TMS320C6000 Memory Test

Dave Bell
Sebastien Tomas

TMS320C6000 Applications

ABSTRACT

This set of programs has been compiled to provide a way to verify the integrity of internal DSP memory and external system memory for all devices currently in the TMS320C6000™ (C6000™) family. Included with the memory test are all source files, the Code Composer Studio™ project file, and the linker command file. The source files contain the necessary parameters to test all devices within the C6000 family; the particular device is selected by passing a preprocessor variable to the assembler during compile time. Internal device memory is tested in its entirety and external memory can be added to the test by including a memory table in one of the source files. Switching between C6000 devices and systems only requires modifying the external memory table and passing the corresponding device name to the assembler. All other device considerations are handled by the code. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SPRA630.

Contents

1 Program Description

In order to facilitate the verification and integrity of internal DSP memory and external system memory for all devices currently in the TMS320C6000 (C6000) family, a test has been put together. The C6000 memory verification program exists for the purpose of verifying the functionality of internal and external memory sections in any of the C6000 DSPs. The test program, at a minimum, verifies all internal memory locations to ensure that data can be read from and written to accurately. If desired, external sections can be included in the test coverage by providing a memory description table, containing the memory section address and size.

Five test patterns are run to verify each section of memory: all 1's, alternating 1-0, alternating 0-1, all 0's, and a ramp function. If all five tests pass, the CPU spins in a pass loop upon completion.

If any test fails, the program is immediately aborted and the CPU branches to a fail loop. The address of the memory fail is held in register A5, the test value is held in A4, and the stored value is held in A3.

1.1 File Descriptions

The memory verification program is made up of several source files, each of which has a particular function. In order to incorporate the ability to test the internal memory of multiple internal architectures, there are two basic file sets: those for the Harvard architecture devices (C6x0x) and those for the two-level cache architecture devices (C6x1x). Both file sets perform the same function: to transfer different patterns of data to all sections of memory and check for failures. A single program controls the program flow, and to verify system memory the user can easily add external sections to the test flow.

The memory verification program contains the following source files:

- c6000memtest.asm: This is the main program, which contains the control code and the interrupt vector table. This control code consists of the calling function for the internal memory test, the pass and fail loops entered upon completion; and the interrupt vector table used to recognize the end of DMA activity during program execution. Several constants used in the other source files are defined in this file as well.
- c6000imem.asm: This file contains information describing the location and sizes of the internal memory on each of the C6000 DSPs. As new C6000 devices become available, descriptions of their internal memory can be added here to expand the test coverage. Likewise, sections of internal memory could potentially be removed from the screen if desired. Current devices for which the internal memory is defined include the TMS320C6201,C6701, C6202, C6203, C6204, C6205, C6211, and C6711.
- c6000emem.asm: This file contains user-defined information describing the external memory sections to be included in the verification. This is in table form, and should be entered by the user according to the system memory available to be tested. Each table entry includes the section size in bytes and section start address. The end of the table is indicated by an entry of zero (a section of zero size). If no external sections are to be tested, then the first entry in the table must be zero. The EMIF control registers are *not* configured by the test program and must be programmed either through the emulator or by a host prior to running the test suite.

- c6x0xint_test.asm: This file contains the internal data memory and internal program memory (block 0) test for all C6x0x DSPs. After completion, the CPU branches to the program memory block 1 test program.
- c6x0xint1_test.asm: This file contains the internal program memory (block 1) test for all C6x0x DSPs. After completion, the CPU branches to the external memory test.
- c6x0xext_test.asm: This file contains the external memory test for all C6x0x DSPs. The memory table defined in c6000emem.asm is used to dictate which external memory sections are tested. The external memory test runs until a section size of zero is loaded from the external memory table in c6000emem.asm. After completion the CPU branches to a pass loop (or fail).
- c6x1xint_test.asm: This file contains the internal L2 memory test for all C6x1x DSPs. Each block of L2 memory is tested. After completion the CPU branches to the external memory test.
- c6x1xext_test.asm: This file contains the external memory test for all C6x1x DSPs. The memory table defined in c6000emem.asm is used to dictate which external memory sections are tested. The external memory test runs until a section size of zero is loaded from the external memory table in c6000emem.asm. After completion the CPU branches to a pass loop (or fail).
- c6000memtest.cmd: This is the linker command file. This file contains two sets of link options—one for C6x0x devices in Map 1 or all C6x1x devices, and one for C6x0x devices in Map 0. The appropriate portion of the file must be uncommented by the user prior to linking.
- c6000memtest.mak: This is the CCS project file. This file contains the project information for the memory verification program. This project should be loaded into Code Composer Studio to edit and build the memory test.
- c6000memtest.out: This is the COFF generated by building the c6000memtest project. This file is loaded into the DSP memory for testing.

2 Edit and Build Process

The project source files can easily be edited to suite different C6000 DSPs as well as different systems. There are several things that must be changed within the source, as well as preprocessor variables that can be passed to the compiler to rebuild the test. The following steps should be followed to build the project to test the memory in a C6000 system:

- Start Code Composer Studio
- Load the project c6000memtest.mak
- Open c6000emem.asm and modify memory table to reflect C6000 system. Comments are provided in the source along with an example memory list.
- Open project options and select the “assembler” tab. Manually insert “-DDEVICE=6xxx” (where xxx is 201, 701, 202, 203, 204, 205, 211, 711) into the options field. This flag is passed to all source files to select the appropriate memory configurations to use for the test.
- Build the project

After following these steps a new c6000memtest.out file is generated and can be loaded into the DSP and run.

2.1 Example Build

An example of how to modify and build the memory test is as follows. Consider a C6211 system with 16MB of SDRAM in external memory CE0 and 8kB of asynchronous memory in CE1. First, the project must be loaded into CCS (c6000memtest.mak). To include the external memory in the test, the table in c6000emem.asm must be modified to the following:

```
EMEM_TBL:
                  ;       section size (bytes), memory address
                  .word        0x01000000     , 0x80000000         ; Section 1
                  .word        0x00004000     , 0x90000000         ; Section 2
                  .word        0x00000000                          ; END
                  .label table_bottom
```

To compile the test for the C6211, the assembler options in Code Composer Studio must be modified to include “-DDEVICE=6211”.

To generate a new c6000memtest.out file simply build the project.

The .out file is then ready to load and be run on the C6211. Note that the EMIF configuration to communicate to the SDRAM and asynchronous memory is *not* included in the test suite. Before executing the code it is necessary to configure the EMIF appropriately. This can be done either with the use of a GEL script or by simply editing the appropriate registers in the debugger. For details on how to write GEL scripts, see [1]. For details on the EMIF control registers, see [2].

3 Summary

In summary, the TMS320C6000 memory test is a configurable test suite capable of being run on all C6000 devices in different systems. The test includes a verification of internal memory as well as any user-defined external memory sections. To select the device and system memory involves creating/modifying the external memory table and passing the correct device number to the assembler as a preprocessor variable in the assembler options.

4 References

1. *Code Composer Studio Reference Guide* (literature number SPRU328)
2. *TMS320C6000 Peripherals Reference Guide* (literature number SPRU190)

