Extended Memory Support for HPC

National Semiconductor
Application Note 577
Raja Gopalan
January 1989

INTRODUCTION

HPC™ family of microcontrollers have maximum addressing capability of 64 kbytes directly by the CPU. If an application requires more than 64k of address space, then the HPC address space can be expanded in terms of banks of memory, using an I/O port to select the memory banks. For example one can use PORTB pins 8, 9, 13 and 14 to select up to 16 banks of memory (which the MOLE development system also supports currently for debugging purposes). Please refer to the application note AN-497 ''Expanding the HPC Address Space'' by Joe Cocovich for hardware details.

The current version of HPC software package (Compiler, Assembler and Linker) however, does not directly support more than 64k of address space. This is mainly due to the Linker, which currently can handle only 64k of address space.

This report describes a method to handle more than 64k of address space from a software point of view. In order to do this, the user has to do multiple linking of modules in different banks and resolve the inter-bank symbol references.

The rest of the report describes the following:

1. Compiler generated selections (of code and data).
2. Programming conventions for bank switching.
3. Switch function to support bank switching.
4. Linking for bank switching.

SECTIONS GENERATED BY THE COMPILER

The compiler generates sections of relocatable assembly code which can be positioned in absolute address using the Linker in two ways:

1. Using the /SECT directive.
2. Using the /RANGE directive.

The following are the sections generated by the compiler for a source file named ''MODULE'':

```
1.   MODULE_code,rom8              Code.
2.   MODULE_code,roml6

3.   MODULE_ram8_bss,ram8         Data area for uninitialized
4.   MODULE_raml6_bss,raml6       static variables.

5.   MODULE_ram8_data,ram8        Data area for initialized
6.   MODULE_raml6_data,raml6      static variables.

7.   MODULE_ram8_strdata,ram 8    Data area for string literals.

8.   MODULE_raml6_init,rom8       Initial value for static
9.   MODULE_ram8_init,rom8        variables.

10.  MODULE_ram8,strinit,rom8     Initial values for string
                                  literals.

11.  MODULE_basel6_bss,base       Base page area for uninitialized
12.  MODULE_base8_bss,base        static variables.

13.  MODULE_basel6_data,base      Base page area for initialized
14.  MODULE_base8_data,base       static variables.

15.  MODULE_basel6_init,roml6     Initial values for Base page
16.  MODULE_base8_init,rom8       initialized variables.

17.  MODULE_roml6_data,roml6      Area for constant storage type.
18.  MODULE_rom8_data,rom8

19.  c_stack,raml6                Stack area in module
                                  containing main( ).

20.  _init_info_                  for each module which has any
                                  static variables defined.
```

HPC™ is a trademark of National Semiconductor Corporation.

 TL/DD10131 RRD-B30M105/Printed in U. S. A.

PROGRAMMING CONVENTIONS TO BE USED FOR BANK SWITCHING

As far as the bank switching hardware is concerned, the HPC addressing space is divided into banks of memory. The Fixed Address space is referred to as shared bank and the switchable address space is called as switchable bank. Any mechanism for bank selection can be used, as long as the conventions mentioned below are strictly followed:

1. All static variables must be placed in the shared memory. Basepage must go in basepage (which is shared).
2. If string literals are not in ROM, they must be placed in the shared memory.
3. Initialization values for static variables or string literals in RAM must be in the shared memory. This includes basepage initializers and __init__info__ sections.
4. If string literals for a bank are in ROM, and are never used as an argument to an inter-bank function call, the literals for that bank can be in the switchable bank.
5. If constants for a bank are never used by passing their address as an argument to an inter-bank function call, the constants for that bank can be in the switchable bank.
6. If the addresses of constants or string literals for a bank are used as arguments to an inter-bank function call, the constants or literals must be in the shared memory.
7. The stack must be in the shared memory.
8. Interrupt vectors must point to routines in the shared memory.
9. Only code and qualified ROM data can be placed in switchable banks.
10. A call to a function placed in the shared memory is always direct.
11. A function call from one switchable bank to another switchable bank must use a switching routine in the shared memory. Such a call cannot pass arguments which are addresses of functions, constants, or string literals in the calling bank. All pointers passed must be to objects in the shared memory.
12. A function which returns a structure cannot be used in an inter-bank function call if the returned structure is in memory in the calling bank. If the returned structure is an argument to another function, has a member of it accessed, or is assigned to a static or local variable, it is legal. If it is placed into switchable memory, by assigning to what is pointed at by a pointer, the operation will fail for an inter-bank function call.
13. The START macro in CRTFIRST must initialize the bankswitching port as necessary, and select the bank containing main( ) if it is not in the shared memory.

FUNCTION IN ASSEMBLER TO HANDLE SWITCHING OF BANKS

When bank switching must occur, the stack is set up by the compiler generated code for a normal function call. Instead of calling the destination function directly, however, the inter-bank link for the destination is called, as a result of the special manipulations with the linker LNHPC. This routine must change banks and then transfer to the destination, and must receive the return from the destination function so as to switch back to the original caller. This must be done transparently—no registers may be modified, and the stack must appear the same.

Included is the code to support the actual switching of banks during inter-bank function calls. This code allows a routine in either the shared memory or one of the switchable banks to make an inter-bank call to a routine in another bank.

The inter-bank link for each destination is created by a macro, invoked for each required linkage. The inter-bank link is simply a subroutine call to a common switching routine, with in-line arguments giving the bank and address of the destination. The common switching routine does the necessary manipulation of the stack to execute the destination and receive the return. The excess information is saved off in a separate software stack; upon return this information is used to restore the situation as if a normal function call had occurred.

Since the inter-bank transfer is completely transparent, it is not limited to handling C function calls. Any subroutine call which does not pass pointers to objects in switchable banks, which does not have in-line arguments, which does not use the Carry bit as either input or return, and which does not use a Return And Skip instruction, can be used with an inter-bank function call. However, the macro generates names using the C convention; an additional form is available for assembly subroutine names.

Also available is a version which allows the bank switching stack to be in 8-bit memory. It differs only in a few places from the 16-bit stack version.

The normal arrangement calls for the common switching routine and all the inter-bank links to be in shared memory. However, order of execution in the bank switching code is such that the inter-bank links for each destination that a bank needs can be in the switchable memory, and only the common routine needs to be in shared memory.

The software stack used by the bank switching is designed to grow downward, in contrast to the hardware stack, which grows upward. This allows the software stack to be placed in the same memory area as the hardware stack, but above it, and the two stacks will share their memory.

LINKAGE PROCEDURE FOR BANK SWITCHING

The actual linking of a multibank program is a series of individual linkages. The result will be a load module representing each bank's code, plus that bank's contribution to the shared memory area. It is essential that command files be used as inputs to LNHPC because each module must be linked several times, and changes would be ruinous:

First, each bank's set of modules must be linked independently. The Map files from each bank's linkage will give the necessary information on:

1. Undefined references, both functions and data.
2. A list of library routines invoked to support the code.
3. The size of the __init__info__ section for the bank.
4. The size of the total code.
5. The entry for the functions defined in that bank.
6. The address of the variables defined in the bank (which is applicable for shared bank only).

This information should be checked and validated. The undefined data references must be only to data which will be in the shared memory. The undefined function references should be for the function calls defined in other banks. The library routines invoked may be reduced by library routines which will be in the shared memory to support code there, or can be placed in shared memory to use the shared ver-

sion for several banks. The size of each bank's __init__info__ section will be used to make dummy sections for the initial shared memory linkage (see next step below). Finally, the total size of the code, allowing for library routines which will be in the shared memory, must fit in the bank.

