Technical Notes on using Analog Devices' DSP components and development tools
Contact our technical support by phone: (800) ANALOG-D or e-mail: dsp.support@analog.com
Or visit our on-line resources http://www.analog.com/dsp and http://www.analog.com/dsp/EZAnswers

# ADSP-2106x EPROM Overlay Support with VisualDSP++ 2.0

*Contributed by G.Ouellette* *June 10, 2002*

## Introduction

Using external memory to store program code and data in overlays is a good way to reduce DSP system costs. Many applications efficiently run using overlays from SDRAM. To reduce system cost further, program code and data can be placed off chip in a boot EPROM or FLASH.

This note discusses the concept of EPROM overlay partitions ADSP-21065L DSP.

The following topics and examples are discussed:

• Overview of Memory Overlays

• Overview of Overlay Manager

• Overview of Boot Kernel

• EPROM Overlay Example

All of the code segments used in the following discussion are segments from the example program that appears at the end of this note.

## Overview of Memory Overlays

Memory overlays provide support for applications whose code is too long to fit into internal memory of the processor. Program instructions and data are partitioned and stored in external memory until they are required for program execution. The partitions are referred to as memory overlays and the routines that call and execute them overlay managers.

Instructions can be placed in external memory ("live" space) and transferred, when needed, to internal memory ("run" space) for execution. Several overlays can be “run” (or execute) in a common location in internal memory, but only one can be in run space at a time. Overlay support is provided by the linker and is partially designed by the user in the linker description file (LDF). The LDF provides the linker with the direction on how to configure the overlays as well as the information necessary for the overlay manager routine to load the overlays. The information provided by the linker includes the following constants:

_ov_startaddress_N
_ov_endaddress_N
_ov_size_N
_ov_word_run_size_N
_ov_word_live_size_N
_ov_runtimestartaddress_N

Where N = the Overlay ID

Each overlay has a word size and an address associated with it, which the overlay manager uses to determine where the overlay resides and where it is executed.

Along with providing constants, the linker redirects overlay symbol references within your code to the overlay manager routine using a procedure linkage table (PLIT). The PLIT is essentially a jump table that executes user-defined code, and then jumps to the overlay manager. The linker replaces an overlay symbol reference (function call) with a jump to a location in the PLIT. The programmer defines the PLIT in the linker description file (LDF).

## Overview of Overlay Manager

The overlay manager is a user defined routine that is responsible for loading a referenced overlay function into internal memory (run time space). This is done with the aid of the linker-generated constants and the PLIT commands. The linker-generated constants tell the overlay manager the address of the live overlay, where the overlay resides for execution, and the number of words in the overlay. The PLIT commands tell the overlay manager such information as which overlay is required and the run time address of the referenced symbol.

## Overview of Boot Kernel

The ADSP-21065L is configured by default to load 256 words of 48-bit width (instruction size) after reset by DMA. DMA channel 8 is initialized for the ADSP-21065L. The ADSP-21065L DMA settings for EPROM booting can be taken from table 1.

| DMA Channel 8 | | ADSP-21065L |
|---|---|---|
| SYSCON | | 0x0000 0020 |
| II6 | IIEP0 | 0x8000 |
| IM6 | IMEP0 | 0x1 |
| C6 | CEP0 | 0x100 |
| EI6 | EIEP0 | 0x8000 0000* |
| EM6 | EMEP0 | 0x1 |
| DMAC6 | ECEP0 | 0x600 |
| IRQ Vector | | 0x8040 |

* 0x20000 for Rev 0.1

*Table 1. DMA settings for EPROM booting*

After RESET, the core processor is held in an IDLE mode until 256 words have been loaded in. The core processor will start execution of the just loaded boot kernel.

Beginning from *Start_Loader* some required registers are set, external memory accesses are setup and wait states are initialized. Having completed the set-up, the DMA engine is used to fetch 48-bit words from the EPROM.