Appendix A Memory Test Source Files

A.1 c6000memtest.asm

```
; c6000mem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this program is to verify the functionality of all internal
; memory on any 'C6000 DSP, as well as any external memory sections specified
; in c6000emem.asm.
;
; Four test values are used to verify the memory: all 1's, alternating 1-0,
; alternating 0-1, all 0's, and a ramp function. If all five tests pass, the CPU
; spins in a pass loop upon completion.
;
; If any test fails, the program is immediately aborted. The CPU branches to
; a fail loop. The address of the memory fail is held in register A5, the
; test value is held in A4, and the stored value is held in A3.
;
; Any new C6000 DSP can be tested by this program by doing the following:
;               - Add internal memory description in c6000imem.asm
;               - If a C6x1x device, add two lines to this source file:
;                           .elseif DEVICE = 6x1x (replace x's with actual)
;                 C6000     .set 1
;
; Assumptions made in the test case include:
;     - C6x0x devices have: 1 internal data memory block
;                           2 internal program memory blocks
;     - C6x1x devices have: n internal memory blocks, where n is specified
;                           in C6000imem.asm
;     - All code is located at the base address of block 0.
;     - Contents of all memory (aside from the memory test code) will be
        overwritten during the course of the test.


            .global  _main
            .global  RESET
            .global  DTEST
            .global  PTEST0
            .global  PTEST1
            .global  L2TEST
            .global  ETEST
            .global  DMA
            .global  QDMA
            .global  EDMA
            .global  EDMA_CTRL
            .global  pass
            .global  fail

                   .if $isdefed ("DEVICE")
                   .if DEVICE = 6211
```

```
C6000       .set   1
                   .elseif DEVICE = 6711
C6000       .set   1
                   .else ; DEVICE = 6x0x
C6000       .set   0
                   .endif
                   .else ; DEVICE not specified...6201
C6000       .set   0
                   .endif

DTEST       .set   0
PTEST0      .set   1
PTEST1      .set   2
ETEST       .set   3
L2TEST      .set   0
DMA         .set   0x01840000
QDMA        .set   0x02000000
EDMA        .set   0x01A00000
EDMA_CTRL   .set   0x01A0FFE0

            .text
_main:
;-------------------------------------------------------------------------------;
; Initialize interrupt 8 for DMA Ch0 and interrupt 12 for DMA Ch3               ;
;-------------------------------------------------------------------------------;

            MVC      CSR, B0          ; Get Control Status Register
            OR       B0, 1, B0        ; Set GIE bit
            MVC      B0, CSR          ; Restore CSR
            MVC      IER, B0          ; Get Interrupt Enable Register
            SET      B0, 1, 1, B0     ; Set NMIE bit
            SET      B0, 8, 8, B0     ; Set IE8 bit
            SET      B0, 12, 12, B0   ; Set IE12 bit
            MVC      B0, IER          ; Restore IER
            MVKL     RESET, B0        ; Base address for vector table
            MVKH     RESET, B0
            MVC      B0, ISTP         ; Set IST address

;-------------------------------------------------------------------------------;
; Branch to internal memory test                                                ;
;-------------------------------------------------------------------------------;

            .if C6000 = 0
            .ref     _c6x0xint_test
            MVKL     _c6x0xint_test, B3
            MVKH     _c6x0xint_test, B3
            .else; if C6000 = 1
            .ref     _c6x1xint_test
            MVKL     _c6x1xint_test, B3
            MVKH     _c6x1xint_test, B3
            .endif

            B        B3
            NOP      5
```

```
;-------------------------------------------------------------------------------;
; PASS -- Loop forever                                                          ;
;-------------------------------------------------------------------------------;

pass:
            B       pass
            NOP     5

;-------------------------------------------------------------------------------;
; FAIL -- Loop forever                                                          ;
;-------------------------------------------------------------------------------;

fail:
            B       fail
            NOP     5

;-------------------------------------------------------------------------------;
; Interrupt Service Table                                                       ;
;-------------------------------------------------------------------------------;

              .sect ".vectors"
              .align 0x400

RESET B       _main
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
NMI   B       NMI
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
RESV1 B       RESV1
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
RESV2 B       RESV2
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
```

```
INT4  B        INT4
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT5  B        INT5
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT6  B        INT6
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT7  B        INT7
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
            .if C6000 = 0
INT8  B        IRP
      STW      B7, *+B4[2]     ; Clear BLOCK COND
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
            .else ; if C6000 = 1
INT8  B        IRP
 ||   LDW      *+B14[1], A11   ; Get CIPR value
      NOP
      NOP
      NOP
      NOP
      STW      A11,*+B14[1]    ; Clear the pending CIP
      NOP
            .endif
```

```
INT9  B         INT9
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT10 B         INT10
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT11 B         INT11
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT12 B         IRP
      ZERO      B0              ; Set loop condition to false
      STW       B7, *+B4[19]    ; Clear BLOCK COND
      NOP
      NOP
      NOP
      NOP
      NOP
INT13 B         INT13
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT14 B         INT14
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
INT15 B         INT15
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
```

A.2 c6000imem.asm

```
; c6000imem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this file is to define various contstants used in the
; c6000memtest program. The constants define the starting addresses and
; sizes of the internal memories for the various C6000 processors. As
; new C6000 DSPs become available, they can be incorporated into this
; test flow by adding entries to this file.
;
; Each C6x0x device needs to have  two program memory blocks and a single
; data memory block defined. C6x1x devices need to provide L2 memory
; block information.
;
; Current devices listed include:
;     C6201, C6701, C6202, C6203, C6204, C6205, C6211, C6711
;
; The LEN_BLOCK defined at the end of the file is the size of the
; working block. This size can be modified if desired.

                    .global DMEM
                    .global PMEM0
                    .global PMEM1
                    .global LEN_D
                    .global LEN_P0
                    .global LEN_P1
                    .global NUM_L2
                    .global LEN_L2
                    .global NUM_CE
                    .global LEN_BLOCK

                    .if $isdefed (”DEVICE”)
                    .if DEVICE = 6201
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .elseif DEVICE = 6701
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .elseif DEVICE = 6202
PMEM0               .set    0x00000000
LEN_P0              .set    0x00020000
PMEM1               .set    0x00020000
LEN_P1              .set    0x00020000
DMEM                .set    0x80000000
```

```
LEN_D               .set    0x00020000
                    .elseif DEVICE = 6203
PMEM0               .set    0x00000000
LEN_P0              .set    0x00040000
PMEM1               .set    0x00040000
LEN_P1              .set    0x00020000
DMEM                .set    0x80000000
LEN_D               .set    0x00040000
                    .elseif DEVICE = 6204
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .elseif DEVICE = 6205
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .elseif DEVICE = 6211
NUM_L2              .set    4
LEN_L2              .set    0x00004000
NUM_CE              .set    4
                    .elseif DEVICE = 6711
NUM_L2              .set    4
LEN_L2              .set    0x00004000
NUM_CE              .set    4
                    .else ; if no or invalid device, use 6201
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .endif ; DEVICE
                    .else ; if no device specified
PMEM0               .set    0x00000000
LEN_P0              .set    0x00008000
PMEM1               .set    0x00008000
LEN_P1              .set    0x00008000
DMEM                .set    0x80000000
LEN_D               .set    0x00010000
                    .endif ; $isdefed (”DEVICE”)

LEN_BLOCK           .set    0x00040000
```

A.3 c6000emem.asm

```
; c6000emem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this file is to location and size information for any ex-
; ternal memory location to be tested by the c6000memtest program. The file
; is simply a table with the section size in bytes and section start ad-
; dress for each entry pair. The end of the table is indicated by an entry
; of zero. If no external sections are to be tested, then the first entry
; in the table must be zero.
                  .global EMEM_TBL
                  .global table_top
                  .global table_bottom

                  .sect  ”memorytable”
                  .align 0x8
                  .label table_top
EMEM_TBL:
                  ; Enter external memory section information here:
                  ;       section size (bytes), memory address

                  ; example:
;                 .word       0x00100000     , 0x00400000        ; Section 1
;                 .word       0x00200000     , 0x02000000        ; Section 2
;                 .word       0x001004c0     , 0x02200000        ; Section 3
;                 .word       0x01000000     , 0x03000000        ; Section 4
;                 .word       0x00100000     , 0x80000000        ; Section 5
;                 .word       0x00200000     , 0x81000000        ; Section 6
;                 .word       0x001004c0     , 0x83000000        ; Section 7

                  ; leave this declaration to indicate the end of the section
                  ; listing.
                  .word       0x00000000                          ; END
                  .label table_bottom
```