Second, an initial linkage of the shared memory is done to determine the addresses of routines and data which will be in there. This requires certain routines to be assembled:

1. The inter-bank switching routine and all the links needed for inter-bank function calls (their bank and address values are left out initially).
2. External references for any additional library routines to be forced into the shared memory.
3. Dummy __init__info__ sections which are each as large as the corresponding bank's real __init__info__ section (or one dummy section as large as all the bank's sections combined).

The shared memory is then linked with all of these items included, and the Map file will give valid addresses of data, functions, and sections.

Third, the banks can be linked to produce actual modules. All entry points in the shared memory are now defined, and need to be provided to the linkages of each bank. Assembly files providing the definitions is the simplest way to go. One file can provide the addresses of all user functions, library routines, and data variables in the shared memory, from the Map of the shared memory. Individual files need to be made to provide the addresses of the inter-bank links, because the links for a bank cannot be given to that bank. Additionally, the next available addresses need to be figured for each memory area. This provides linkage and layout by creating the new names and values to resolve the undefined references in the linkage; the linker will do the work of substituting the link address for the undefined function address. Then each bank can be linked, with the addresses for memory areas given to the linker, and the additional files defining shared memory and the other banks inter-bank links being linked in. After each bank, the next available addresses must be updated. Note that the __init__info__ sections must be contiguous and in the exact space created by the dummy routines.

Finally, the shared memory can be linked to produce the actual module. The banks and addresses must be provided for each inter-bank link and that module reassembled. The external references for additional library routines remains the same, and the dummy section for __init__info__ are unchanged. The Map of this linkage must be checked against the Map of initial linkage and/or against all addresses fed to the bank switched modules.

EXAMPLE CODE DISTRIBUTION

The example is a skeleton for a realtime program which accumulates time data into tables, then processes those tables by regression fit into a table of coefficients. The system then monitors further events and uses the coefficient to predict behavior as it occurs. The following files are to be in a system with two banks, from 0x4000 to 0x7fff.

TABLES.H	Data structure
MAIN.C	Main program, for shared bank
TABLES.C	Table accumulation and processing
MONITOR.C	Monitor external events and predict
ERRORS.C	Error routines
TIMERS.C	Timer initialization and interrupt service
UART.C	UART processing and interrupt service
CRTFIRST.ASM	Modified to set up Port B for Bankswitching
CRTFIRST.INC	<Standard module, unchanged>
BANKSWIT.ASM	<Standard module, unchanged>
BANKLINK.INC	Modified for inter-bank linkages
BANKDEFS.INC	Macro definitions to simplify linkages

The distribution shown in Table I is intended as an initial starting point. The monitor and prediction code is very large, and fills the bank. The table processing code has room left so the error routines (which are seldom called) are fit in there. This bank has RAM in it, which is not known to the compiler but is managed by the program. Main is in shared memory because it is the major loop of the program. Timers and UART are in shared because they contain the Interrupt Service Routines.

Shared	Bank 0	Bank 1
Main	Tables	Monitor
Timers	Errors	Strings
UART	Strings	Constants
Crtfirst	Constants	
Statics	Table RAM	
Initialization		
Printf		
Stack		
Bank Stack		
Crtinit		

All statics will be in shared memory. Initialization data is in shared memory. The string literals are all in ROM, and will be in banks; since these are passed as arguments to printf( ), printf( ) must either be in both banks or in shared memory in this case, to avoid duplication of memory usage and to save room in Bank 1. Constants are in banks, since inter-bank calls can be avoided when using constants and string literals. The stack and the bank stack are in the shared memory. The crtfirst routine is modified, and crtinit is with it in shared memory (although it may be possible to have crtinit in the bank selected by the START macro, this would require more manual linkage for the call in crtfirst).

LINKAGE PROCEDURE

Each bank load module is created by linking the banks separately. The linking is done in two steps. The first step is trial linkage and the parameters are specified in BANK0__1.CMD,BANK1__1.CMD and SHARED__1.CMD for linker. The information from this trial linkage is used in the second attempt where the load module is actually created. The command files used are BANK0__2.CMD, BANK1__2.CMD and SHARED__2.CMD.

Initial linker command files are:

```
SHARED_1.CMD
BANK0_1.CMD
BANK1_1.CMD
```

describing memory as

```
0000-01ff  shared: onchip RAM & I/O
0200-0fff  shared: offchip RAM
1000-3fff  shared: ROM
4000-7fff  banks
           bank 0: 4000-5fff  ROM
                   6000-7fff  RAM, private
           bank 1: 4000-7fff  ROM
8000-ffff  shared: ROM
```

where the private RAM is not mentioned to the linker. The private RAM is defined to the compiler using constants; another alternative would have been to define an assembler module of the proper size allocating the space, and place it with the linker. This would require another piece of assembly code, but would limit the address information to the linker command files.

During the trial linkage Bank 0 links but contains printf( ), which was desired to be in shared memory so it can be passed string literals; putchar( ) will also be there. This leaves only the variable live, which is just fine, will be placed in shared bank. The size of __init__info__ is 0x4136 to 0x4147 or 18 bytes (this information is best taken from the Section Table of the map). The code is not present; it is assumed to fit. For Bank 1, printf( ) will again be defined in shared; putchar( ) will not be referenced. The undefined for live, capture__table( ), and error( ) are correct. The size of __init__info__ is 0x409A to 0x409F or 6 bytes. The code is assumed to fit.

The initial linkage of the shared memory requires the creation of the linkage files. The linkages have to be put into BANKLINK.INC for all inter-bank entry points, including from shared to a bank. The sizes for the __init__info__ sections and the library access forcing requests are put in a file, using BANKDEFS.INC to make things easier. These are linked together, with the C stack and the switch stack in the off-chip RAM, with the switch stack on top so that they can share the same memory. There are few inter-bank calls, so the SWITCH__STACK__DEPTH used is 10. Linking this finishes the initial sequence, and the values are now available for the real second attempt of linking whereby the actual modules will be created.

Now the definitions to complete each bank are created. The module BANKDEFS.INC makes this easier. Each bank defines the linkages to entry points within that bank. The shared defines publics within the shared memory. (These values are best taken from the Symbol Table portion of the map.) Then the linker command files need to be modified (in the example new file names are used, but the user will probably not use new files, rather simply modify the existing files). The definition files needed for each bank will be added; these are the file for shared memory and for every other bank but this one. The No Output option is changed to giving a name for the object file, if desired, and the Ignore Errors is added because there is still no reset vector for a bank.

Finally, the memory addresses have to be determined from the shared load map and put into the command file (these values are best taken from the Memory Order Map, Memory Type Map, or the Section Table). The positioning of __init__info__ is critical, the others can have gaps. A trial linkage shows where the linker places modules, and final adjustments are required to ensure such placements meet the requirements. Bank 0 requires only that the initialization data be moved to shared memory. The updated addresses from Bank 0 are used in Bank 1. Bank 1 is placed acceptably by the linker.

The final linkage of the shared memory can now be done. Address and bank information is added to the linkage list. The remaining parts don't change. This linkage must be checked against the first linkage of shared to be certain no addresses have changed. Finally, the addresses used in each bank or shared should be checked against other banks to check for overlaps, and the types of sections in each memory should be checked to make sure all conventions have been met.