Having determined the offset to find the start address of the boot section for this processor, the DMA parameters are set up to read data word by word. Each 48-bit word is transferred into address 0x8004 for dispatching. As the image in the EPROM contains program memory code sections and data memory sections, a 16-bit initialization tag is placed before each block. The initialization headers are organized as shown below:

| Tag Number | | Initialization Type |
|---|---|---|
| 0 | 0x0 | FINAL INIT |
| 1 | 0x1 | ZERO DM16 |
| 2 | 0x2 | ZERO DM32 |
| 3 | 0x3 | ZERO DM40 |
| 4 | 0x4 | INIT DM16 |
| 5 | 0x5 | INIT DM32 |
| 6 | 0x6 | INIT DM40 |
| 7 | 0x7 | ZERO PM16 |
| 8 | 0x8 | ZERO PM32 |
| 9 | 0x9 | ZERO PM40 |
| 10 | 0xA | ZERO PM48 |
| 11 | 0xB | INIT PM16 |
| 12 | 0xC | INIT PM32 |
| 13 | 0xD | INIT PM40 |
| 14 | 0xE | INIT PM48 |

*Table 2. Section Initialization Headers*

Any initialization of 48-bit PM memory uses a write with the PX register set to zero. The kernel enters a loop, which reads one 48-bit word from the EPROM and writes the appropriate width value to memory.

## EPROM Overlay Example

The tools shipped with VisualDSP aren't currently equipped to handle EPROM overlays. When trying to specify the live location of the overlay in EPROM space (0x8000 0000), the linker generates an error. The linker isn't capable of determining the live address on the EPROM because it is user defined.

To work around this limitation, the kernel has been modified. Any initialization of 48-bit PM memory (INIT PM48), the kernel performs a check to identify if the 48-bit PM data is to be loaded into 48-bit external memory. Following this, a check has been added to determine if the external memory is in fact in EPROM space (0x8000 0000). However, as noted above, the linker is unable to generate the live location of the overlay in EPROM space. It is up to the programmer to alert the kernel of the presence of a EPROM overlay.

As an example, the kernel has been modified as follows:

```
pm48_init:
        r13 = 0x20000;
        r13=r3-r13;
        if ge jump pm48_externalMemoryInit;

        I8=R3;
        LCNTR=R2,DO pm48_init_loop UNTIL LCE;
          CALL read_PROM_word (DB);
          NOP;
          NOP;

pm48_init_loop:
        PM(I8,M14)=PX;
        JUMP read_boot_info;

pm48_externalMemoryInit:
        I0=R3;
```

```
        BTST R3 by 0x17;
        IF sz JUMP pm48_loop;


/**** EPROM Overlay ****/

prom_ovrly:
        R9=DM(EIEP0);
        DM(I2,M6)=R9;
        R8=0x6;
        R8=R2*R8(SSI);
        R9=R8+R9;
        DM(EIEP0)=R9;
        JUMP read_boot_info; /* read boot info */
```

*Listing 1. Modified Kernel Example*

Following *pm48_init*, the kernel performs a comparison to determine whether the 48-bit word is to reside in external PM memory. It checks to see if the location is larger than the first external memory location of the ADSP-21065L (0x0002 0000). If the data is indeed to be resident in external memory, a jump to *pm48_externalMemoryInit* is executed.

A check is performed to determine if the data is to reside in EPROM memory.

The following code segment from the LDF used in the example configures the overlay live address:

```
MEMORY

{
   isr_tabl {
     TYPE(PM RAM) START(0x00008000)
     END(0x000080FF) WIDTH(48)
   }

   pm_code {
     TYPE(PM RAM) START(0x00008100)
     END(0x000087ff) WIDTH(48)
   }

    pm_data {
     TYPE(PM RAM) START(0x00009000)
     END(0x000097ff) WIDTH(32)
   }

   pm_idat {
     TYPE(PM RAM) START(0x00009800)
     END(0x00009fff) WIDTH(32)
   }

   dm_data {
     TYPE(DM RAM) START(0x0000C000)
     END(0x0000Cfff) WIDTH(32)
   }

   dm_rdat {
     TYPE(DM RAM) START(0x0000D000)
     END(0x0000Dfff) WIDTH(32)
   }

   ovl_code {
     TYPE(PM RAM) START(0x00820000)
     END(0x00FFFFFF) WIDTH(48)
   }

}  // End MEMORY
```

*Listing 2. LDF Overlay Example*

The overlay declaration in Listing 2 configures the overlay's live address to be somewhere in external memory, *ovl_code*, starting at 0x00820000.

