODE_START:

            ;     set PortE bit 7 to an output to eliminate possible noise
            ;     problems associated with unterminated input pins.
```

```
FD80 4C0980             bset  DDRE,80h          ; set the data direction to
                                                ; configure PortE, bit 7 as an
                                                ; output.
FD83 4C0880             bset  PORTE,80h         ; set PortE, bit 7 to logic 1.

FD86 CF0C00             lds   #STACKTOP         ; initialize D-Bug12 stack
                                                ; pointer
FD89 4F6F0103           brclr PORTAD,01h,DEBUG12; if bit 0 of A/D port is 1,
FD8D 061000             jmp   EE_START          ; then jump to the start of
                                                ; internal EEPROM
                                                ; otherwise, remain in D-Bug12
FD90        DEBUG12:

            ;     Clear all monitor RAM to start from a known state

FD90 CE0A00             ldx   #MonRAMStart
FD93 6930   ClrRAM:     clr   1,x+              ; clear one and inc pointer
FD95 8E0C00             cpx   #MonRAMStart+MonRAMSize
FD98 26F9               bne   ClrRAM            ; loop till RAM clear

            ;     Enable pipe signals, E, low strobe and read/write in port E
            ;     PIPOE, NECLK, LSTRE and RDWE are write once in normal modes
            ;     PEAR  [ARSIE:CDLTE :PIPOE :NECLK !LSTRE : RDWE :  0   :  0   ]$0A

FD9A 862C               ldaa  #$2c              ; prevent later protection
                                                ; lock
FD9C 5A0A               staa  PEAR              ; PROTLK is write-once

            ;     Without changing modes, enable internal visibility
            ;     MODE  [SMODN: MODB : MODA : ESTR ! IVIS :  0   : EMD  : EME  ]$0B

FD9E 4C0B08             bset  MODE,$08          ; set IVIS

            ;     Disable the COP watchdog by CR2:CR1:CR0 = 0:0:0
            ;     COPCTL = $07 when reset in normal modes
            ;     FCME and CRx bits are write once in normal modes
            ;     COPCTL[ CME : FCME : FCM  : FCOP ! DISR : CR2  : CR1  : CR0  ]$16

FDA1 790016             clr   COPCTL            ; disable watchdog

            ;     Enable Program chip select 0 and Data chip select
            ;     CSCTL0 = $20 after reset (CSP0 on others off)
            ;     also set data chip select to cover $0000-7FFF (will mirror
            ;     to fill space)
            ;     internal resources have higher priority in case of overlaps
            ;
            ;     CSCTL0[  0  :CSP1E :CSP0E : CSDE ! CS3E : CS2E : CS1E : CS0E ]$3C
            ;     CSCTL1[  0  :CSP1FL:CSPA21:CSDHF !CS3EP :  0   :  0   :  0   ]$3D

FDA4 8630               ldaa  #$30
FDA6 5A3C               staa  CSCTL0            ; CSP0E and CSDE on
FDA8 8610               ldaa  #$10
FDAA 5A3D               staa  CSCTL1            ; CSD to cover $0000-7FFF
```

```
            ;     Set stretch for CSP0 and CSD to 1 extra E-speed cycle per
            ;     access (to accomodate slower external RAM and EPROM)
            ;
            ;     CSSTR0[  0  :  0   :SRP1A :SRP1B !SRP0A :SRP0B :STRDA :STRDB ]$3E

FDAC 8605               ldaa  #$05
FDAE 5A3E               staa  CSSTR0            ; CSP0E and CSDE on

            ;     Enable EEPROM so D-Bug12 can program/erase bytes
            ;     EEMCR [  1  :  1   :  1   :  1   !  1   :  1   :PROTLK: EERC ]$F0
            ;     BPROT [  1  :BPROT6:BPROT5:BPROT4!BPROT3:BPROT2:BPROT1:BPROT0]$F1

FDB0 86FC               ldaa  #$fc              ; prevent later protection
                                                ; lock
FDB2 5AF0               staa  EEMCR             ; PROTLK is write-once
FDB4 7900F1             clr   BPROT             ; allow EE program and erase

FDB7 CEFE00             ldx   #$fe00            ; point to the table of user
                                                ; accessible routines.
FDBA 05E30000           jmp   [0,x]             ; the first entry is a pointer
                                                ; to main. GO.........


            ;     The following subroutine produces a delay of approximately
            ;     20 mS, based on the following conditions:

            ;     1.) An 8.00 MHz E-clock
            ;     2.) Subroutine located in external EPROM - selected by CSP0
            ;     3.) CSP0 programmed for 1 E-clock stretch
            ;
            ;     This routine is called by D-Bug12's WriteEEByte() function
            ;     through a pointer stored in the Customization Data Table.