If everything is correct, you have load modules for the system.

```
Contents of Linker command file BANK0__1.CMD:

/Echo
/libfile=\hpc\library
/Format=lm
/Map=bank0__1.map
/Table
/Cr
/Range=BASE=(0x0002:0x00BF)
/Range=RAM16=(0x0200:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x4000:0x5FFF,0x8000:0xFFCF,0x1000:0x3FFF)
/Range=ROM8=ROM16
tables,
errors
/NoOutput
```

TL/DD/10131–1

```
Contents of the Linker map file BANK0__1.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:37


Reset Vector:  0000

-- Range Definitions --

BASE    0002:00BF
ROM16   4000:5FFF
ROM16   8000:FFCF
ROM16   1000:3FFF
RAM16   0200:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0200  0211   RAM16
4000  4508   ROM16
450A  4687   ROM16
4688  47BF   ROM8


-- Memory Type Map --

BASE
  [size = 0000]

RAM16
  0200  0211
  [size = 0012]

RAM8
  [size = 0000]

ROM16
  4000  4508
  450A  4687
  [size = 0687]

ROM8
  4688  47BF
  [size = 0138]

VECTOR
  [size = 0000]
```

TL/DD/10131–2

```
-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0200  0211
  [size = 0012]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  4000  4508
  450A  4687
  4688  47BF
  [size = 07BF]



-- Section Table --

start end    attributes       Section
                                Module

0200  020D   RAM16 WORD       TABLES_RAM16_BSS
0200  020D                     tables
4000  4135   ROM16 WORD       TABLES_CODE
4000  4135                     tables
4136  4147   ROM16 WORD       _INIT_INFO_
4136  413B                     tables
413C  4147                     errors
020E  020F   RAM16 WORD       ERRORS_RAM16_DATA
020E  020F                     errors
0210  0211   RAM16 WORD       ERRORS_RAM16_BSS
0210  0211                     errors
4148  41C3   ROM16 WORD       ERRORS_CODE
4148  41C3                     errors
4688  4689   ROM8  WORD       ERRORS_RAM16_INIT
4688  4689                     errors
468A  4703   ROM8  BYTE       ERRORS_ROM8_STRDATA
468A  4703                     errors
41C4  4508   ROM16 WORD       LIBI_CODE
41C4  4508                     libi
450A  4687   ROM16 WORD       LIBP_CODE
450A  4687                     libp
4704  47BF   ROM8  BYTE       LIBRARY
4704  47BF                     LIBIDVL

Error:   Undefined External:  _live
         Address:  0096
         Module:  tables
Error:   Undefined External:  _putchar
         Address:  0044
         Module:  errors
Error:   Undefined External:  _putchar
         Address:  004A
```

TL/DD/10131–3

```
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  0086
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  0190
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  028A
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  02D9
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  0337
        Module:  libi
Error:  Undefined External:  _putchar
        Address:  0027
        Module:  libp
Error:  Undefined External:  _putchar
        Address:  0057
        Module:  libp
Error:  Undefined External:  _putchar
        Address:  0146
        Module:  libp
Error:  Undefined External:  _putchar
        Address:  0175
        Module:  libp
Error:  No End Address has been specified


signed_divide_32 . . . .  4704 Null ROM8
   -LIBIDVL
signed_remainder_32  . .  4708 Null ROM8
   -LIBIDVL
unsigned_divide_32 . . .  4739 Null ROM8
   -LIBIDVL    libp
unsigned_remainder_32  .  473D Null ROM8
   -LIBIDVL    libp
_build_tables  . . . . .  404B Null ROM16
   -tables
_capture_table . . . . .  404E Null ROM16
   -tables
_compute_coefficients  .  40AB Null ROM16
   -tables
_d_printf  . . . . . .  453C Null ROM16
   -libp       libi
_error . . . . . .  4148 Null ROM16
   -errors
_fatal_error . . . . . .  416B Null ROM16
   -errors
_initialize_table_memory  4000 Null ROM16
   -tables
_live  . . . . . .  **** Null
```

TL/DD/10131–4

```
    tables
_printf  . . . . . .  41C4 Null ROM16
   -libi       errors
_putchar . . . . . .  **** Null
    errors     libi       libp
_quit  . . . . . .  41B0 Null ROM16
   -errors
_s_printf  . . . . . .  450A Null ROM16
   -libp       libi
_u_printf  . . . . . .  459A Null ROM16
   -libp       libi
```

TL/DD/10131–5

Information obtained from BANK0__1.MAP are:

1) The _INIT_INFO_ section size for Bank0 linkage is 18 bytes,
   ie from 0x4136 to 0x4147.

2) The entry address for functions obtained here as follows:

Function	Address
initialize_table_memory	0x4000
build_tables	0x404b
capture_table	0x404e
compute_coefficients	0x40ab
error	0x4136
fatal_error	0x4159

   These addresses will be used by the SWITCH_TO_FUNCTION assembly
   macro calls in the file BANKLINK.INC.

3) The undefined external reference for the variable live is expected,
   which will be defined in the SHARED bank. The undefined function
   putchar will also be defined in the shared bank.

TL/DD/10131–6

```
Contents of Linker command file BANK1__1.CMD:

/Echo
/libfile=\hpc\library
/Format=lm
/Map=bank1__1.map
/Table
/Cr
/Range=BASE=(0x0002:0x00BF)
/Range=RAM16=(0x0200:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x4000:0x7FFF,0x8000:0xFFCF,0x1000:0x3FFF)
/Range=ROM8=ROM16
monitor
/NoOutput
```

TL/DD/10131–7

```
Contents of the Linker map file BANK1__1.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:37


Reset Vector:  0000

-- Range Definitions --

BASE    0002:00BF
ROM16   4000:7FFF
ROM16   8000:FFCF
ROM16   1000:3FFF
RAM16   0200:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0200  0201   RAM16
4000  43E4   ROM16
43E6  4563   ROM16
4564  465F   ROM8


-- Memory Type Map --

BASE
  [size = 0000]

RAM16
  0200  0201
  [size = 0002]

RAM8
  [size = 0000]

ROM16
  4000  43E4
  43E6  4563
  [size = 0563]

ROM8
  4564  465F
  [size = 00FC]

VECTOR
  [size = 0000]
```

TL/DD/10131–8

```
-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0200  0201
  [size = 0002]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  4000  43E4
  43E6  4563
  4564  465F
  [size = 065F]



-- Section Table --

start end    attributes       Section
                                Module

0200  0201   RAM16 WORD       MONITOR_RAM16_BSS
0200  0201                     monitor
4000  4099   ROM16 WORD       MONITOR_CODE
4000  4099                     monitor
4564  45A3   ROM8  BYTE       MONITOR_ROM8_STRDATA
4564  45A3                     monitor
409A  409F   ROM16 WORD       _INIT_INFO_
409A  409F                     monitor
40A0  43E4   ROM16 WORD       LIBI_CODE
40A0  43E4                     libi
43E6  4563   ROM16 WORD       LIBP_CODE
43E6  4563                     libp
45A4  465F   ROM8  BYTE       LIBRARY
45A4  465F                     LIBIDVL

Error:   Undefined External:  _live
         Address:  0002
         Module:  monitor
Error:   Undefined External:  _live
         Address:  0012
         Module:  monitor
Error:   Undefined External:  _live
         Address:  0026
         Module:  monitor
Error:   Undefined External:  _capture_table
         Address:  0030
         Module:  monitor
Error:   Undefined External:  _error
         Address:  0091
         Module:  monitor
Error:   Undefined External:  _putchar
         Address:  004A
         Module:  libi
```