A.4 c6x0xint_mem.asm

```
; c6x0xint_mem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this code is to verify the internal data memory and block
; 0 of the internal program memory of a C6x0x DSP. When complete, the CPU
; branches to the test of block 1 of the C6x0x internal program memory.
;
; The attributes of program memory block 0 are located in c6000.imem.asm,
; and can be modified to suit any new c6x0x-style devices that become
; available.
;
; Assumptions made in this code are:
;     - The size of block 0 is larger than the working data block
;       size (LEN_BLOCK).
;     - The program code is located within program memory block 0, and is
;       smaller than the working data block size.

                    .text

                    .if $isdefed ("DEVICE")
                    .if DEVICE = 6211
C6000       .set    1
                    .elseif DEVICE = 6711
C6000       .set    1
                    .else ; DEVICE = 6x0x
C6000       .set    0
                    .endif
                    .else ; DEVICE not specified...6x0x
C6000       .set    0
                    .endif

                    .if C6000 = 0

                    .ref    DMA
                    .ref    PMEM0
                    .ref    LEN_P0
                    .ref    PMEM1
                    .ref    DMEM
                    .ref    LEN_D
                    .ref    DTEST
                    .ref    PTEST0
                    .ref    LEN_BLOCK
                    .ref    _c6x0xint1_test
                    .ref    table_bottom
                    .ref    RESET
                    .ref    fail

                    .global _c6x0xint_test
                    .global c6x0xint_loop
                    .global data_fill
                    .global data_check
                    .global pmem0_check0
```

```
                .global pmem0_check1
                .global pmem0_loop
                .global pmem0_end
                .global pass_c6x0xint
                .global fail_c6x0xint

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_c6x0xint_test:
;-------------------------------------------------------------------------------;
; Init test value                                                               ;
;-------------------------------------------------------------------------------;

            MVK     4, B2             ; Set program test count
c6x0xint_loop:
            ZERO    B0
            ZERO    B1
            ZERO    A1
            ZERO    A2
            CMPEQ   B2, 1, B0
            CMPEQ   B2, 2, B1
            CMPEQ   B2, 3, A1
            CMPEQ   B2, 4, A2

      [ A2] MVKL    0xFFFFFFFF, A4    ; Test 4: Fill with F's
      [ A2] MVKH    0xFFFFFFFF, A4
      [ A1] MVKL    0xAAAAAAAA, A4    ; Test 3: Fill with A's
      [ A1] MVKH    0xAAAAAAAA, A4
      [ B1] MVKL    0x55555555, A4    ; Test 2: Fill with 5's
      [ B1] MVKH    0x55555555, A4

      [B0]  ZERO    A4                ; Test 1: Step beginning with 1
      [!B2] MVKL    0x00020001, A4    ; Test 0: Step beginning with 1
      [!B2] MVKH    0x00020001, A4

            MVKL    DMEM,    A5      ; Get base address for dmem
            MVKH    DMEM,    A5
            MVKL    LEN_D    A6      ; Get length of working block
            MVKH    LEN_D,   A6

;-------------------------------------------------------------------------------;
; Fill data memory                                                              ;
;-------------------------------------------------------------------------------;

            SHR     A6, 2, A1         ; Set loop count to word count
            MV      A5, A9
            MVK     1, A2             ; Set loop condition to TRUE
            ADD     A4, 1, B14        ; increment by 0x00020002
data_fill:
      [ A1] B       data_fill
      [ A1] STW     A4, *A9++
      [ A1] SUB     A1, 1, A1         ; Decrement count
      [!B2] ADD2    ADD2     A4, B14, A4       ; Increment step if test 0
            NOP
            NOP
```

```
;-------------------------------------------------------------------------------;
; Check data memory for failures                                                ;
;-------------------------------------------------------------------------------;

            SHR     A6, 2, A1         ; Set loop count to word count
            MV      A5, A9            ; Set A9 to DMEM
            MVK     1, A2             ; Set loop condition to TRUE
      [!B2] SUB     B14, 1, A4        ; Return original A4 value

data_check:
      [ A1] LDW     *A9++, A3         ; Load data value
            NOP     4
      [ A1] CMPEQ   A3, A4, A2        ; Compare data to test value
      [!A2] B       fail_c6x0xint     ; Branch to fail loop
      [!A2] SUB     A9, 4, A5         ; Correct address of fail
      [!A2] MVK     DTEST, B3
      [!A2] SHL     B3,4,B3           ; Set error code = 0x000000MT.
      [!A2] ADD     B3, B2, B3        ;    M = memory block, T = test count
      [ A1] SUB     A1, 1, A1         ; Decrement count
      [ A1] B       data_check        ; Loop until done
      [!B2] ADD2    A4, B14, A4       ; Increment step if test 0
            NOP     4

;-------------------------------------------------------------------------------;
; Verify program memory where the code is located                               ;
; To do this, it is necessary to do the following steps:                        ;
;     1. DMA program out of pmem0 to known good dmem                            ;
;     2. DMA test pattern from dmem to pmem0                                    ;
;     3. DMA test pattern back from pmem0 to dmem                               ;
;     4. DMA code back from dmem to pmem0                                       ;
;                                                                               ;
; Setup DMA from PMEM0 to DMEM (step 1)                                         ;
;-------------------------------------------------------------------------------;

            MVKL    DMA, B4
            MVKH    DMA, B4
            MVKL    RESET, A8
            MVKH    RESET, A8
            MVKL
                    DMEM, B5
            MVKH    DMEM, B5
            MVKL    LEN_BLOCK, B6
            MVKH    LEN_BLOCK, B6
            ADD     B5, B6, B9      ; Get location just after data block
                                    ; DMEM + LEN_BLOCK
            SHR     B6, 2, A13      ; Convert from bytes to words
            MVKL    0x00000050, A6  ; Src inc, Dst inc
            MVKH    0x00000050, A6  ; TCINT = 0
            MVK     0x0000, B7      ; BLOCK IE = 0
            STW     A6, *+B4[0]     ; Set pri ctrl
            STW     B7, *+B4[2]     ; Set sec ctrl
            STW     A8, *+B4[4]     ; Set source = PMEM0
            STW     B9, *+B4[6]     ; Set dest = DMEM + LEN_BLOCK
            STW     A13, *+B4[8]     ; Set count = block size
```

```
;-------------------------------------------------------------------------------;
; Setup DMA from DMEM to PMEM0 + (step 2)                                       ;
;-------------------------------------------------------------------------------;

            MVKL     PMEM0, B8
            MVKH     PMEM0, B8
            MVKL     LEN_P0, A7
            MVKH     LEN_P0, A7

            STW      A6, *+B4[16]    ; Set pri ctrl
            STW      B7, *+B4[18]    ; Set sec ctrl
            STW      A5, *+B4[20]    ; Set source = DMEM
            STW      B8, *+B4[22]    ; Set dest = PMEM0
            STW      A13, *+B4[24]   ; Set count = Block size

;-------------------------------------------------------------------------------;
; Setup DMA from PMEM0 to DMEM  + (step 3)                                      ;
;-------------------------------------------------------------------------------;

            STW      A6, *+B4[1]    ; Set pri ctrl
            STW      B7, *+B4[3]    ; Set sec ctrl
            STW      B8, *+B4[5]    ; Set source = PMEM
            STW      A5, *+B4[7]    ; Set dest = DMEM
            STW      A13, *+B4[9]    ; Set count = Block size

;-------------------------------------------------------------------------------;
; Setup DMA from DMEM back to PMEM0 (step 4)                                    ;
;-------------------------------------------------------------------------------;

            MVKL     0x02000050, A9  ; Src inc, Dst inc