FDBE        _EEDelay:
FDBE CE2710             ldx   #10000            ; load delay count into x
FDC1 09     DlyLoop:    dex                     ; decrement count
FDC2 26FD               bne   DlyLoop           ; loop till done.
FDC4 3D                 rts                     ; return.
```

APPENDIX D
D-BUG12 CUSTOMIZATION DATA

The Customization Data area, located in EPROM from `$FE80 to $FEFF`, allows users to change default data parameters used by D-Bug12. The data contained in this area is described by C data structure. The `CustomData` typedef is shown below. For those unfamiliar with C an assembly language equivalent is also shown. The purpose of each field is explained in the following paragraphs.

```
typedef struct {
                Byte UserCCR;               /* User CPU Condition Code Register */
                Byte UserB;                 /* User CPU B-accumulator */
                Byte UserA;                 /* User CPU A-accumulator */
                Address UserX;              /* User CPU X-index register */
                Address UserY;              /* User CPU Y-index register */
                Address UserPC;             /* User CPU Program Counter */
                Address UserSP;             /* User CPU Stack Pointer */
                unsigned long SysClk;       /* System Clock frequency (in Hz) */
                Address IOBase;             /* Base address of the I/O registers */
                unsigned int SCIBaudRegVal; /* Initial SCI BAUD register value */
                Address EEBase;             /* Base address of on-chip EEPROM */
                unsigned int EESize;        /* size of the on-chip EEPROM */
                void (*Delay)(void);        /* pointer to EEPROM program/erase */
                                            /* delay routine */
                int AuxCmdCount;            /* number of commands in the */
                                            /* auxiliary command table */
                CmdTblEntryP AuxCmdTableP;  /* pointer to the auxiliary command */
                                            /* table */
                } CustomData;
```

```
                    org    $FE80
;
CustData            equ    *
UserCCR             dc.b   $90          ; User CPU Condition Code Register
UserB               dc.b   $00          ; User CPU B-accumulator
UserA               dc.b   $00          ; User CPU A-accumulator
UserX               dc.w   $0000        ; User CPU X-index register
UserY               dc.w   $0000        ; User CPU Y-index register
UserPC              dc.w   $0000        ; User CPU Program Counter
UserSP              dc.w   $0A00        ; User CPU Stack Pointer
SysClk              dc.l   8000000      ; System Clock frequency (in Hz)
IOBase              dc.w   $0000        ; Base address of the I/O registers
SCIBaudRegVal dc.w  52           ; Initial SCI BAUD register value
EEBase              dc.w   $1000        ; Base address of the on-chip EEPROM
EESize              dc.w   4096         ; Size of the on-chip EEPROM
EEDelay             dc.w   _EEDELAY     ; Address of EEPROM program/erase delay
                                        ; routine
AuxCmdCount         dc.w   0            ; Number of commands in the auxiliary
                                        ; command table
AuxCmdTableP        dc.w   $0000        ; Pointer to the auxiliary command table
```

Initial User CPU Register Values

The first seven fields in the `CustomData typedef struct` are used to provide default values for the user CPU12 registers. The user CCR value is set to 0x90. This sets the S-bit, disabling the STOP instruction, and the I-bit, inhibiting IRQ interrupts. The X-bit is cleared to allow the use of the XIRQ interrupt as a programmer's abort switch. The user SP value is set to 0x0a00, which is one byte beyond the last on-chip RAM location available to the user. The CPU12 stack pointer points to the last byte pushed onto the stack. All of the other registers contain the value zero.

SysClk Field

The `SysClk` field is used to inform D-Bug12 of the system clock frequency, M. Its value, in Hz, is set to 8,000,000. The E-clock frequency is the same as the system clock frequency, M. `SysClk` is used by the D-Bug12 BAUD command in calculating the new value of the SCI Baud register for the requested baud rate.

NOTE

It is the responsibility of the startup code to perform any actions necessary to set the system clock frequency. D-Bug12 DOES NOT set or change the system clock frequency using the `SysClk` value.

IOBase Field

The `IOBase` field defines the base address of the I/O registers. This address is used by D-Bug12 when accessing the I/O registers associated with the SCI and when programming or erasing the on-chip EEPROM. On the MC68HC812A4 the I/O registers are mappable to any 2k memory space. Therefore, the `IOBase` entry should only be a multiple of 2048. The value of `IOBase` is set to 0x0000 which is the default address of the I/O registers for the MC68HC812A4.

NOTE

It is the responsibility of the startup code to set the base address of the I/O registers. D-Bug12 DOES NOT set or change the I/O register base address.

SCIBaudRegVal Field

The `SCIBaudRegVal` field is used to set the initial baud rate of the SCI used for console I/O by D-Bug12. Note that the value in `SCIBaudRegVal` is written directly to the Baud register of the console SCI. The value is NOT the desired baud rate. The calculation of this value is NOT made by D-Bug12 because of the possibility of an invalid Baud register value. Without a valid Baud register value during SCI initialization, D-Bug12 would have no way to inform the user that a problem existed. Not all combinations of baud rates and system clock frequencies produce a valid Baud register value. The formula used to calculate the Baud register value is:

```
   BaudRegVal = MCLK ÷ ( 16 * SCIBaudRate)
```

The initial Baud register value is 52 (0x0034). At a system clock frequency of 8.0 MHz, this sets the communications rate of 9600 baud.

NOTE

Because of the ability to choose either SCI0 or SCI1 for use as the control console, D-Bug12 takes care of initializing the SCI registers. The chosen SCI is set to 8-data bits, 1-start bit, 1-stop bit, and no parity.

EEBase and EESize Fields

The `EEBase` and `EESize` fields are used to describe the base address and range of the M68HC12's on-chip EEPROM. This information is used by D-Bug12's `WriteMem()` function to determine when a byte is being written to the on-chip EEPROM. D-Bug12 then calls its WriteEEByte() function to place the data in the on-chip EEPROM. On the MC68HC812A4 the EEPROM base address is mappable to any 4k memory space. Therefore, the `EEBase` entry should only be a multiple of 0x1000. The value of `EEBase` is set to 0x1000 which is the default base address of the on-chip EEPROM for the MC68HC812A4. The value of `EESize` is also set to 0x1000 (4096) which is the size of the on-chip EEPROM. Setting the value of `EESize` to zero disables the `WriteMem()` function's ability to write to on chip EEPROM.

NOTE

It is the responsibility of the startup code to set the base address of the EEPROM. D-Bug12 DOES NOT set or change the EEPROM base address.

EEPROM Erase/Program Delay Function Pointer Field

The `(void)(* Delay)(void)` field is a function pointer that points to an EEPROM program/erase delay routine. For the MC68HC812A4, the routine should produce a delay of 20 mS before it returns. The delay routine is nothing more than a software delay loop. The subroutine is located in the startup code area of the D-Bug12 EPROM from `$FD80 - $FDFF`. See Appendix C, D-Bug12 Startup Code.

Auxiliary Command Table Entries

The last two entries in this table provide a mechanism to extend the command set of D-Bug12. The `AuxCmdTableP` points to an auxiliary command table, and `AuxCmdCount` contains the number of entries in the auxiliary command table. The table consists of an array of `CmdTblEntry's.` Each `CmdTblEntry` in the auxiliary command table has the following structure:

```
typedef struct {
      const char *CommandStr;                  /* pointer to the  command */
                                               /*string */
      int (*ExecuteCmd)(int argC, char *argV[]);/* pointer to function that*/
                                               /* implements the command */
               } CmdTblEntry, * CmdTblEntryP;
```

As the `typedef` shows, the first field is a character pointer pointing to a null terminated character array containing the command name. The command name string *must be in upper case*. The second field, a function pointer, points to a function that implements the new D-Bug12 command. The first parameter to this function is a count of the number of command line arguments that the command line interpreter found on the command line. This count *includes* the command name itself. The command line may contain no more than a total of 10 parameters. The second function parameter is a pointer to an array of `char *`. Each `char *` points to one of the command line parameters parsed by the command line interpreter.

The function implementing the new command can report any error conditions to the user in one of two ways. If the error condition can be described by one of the error messages in the enumerated constant list below, the user defined command should return the appropriate constant. If some other message text needs to be conveyed to the user, the command should communicate the error message directly to the user by using the `printf()` function which is one of the available user callable functions. In this case, the user defined command should return an error code of noErr.