TL/DD/10131–9

```
Error:   Undefined External:  _putchar
         Address:  0086
         Module:  libi
Error:   Undefined External:  _putchar
         Address:  0190
         Module:  libi
Error:   Undefined External:  _putchar
         Address:  028A
         Module:  libi
Error:   Undefined External:  _putchar
         Address:  02D9
         Module:  libi
Error:   Undefined External:  _putchar
         Address:  0337
         Module:  libi
Error:   Undefined External:  _putchar
         Address:  0027
         Module:  libp
Error:   Undefined External:  _putchar
         Address:  0057
         Module:  libp
Error:   Undefined External:  _putchar
         Address:  0146
         Module:  libp
Error:   Undefined External:  _putchar
         Address:  0175
         Module:  libp
Error:   No End Address has been specified


 signed_divide_32 . .   45A4 Null ROM8
    -LIBIDVL
 signed_remainder_32    45A8 Null ROM8
    -LIBIDVL
 unsigned_divide_32 .   45D9 Null ROM8
    -LIBIDVL    libp
 unsigned_remainder_32  45DD Null ROM8
    -LIBIDVL    libp
 _capture_table . . .   **** Null
     monitor
 _compute_prediction    4032 Null ROM16
    -monitor
 _d_printf  . . . . .   4418 Null ROM16
    -libp       libi
 _error . . . . . .    **** Null
     monitor
 _live  . . . . . .    **** Null
     monitor
 _monitor . . . . . .   4000 Null ROM16
    -monitor
 _printf  . . . . . .   40A0 Null ROM16
    -libi       monitor
 _putchar . . . . . .   **** Null
     libi       libp
```

TL/DD/10131–10

```
_s_printf  . . . . .   43E6 Null ROM16
   -libp       libi
_u_printf  . . . . .   4476 Null ROM16
   -libp       libi
_validate_calculation  4053 Null ROM16
   -monitor
```

TL/DD/10131–11

The informations derieved from this file are:

1) The _INIT_INFO_ section size for Bank0 linkage is 6 bytes.
   ie, from 0x409a to 0x409f.

2) The entry address for functions obtained here as follows:

Function	Address
monitor	0x4000

   These addresses will be used by the SWITCH_TO_FUNCTION assembly
   macro calls in the file BANKLINK.INC.

3) The undefined external reference for the variable live is expected,
   which will be defined in the SHARED bank. The undefined function
   putchar will also be defined in the shared bank. The undefined external
   references for functions defined in Bank0 and Shared will be taken care
   by proper link addresses during second pass of linkage.

TL/DD/10131–12

Contents of the Linker command file SHARED_1.CMD:

```
/Echo
/libfile=\hpc\library
/Format=lm
/Map=shared_1.map
/Table
/Cr
/Range=BASE=(0x0002:0x00BF)
/Range=RAM16=(0x0200:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x8000:0xFFCF,0x1000:0x3FFF)
/Range=ROM8=ROM16
/Sect=c_stack=0x0200:0x0FFF
/Sect=switch_stack=c_stack
main,
timers,
uart,
crtfirst,
bankswit, shared_1
/NoOutput
```

Note that we include the files BANKSWIT.ASM and SHARED_1.ASM.
BANKSWIT.ASM includes BANKLINK.INC file in which we have made the switch_to_function macro calls for the inter bank function refernces.
SHARED_1.ASM contains the init_dummy macro call to create continuous space for _INIT_INFO_ section in shared memory. Also it contains the force_library macro call to force PUTCHAR and PRINTF in shared address space.

TL/DD/10131–13

```
Contents of the Linker output file SHARED_1.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:37


Reset Vector:  FFAF

-- Range Definitions --

BASE    0002:00BF
ROM16   8000:FFCF
ROM16   1000:3FFF
RAM16   0200:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0002  0003   BASE
0200  0ABF   RAM16
8000  80A8   ROM16
80AA  8118   ROM16
811A  845E   ROM16
8460  85DD   ROM16
85DE  873C   ROM8
FFAF  FFBF   ROM8
FFF4  FFF5   VECTOR
FFFA  FFFB   VECTOR
FFFE  FFFF   VECTOR


-- Memory Type Map --

BASE
  0002  0003
  [size = 0002]

RAM16
  0200  0ABF
  [size = 08C0]

RAM8
  [size = 0000]

ROM16
  8000  80A8
  80AA  8118
  811A  845E
  8460  85DD
  [size = 05DB]
```

TL/DD/10131–14

```
ROM8
  85DE  873C
  FFAF  FFBF
  [size = 0170]

VECTOR
  FFF4  FFF5
  FFFA  FFFB
  FFFE  FFFF
  [size = 0006]



-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0002  0003
  0200  0ABF
  [size = 08C2]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  8000  80A8
  80AA  8118
  811A  845E
  8460  85DD
  85DE  873C
  FFAF  FFBF
  FFF4  FFF5
  FFFA  FFFB
  FFFE  FFFF
  [size = 0751]



-- Section Table --

start end    attributes       Section
                                Module

0200  09FF   RAM16 WORD       C_STACK
0200  09FF                      main
0A00  0A27   RAM16 WORD       SWITCH_STACK
0A00  0A27                      Bank_Switch
0A28  0A2D   RAM16 WORD       MAIN_RAM16_DATA
0A28  0A2D                      main
0A2E  0A41   RAM16 WORD       MAIN_RAM16_BSS
0A2E  0A41                      main
8000  8031   ROM16 WORD       MAIN_CODE
8000  8031                      main
85DE  85E3   ROM8  WORD       MAIN_RAM16_INIT
85DE  85E3                      main
8032  8061   ROM16 WORD       _INIT_INFO_
```

TL/DD/10131–15

```
8032  803D                      main
803E  8043                      timers
8044  8049                      Bank_Switch
804A  8061                      SHARED_1
0A42  0ABF   RAM16 WORD       TIMERS_RAM16_BSS
0A42  0ABF                      timers
8062  80A8   ROM16 WORD       TIMERS_CODE
8062  80A8                      timers
80AA  8118   ROM16 WORD       UART_CODE
80AA  8118                      uart
FFAF  FFBF   ROM8  ABS        CRTFIRST
FFAF  FFBF                      crtfirst
0002  0003   BASE  WORD       SWITCH_POINTER
0002  0003                      Bank_Switch
85E4  85E5   ROM8  BYTE       SWITCH_INIT
85E4  85E5                      Bank_Switch
85E6  8659   ROM8  BYTE       SWITCH_CODE
85E6  8659                      Bank_Switch
865A  873C   ROM8  BYTE       LIBRARY
865A  8680                      crtinit
8681  873C                      LIBIDVL
811A  845E   ROM16 WORD       LIBI_CODE
811A  845E                      libi
8460  85DD   ROM16 WORD       LIBP_CODE
8460  85DD                      libp


 initialize_memories  . .  865A Null ROM8
    -crtinit    crtfirst
 PROGRAM_exit . . . . . .  FFBF Null ROM8
    -crtfirst
 PROGRAM_start  . . . . .  FFAF Null ROM8
    -crtfirst   main
 STACK_end  . . . . . . .  0A00 Null RAM16
    -main
 STACK_start  . . . . . .  0200 Null RAM16
    -main       crtfirst
 signed_divide_32 . . . .  8681 Null ROM8
    -LIBIDVL
 signed_remainder_32  . .  8685 Null ROM8
    -LIBIDVL
 unsigned_divide_32 . . .  86B6 Null ROM8
    -LIBIDVL    libp
 unsigned_remainder_32  .  86BA Null ROM8
    -LIBIDVL    libp
 _build_tables  . . . . .  85EB Null ROM8
    -Bank_Switch           main
 _button_service  . . . .  8086 Null ROM16
    -timers
 _calibrating . . . . . .  0A2A Byte RAM16
    -main
 _capture_table . . . . .  85F0 Null ROM8
    -Bank_Switch
```