            MVKH     0x02000050, A9  ; TCINT = 1
            MVK      0x80, B7        ; BLOCK_IE = 1
            STW      A9, *+B4[17]    ; Set pri ctrl
            STW      B7, *+B4[19]    ; Set sec ctrl
            STW      B9, *+B4[21]    ; Set source = DMEM1 + LEN_BLOCK
            STW      A8, *+B4[23]    ; Set dest = PMEM0
            STW      A13, *+B4[25]    ; Set count = block size


            ADD      A6, 1, A6       ; Set start = 01b
            ADD      A9, 1, A9       ; Set start = 01b
            STW      A6, *+B4[0]     ; Start DMA 0
            STW      A6, *+B4[16]    ; Start DMA 1
            STW      A6, *+B4[1]     ; Start DMA 2
            STW      A9, *+B4[17]    ; Start DMA 3

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE
```

```
;-------------------------------------------------------------------------------;
; Check internal data memory for program memory failures                        ;
;-------------------------------------------------------------------------------;

            MV       A13, A1          ; Set loop count section word count
            MV       A5, B9           ; Set B9 to DMEM
            MVK      1, A2            ; Set loop condition to true
      [!B2] SUB      B14, 1, A4       ; Return original A4 value
pmem0_check0:
      [ A1] LDW      *B9++, A3        ; Load data value
            NOP      4
      [ A1] CMPEQ    A3, A4, A2       ; Compare data to test value
      [!A2] B        fail_c6x0xint    ; Branch to fail loop
      [!A2] SUB      B9, 4, A5        ; Correct address of fail
      [!A2] MVK      PTEST0, B3
      [!A2] SHL      B3,4,B3          ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3       ;    M = memory block, T = test count
      [ A1] SUB      A1, 1, A1        ; Decrement count
      [ A1] B        pmem0_check0     ; Loop until done
      [!B2] ADD2     A4, B14, A4      ; Increment step if test 0
            NOP      4

;-------------------------------------------------------------------------------;
; Verify remainder of program memory block 0.                                   ;
;     To do this, it is necessary to do the following steps:                    ;
;         1. DMA test pattern from dmem to pmem0                                ;
;         2. DMA test pattern back from pmem0 to dmem                           ;
;     The code must loop, testing the appropriate number of test blocks, as     ;
;     determined in A11, plus any remainder in A12.                             ;
;-------------------------------------------------------------------------------;

            MVKL     LEN_P0, B7
            MVKH     LEN_P0, B7
            LMBD     1, B6, B10       ; Get left-most bit of block size
            MVK      31, A8           ; Set MS bit position
            SUB      A8, B10, A8      ; Find bit position of lmb
            SHL      B10, 5, B11      ; Place lmbd val in bits 9:5 (csta)
            ADD      B11, B10, B11    ; Set up for extu (cstb = csta)
            EXTU     B7, B11, B11     ; A12 = LEN_PO % LEN_BLOCK
            SHR      B7, A8, A11      ; A11 = LEN_PO / LEN_BLOCK
            SHR      B11, 2, A12      ; Shift bytes to words

            MVKL     DMA, B4
            MVKH     DMA, B4
            MVKL     RESET, A8
            MVKH     REST, A8
            MVKL     DMEM, B5
            MVKH     DMEM, B5
            MVKL     LEN_BLOCK, B6
            MVKH     LEN_BLOCK, B6
            ADD      B5, B6, B9       ; Get location just after data block
                                      ; DMEM + LEN_BLOCK
```

```
            MVKL     PMEM0, B8
            MVKH     PMEM0, B8
            MVKL     LEN_P0, A7
            MVKH     LEN_P0, A7
            MV       A13, B0         ; Set conditional req to block size
            MV       A11, B1         ; Set conditional reg to block count
pmem0_loop:

;-------------------------------------------------------------------------------;
; Setup DMA from DMEM to PMEM0 (step 1)                                         ;
;-------------------------------------------------------------------------------;

      [!B1] SUB      B1, 1, B1
      [ B1] MV       A13, B0          ; Set conditional reg to block size
      [!B1] MV       A12, B0          ; If no more blocks, set count to remainder
      [!B0] B        pmem0_end        ; Skip pmem test if no remaining count
            NOP      5

            ADD      B8, B6, B8       ; Increment pmem0 start
            MVKL     0x00000050, A6   ; Src inc, Dst inc
            MVKH     0x00000050, A6   ; TCINT = 0
            MVK      0x0000, B7       ; BLOCK_IE = 0
            STW      A6, *+B4[16]     ; Set pri ctrl
            STW      B7, *+B4[18]     ; Set sec ctrk
            STW      A5, *+B4[20]     ; Set source = DMEM
            STW      B8, *+B4[22]     ; Set dest = PMEM0
            STW      A13, *+B4[24]    ; Set count = Block size

;-------------------------------------------------------------------------------;
; Setup DMA from PMEM0 to DMEM (step 2)                                         ;
;-------------------------------------------------------------------------------;

            MVKL     0x00000050, A9  ; Src inc, Dst inc
            MVKH     0x00000050, A9  ; TCINT = 0
            MVK      0x0080, B7      ; BLOCK_IE = 1
            STW      A9, *+B4[17]    ; Set pri ctrl
            STW      B7, *+B4[19]    ; Set sec ctrk
            STW      B8, *+B4[21]    ; Set source = PMEM0
            STW      A5, *+B4[23]    ; Set dest = PMEM0
            STW      A13, *+B4[25]   ; Set count = block size

            ADD      A6, 1, A6       ; Set start = 01b
            ADD      A9, 1, A9       ; Set start = 01b
            STW      A6, *+B4[16]    ; Set DMA 1
            STW      A9, *+B4[17]    ; Set DMA 3

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE
```

```
;-------------------------------------------------------------------------------;
; Check internal data memory for program memory failures                        ;
;-------------------------------------------------------------------------------;

            MV       A13, A1          ; Set loop count section to word count
            MV       A5, B9           ; Set B9 to DMEM
            MVK      1, A2            ; Set loop condition to true
      [!B2] SUB      B14, 1, A4       ; Return original A4 value

pmem0_check1:

      [ A1] LDW      *B9++, A3        ; Load data value
            NOP      4
      [ A1] CMPEQ    A3, A4, A2       ; Compare data to test value
      [!A2] B        fail_c6x0xint    ; Branch to fail loop
      [!A2] SUB      B9, 4, A5        ; Correct address of fail
      [!A2] MVK      PTEST0, B3
      [!A2] SHL      B3, 4, B3        ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3       ; M = memory block, T = test count
      [ A1] SUB      A1,  1, A1       ; Decrement count
      [ A1] B        pmem0_check1     ; Loop until done
      [!B2] ADD2     A4, B14, A4      ; Increment step if test 0
            NOP      4
            B        pmem0_loop       ; Go back to start of loop
            NOP      5

;-------------------------------------------------------------------------------;
; Loop back for another test                                                    ;
;-------------------------------------------------------------------------------;

pmem0_end:
 [ B2] B        c6x0xint_loop
      [ B2] SUB      B2, 1, B2
            NOP      4

;-------------------------------------------------------------------------------;
; Pass -- Branch to test of second program memory block                         ;
;-------------------------------------------------------------------------------;

pass_c6x0xint:
            MVKL     _c6x0xint1_test, B12  ; Address of 2nd memory test
            MVKH     _c6x0xint1_test, B12  ;
            B        B12
            NOP      5

;-------------------------------------------------------------------------------;
; Fail -- Branch to fail routine                                                ;
;-------------------------------------------------------------------------------;

fail_c6x0xint:
            MVKL     fail, B12
            MVKH     fail, B12
            B        B12
            NOP      5

                  .endif
```