In this example, to determine if the data is to reside in EPROM memory, the kernel tests for the presence of a set bit in location 0x17 of the 48-bit external memory address.

```
pm48_externalMemoryInit:
        I0=R3;
        BTST R3 by 0x17;
        IF sz JUMP pm48_loop;
```

If the test fails, a jump to pm48_loop is executed and the kernel places the data into external memory at the address specified. If the test passes, the overlay is assumed to have a live address in EPROM space. At this stage, the kernel performs several steps to overcome the limitations of the linker.

```
 /**** EPROM Overlay ****/

prom_ovrly:
        R9=DM(EIEP0);
        DM(I2,M6)=R9;
        R8=0x6;
        R8=R2*R8(SSI);
        R9=R8+R9;
        DM(EIEP0)=R9;
        JUMP read_boot_info; /* read boot info */
```

The current external DMA index points to the live location of an overlay in external memory. For this example, its live location is in the EPROM. This address is copied from the EIEP0 and replaces the *liveAddresses* location in DM memory. If this was the first overlay, the contents of *liveAddresses[1]* would be replaced by the contents of EIEP0.

The programmer must be aware that the location of the table of *liveAddresses* in DM memory must be known in advance and loaded into I2. The current overlay manager delivered with the tools initializes the *liveAddresses* to begin at location 0xC000 in DM memory. Due to this, I2 is initialized to 0xC000 at the start of the loader kernel.

The index register, I2, is incremented to the next *liveAddresses* in DM memory in preparation for the next EPROM overlay live address.

Since the overlay is to remain resident on the EPROM, the external port DMA index, EIEP0, needs to be adjusted to point to the next location following the overlay segment.

```
        R8=0x6;
        R8=R2*R8(SSI);
        R9=R8+R9;
        DM(EIEP0)=R9;
```

The data register, *R2*, has been previously loaded with the number of words in the overlay segment. Each word is 48-bits long, but is packed into 8-bits on the EPROM. Thus the number of words in the segment must be multiplied by 6 and added to the current value in EIEP0 to advance the

external index to the address following the current overlay segment.

The following example is intended to help explain a method to perform EPROM overlays so you can get started on your own applications.

## References:

ADSP-21065L SHARC USER'S MANUAL

EE-66 *'Using Memory Overlays'*

EE-56 *' Tips & Tricks on the ADSP-2106x EPROM and HOST bootloader'*

# Appendix A

Listed below are the source codes used to illustrate the EPROM Overlay initialization and operation for the ADSP-21065L. *(Please note that these included code modules were built using the VisualDSP2.0++ development tools for the ADSP-2106x processor family and the ADSP-21065L EZ-Kit Lite Evaluation kit.)*