TL/DD/10131–16

```
_coefficients  . . . . .  0A2E Byte RAM16
   -main
_compute_coefficients  .  85F5 Null ROM8
   -Bank_Switch           main
_d_printf  . . . . . .  8492 Null ROM16
   -libp       libi
_error . . . . . .  85FF Null ROM8
   -Bank_Switch
_fatal_error . . . . . .  8604 Null ROM8
   -Bank_Switch
_initialize_inputs . . .  8062 Null ROM16
   -timers     main
_initialize_outputs  . .  80AA Null ROM16
   -uart       main
_initialize_table_memory  85E6 Null ROM8
   -Bank_Switch           main
_live  . . . . . .  0A42 Byte RAM16
   -timers
_main  . . . . . .  8000 Null ROM16
   -main       crtfirst
_monitor . . . . . .  85FA Null ROM8
   -Bank_Switch           main
_operational . . . . . .  0A28 Byte RAM16
   -main
_predicting  . . . . . .  0A2C Byte RAM16
   -main
_printf  . . . . . .  811A Null ROM16
   -libi       SHARED_1
_put_uart  . . . . . .  810E Null ROM16
   -uart
_putchar . . . . . .  80AB Null ROM16
   -uart       libi       libp
_s_printf  . . . . . .  8460 Null ROM16
   -libp       libi
_timer_service . . . . .  8063 Null ROM16
   -timers
_u_printf  . . . . . .  84F0 Null ROM16
   -libp       libi
```

Notice that there is entry for each function that is actully placed in switchable bank being called from other banks. These entries are used as link addresses for the respective functions when linking the banks individually. Refer the files shared.asm, bank0.asm and bank1.asm.

TL/DD/10131–17

Contents of the linker command file BANK0__2.CMD:

```
/Echo
/libfile=\hpc\library
/Format=lm
/Map=bank0__2.map
/Table
/Cr
/Range=BASE=(0x0004:0x00BF)
/Range=RAM16=(0x0B00:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x4000:0x5FFF,0x8740:0xFFAE,0x1000:0x3FFF)
/Range=ROM8=ROM16
tables,
errors,
shared, bank1
/Sect=_init_info_=0x804A:0x8061
/Sect=errors_ram16_init=0x8740
/Output=bank0
/Ignore
```

Notice the ROM address is space defined as 0x4000:0x5fff for the BANK0 space. In shared memory address range 0x8740:0xffae is available, which is obtained from shared_1.map. Also _init_info_ goes into the range 0x804a:0x8061 which was reserved by the init_dummy macro and the address is obtained from shared_1.map. Also the section errors_ram16_init goes to address 0x8740 onwards. The link addresses for the functions and variables are specified in shared.asm and bank1.asm.

TL/DD/10131–18

```
Contents of the Linker output file BANK0__2.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:38


Reset Vector:  0000

-- Range Definitions --

BASE    0004:00BF
ROM16   4000:5FFF
ROM16   8740:FFAE
ROM16   1000:3FFF
RAM16   0B00:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0B00  0B11   RAM16
4000  41B1   ROM16
41B2  422B   ROM8
804A  805B   ROM16
8740  8741   ROM8


-- Memory Type Map --

BASE
  [size = 0000]

RAM16
  0B00  0B11
  [size = 0012]

RAM8
  [size = 0000]

ROM16
  4000  41B1
  804A  805B
  [size = 01C4]

ROM8
  41B2  422B
  8740  8741
  [size = 007C]

VECTOR
  [size = 0000]
```

TL/DD/10131–19

```
-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0B00  0B11
  [size = 0012]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  4000  41B1
  41B2  422B
  804A  805B
  8740  8741
  [size = 0240]



-- Section Table --

start end    attributes       Section
                                Module

804A  805B   ROM16 WORD       _INIT_INFO_
804A  804F                      tables
8050  805B                      errors
8740  8741   ROM8  WORD       ERRORS_RAM16_INIT
8740  8741                      errors
0B00  0B0D   RAM16 WORD       TABLES_RAM16_BSS
0B00  0B0D                      tables
4000  4135   ROM16 WORD       TABLES_CODE
4000  4135                      tables
0B0E  0B0F   RAM16 WORD       ERRORS_RAM16_DATA
0B0E  0B0F                      errors
0B10  0B11   RAM16 WORD       ERRORS_RAM16_BSS
0B10  0B11                      errors
4136  41B1   ROM16 WORD       ERRORS_CODE
4136  41B1                      errors
41B2  422B   ROM8  BYTE       ERRORS_ROM8_STRDATA
41B2  422B                      errors

Error:   No End Address has been specified



 _build_tables  . . . . .  404B Null ROM16
    -tables
 _capture_table . . . . .  404E Null ROM16
    -tables
 _coefficients  . . . . .  0A2E Null
    -SHARED
 _compute_coefficients  .  40AB Null ROM16
    -tables
```

TL/DD/10131–20

```
    _error . . . . . .  4136 Null ROM16
       -errors
    _fatal_error . . . . . .  4159 Null ROM16
       -errors
    _initialize_table_memory  4000 Null ROM16
       -tables
    _live  . . . . . .  0A42 Null
       -SHARED     tables
    _monitor . . . . . .  85FC Null
       -BANK1
    _printf  . . . . . .  811A Null
       -SHARED     errors
    _putchar . . . . . .  80AB Null
       -SHARED     errors
    _quit  . . . . . .  419E Null ROM16
       -errors
```

Notice that there is no undefined external references errors.

TL/DD/10131–21

Since the function Main is not defined in this bank there is no reset vector address defined and hence the Linker gives the 'no end address specified' error message, which can be ignored.

Contents of the Linker command file BANK1__1.CMD:

```
/Echo
/libfile=\hpc\library
/Format=lm
/Map=bank1__2.map
/Table
/Cr
/Range=BASE=(0x0004:0x00BF)
/Range=RAM16=(0x0C00:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x4000:0x7FFF,0x8742:0xFFAE,0x1000:0x3FFF)
/Range=ROM8=ROM16
monitor,
shared, bank0
/Sect=_init_info_=0x805C:0x8061
/Output=bank1
/Ignore
```

Notice the _init_info_ section is placed in address space 0x805c to 0x8061.
This is basically the rest of the space after Bank0 _init_info_ usage.
Also the Link addresses for BANK0 and SHARED are appropriately mentioned
in the assembly files bank0.asm and shared.asm and they are also linked.
The shared address (ROM16 and RAM16) space is properly updated with the
information from bank0__1.map.

TL/DD/10131–22

```
Contents of the Linker output file BANK1__2.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:38


Reset Vector:  0000

-- Range Definitions --

BASE    0004:00BF
ROM16   4000:7FFF
ROM16   8742:FFAE
ROM16   1000:3FFF
RAM16   0C00:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0C00  0C01   RAM16
4000  4099   ROM16
409A  40D9   ROM8
805C  8061   ROM16


-- Memory Type Map --

BASE
  [size = 0000]

RAM16
  0C00  0C01
  [size = 0002]

RAM8
  [size = 0000]

ROM16
  4000  4099
  805C  8061
  [size = 00A0]

ROM8
  409A  40D9
  [size = 0040]

VECTOR
  [size = 0000]
```

TL/DD/10131–23

```
-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0C00  0C01
  [size = 0002]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  4000  4099
  409A  40D9
  805C  8061
  [size = 00E0]



-- Section Table --

start end    attributes       Section
                                Module

805C  8061   ROM16 WORD       _INIT_INFO_
805C  8061                     monitor
0C00  0C01   RAM16 WORD       MONITOR_RAM16_BSS
0C00  0C01                     monitor
4000  4099   ROM16 WORD       MONITOR_CODE
4000  4099                     monitor
409A  40D9   ROM8  BYTE       MONITOR_ROM8_STRDATA
409A  40D9                     monitor

Error:   No End Address has been specified


 _build_tables  . . . . .  85ED Null
    -BANK0
 _capture_table . . . . .  85F2 Null
    -BANK0      monitor
 _coefficients  . . . . .  0A2E Null
    -SHARED
 _compute_coefficients  .  85F7 Null
    -BANK0
 _compute_prediction  . .  4032 Null ROM16
    -monitor
 _error . . . . . .  8601 Null
    -BANK0      monitor
 _fatal_error . . . . . .  8606 Null
    -BANK0
 _initialize_table_memory  85E8 Null
    -BANK0
 _live  . . . . . .  0A42 Null
    -SHARED     monitor
 _monitor . . . . . .  4000 Null ROM16
    -monitor
```

TL/DD/10131–24

```
_printf  . . . . . .  811A Null
   -SHARED     monitor
_putchar . . . . . .  80AB Null
   -SHARED
_validate_calculation  .  4053 Null ROM16
   -monitor
```

Notice that there is no undefined external reference error messages.