A.5 c6x0xint1_mem.asm

```
; c6x0xint1_mem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this code is to verify the second half of a C6x0x DSP's
; internal program memory. When complete, the CPU branches to the C6x0x
; external memory test.

; The attributes of program memory block 1 are located in c6000imem.asm,
; and can be modified to suite any new c6x0x-style devices that become
; available
;
; Assumptions made in this code are:
;     - The size of block 1 is larger than the working data block
;       size (LEN_BLOCK).

                    .text

                    .if $isdefed ("DEVICE")
                    .if DEVICE = 6211
C6000       .set    1
                    .elseif DEVICE = 6711
C6000       .set    1
                    .else ; DEVICE = 6x0x
C6000       .set    0
                    .endif
                    .else ; DEVICE not specified...6202
C6000       .set    0
                    .endif

                    .if C6000 = 0

                    .ref    DMA
                    .ref    PMEM1
                    .ref    LEN_P1
                    .ref    LEN_BLOCK
                    .ref    DMEM
                    .ref    PTEST1
                    .ref    _c6x0xext_test
                    .ref    fail

                    .global _c6x0xint1_test
                    .global c6x0xint1_loop
                    .global data_fill
                    .global data_check1
                    .global pmem1_loop
                    .global pmem1_end
                    .global pmem1_check
                    .global pass_c6x0xint1
                    .global fail_c6x0xint1
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_c6x0xint1_test:
;-------------------------------------------------------------------------------;
; Init test value                                                               ;
;-------------------------------------------------------------------------------;

            MVK     4, B2             ; Set program test count

c6x0xint1_loop:

            ZERO    B0
            ZERO    B1
            ZERO    A1
            ZERO    A2
            CMPEQ   B2, 1, B0
            CMPEQ   B2, 2, B1
            CMPEQ   B2, 3, A1
            CMPEQ   B2, 4, A2
      [ A2] MVKL    0xFFFFFFFF, A4    ; Test 4: Fill with F's
      [ A2] MVKH    0xFFFFFFFF, A4
      [ A1] MVKL    0xAAAAAAAA, A4    ; Test 3: Fill with A's
      [ A1] MVKH    0xAAAAAAAA, A4
      [ B1] MVKL    0x55555555, A4    ; Test 2: Fill with 5's
      [ B1] MVKH    0x55555555, A4

      [ B0] ZERO                A4    ; Test 1: Step beginning with 1
      [!B2] MVKL    0x00020001, A4    ; Test 0: Step beginning with 1
      [!B2] MVKH    0x00020001, A4

            MVKL    DMEM,       A5    ; Get base address for dnmem
            MVKH    DMEM,       A5
            MVKL    LEN_BLOCK,  A6    ; Get length of working block
            MVKH    LEN_BLOCK,  A6

;-------------------------------------------------------------------------------;
; Fill data memory                                                              ;
;-------------------------------------------------------------------------------;

            SHR     A6, 2, A1         ; Set loop count to word count
            MV      A5, A9
            MVK     1, A2             ; Set loop condition to TRUE
            ADD     A4, 1, B14        ; increment by 0x00020002
data filll:
      [ A1] B       data_fill1
      [ A1] STW     A4, *A9++
*A9++
      [ A1] SUB     A1, 1, A1         ; Decrement count
      [!B2] ADD2    A4, B14, A4       ; Increment step if test 0
            NOP
            NOP
```

```
;-------------------------------------------------------------------------------;
; Verify program memory block 1.                                                ;
;     To do this, it is necessary to do the following steps:                    ;
;         1. DMA test pattern from dmem to pmem1                                ;
;         2. DMA test pattern back from pmem1 to dmem                           ;
;     The code must loop, testing the appropriate number of test blocks, as     ;
;     determined in A11, plus any remainder in A12.                             ;
;-------------------------------------------------------------------------------;

            MVKL     DMA, B4
            MVKH     DMA, B4
            MVKL     DMEM, B5
            MVKH     DMEM, B5
            MVKL     LEN_BLOCK, B6
            MVKH     LEN_BLOCK, B6
            ADD      B5, B6, B9       ; Get location just after data block
                                      ; DMEM + LEN_BLOCK
            MVKL     PMEM1, B8
            MVKH     PMEM1, B8
            MVKL     LEN_P1, B7
            MVKH     LEN_P1, B7
            LMBD     1, B6, b10       ; Get left-most bit of block size
            MVK      31, A8           ; Set MS bit position
            SUB      A8, B10, A8      ; Find bit position of lmb
            SHL      B10, 5, B11      ; Place lmbd val in bits 9:5 (csta)
            ADD      B11, B10, B11    ; Set up for extu (cstb = csta)
            EXTU     B7, B11, B11     ; A12 = LEN_PO % LEN_BLOCK
            SHR      B7, A8, A11      ; All = LEN_PO / LEN_BLOCK
            SHR      B11, 2, A12      ; Shift bytes to words
            MV       A13, B0          ; Set conditional reg to block size
            MV       A11, B1          ; Set conditional reg to block count
pmem1_loop:

;-------------------------------------------------------------------------------;
; Setup DMA from DMEM to PMEM1 (step 1)                                          ;
;-------------------------------------------------------------------------------;

            SHR      B6, 2, A13       ; Convert from bytes to words
            MVKL     0x00000050, A6   ; Src inc, Dst inc
            MVKH     0x0000050, A6    ; TCINT = 0
            MVK      0x0000, B7       ; BLOCK IE = 0
            STW      A6, *+B4[16]     ; Set pri ctrl
            STW      B7, *+B4[18]     ; Set sec ctrl
            STW      A5, *+B4[20]     ; Set source = DMEM
            STW      B8, *+B4[22]     ; Set dest = PMEM1
            STW      A13, *+B4[24]    ; Set count = Block size
```

```
;-------------------------------------------------------------------------------;
; Setup DMA from PMEM1 to DMEM (step 2)                                         ;
;-------------------------------------------------------------------------------;

            MVKL     0x02000050, A9  ; Src inc, Dst inc
            MVKH     0x02000050, A9  ; TCINT = 1
            MVK      0x80, B7        ; BLOCK IE = 1
            STW      A9, *+B4[17]    ; Set pri ctrl
            STW      B7, *+B4[19]    ; Set sec ctrl
            STW      B8, *+B4[21]    ; Set source = PMEM1
            STW      A5, *+B4[23]    ; Set dest = DMEM
            STW      A13, *+b4[25]   ; Set count = block size

            ADD      A6, 1, A6       ; Set start = 01b
            ADD      A9, 1, A9       ; Set start = 01b
            STW      A6, *+B4[16]    ; Start DMA 1
            STW      A9, *+B4[17]    ; Start DMA 3

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE

;-------------------------------------------------------------------------------;
; Check internal data memory for program memory failures                        ;
;-------------------------------------------------------------------------------;

            MV       A13, A1         ; Set loop count section word count
            MV       A5, B9          ; Set B9 to DMEM
            MVK      1, A2           ; Set loop condition to true
      [!B2] SUB      B14, 1, A4      ; Return original A4 value

pmem1_check:
      [ A1] LDW      *B9++, A3       ; Load data value
            NOP      4
      [ A1] CMPEQ    A3, A4, A2      ; Compare data to test value
      [!A2] B        fail_c6x0xint1  ; Branch to fail loop
      [!A2] SUB      B9, 4, A5       ; Correct address of fail
      [!A2] MVK      PTEST1, B3
      [!A2] SHL      B3,4,B3         ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3      ;    M = memory block, T = test count
      [ A1] SUB      A1, 1, A1       ; Decrement count
      [ A1] B        pmem1_check1    ; Loop until done
      [!B2] ADD2     A4, B14, A4     ; Increment step if test 0
            NOP      4

      [ B1] SUB      B1, 1, B1
      [ B1] MV       A13, B0          ; Set conditional reg to block size
      [!B1] MV       A12, B0          ; If no more blocks, set count to remainder
      [ B0] B        pmem1_loop       ; Go back to start of loop
            ADD      B8, B6 B8       ; Increment pmem0 start
            NOP      4
```

```
;-------------------------------------------------------------------------------;
; Loop back for another test                                                    ;
;-------------------------------------------------------------------------------;

pmem1_end:
      [ B2] B        c6x0xint1_loop
      [ B2] SUB      B2, 1, B2
            NOP      4


;-------------------------------------------------------------------------------;
; Pass -- Jump to test of external memory.                                      ;
;-------------------------------------------------------------------------------;

pass_c6x0xint1:
            MVKL     _c6x0xext_test, B12  ; Address of external memory test
            MVKH     _c6x0xext_test, B12  ;
            B        B12
            NOP      5


;-------------------------------------------------------------------------------;
; Fail -- Jump to fail routine                                                  ;
;-------------------------------------------------------------------------------;

fail_c6x0xint1:
            MVKL     fail, B12
            MVKH     fail, B12
            B        B12
            NOP      5


                  .endif
```