ROM_Overlay_kernel.asm

```
/*****************************************************************************
      065L_prom.asm

      PROM based boot loader of the ADSP21065L processor.

      Copyright (c) 1998 Analog Devices Inc.  All rights reserved.

Modified: October-5-1999
            June-28-2000
            Changes Made: Per the 8-25-99 21065L Anomaly List, set the
EIEPx bit A23 for PROM Booting

Modified code to accomodate EPROM overlays - GJO 2000
*****************************************************************************/

/* Warning: After booting process, EP0I bit in IMASK is set.          */

/* Define the addresses of various IOP registers.                     */
#include <def21065l.h>

/* ADSP-21065L specific reset vector start address.*/
#define START_ADDR   0x8000
#define START_DM     0xC000

.SEGMENT/PM          seg_ldr;

/* The loader begins with the interrupts up to and including the low  */
/* priority timer interrupt.                                          */

          NOP;NOP;NOP;NOP;                    /* Reserved interrupt       */
___lib_RSTI:
           IDLE;                              /* Implicit IDLE instruction */
           JUMP Start_Loader (DB);            /* Begin loader             */
           NOP;                               /* Pad to next interrupt    */
           NOP;                               /* Pad to next interrupt    */
           NOP;NOP;NOP;NOP;

/* Vector for status stack/loop stack overflow or PC stack full:      */
___lib_SOVFI:   RTI;
          RTI;
          RTI;
          RTI;

/* Vector for high priority timer interrupt: */
___lib_TMZHI:   RTI;
          RTI;
          RTI;
          RTI;

/* Vector for external interrupts:          */
___lib_VIRPTI:  RTI;
          RTI;
          RTI;
          RTI;

___lib_IRQ2I:   RTI;
          RTI;
          RTI;
          RTI;

___lib_IRQ1I:   RTI;
          RTI;
          RTI;
          RTI;

___lib_IRQ0I:   RTI;
          RTI;
          RTI;
          RTI;

          NOP;NOP;NOP;NOP;

/* Vector for Serial port DMA channels:                    */

___lib_SPROI:   RTI;
          RTI;
          RTI;
          RTI;
```

```
___lib_SPR1I:   RTI;
                RTI;
                RTI;
                RTI;

___lib_SPT0I:   RTI;
                RTI;
                RTI;
                RTI;

___lib_SPT1I:   RTI;
                RTI;
                RTI;
                RTI;

/* Reserved */          RTI;
                RTI;
                RTI;
                RTI;

/* Reserved */          RTI;
                RTI;
                RTI;
                RTI;

/* Vectors for External port DMA channels:          */
___lib_EP0I:
                 R2=DM(DMAC0);                          /* Get DMAC Control setting*/
                 RTI (DB);
                 R6=0;
                 DM(DMAC0)=R6;                          /*  zeroed between uses.       */

Start_Loader:
/* After power up or reset, default value in SYSCON register is 0x0000 0010.       */
/* The WAIT register for revision 0.0 defaults to 0x200d6b5a. Rev 0.0 requires      */
/* that waitstates be inserted using external logic to assert the ACK pin (see     */
/* anomaly list).  For revisions 0.1 and greater, the WAIT register defaults       */
/* to 0x21ad6b5a, which correctly sets 6 waitstates plus an idle cycle for ROM     */
/* boot memory. If for any reason these two value should be modified, do it        */
/* here as following example. Make sure this file does not exceed 256 words.       */
/*                                                                                 */
/*              ustat1=0Xxxxxxxxx;                                                 */
/*              DM(SYSCON)=ustat1;      Modify SYSCON                              */
/*              ustat1=0Xxxxxxxxx;                                                 */
/*              DM(WAIT)=ustat1;        Modify WAIT                                */

/*****************************************************************/
/* SETUP SDRAM on EZ-LAB */
/*****************************************************************/
                ustat1=dm(WAIT);
                bit clr ustat1 0x000f801f;                      /*clear MS0 amd MS3 waitstate and mode*/
                dm(WAIT)=ustat1;
                ustat1=937;                                     /*refresh rate*/
                dm(SDRDIV)=ustat1;
                ustat1=dm(IOCTL);                               /*mask in SDRAM settings*/