TL/DD/10131–25

Since the function Main is not defined in this bank there is no reset vector address defined and hence the Linker gives the 'no end address specified' error message, which can be ignored.

Contents of the Linker command file SHARED_2.CMD which is same as SHARED_1.CMD:

```
/Echo
/libfile=\hpc\library
/Format=lm
/Map=shared_2.map
/Table
/Cr
/Range=BASE=(0x0002:0x00BF)
/Range=RAM16=(0x0200:0x0FFF,0x01C0:0x01FF,BASE)
/Range=RAM8=RAM16
/Range=ROM16=(0x8000:0xFFCF,0x1000:0x3FFF)
/Range=ROM8=ROM16
/Sect=c_stack=0x0200:0x0FFF
/Sect=switch_stack=c_stack
main,
timers,
uart,
crtfirst,
bankswit, shared_1
/Output=shared
```

TL/DD/10131–26

```
Contents of the Linker output file SHARED_2.MAP which should be identical to
SHARED_1.MAP:

NSC LNHPC, Version E2 (Nov 02 15:46 1987)                    09-May-88 08:38


Reset Vector:  FFAF

-- Range Definitions --

BASE    0002:00BF
ROM16   8000:FFCF
ROM16   1000:3FFF
RAM16   0200:0FFF
RAM16   01C0:01FF
RAM16   BASE
ROM8    ROM16
RAM8    RAM16

-- Memory Order Map --

0002  0003   BASE
0200  0ABF   RAM16
8000  80A8   ROM16
80AA  8118   ROM16
811A  845E   ROM16
8460  85DD   ROM16
85DE  873C   ROM8
FFAF  FFBF   ROM8
FFF4  FFF5   VECTOR
FFFA  FFFB   VECTOR
FFFE  FFFF   VECTOR


-- Memory Type Map --

BASE
  0002  0003
  [size = 0002]

RAM16
  0200  0ABF
  [size = 08C0]

RAM8
  [size = 0000]

ROM16
  8000  80A8
  80AA  8118
  811A  845E
  8460  85DD
  [size = 05DB]
```

TL/DD/10131–27

```
ROM8
  85DE  873C
  FFAF  FFBF
  [size = 0170]

VECTOR
  FFF4  FFF5
  FFFA  FFFB
  FFFE  FFFF
  [size = 0006]



-- Total Memory Map --

TOTAL RAM = BASE + RAM16 + RAM8
  0002  0003
  0200  0ABF
  [size = 08C2]


TOTAL ROM = ROM16 + ROM8 + VECTOR
  8000  80A8
  80AA  8118
  811A  845E
  8460  85DD
  85DE  873C
  FFAF  FFBF
  FFF4  FFF5
  FFFA  FFFB
  FFFE  FFFF
  [size = 0751]



-- Section Table --

start end    attributes       Section
                                Module

0200  09FF   RAM16 WORD       C_STACK
0200  09FF                      main
0A00  0A27   RAM16 WORD       SWITCH_STACK
0A00  0A27                      Bank_Switch
0A28  0A2D   RAM16 WORD       MAIN_RAM16_DATA
0A28  0A2D                      main
0A2E  0A41   RAM16 WORD       MAIN_RAM16_BSS
0A2E  0A41                      main
8000  8031   ROM16 WORD       MAIN_CODE
8000  8031                      main
85DE  85E3   ROM8  WORD       MAIN_RAM16_INIT
85DE  85E3                      main
```

TL/DD/10131–28

```
8032  8061   ROM16 WORD       _INIT_INFO_
8032  803D                     main
803E  8043                     timers
8044  8049                     Bank_Switch
804A  8061                     SHARED_1
0A42  0ABF   RAM16 WORD       TIMERS_RAM16_BSS
0A42  0ABF                     timers
8062  80A8   ROM16 WORD       TIMERS_CODE
8062  80A8                     timers
80AA  8118   ROM16 WORD       UART_CODE
80AA  8118                     uart
FFAF  FFBF   ROM8  ABS        CRTFIRST
FFAF  FFBF                     crtfirst
0002  0003   BASE  WORD       SWITCH_POINTER
0002  0003                     Bank_Switch
85E4  85E5   ROM8  BYTE       SWITCH_INIT
85E4  85E5                     Bank_Switch
85E6  8659   ROM8  BYTE       SWITCH_CODE
85E6  8659                     Bank_Switch
865A  873C   ROM8  BYTE       LIBRARY
865A  8680                     crtinit
8681  873C                     LIBIDVL
811A  845E   ROM16 WORD       LIBI_CODE
811A  845E                     libi
8460  85DD   ROM16 WORD       LIBP_CODE
8460  85DD                     libp


 initialize_memories  . .  865A Null ROM8
    -crtinit    crtfirst
 PROGRAM_exit . . . . . .  FFBF Null ROM8
    -crtfirst
 PROGRAM_start  . . . . .  FFAF Null ROM8
    -crtfirst   main
 STACK_end  . . . . . . .  0A00 Null RAM16
    -main
 STACK_start  . . . . . .  0200 Null RAM16
    -main       crtfirst
 signed_divide_32 . . . .  8681 Null ROM8
    -LIBIDVL
 signed_remainder_32  . .  8685 Null ROM8
    -LIBIDVL
 unsigned_divide_32 . . .  86B6 Null ROM8
    -LIBIDVL    libp
 unsigned_remainder_32  .  86BA Null ROM8
    -LIBIDVL    libp
 _build_tables  . . . . .  85EB Null ROM8
    -Bank_Switch           main
 _button_service  . . . .  8086 Null ROM16
    -timers
 _calibrating . . . . . .  0A2A Byte RAM16
    -main
 _capture_table . . . . .  85F0 Null ROM8
```

TL/DD/10131–29

```
   -Bank_Switch
_coefficients  . . . . .  0A2E Byte RAM16
   -main
_compute_coefficients  .  85F5 Null ROM8
   -Bank_Switch           main
_d_printf  . . . . . .  8492 Null ROM16
   -libp       libi
_error . . . . . .  85FF Null ROM8
   -Bank_Switch
_fatal_error . . . . . .  8604 Null ROM8
   -Bank_Switch
_initialize_inputs . . .  8062 Null ROM16
   -timers     main
_initialize_outputs  . .  80AA Null ROM16
   -uart       main
_initialize_table_memory  85E6 Null ROM8
   -Bank_Switch           main
_live  . . . . . .  0A42 Byte RAM16
   -timers
_main  . . . . . .  8000 Null ROM16
   -main       crtfirst
_monitor . . . . . .  85FA Null ROM8
   -Bank_Switch           main
_operational . . . . . .  0A28 Byte RAM16
   -main
_predicting  . . . . . .  0A2C Byte RAM16
   -main
_printf  . . . . . .  811A Null ROM16
   -libi       SHARED_1
_put_uart  . . . . . .  810E Null ROM16
   -uart
_putchar . . . . . .  80AB Null ROM16
   -uart       libi       libp
_s_printf  . . . . . .  8460 Null ROM16
   -libp       libi
_timer_service . . . . .  8063 Null ROM16
   -timers
_u_printf  . . . . . .  84F0 Null ROM16
   -libp       libi
```

After final linkages the shared bank address space in the map files BANK0__2.MAP, BANK1__2.MAP and SHARED_2.MAP should be verified for no memory overlap.