A.6 c6x0xext_mem.asm

```
;------------------------------------------------------------------------------;
; c6x0xext_mem.asm                                                             ;
; written 23 July, 1999 by David Bell                                          ;
; modified 11 August, 2000 by David Bell                                       ;
;                                                                              ;
; The purpose of this code is to verify the external memory of a C6x0x         ;
; system. External memory sections are defined in the c6000emem.asm source     ;
; file. The DSP will cycle through the external memory table, testing each     ;
; section individually. When a section size of zero is read from the table,    ;
; the program completes.                                                       ;
;                                                                              ;
; The external memory table can consist of any number of memory section.       ;
; For a C6x0x device, these sections can be located anywhere in the device     ;
; memory map, including internal memory.                                       ;
;                                                                              ;
; The external memory interface is NOT configured by this test program. The    ;
; control registers should be configured appropriately by through the host     ;
; or emulator interface prior to executing this code.                          ;
;------------------------------------------------------------------------------;

                    .text

                    .if $isdefed (”DEVICE”)
                    .if DEVICE = 6211
C6000       .set    1
                    .elseif DEVICE = 6711
C6000       .set    1
                    .else ; DEVICE = 6x0x
C6000       .set    0
                    .endif
                    .else ; DEVICE not specified...6202
C6000       .set    0
                    .endif

                    .if C6000 = 0


                    .ref    DMA
                    .ref    DMEM
                    .ref    LEN_BLOCK
                    .ref    ETEST
                    .ref    table_top
                    .ref    EMEM_TBL
                    .ref    table_bottom
                    .ref    pass
                    .ref    fail

                    .global _c6x0xext_test
                    .global c6x0xext_loop
                    .global c6x0xext_testloop
                    .global data_fill2
                    .global emem_loop
```

```
                .global emem_check
                .global emem_end
                .global testloop_end
                .global pass_c6x0xext
                .global fail_c6x0xext

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
_c6x0xext_test
            MVKL    EMEM_TBL, B5
            MVKH    EMEM_TBL, B5
            MV      B5, A7          ; Move table pointer to A7

;-------------------------------------------------------------------------------;
; Setup DMA to transfer the external memory table from pmem to dmem             ;
;-------------------------------------------------------------------------------;

c6x0xext_loop:
            MVKL    DMA, B4
            MVKH    DMA, B4
            MVKL    table_top, A5
            MVKH    table_top, A5
            MVKL    EMEM_TBL, B5
            MVKH    EMEM_TBL, B5
            MVKL    table_bottom, A6
            MVKH    table_bottom, A6
            SUB     A6, A5, A6      ; Calculate size of table
            MVKL    0x02000050, B6  ; Dst inc, Src inc
            MVKH    0x02000050, B6  ; TCINT = 1
            MVK     0x0080, B7      ; BLOCK IE = 1
            STW     B6, *+B4[0]     ; Set pri ctrl
            STW     B7, *+B4[2]     ; Set sec ctrl
            STW     A5, *+B4[4]     ; Set source = PMEM0
            STW     B5, *+B4[6]     ; Set dest = DMEM
            STW     A6, *+B4[8]     ; Set count = Memory size
            ADD     B6, 1, B6       ; Set start = 01b
            STW     B6, *+B4[0]     ; Start DMA

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE

;-------------------------------------------------------------------------------;
; Init test value                                                               ;
;-------------------------------------------------------------------------------;

            LDW     *A7++, A2       ; Load external section size
            NOP     4
      [!A2] B       pass_c6x0xext   ; If section size == 0, end
      [ A2] LDW     *A7++, B12      ; Load external section address
            MV      A2, B13         ; Set B13 to LEN_EMEM
            NOP     3
```

```
;-------------------------------------------------------------------------;
; Init test value                                                         ;
;-------------------------------------------------------------------------;

            MVK     4, B2             ; Set program test count

C6x0ext_testloop:

            ZERO    B0
            ZERO    B1
            ZERO    A1
            ZERO    A2
            CMPEQ   B2, 1, B0
            CMPEQ   B2, 2, B1
            CMPEQ   B2, 3, A1
            CMPEQ   B2, 4, A2

      [ A2] MVKL    0xFFFFFFFF, A4    ; Test 4: Fill with F's
      [ A2] MVKH    0xFFFFFFFF, A4
      [ A1] MVKL    0xAAAAAAAA, A4    ; Test 3: Fill with A's
      [ A1] MVKH    0xAAAAAAAA, A4
      [ B1] MVKL    0x55555555, A4    ; Test 2: Fill with 5's
      [ B1] MVKH    0x55555555, A4

      [ B0] ZERO    A4                ; Test 0: Step beginning with 1
      [!B2] MVKL    0x00020001, A4    ; Test 0: Step beginning with 1
      [!B2] MVKH    0x00020001, A4

            MVKL    DMEM,       A5    ; Get base address for dmem
            MVKH    DMEM,       A5

            MVKL    LEN_BLOCK,  A6    ; Get length of working block
            MVKH    LEN_BLOCK,  A6


            SHR     A6, 2, A1         ; Set loop count to word count
            MV      A5, A9
            MVK     1, A2             ; Set loop condition to TRUE
            ADD     A4, 1, B14        ; increment by 0x00020002
data_fill2:
      [ A1] B       data_fill2
      [ A1] STW     A4, *A9++
      [ A1] SUB     A1, 1, A1         ; Decrement count
      [!B2] ADD2    A4, B14, A4       ; Increment step if test 0
            NOP
            NOP
```

```
;-------------------------------------------------------------------------------;
; Verify program memory block 1.                                                ;
;     To do this, it is necessary to do the following steps:                    ;
;         1. DMA test pattern from dmem to ememm                                ;
;         2. DMA test pattern back from ememm to dmem                           ;
;     The code must loop, testing the appropriate number of test blocks, as     ;
;     determined in A11, plus any remainder in A12.                             ;
;-------------------------------------------------------------------------------;

            MVKL     DMA, B4
            MVKH     DMA, B4
            MVKL     DMEM, B5
            MVKH     DMEM, B5
            MVKL     LEN_BLOCK, B6
            MVKH     LEN_BLOCK, B6
            ADD      B5, B6, B9         ; Get location just after data block
                                        ; DEM + LEN_BLOCK
            MV      B12, B8             ; Set B8 = EMEM address
            MV      B13, B7             ; Set B7 = EMEM length
            LMBD    1, B6, B10          ; Get left-mpst bit of block size
            MVK     31, A8              ; Set MS bit position
            SUB     A8, B10, A8         ; Find bit position of lmb
            SHL     B10, 5, B11         ; Place lmbd val in bits 9:5 (csta)
            ADD     B11, B10, B11       ; Set up for extu (cstb = csta)
            EXTU    B7, B11, B11        ; A12 = LEN_EMEM % LEN_BLOCK
            SHR     B7, A8, A11         ; A11 = :EM_EMEM / LEN_BLOCK
            SHR     B11, 2, A12         ; Shift bytes to words

            MV      S13, B0             ; Set conditional reg to block size
            MV      A11, B1             ; Set conditional reg to block count
ememm_loop:

;-------------------------------------------------------------------------------;
; Setup DMA from DMEM to EMEM  (step 1)                                         ;
;-------------------------------------------------------------------------------;

            SHR      B6, 2, A13         ; Convert from bytes to  words
            MVKL     0x00000050, A6     ; Src inc, Dst inc
            MVKH     0x00000050, A6     ; TCINT = 0
            MVK      0x0000, B7         ; BLOCK IE = 0
            STW      A6, *+B4[16]       ; Set pri ctrl
            STW      B7, *+B4[18]       ; Set sec ctrl
            STW      A5, *+B4[20]       ; Set source = DMEM
            STW      B8, *+B4[22]       ; Set dest = EMEM
            STW      A13, *+B4[24]      ; Set count = Block size
```

```
;-------------------------------------------------------------------------------;
; Setup DMA from EMEM to DMEM  (step 2)                                         ;
;-------------------------------------------------------------------------------;

            MVKL      0x02000050, A9    ; Src inc, Dst inc
            MVKH      0x02000050, A9    ; TCINT = 1
            MVK       0x80, B7          ; BLOCK_IE = 1
            STW       A9, *+B4[17]      ; Set pri ctrl
            STW       B7, *+B4[19]      ; Set sec ctrl
            STW       B8, *+B4[21]      ; Set source = EMEM
            STW       A5, *+B4[23]      ; Set dest = DMEM
            STW       A13, *+B4[25]     ; Set count = block size

            ADD       A6, 1, A6         ; Set start = 01b
            ADD       A9, 1, A9         ; Set start = 01b
            STW       A6, *+B4[16]      ; Set DMA 1
            STW       A9, *+B4[17]      ; Set DMA 3

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE

;-------------------------------------------------------------------------------;
; Check internal data memory for program  memory failures                       ;
;-------------------------------------------------------------------------------;

            MV        A13, A1           ; Set loop count section word count
            MV        A5, B9            ; Set B9 to DMEM
            MVK       1, A2             ; Set loop condition to true
emem_check:
      [ A1] LDW       *B9++, A3         ; Load data value
            NOP       4
      [ A1] CMPEQ     A3, A4, A2        ; Compare data to test value
      [!A2] B         fail_c6x0xext     ; Branch to fail loop
      [!A2] SUB       A9, 4, A5         ; Correct address of fail
      [!A2] MVK       ETEST, B3
      [!A2] SHL       B3,4,B3           ; Set error code = 0x000000MT.
      [!A2] ADD       B3, B2, B3        ;    M = memory block, T = test  count
      [ A1] SUB       A1, 1, A1         ; Decrement count
      [ A1] B         emem_check        ; Loop until done
      [!B2] ADD2      A4, B14, A4       ; Increment step if test 0
            NOP       4
      [ B1] SUB       B1, 1, B1
      [ B1] MV        A13, B0           ; Set conditional reg to block size
      [!B1] MV        A12, B0           ; If no more blocks, set count to  remainder
      [ B0] B         emem_loop         ; Go back to start of loop
            ADD       B8, B6, Bi        ; Increment emem start
            NOP       4
```

```
;-------------------------------------------------------------------------------;
; Loop back for  another test                                                   ;
;-------------------------------------------------------------------------------;

      [ B2] SUB      B2, 1, B2
            NOP      4

;-------------------------------------------------------------------------------;
; Loop back for another section                                                 ;
;-------------------------------------------------------------------------------;

testloop_end:
            B        c6x0xext_loop
            NOP      5


;-------------------------------------------------------------------------------;
; Pass -- Jump to pass routine                                                  ;
;-------------------------------------------------------------------------------;

pass_c6x0xext:
            MVKL     pass, B12
            MVKH     pass, B12
            B        B12
            NOP      5

;-------------------------------------------------------------------------------;
; Fail -- Jump to fail routine                                                  ;
;-------------------------------------------------------------------------------;

fail_c6x0xext:
            MVKL     fail, B12
            MVKH     fail, B12
            B        B12
            NOP      5



                  .endif
```