                bit set ustat1 SDPSS|SDBN2|SDBS0|SDTRP3|SDTRAS5|SDCL2|SDPGS256|DSDCK1;
                dm(IOCTL)=ustat1;
/*****************************************************************/
/*****************************************************************/

                I2 = START_DM;          /* Used for EPROM overlay  */

                L0=0;                   /* Zero out L-registers so they            */
                L4=0;                   /*  can be used without wrap               */
                L7=0;
                L8=0;
                L12=0;
                L15=0;

                M5=0;                   /* Setup M-registers to use for            */
                M6=1;                   /*  for various memory transfers           */
                M13=0;
                M14=1;
                R10=DM(SYSCON);         /* Read current SYSCON setting             */
                R12=PASS R10;           /* Hold Initial SYSCON setting             */
                R11=BSET R10 BY 1;      /* Set BSO bit for reading ROM             */
                R10=BCLR R10 BY 1;      /* Clear BSO bit for ext write             */

                DM(SYSCON)=R10;                                 /* Clear BSO bit for writing RAM        */

                BIT SET IMASK 0x10000;                          /* Enable EP0 interrupt                 */
                BIT SET MODE1 0x1800;                           /* Enable interrupts and nesting        */

                R14=0x0221;                                     /* For DMAC0 setting                    */

                R15=6;                                          /* EC to load one 48-bit word           */
                R0=DM(SYSTAT);                                  /* Load the Host ID                     */
                R0=FEXT R0 BY 8:3;

                DM(IMEP0)=M13;                                  /* Setup the DMA registers              */
                DM(EMEP0)=M14;

                I15 = START_ADDR+0x4;                           /* DMA destination address              */

        /** kernel modification (next 3 instructions) 10/5/99 **/
                ustat1 = DM(EIEP0);             /* Get current index addr, points to 1st addr after last 256 kernel byte       */
                bit set ustat1 0x800000;        /* set bit A23 to ensure proper number of waitstates with BS0 set               */
```

```
                DM(EIEP0) = ustat1;                     /* write back to the external DMA index register to read the next prom word     */
        /** ************************************************** **/

get_addr:  CALL read_PROM_word;
           COMP(R0, R2);

           IF NE JUMP get_addr;

           /* ********************* Workaround for VDSP 4.1 elf-loader 8/25/99 ******************* */
           /*   Reset state of EIEP0 in PROM boot mode is different for early revisions */
           /*   test bits 25-29 in the MODE2 register to determine revision of the DSP */
           /*   Revision 0.0, 0.1 --> MODE2[31-25]=1100000  */
           /*   Revision 0.2      --> MODE2[31-25]=1100001  */
           /*   Revision 0.3      --> MODE2[31-25]=1100010  */

           R4=MODE2;                            /* read MODE2 register to determine the DSP revision */
           R4=FEXT R4 BY 25:5;                  /* extract bits 25-29, place into bits 0-4 back in same reg R4 */
           R4=PASS R4;                          /* Pass throught the ALU to test the AZ bit in ASTAT for zero */
           IF EQ R3=BSET R3 BY 17;              /* If bits 25-20 are zero, the DSP is a rev 0.0 or 0.1 */

           /* Setting this is necessary for these revs since EIEP0 resets */
           /* to 0x20000. Setting bit 17 is not required if revision */
           /* is >= 0.2 because for these EIEP0 resets to 0x000000 */

           R3 = BSET R3 BY 23;                  /* Setting A23 in external DMA index register is required  */
                                                /* for generating correct # of waitstates offchip with the BSO bit */
                                                /* (in SYSCON) set. Refer to Doc changes section in the 21065L anomaly list (8-25-99). */
                                                /* NOTE: A23 is set internally and is not driven out on an */
                                                /* external address pin, therefore it has no external affect */
                                                /* on the booting device */