TL/DD/10131–30

```
;       ********************************************************
;       *                                                      *
;       *       National Semiconductor MicroController Group   *
;       *                                                      *
;       *       HPC C Compiler Support and Library Routines    *
;       *       C Run Time Initialization User Tunable Code    *
;       *       CRTFIRST.ASM -  C run time initialization      *
;       ********************************************************

;Copyright (c) 1987, National Semiconductor, Santa Clara Ca 95051

;See CRTFIRST.INC source code for explanation of macros and usage

;Code origin
        .sect   crtfirst,rom8,abs=0xffaf

        ld      0x00f3.b,#0xff          ;output pins for upper Port B
        ld      0x00e3.b,#0x00          ;select bank 0


        jp      .

        .incld  crtfirst.inc

        .end    PROGRAM_start
```

TL/DD/10131–31

```
; SHARED_1.ASM - Bank switch support function
; To force library functions onto shared bank and to
; allocate continuous space for _init_info_ section on the
; shared bank.
;
.incld  bankdefs.inc

force_library   printf

init_dummy      18, 6

.end

; BANK0.ASM - Link address for functions actually defined
; in bank0

.incld  bankdefs.inc

link_address    initialize_table_memory, 0x85e8
link_address    build_tables, 0x85ed
link_address    capture_table, 0x85f2
link_address    compute_coefficients, 0x85f7
link_address    error, 0x8601
link_address    fatal_error, 0x8606

.end


; BANK1.ASM - Link address for the function actually defined
; in bank1.

.incld  bankdefs.inc

link_address    monitor, 0x85fc

.end


; SHARED.ASM - Link address for the functions and variables
; defined in shared address space.

.incld  bankdefs.inc

link_address    printf, 0x811a
link_address    putchar, 0x80ab
link_address    live, 0x0a42
link_address    coefficients, 0x0a2e

.end
```

TL/DD/10131–41

```
        .title  crtfirst, 'C Run Time Initialization'
;       ************************************************************
;       *                                                          *
;       *       National Semiconductor MicroController Group       *
;       *                                                          *
;       *       HPC C Compiler Support and Library Routines        *
;       *       CRTFIRST.INC - C Run Time Initialization           *
;       *                                                          *
;       ************************************************************

;Copyright (c) 1987, National Semiconductor, Santa Clara Ca 95051

;Edit History
; 12/15/86 DKL  Create from CCHPC startup output
; 2/6/87 DKL    Convert to new Assembler Syntax
; 2/9/87 DKL    Seperate out Tunable Code
; 3/4/87 RPG    Modify to suit new compiler
; 3/10/87 DKL   Changes to DKL arrangement, initialize memory
; 3/20/87 DKL   Stack out, efficient list order in
; 5/6/87 DKL    Make this the included, not includer, file
; 7/27/87 DKL   Move Initialization of RAM to separate subroutine
;

        .public PROGRAM_start, PROGRAM_exit
        .extrn  _main
     .ifndef memories_8bit
        .extrn  initialize_memories
     .else
        .extrn  initialize_memories_8bit
     .endif
        .extrn  STACK_start

        .form
;This routine provides the standard C RunTime Routine for starting a
;compiled and linked program.  It initializes the stack pointer and
;RAM memories, and enters the compiler generated code in function
;"main()" with no arguments.

;Four macros are used to allow the end user to have control of the start
;process at key moments, before the C code begins execution.  The macros
;used are ORIGIN, START, READY, and HALT, in the following fashion:
;
;       ORIGIN
;PROGRAM_start:
;       ld      sp,<stack>
;       START
;       jsrl    initialize_memories
;       READY
;       jsrl    _main
;PROGRAM_exit:
;       HALT
;
;Code size is tested to ensure that the code does not overwrite any
;dedicated addresses (e.g., subroutine jump table), and optionally to
```

TL/DD/10131–32

```
;ensure that no space is wasted between the end and the dedicated area.
;The dedicated address is defined as ADDRESS_limit, and the check for
;waste space is controlled by ORIGIN_check being non-zero.  Either of
;these may be redefined by the user in the ORIGIN macro.

;ORIGIN macro
;Must declare the section and set the absolute origin for the startup
;code.  Code must end before any dedicated addresses (ADDRESS_limit),
;and should not waste any space.  If any of the other macros here are
;lengthened, this must be adjusted.  Might optionally redefine values
;of ADDRESS_limit or ORIGIN_check.
;

;START macro
;Code to execute after the stack pointer is initialized, and before the
;memories are initialized.  Must enable the appropriate configuration
;options for the chip, so that memories can be accessed.  Since all
;memories can be accessed, the list of RAM memories can be accessed
;where ever it may be.
;

;READY macro
;Code to execute after memory is initialized, but before the C code is
;entered.
;

;HALT macro
;Code to execute when the C code terminates.
;

;Limit address of code for this routine (first dedicated address)

;Whether to check that the origin provided is exactly correct

        .form
;C RunTime Initialization Startup Code
        ORIGIN  ;declares absolute section and defines address

PROGRAM_start:
        ld      sp,#STACK_start         ;Initialize stack
        START                           ;User code option
 .ifndef memories_8bit
        jsrl    initialize_memories
 .else
        jsrl    initialize_memories_8bit
 .endif
        READY
        jsrl    _main
PROGRAM_exit:
        HALT

origin= ADDRESS_limit - . + PROGRAM_start
 .if . > ADDRESS_limit
        .ERROR  'Startup Routine overlaps Subroutine Jump Table'
 .else
```

TL/DD/10131–33

```
 .if . < ADDRESS_limit & ORIGIN_check
      .ERROR  'Startup Routine not contiguous to Subroutine Jump Table'
 .endif
.endif
```

TL/DD/10131–34

```
        .Title  Bank_Switch, 'Bank Switch Function for Function Calls'
;       ********************************************************
;       *                                                      *
;       *       National Semiconductor MicroController Group   *
;       *                                                      *
;       *       HPC Code to Support Inter-Bank Function Calls  *
;       *       BANKSWIT.ASM -  Bank switch support functions  *
;       ********************************************************
;

;Copyright (c) 1988, National Semiconductor, Santa Clara Ca 95051

;Edit History
; 3/10/88 DKL   Create for Memo/Apps note
; 3/15/88 DKL   Add direct support for
;               C function names, assembler special
;

;This is the main switching function to allow inter-bank function calls
;transparent to the compiler and assembler.

;Requires compilation with the value SWITCH_STACK_DEPTH defined, for the
;number of levels of inter-bank function call nesting to be allowed.  The
;value should take into account any interrupt nesting from any interrupt
;service routines which may switch banks.

;Is called with stack as
;       SP -----> Next free location
;       SP-2 ---> Intermediate Switch Function Return Address
;       SP-4 ---> Destination's Return Address
;       SP-6 ---> Destination's Argument 1
;       ...       Destination's Argument Space
;       old sp -> Destination's Argument n
;       ...       Caller's Local Variable Space
;       FP -----> Caller's First Local Variable
;       FP-2 ---> Caller's Parent's Frame Pointer
;       FP-4 ---> Caller's Return Address
;       FP-6 ---> Caller's Argument 1
;       ...       Caller's Argument Space
;and must call Destination Function with stack in same form, but the
;Destination's Return Address must cause return to the switcher function.

;An additional stack is necessary to store the additional information so
;the main stack is not polluted.  This also requires an additional stack
;pointer.

        .form
 .macro  switch_to       function, bank, address
        .public _~function
_~function:
        jsr     function_call_switcher
 .if @ > 1
        .byte   low(address)
        .byte   high(address)
        .byte   bank
```