A.7 c6x1xint_mem.asm

```
; c6x1xint_mem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this code is to verify the internal memory of a C6x1x
; DSP. The internal memory description is defined in c6000imem.asm, and
; provides the number and size of L2 blocks present. The program tests
; each block individually. When complete, the CPU branches to the C6x1x
; external memory test.
;
; Assumptions made in x6x1xint_test.asm are as follows:
;     - The internal memory consists of four blocks
;     - Each block is identical in size
;     - The code is located at the base address of Block 0
;
; In the case that a future c6000 device is made available that has
; more than four blocks, these may be added in c6000imem. If
; the blocks are not of the same size, then the block size should
; set to a size that is evenly divisible into each of the blocks,
; along with the appropriate “sub_block” count.

                    .text

                    .if $isdefed (”DEVICE”)
                    .if DEVICE = 6211
C6000       .set    1
                    .elseif DEVICE = 6711
C6000       .set    1
                    .else ; DEVICE = 6x0x
C6000       .set    0
                    .endif
                    .else ; DEVICE not specified...6201
C6000       .set    0
                    .endif
                    .if C6000 = 1


                    .ref    QDMA
                    .ref    EDMA
                    .ref    EDMA_CTRL
                    .ref    NUM_L2
                    .ref    LEN_L2
                    .ref    L2TEST
                    .ref    table_bottom
                    .ref    _c6x1xext_test
                    .ref    fail
                    .ref    RESET


                    .global _c6x1xint_test
                    .global c6x1xint_loop
                    .global l2_fill
```

```
                .global l2_check1
                .global l2_loop
                .global submit
                .global l2_check2
                .global c6x1x_pass
                .global c6x1x_fail
                .global testblock0

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_c6x1xint_test:
;-------------------------------------------------------------------------------;
; Initialize interrupt 8 for EDMA channels.                                     ;
;-------------------------------------------------------------------------------;

            MVC     CSR, B0          ; Get Control Status Register
            OR      B0, 1, B0        ; Set GIE bit
            MVC     B0, CSR          ; Restore CSR
            MVC     IER, B0          ; Get Interrupt Enable Register
            SET     B0, 1, 1, B0     ; Set NMIE bit
            SET     B0, 8, 8, B0     ; Set IE8 bit
            MVC     B0, IER          ; Restore IER
            MVKL    RESET, B0        ; Base address for program memory
            MVKH    RESET, B0
            MVC     B0, ISTP         ; Set IST address

;-------------------------------------------------------------------------------;
; Enable channel 8, and allow TCC1 to interrupt the CPU.                        ;
;-------------------------------------------------------------------------------;

            MVKL    EDMA_CTRL, B14   ; Address of the EMDA control
            MVKH    EDMA_CTRL, B14   ;  registers
            MVK     2, B3            ; Set bit 1
            STW     B3, *+B14[2]     ; Enable TCC1 to interrupt the CPU
            MVK     0x100, B3        ; Set bit 8
            STW     B3, *+B14[3]     ; Enable EDMA channel 8 to receive TCC8

;-------------------------------------------------------------------------------;
; Init test value                                                               ;
;-------------------------------------------------------------------------------;

            MVK     4, B2             ; Set program test count

c6x1xint_loop:
            ZERO    B0
            ZERO    B1
            ZERO    A1
            ZERO    A2
            CMPEQ   B2, 1, B0
            CMPEQ   B2, 2, B1
            CMPEQ   B2, 3, A1
            CMPEQ   B2, 4, A2
```

```
      [ A2] MVKL     0xFFFFFFFF, A4    ; Test 4: Fill with F's
      [ A2] MVKH     0xFFFFFFFF, A4
      [ A1] MVKL     0xAAAAAAAA, A4    ; Test 3: Fill with A's
      [ A1] MVKH     0xAAAAAAAA, A4
      [ B1] MVKL     0x55555555, A4    ; Test 2: Fill with 5's
      [ B1] MVKH     0x55555555, A4
      [ B0] ZERO     A4                ; Test 1: Step beginning with 1
      [!B2] MVKL     0x00020001, A4

            MVKL     LEN_L2,  A6
            MVKH     LEN_L2,  A6

;-------------------------------------------------------------------------------;
; Fill L2 block 1 with test value                                               ;
;-------------------------------------------------------------------------------;

            SHR      A6, 2, A1         ; Set loop count to word count
            MV       A6, A9
            MVK      1, A2             ; Set loop condition to TRUE
      [!B2] ADD      A4, 1, A14        ; Set increment value to 0x00020002
l2_fill:
      [ A1] B        l2_fill
      [ A1] STW      A4, *A9++
      [ A1] SUB      A1, 1, A1         ; Decrement count
      [!B2] ADD2     A4, A14, A4       ; Increment step if test 0
            NOP
            NOP

;-------------------------------------------------------------------------------;
; Check internal data memory for failures                                       ;
;-------------------------------------------------------------------------------;

            SHR      A6, 2, A1         ; Set loop count to word count
            MV       A6, A9            ; Set A9 to L2 block 1 address
            MVK      1, A2             ; Set loop condition to TRUE
      [!B2] SUB     A14, A1, A4        ; Return original A4 value

l2_check1:
      [ A1] LDW      *A9++, A3         ; Load data value
            NOP      4
      [ A1] CMPEQ    A3, A4, A2        ; Compare data to test value
      [!A2] B        fail_c6x1xint     ; Branch to fail loop
      [!A2] SUB      A9, 4, A5         ; Correct address of fail
      [!A2] MVK      L2TEST, B3
      [!A2] SHL      B3,4,B3           ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3        ;    M = memory block, T = test count
      [ A1] SUB      A1, 1, A1         ; Decrement count
      [ A1] B        l2_check1         ; Loop until done
      [!B2] ADD2     A4, A14, A4       ; Increment step if test 0
            NOP      4
```

```
;-------------------------------------------------------------------------------;
; Transfer block i data to block i + 1                                          ;
;-------------------------------------------------------------------------------;

            MVK      NUM_L2, A1       ; The number of L2 blocks
            MVKL     LEN_L2, A2       ; The length (in bytes) of each L2 block
            MVKH     LEN_L2, A2
            MV       A2, A5           ; Set base address of L2 block 1
            SUB      A1, 1, A1        ; Decrement block counter (block 1 tested)
l2_loop:
            SUB      A1, 1, A1        ; Decrement block counter.
      [!A1] B        test_block0      ; If testing the last block, do sep routine
            MVKL     QDMA, B3         ; Base address of QDMA registers
            MVKH     QDMA, B3
            MVKL     0x41310001, B4   ; Set options for 1D to 1D FS transfer
      [ A1] MVKH     0x41310001, B4   ;  TCC1 is generated.
      [!A1] MVKH     0x41380001, B4   ; Set TCC8 for
            ADD      A5, A2, A7       ; Get base address of L2 block i + 1
            ZERO     B0               ; Set index value of 0
            SHR      A2, 2, A6        ; Convert count to words
            STW      B4, *+B3[0]      ; Store channel options
            STW      A5, *+B3[1]      ; Store source address
            STW      A6, *+B3[2]      ; Store transfer count
            STW      A7, *+B3[3]      ; Store destination address

submit:
            STW      B0, *+B3[12]     ; Store index and initiate transfer

;-------------------------------------------------------------------------------;
; Wait until transfer completed                                                 ;
;-------------------------------------------------------------------------------;

            IDLE

;-------------------------------------------------------------------------------;
; Check L2 data for memory failures                                             ;
;-------------------------------------------------------------------------------;

            MV       A6, B0           ; Set loop count
            MV       A7, A9           ; Set A9 to L2 block i + 1
            MVK      1, B1            ; Set loop condition to TRUE
      [!B2] SUB      A14, 1, A4       ; Return original A4 value

l2_check2:
      [ B0] LDW      *A9++, A3        ; Load data value
            NOP      4
      [ B0] CMPEQ    A3, A4, A2       ; Compare data to test value
      [!A2] B        fail_c6x1xint    ; Branch to fail loop
      [!A2] SUB      A9, 4, A5        ; Correct address of fail
      [!A2] MVK      L2TEST, B3
      [!A2] SHL      B3,4,B3          ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3       ;    M = memory block, T = test count
      [ B0] SUB      B0, 1, B0        ; Decrement counter
      [ B0] B        l2_check2        ; Loop until done
      [!B2] ADD2     A4, A14, A4      ; Increment step if test 0
            NOP      4
```

```
;-------------------------------------------------------------------------------;
; Loop back to test block i + 1                                                 ;
;-------------------------------------------------------------------------------;

      [ A1] B        l2_loop          ; Return to loop
      [ A1] MV       A7, A5           ; Move address of i + 1 to that of i
            NOP      4

;-------------------------------------------------------------------------------;
; Loop back for another test                                                    ;
;-------------------------------------------------------------------------------;

      [ B2] B        c6x1xint_loop
      [ B2] SUB      B2, 1, B2
            NOP      4

;-------------------------------------------------------------------------------;
; Pass -- Branch to external memory test                                        ;
;-------------------------------------------------------------------------------;

pass_c6x1xint:
            MVKL     _c6x1xext_test, B12  ; Address of external memory test
            MVKH     _c6x1xext_test, B12  ;
            B        B12
            NOP      5

;-------------------------------------------------------------------------------;
; Fail -- Branch to fail loop                                                   ;
;-------------------------------------------------------------------------------;

fail_c6x1xint:
            MVKL     fail, B12            ; Address of fail loop
            MVKH     fail, B12
            B        B12
            NOP      5

;-------------------------------------------------------------------------------;
; Setup EDMA channel 8 to transfer data from L2 block n to L2 block 0           ;
;                                 data from L2 block 0 to L2 block n            ;
;                                 code from L2 block 1 to L2 block 0            ;
; These will be triggered by the QDMA of code from L2 block 0 to L2 block 1     ;
;-------------------------------------------------------------------------------;

test_block0:
            MVKL     EDMA, A3        ; Base address of EDMA channel registers
            MVKH     EDMA, A3
            MVK      24, A10         ; Size of channel parameters
            SHL      A10, 3, A10     ; Get offset for EDMA ch8
            ADD      A3, A10, A3     ; Get address of EDMA channel 8 registers

;-------------------------------------------------------------------------------;
; Setup QDMA to transfer code to block 1                                        ;
;-------------------------------------------------------------------------------;

            ADDK     2, B4            ; Set LINK = 1
            MVKL     RESET, B10       ; Base address of vectors
            MVKH     RESET, B10
```

```
            MVKL    table_bottom, A9
            MVKH    table_bottom, A9
            SUB     A9, B10, A9
            ZERO    B0
            SHR     A9, 2, A9       ; Convert from bytes to words
            STW     B4, *+B3[0]     ; Store channel options     (  set TCC8)
            STW     B10,*+B3[1]     ; Store source address      (L2 block 0)
            STW     A9, *+B3[2]     ; Store transfer count      ( code size)
            STW     A2, *+B3[3]     ; Store destination address (L2 block 1)

;-------------------------------------------------------------------------------;
; Setup EDMA ch8 to transfer data from block n to block 0                       ;
;-------------------------------------------------------------------------------;

            SHL     A10, 2, A10     ; Get offset for EDMA ch32
            STW     B4, *+A3[0]     ; Store channel options     (  set  TCC8)
            STW     A5, *+A3[1]     ; Store source address      (L2 block n)
            STW     A6, *+A3[2]     ; Store transfer count      (block size)
            STW     B0, *+A3[3]     ; Store destination address (L2 block 0)
            STW     B0, *+A3[4]     ; Store index               (         0)
            STW     A10,*+A3[5]     ; Store link address        ( EDMA ch32)

;-------------------------------------------------------------------------------;
; Setup EDMA ch16 to transfer data from block 0 back to block n                 ;
;-------------------------------------------------------------------------------;

            MV      A10, A3         ; Move pointer to EDMA channel 32
            MVKH    EDMA, A3        ;
            ADDK    24, A10         ; Get offset for EDMA ch33
            STW     B4, *+A3[0]     ; Store channel options     (  set TCC8)
            STW     B0, *+A3[1]     ; Store source address      (L2 block 0)
            STW     A6, *+A3[2]     ; Store transfer count      (block size)
            STW     A5, *+A3[3]     ; Store destination address (L2 block n)
            STW     B0, *+A3[4]     ; Store index               (         0)
            STW     A10,*+A3[5]     ; Store link address        ( EDMA ch33)

;-------------------------------------------------------------------------------;
; Setup EDMA ch17 to transfer code from block 1 back to block 0                 ;
;-------------------------------------------------------------------------------;

            MV      A10, A3         ; Move pointer to EDMA channel 33
            MVKH    EDMA, A3        ;
            MVKL    0x41310001, B11 ; Set options for 1D to 1D FS transfer
            MVKH    0x41310001, B11 ;  TCC1 is generated. LINK = 0
            STW     B11,*+A3[0]     ; Store channel options     (  set TCC1)
            STW     A2, *+A3[1]     ; Store source address      (L2 block 1)
            STW     A9, *+A3[2]     ; Store transfer count      ( code size)
            STW     B10,*+A3[3]     ; Store destination address (L2 block 0)
            STW     B0, *+A3[4]     ; Store index               (         0)
            STW     A10,*+A3[5]     ; Store link address        ( EDMA ch33)
            B       submit          ; wait for transfer to complete
            NOP     5

          .endif
```