           /* ********************************************************************************** */

           DM(EIEP0)=R3;                /* Point to address in PROM */

read_boot_info:
           CALL read_PROM_word;
           R0=PASS                                                                                                          R2;
           CALL read_PROM_word;

load_memory:
           R0=PASS R0;
           IF NE JUMP (PC, test_dm16_zero);

/* After the IDLE completes, the following sequence of instructions will        */
/* be executed: (Remember these are in a loop)                                  */
/*           R0=R0-R0, DM(I4,M5)=R9, PM(I12,M13)=R11                            */
/* This instructions sets the EQ flag to terminate the loop, writes             */
/* the original value to SYSCON, and writes a *new* instruction over            */
/* itself.  The new instruction is:                                             */
/*           PM(0, I8)=PX;                                                      */
/* This instruction resets the DMA8 vector to whatever it should be.            */
/* The loop will terminate, because of the previous set EQ.  Instruction        */
/* flow will continue with START_ADDR+0x5, just like nothing happened!          */

final_init:  R9=0xb1db0000;                     /* Load instruction PM(0,I8)=PX;        */
             R11=BSET R11 BY 8;                 /* Set IMDW to 1 for inst write         */
             DM(SYSCON)=R11;                    /* Setup to read PROM, inst wrt         */

             R1=START_ADDR;                     /* Point to destination                 */
             R2=0x100;                          /* Load length of last init             */
             R4=R2*R15 (SSI);                   /* Compute external length              */
             DM(IMEP0)=M14;                     /* Set to increment internal ptr        */
             I4=START_ADDR+0x4;                 /* Point to START_ADDR+0x4 for patch    */
             I8=START_ADDR+0x40;                /* Point to DMA8 vector to patch        */
             I12 = SYSCON;                      /* Address of SYSCON                    */

             R4=PASS R4, R11=R12;               /* Clear AZ, hold initial SYSCON        */

             DO ___lib_RSTI UNTIL EQ;           /* Setup dummy loop                     */

             FLUSH CACHE;                       /* Clean off old top-of-loop            */
                  R0=START_ADDR+0x4;            /*  and replace with new                */
                  PCSTK=R0;                     /*  top-of-loop value                   */

                  DM(IIEP0)=R1;                 /* Setup DMA to load over ldr           */
                  DM(CEP0)=R2;                  /* Load internal count                  */
                  DM(ECEP0)=R4;                 /* Load external count                  */
                  JUMP START_ADDR+0x4 (DB);     /* Jump to start                        */
                  DM(DMAC0)=R14;                /* Start DMA transfer                   */
                  IDLE;                         /* After IDLE, patch then start        */

test_dm16_zero:   R0=R0-1;
                  IF EQ JUMP (PC, dm16_zero);
                  R0=R0-1;
                  IF NE JUMP (PC, test_dm40_zero);

dm32_zero:
dm16_zero:        R0=R0-R0, I0=R3;
                  LCNTR=R2, DO dm16_zero_loop UNTIL LCE;
dm16_zero_loop:   DM(I0,M6)=R0;

                  JUMP read_boot_info;

test_dm40_zero:   R0=R0-1;
                  IF NE JUMP (PC, test_dm16_init);

dm40_zero:        PX1=0;
                  PX2=0;
```

```
                        R0=R0-R0, I0=R3;
                        LCNTR=R2, DO dm40_zero_loop UNTIL LCE;
dm40_zero_loop:         DM(I0,M6)=PX;

                        JUMP read_boot_info;

test_dm16_init:         R0=R0-1;
                        IF NE JUMP (PC, test_dm32_init);

dm16_init:              I0=R3;
                        LCNTR=R2, DO dm16_init_loop UNTIL LCE;
                                CALL read_PROM_word (DB);
                                NOP;                            /* NOP's are required for   */
                                NOP;                            /* loop restriction         */
dm16_init_loop:                 DM(I0,M6)=R3;

                        JUMP read_boot_info;

test_dm32_init:         R0=R0-1;
                        IF NE JUMP (PC, test_dm40_init);