```
enum Error {
            WrongNumArgs = 6,    /* Wrong Number of Arguments */
            BadStartAddress = 7, /* Invalid Starting Address */
            BadEndAddress = 8,   /* Invalid Ending Address */
            StartEndError = 9,   /* Start Address Greater Than End Address */
            BadHexData = 10,     /* Invalid Hex Data */
            DataSizeError = 11,  /* Data Out Of Range */
            NoTargetWrite = 12,  /* Can't Write Target Memory */

           };
```

APPENDIX E
CUSTOMIZING THE EPROMS

The following blocks in the factory-supplied EPROMs can be reprogrammed with user code or D-Bug12 code that has been modified for custom operation:

$8000 - $9FFF — available for user programs

$FD80 - $FDFF — D-Bug12 startup code. See Appendix C.

$FE80 - $FEFF — D-Bug12 customization data. See Appendix D.

$FF00 - $FFBF — available for user programs

Since the EPROMs also contain D-Bug12 and other EVB operating firmware, the factory programming must be retained and burned into the custom chips along with the custom code. The table below maps the EVB's logical addresses (from Table 3-5) to the pin-level physical addresses of U7 and U9A.

Note that the lower half of each EPROM — from $0000 to $3FFF — is unused and is filled with ones. This is necessary because of the chip select, CSP0*, used by the MCU for EPROM access. For more information on this subject, refer to **4.6.2 Chip Selects**.

NOTE

Do not reprogram the factory-supplied EPROMs. Keep them as masters, using expendable chips for new programming.

Physical EPROM Addresses

MCU Logical Address	Data	U9A Physical Address	U7 Physical Address
—	$FF	$0000 - $3FFF	$0000 - $3FFF
$8000 - $9FFE even addresses	custom	$4000 - $4FFF	—
$8001 - $9FFF odd addresses	custom	—	$4000 - $4FFF
$A000 - $FD7E even addresses	factory	$5000 - $7EBF	—
$A001 - $FD7F odd addresses	factory	—	$5000 - $7EBF
$FD80 - $FDFE even addresses	factory or modified	$7EC0 - $7EFF	—
$FD81 - $FDFF odd addresses	factory or modified	—	$7EC0 - $7EFF
$FE00 - $FE7E even addresses	factory	$7F00 - $7F3F	—
$FE01 - $FE7F odd addresses	factory	—	$7F00 - $7F3F
$FE80 - $FEFE even addresses	factory or modified	$7F40 - $7F7F	—
$FE81 - $FEFF odd addresses	factory or modified	—	$7F40 - $7F7F
$FF00 - $FFBE even addresses	custom	$7F80 - $7FBF	—
$FF01 - FFBF odd addresses	custom	—	$7F80 - $7FBF
$FFC0 - $FFFE even addresses	factory	$7FC0 - $7FFF	—
$FFC1 - $FFFF odd addresses	factory	—	$7FC0 - $7FFF

APPENDIX F
SDI CONFIGURATION

To configure the EVB for use with Motorola's Serial Debug Interface (SDI), follow these steps:

1. Remove the jumper on header W11 from CSD*.
2. Move the CSP0* jumper on W11 to pins 2-3.

   Steps 1 and 2 disable the external EPROM and map the CSP0* chip select to external RAM.
3. Remove the jumper from W30.

   Step 3 allows the SDI to drive the MCU's BKGD pin low at reset.
4. Move the jumper on W34 to pins 1-2.
5. Move the jumper on W42 to pins 1-2.

   Steps 4 and 5 place the MCU in Special Single Chip mode.
6. Move the base address of the MCU's on-chip EEPROM from $F000 (the default in Special Single Chip mode) to $1000. To do this, change the data at address $0012 to a value of $11 using the appropriate debugging tool. For MCUdebug, the correct command is:

   ```
   MM 12 11
   ```

   Step 6 must be repeated each time the EVB is reset in this mode, as the EEPROM's base address defaults to $F000 at reset.

Table 4-1 provides full descriptions of these jumper changes. See Figure 4-2 for details of header W11. See Figure 1-1 for header locations on the EVB.

Note that CSP0* covers the address range from $8000 to $FFFF. The 16 Kbytes of RAM appear in the new memory map from $C000 to $FFFF. This SDI memory map is shown in the table below.

This configuration provides the following enhancements when using the SDI:

- The MCU's on-chip RAM, from $0800 to $0BFF, is entirely available for user data.
- Data can be loaded into the vector area, which was reserved under the D-Bug12 operating configuration.

For information on using the SDI, refer to the *Motorola Serial Debug Interface User's Manual*.

SDI Memory Map

Address Range	Description	Location
$0000 - $01FF	CPU registers	on-chip (MCU)
$0800 - $0BFF	user data area	1K on-chip RAM (MCU)
$1000 - $1FFF	user code area	4K on-chip EEPROM (MCU)
$C000 - $FFFF	user code/data area	16K external RAM (U4, U6A)

INDEX