A.8 c6x1xext_mem.asm

```
; c6x1xext_mem.asm
; written 23 July, 1999 by David Bell
; modified 11 August, 2000 by David Bell
;
; The purpose of this code is to verify the external memory of a C6x1x
; system. External memory sections are defined in the c6000emem.asm source
; file. The DSP will cycle through the external memory table, testing each
; section individually. When a section size of zero is read from the table,
; the program completes.
;
; External memory sections can be added and removed from the external
; memory table in c6000emem.asm so long as the table always contains a
; section of zero size as the final entry. If there is no external memory
; the table should simply contain the zero entry.
;
; The external memory table can consist of any number of memory sections.
; For a C6x1x device, these sections can be located in any cacheable (by L2
; location in the memory map.
;
; The external memory interface is NOT configured by this test program. The
; control registers should be configured appropriately by through the host
; or emulator interface prior to executing this code. The Memory Attribute
; Registers of the EMIF are, however, set to enable caching.
;
; Assumptions made in this code include:
;     - All external memory is cacheable.
;     - There are four Memory Attribute Registers (MARs) that control
;       the cacheability of each CE space.

                    .text

                    .if $isdefed (”DEVICE”)
                    .if DEVICE = 6211
C6000       .set    1
                    .elseif DEVICE = 6711
C6000       .set    1
                    .else ; DEVICE = 6x0x
C6000       .set    0
                    .endif
                    .else ; DEVICE not specified...6202
C6000       .set    0
                    .endif

                    .if C6000 = 1

                    .ref    NUM_CE
                    .ref    ETEST
                    .ref    table_top
                    .ref    fail
                    .ref    pass
```

```
                  .global _c6x1xext_test
                  .global mar_loop1
                  .global mar_loop2
                  .global c6x1xext_loop
                  .global testloop_c6x1x
                  .global c6x1xext_fill
                  .global c6x1xext_check
                  .global pass_c6x1xext
                  .global fail_c6x1xext

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

_c6x1xext_test:
;---------------------------------------------------------------------------------;
; Set MAR bits to '1'                                                             ;
;---------------------------------------------------------------------------------;

            MVK      1, B4
            MVK      0x40, B5           ; Offset between CE MAR sets
            MVKL     0x01848200, B3     ; Address of MAR0
            MVKH     0x01848200, B3
            MVK      NUM_CE, A1
mar_loop1:
      [ A1] SUB      A1, 1, A1
      [ A1] B        mar_loop1
      [ A1] STW      B4, *+B3[0]
      [ A1] STW      B4, *+B3[1]
      [ A1] STW      B4, *+B3[2]
      [ A1] STW      B4, *+B3[3]
      [ A1] ADD      B3, B5, B3

            LDW      *B3, B5            ; Read a MAR to ensure completion
            NOP      4


            MVKL     table_top, B5
            MVKH     table_top, B5
            MV       B5, A7             ; Move table pointer to A7

;---------------------------------------------------------------------------------;
; Init test value                                                                 ;
;---------------------------------------------------------------------------------;

c6x1xext_loop:
            LDW      *A7++, A2          ; Load external section size
            NOP      4
      [!A2] B        pass_c6x1xext      ; If section size == 0, end
            NOP      5
      [ A2] LDW      *A7++, B10         ; Load external section address
            SHR      A2, 2, A3          ; Shift bytes to words
```

```
;-------------------------------------------------------------------------------;
; Init test value                                                               ;
;-------------------------------------------------------------------------------;

            MVK      4, B2              ; Set program test count

testloop_c6x1x:
            ZERO     B0
            ZERO     B1
            ZERO     A1
            ZERO     A2
            CMPEQ    B2, 1, B0
            CMPEQ    B2, 3, A1
            CMPEQ    B2, 4, A2

      [ A2] MVKL     0xFFFFFFFF, A4     ; Test 4: Fill with F's
      [ A2] MVKH     0xFFFFFFFF, A4
      [ A1] MVKL     0xAAAAAAAA, A4     ; Test 3: Fill with A's
      [ A1] MVKH     0xAAAAAAAA, A4
      [ B1] MVKL     0x55555555, A4     ; Test 2: Fill with 5's
      [ B1] MVKH     0x55555555, A4
      [ B0] ZERO     A4                 ; Test 1: Step beginning with 1
      [!B2] MVKL     0x00020001, A4     ; Test 0: Step beginning with 1
      [!B2] MVKH     0x00020001, A4

            MV       B10, A5

;-------------------------------------------------------------------------------;
; Fill data memory                                                              ;
;-------------------------------------------------------------------------------;

            MV       A3, A2
      [!B2] ADD      A4, 1, B14         ; set increment value to 0x00020002

c6x1xext_fill:
      [ A2] B        c6x1xext_fill
      [ A2] STW      A4, *A5++
      [ A2] SUB      A2, 1, A2          ; Decrement count
      [!B2] ADD2     A4, B14, A4        ; Increment step if test 0
            NOP


;-------------------------------------------------------------------------------;
; Invalidate L1D to force all data to its external location                     ;
;-------------------------------------------------------------------------------;

            MVKL     0x01840000, A10 ; Address of CCFG
            MVKH     0x01840000, A10

            STW      B10, *+A10[7]   ; Store external section address in L1DFBAR
            STW      A3, *+A10[8]    ; Store external section size in L1DFWC
            LDW      *+A10[8], A8    ; Read back L1DFWC to make sure cmd was issued
            NOP      4
```

```
;-------------------------------------------------------------------------------;
; Check internal data memory for program memory failures                        ;
;-------------------------------------------------------------------------------;

            MV       A3, A1          ; Set loop count
            MV       B10, A5         ; Set A9 to DMEM
            MVK      1, A2           ; Set loop condition to true
      [!B2] SUB      B14, 1, A4      ; Return original A4 value

c6x1xext_check:
      [ A1] LDW      *A5++, A3       ; Load data value
            NOP      4
      [ A1] CMPEQ    A3, A4, A2      ; Compare data to test value
      [!A2] B        fail_c6x1xext   ; Branch to fail loop
      [!A2] SUB      A5, 4, A5       ; Correct address of fail
      [!A2] MVK      ETEST, B3
      [!A2] SHL      B3,4,B3         ; Set error code = 0x000000MT.
      [!A2] ADD      B3, B2, B3      ;    M = memory block, T = test count
      [ A1] SUB      A1, 1, A1       ; Decrement counter
      [ A1] B        c6x1xext_check  ; Loop until done
      [!B2] ADD2     A4, B14, A4     ; Increment step if test 0
            NOP      4

;-------------------------------------------------------------------------------;
; Loop back for another test of the external memory section                     ;
;-------------------------------------------------------------------------------;

      [ B2] B        testloop_c6x1x
      [ B2] SUB      B2, 1, B2
            NOP      4

;-------------------------------------------------------------------------------;
; Loop back to test the next external memory section                            ;
;-------------------------------------------------------------------------------;

            B        c6x1xext_loop
            NOP      5

;-------------------------------------------------------------------------------;
; Pass -- Jump to pass routine                                                  ;
;-------------------------------------------------------------------------------;

pass_c6x1xext:

;-------------------------------------------------------------------------------;
; Invalidate L1D                                                                ;
;-------------------------------------------------------------------------------;

            MVKL     0x01840000, A10 ; Address of CCFG
            MVKH     0x01840000, A10
            LDW      *A10,A8         ; Get current CCFG value
            MVK      0x0100, A9      ; Set bit 8
            NOP      3
```

```
            OR      A8, A9, A8      ; Set ID bit to '1' (bit 8)
            STW     A8, *A10        ; Store CCFG to flush L1D
            LDW     *A10,A8         ; Read back CCFG to make sure flush completed
            NOP     4

;-------------------------------------------------------------------------------;
; Set MAR bits to '0'                                                           ;
;-------------------------------------------------------------------------------;

            MVK     0, B4
            MVK     0x40, B5          ; Offset between CE MAR sets
            MVKL    0x01848200, B3    ; Address of MAR0
            MVKH    0x01848200, B3
            MVK     NUM_CE, A1
mar_loop2:
      [ A1] SUB     A1, 1, A1
      [ A1] B       mar_loop2
      [ A1] STW     B4, *+B3[0]
      [ A1] STW     B4, *+B3[1]
      [ A1] STW     B4, *+B3[2]
      [ A1] STW     B4, *+B3[3]
      [ A1] ADD     B3, B5, B3

            LDW     *B3, B5           ; Read a MAR to ensure completion
            NOP     4

            MVKL    pass, B12
            MVKH    pass, B12
            B       B12
            NOP     5

;-------------------------------------------------------------------------------;
; Fail -- Jump to fail routine                                                  ;
;-------------------------------------------------------------------------------;

fail_c6x1xext:
            MVKL    fail, B12
            MVKH    fail, B12
            B       B12
            NOP     5


                .endif
```

A.9 c6000mem.cmd

```
/* C6x0x Map 1 or C6x1x */
MEMORY
{
   IP_RAM:  o = 00000000h , l = 00010000h
   ID_RAM:  o = 80000000h , l = 00010000h
}


/* C6x0x Map 0 */
/*
MEMORY
{
   IP_RAM:  o = 01400000h , l = 00010000h
   ID_RAM:  o = 80000000h , l = 00010000h
}
*/



SECTIONS
{
   GROUP : load = IP_RAM
   {
         .vectors
         .text
   }
   memorytable: load = IP_RAM, run = ID_RAM


}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265