dm32_init:              I0=R3;
                        LCNTR=R2, DO dm32_init_loop UNTIL LCE;
                                CALL read_PROM_word (DB);
                                NOP;
                                NOP;
dm32_init_loop:                 DM(I0,M6)=R3;

                        JUMP read_boot_info;

test_dm40_init:         R0=R0-1;
                        IF NE JUMP (PC, test_pm16_zero);

dm40_init:              I0=R3;
                        LCNTR=R2, DO dm40_init_loop UNTIL LCE;
                                CALL read_PROM_word (DB);
                                NOP;
                                NOP;
dm40_init_loop:                 DM(I0,M6)=PX;

                        JUMP read_boot_info;

test_pm16_zero:         R0=R0-1;
                        IF EQ JUMP (PC, dm16_zero);
                        R0=R0-1;
                        IF EQ JUMP (PC, dm32_zero);

test_pm40_zero:         R0=R0-1;
                        IF EQ JUMP (PC, dm40_zero);
                        R0=R0-1;
                        IF NE JUMP (PC, test_pm16_init);

pm48_zero:              PX1=0;
                        PX2=0;
                        R0=R0-R0, I8=R3;
                        LCNTR=R2, DO pm40_zero_loop UNTIL LCE;
pm40_zero_loop:                 PM(I8,M14)=PX;

                        JUMP read_boot_info;

test_pm16_init:         R0=R0-1;
                        IF EQ JUMP (PC, dm16_init);

test_pm32_init:         R0=R0-1;
                        IF EQ JUMP (PC, dm32_init);

test_pm40_init:         R0=R0-1;
                        IF EQ JUMP (PC, dm40_init);

test_pm48_init:         R0=R0-1;
                        IF NE JUMP read_boot_info;

pm48_init:
                        r13 = 0x20000;
                        r13=r3-r13;
                        if ge jump pm48_externalMemoryInit;

                        I8=R3;
                        LCNTR=R2, DO pm48_init_loop UNTIL LCE;
                                CALL read_PROM_word (DB);
                                NOP;
                                NOP;
pm48_init_loop:                 PM(I8,M14)=PX;

                        JUMP read_boot_info;

pm48_externalMemoryInit:
                        I0=R3;
                        BTST R3 by 0x17;                /* Check if Live address is in boot space: ie. R3=0x0008xxxxx   */
                        IF sz JUMP pm48_loop;           /* jump to eprom overlay                                        */

/**************** Eprom Overlay ******************/
prom_ovrly:
                        R9=DM(EIEP0);                   /* copy first locxation of Live space on eprom                  */
                        DM(I2,M6)=R9;
                        R8=0x6;                         /* adjust external index next location of boot info             */
                        R8=R2*R8(SSI);
                        R9=R8+R9;
```

```
                        DM(EIEP0)=R9;
                        JUMP read_boot_info;            /* read boot info                                   */

/***************** Eprom Overlay ********************/
pm48_loop:
                        LCNTR=R2, DO pm48_init_external_loop UNTIL LCE;
                                CALL read_PROM_word (DB);
                                NOP;
                                NOP;
                                dm(i0,m6)=r2;
pm48_init_external_loop:        DM(I0,M6)=R3;
                        JUMP read_boot_info;


read_PROM_word:         R13=DM(SYSCON);             /* Save old value of SYSCON         */
                        DM(SYSCON)=R11;             /* Set BSO bit for ROM read         */
                        DM(IIEP0)=I15;              /* Setup DMA destination address    */
                        DM(CEP0)=M14;               /* Setup DMA internal length        */
                        DM(ECEP0)=R15;              /* Setup DMA external count         */
                        DM(DMAC0)=R14;              /* Start DMA                        */
                        IDLE;                       /* Wait for DMA to complete         */
                        PX=PM(START_ADDR+0x4);      /* Read word from scratch           */
                        DM(SYSCON)=R13;             /* Reset SYSCON to previous         */
                        RTS (DB);
                        R2=PX1;                     /* Copy PX values into DREGS        */
                        R2=PASS R2, R3=PX2;         /* Test the length                  */

.ENDSEG;
```

## Ovly_mgr_65L.asm

```
/*  The OVLY_MGR.ASM file is the overlay manager.  When a symbol            */
/*  residing in overlay is referenced, the ovelay manager loads             */
/*  the overlay code and begins execution.  (This overlay manager           */
/*  does not check to see if the overlay is already in internal             */
/*  memory, this option will be added later.)  A DMA transfer is            */
/*  performed to load in the meory overlay.                                 */

#include "def21065L.h"


.SEGMENT/DM         dm_data;

/*  The following constants are defined by the linker.          */
/*  These constants contain the word size, live locaton         */
/*  and run location of the overlay functions.                  */

.EXTERN _ov_word_run_size_1;
.EXTERN _ov_word_run_size_2;
.EXTERN _ov_word_run_size_3;
.EXTERN _ov_word_live_size_1;
.EXTERN _ov_word_live_size_2;
.EXTERN _ov_word_live_size_3;
.EXTERN _ov_startaddress_1;
.EXTERN _ov_startaddress_2;
.EXTERN _ov_startaddress_3;
.EXTERN _ov_runtimestartaddress_1;
.EXTERN _ov_runtimestartaddress_2;
.EXTERN _ov_runtimestartaddress_3;
/*  Placing the linker constants in an array so the overlay     */
/*  manager can use the appropriate constant based on the       */
/*  overlay id.                                                 */

.VAR  liveAddresses[3] = _ov_startaddress_1,        _ov_startaddress_2,             _ov_startaddress_3;
.VAR  runAddresses[3]  = _ov_runtimestartaddress_1, _ov_runtimestartaddress_2,      _ov_runtimestartaddress_3;
.VAR  runWordSize[3]   = _ov_word_size_run_1,       _ov_word_size_run_2,            _ov_word_size_run_3;
.VAR  liveWordSize[3]  = _ov_word_size_live_1,      _ov_word_size_live_2,           _ov_word_size_live_3;

/*  software stack to temporarily store registers corrupted by overlay manager */
.VAR  ov_stack[10];
.VAR  ov_id_loaded = -1;

.ENDSEG;

/*************************************************************************/
/*                      Overlay Manager Function                                                    */

.SEGMENT/PM         pm_code;

_OverlayManager:
.GLOBAL _OverlayManager;

/*  _overlayID has been defined as R0.  R0 is set in the PLIT of LDF.   */
/*  Set up DMA transfer to internal memory through the external port.   */


/*  Store values of registers used by the overlay manager in to the     */
/*  software stack.                                                     */
dm(ov_stack)=i8;
```

```
dm(ov_stack+1)=m8;
dm(ov_stack+2)=l8;
dm(ov_stack+3)=r2;

/* Use the overlay id as an index (must subtract one)                    */
R0=R0-1;   /* Overlay ID -1                                              */
m8=R0;     /*  Offset into the arrays containing linker                  */
/*  defined overlay constants.                                           */

r2=dm(ov_id_loaded);
r0=r0-r2;
if EQ jump continue;
dm(ov_id_loaded)=m8;

r0=i0;       dm(ov_stack+4)=r0;
r0=m0;       dm(ov_stack+5)=r0;
r0=l0;       dm(ov_stack+6)=r0;

l8=0;
l0=0;

m0=m8; /* Overlay ID - 1 */

/*  Get overlay run and live addresses from memory and use to           */
/*  set up the master mode DMA.                                         */
i8 = runAddresses;
i0 = liveAddresses;

/* Disable DMA */
r0=0;
dm(DMAC0) = r0;

/* Set DMA external pointer to overlay live address */
r0=dm(m0,i0);
dm(EIEP0)=r0;

/* Set DMA internal pointer to overlay run address  */
r0=pm(m8,i8);
dm(IIEP0)=r0;

i0=runWordSize;          /*  Number of words stored in internal memory      */
                         /*  Most likely the word size will be 48 bits      */
                         /*  for instructions.                              */

/* Set DMA external modifier */
r0=1;
dm(EMEP0)=r0;

i8=liveWordSize;         /*  Number of words stored in external memory      */
                         /*  Most likely the word size will be 32 or16      */
                         /*  bits for external storage.                     */

/* Set DMA internal modify to 1 */
dm(IMEP0)=r0;

/* Set DMA internal count to Overlay run size.          */
r0=dm(m0,i0);
dm(CEP0)=r0;

/* Set DMA external count to Overlay live size.         */
r0=pm(m8,i8);

/********************** BOOT SPACE OVERLAYS  **********************/
ustat1=dm(SYSCON);
bit set ustat1 0x00000002;                    /* Select Boot Override forcing 8-48 bit external reads  */
dm(SYSCON)=ustat1;
r2=0x6;                                       /* Each 48-bit word is 6 bytes wide in boot space        */
r0=r0*r2(SSI);
/*****************************************************************/

dm(ECEP0)=r0;

/* DMA enabled, instruction word, Master, 48-32 packing - Overriden is BMO set */
r0=0x0221;
dm(DMAC0)=r0;

/*  Enable DMA interrupt */
bit set mode1 IRPTEN;
bit set imask EP0I;

/*  Restore register values from stack  */
r0=dm(ov_stack+6);       l0=r0;
r0=dm(ov_stack+5);       m0=r0;
r0=dm(ov_stack+4);       i0=r0;

/*  Wait for DMA to complete  */
dma1_wait: idle;

continue:
r2=dm(ov_stack+3);
l8=dm(ov_stack+2);
i8=r1;
m8=0;
r1=dm(ov_stack+1);
r0=dm(ov_stack);

/*  Flush the cache.  If an instruction in previous overlay  */
/*  had been cached, it may be executed instead of the       */
```

```
/*  current overlays instruction.  (If pm transfers align.)  */
flush cache;

/*  Jump to the location of the function to be executed.  */
jump (m8,i8) (db);
i8=r0;
m8=r1;

.ENDSEG;
/*************************************************************************/
```

## Document History

| Version | Description |
|---|---|
| June 10, 2002 by G.Ouellette | Released. |
| June 12, 2000 by G.Ouellette. | Created. |