TL/DD/10131–35

```
 .else
        .byte   0,0,0           ;temporary place holders
 .endif
 .endm   ;switch_to

 .macro  switch_assembly function, bank, address
        .public function
function:
        jsr     function_call_switcher
 .if @ > 1
        .byte   low(address)
        .byte   high(address)
        .byte   bank
 .else
        .byte   0,0,0           ;temporary place holders
 .endif
 .endm   ;switch_assembly

        .form
;Bank Switching Control Port
bank_switch_port= 0x00e3:b   ;must not touch low byte of Port B

;Values for Bank Switching Control Port
bank0   = 0x00
bank1   = 0x01
bank2   = 0x02
bank3   = 0x03
bank4   = 0x20
bank5   = 0x21
bank6   = 0x22
bank7   = 0x23
bank8   = 0x40
bank9   = 0x41
bank10  = 0x42
bank11  = 0x43
bank12  = 0x60
bank13  = 0x61
bank14  = 0x62
bank15  = 0x63

;Switch stack
        .sect   switch_stack, ram16, rel
        .dsw    SWITCH_STACK_DEPTH * 2
growth  = 4
        .endsect

;Switch stack pointer
        .sect   switch_pointer, base, rel
switch_stack_pointer:   .dsw    1
        .endsect

;Initialization value for switch stack pointer
        .sect   switch_init, rom8, rel
        .byte   low(e_sect(switch_stack))
        .byte   high(e_sect(switch_stack))
```

TL/DD/10131–36

```
        .endsect

;Initialization control for switch stack pointer
        .sect   _init_info_, rom16, rel
        .word   b_sect(switch_pointer)
        .word   e_sect(switch_pointer) -1
        .word   b_sect(switch_init)
        .endsect

        .sect   switch_code, rom8, rel
;Linkages
        .incld  banklink.inc

;Switch from caller's bank to destination bank, transparently
;All registers must be preserved
;
function_call_switcher:
        push    a               ;free up registers
        push    x
        add     switch_stack_pointer,#-growth ;get switch stack room
        ld      x,switch_stack_pointer
        ld      a,bank_switch_port ;put caller bank on switch stack
        x       a,[x+].w
        ld      a,-8[sp].w      ;put caller return on switch stack
        x       a,[x+].w
        ld      a,-6[sp].w      ;access destination information
        st      a,x
        ld      a,[x+].b        ;get destination address onto stack
        st      a,-6[sp].b
        ld      a,[x+].b        ;(as bytes because no alignment)
        st      a,-5[sp].b
        ld      a,[x+].b        ;put destination bank in port
        st      a,bank_switch_port
        ld      a,#function_call_returner ;put switcher return on stack
        st      a,-8[sp].w
        pop     x
        pop     a
        ret                     ;transfer to destination in new bank
;
;Return to caller's bank from destination bank, transparently
;All registers must be preserved
;
function_call_returner:
        push    a               ;space for return address
        push    a               ;free up register
        ld      a,[switch_stack_pointer].w ;restore caller bank
        st      a,bank_switch_port
        ld      a,2[switch_stack_pointer].w ;restore caller return
        st      a,-4[sp].w
        add     switch_stack_pointer,#growth ;give up switch stack room
        pop     a
        ret                     ;return to caller in original bank
;
        .endsect
        .end
```

TL/DD/10131–37

```
;       ********************************************************
;       *                                                      *
;       *       National Semiconductor MicroController Group   *
;       *                                                      *
;       *       Definitions of Inter-Bank Function Call Links  *
;       *       BANKLINK.INC -  Bank switch support functions  *
;       ********************************************************
;

;For every inter-bank link required, enter a defining line
;
;       switch_to       function, bank<n>, address
;
;where the function name is the name of the destination function,
;bank<n> is the name of the bank number (bank0, bank1, ...), and
;the address is a numeric constant for the address of the actual
;destination function code in its bank.
;Assembly language functions can be linked using
;
;       switch_assembly function, bank<n>, address
;
;instead.

        switch_to       initialize_table_memory, bank0, 0x4000
        switch_to       build_tables, bank0, 0x404b
        switch_to       capture_table, bank0, 0x404e
        switch_to       compute_coefficients, bank0, 0x40ab
        switch_to       monitor, bank1, 0x4000
        switch_to       error, bank0, 0x4136
        switch_to       fatal_error, bank0, 0x4159
```

TL/DD/10131–38

```
;       ********************************************************
;       *                                                      *
;       *       National Semiconductor MicroController Group   *
;       *                                                      *
;       *       Macros to Assist Bank Switching Linkages       *
;       *       BANKDEFS.INC -  Bankswitch support functions   *
;       ********************************************************

;For every inter-bank link into a module, substitute definitions
;are needed using the values of the inter-bank link in shared
;memory.  These macros make it easier.
;
;       link_address    function, address
;       link_assembly   function, address
;
;where function is the name of the linked function and address is the
;address of the link code in the shared bank.

 .macro  link_address    function, address
        .public _~function
_~function = address
 .endm

 .macro  link_assembly   function, address
        .public function
function = address
 .endm

;For forcing a library routine to be linked, even though not accessed.
;
;       force_library   routine, routine, routine, ...
;       force_assembly  routine, routine, routine, ...
;
;Multiple lines may be used.

 .macro  force_library   list
 .set $count, 0
 .do @
 .set $count, $count + 1
        .extrn  _~@$count
 .enddo
 .endm   ;force_library

 .macro  force_assembly  list
 .set $count, 0
 .do @
 .set $count, $count + 1
        .extrn  @$count
 .enddo
 .endm   ;force_assembly

;To create the dummy place holders for the initialization information
;sections.
```

TL/DD/10131–39

```
;
;       init_dummy      size, size, size, ...
;
;Multiple lines may be used.

 .macro  init_dummy      list
        .sect   _init_info_, rom16, rel
 .set $count, 0
 .do @
 .set $count, $count + 1
        .dsb    @$count
 .enddo
        .endsect
 .endm   ;init_dummy
```

TL/DD/10131–40

```
/*
        tables.c        Placed in Bank0.
*/

#include "tables.h"

extern int coefficients[10];
extern struct table_entry live;

#define table_memory        (* ((struct table_entry *) 0x6000))
#define table_memory_end    (* ((struct table_entry *) 0x8000))

static int table_entries, table_values;
static struct table_entry * first_table;

/* this initializes special RAM memory in the bank for tables */

NOLOCAL
initialize_table_memory()
{
    static struct table_entry * p;

    /* initialize memory as an array of structure */
    for( p = &table_memory, table_entries = 0;
        p < &table_memory_end;
        p++, table_entries++ )
    {
        p->spins = 0;
        p->rolls = 0;
        p->result = 0;
    }
    /* record initial state */
    first_table = &table_memory;
    table_values = 0;
}

/* builds a series of table entries in the RAM memory from inputs */

NOLOCAL
build_tables()
{
    /*
    ...
    decide when table is ready
    ...
    */
    capture_table();
}

NOLOCAL
capture_table()
{
    static struct table_entry * next;

    if( table_values < table_entries )
```

TL/DD/10131–42

```
    {
        /* table not full, locate next and add one */
        next = first_table + table_values;
    }
    else
    {
        /* table full, advance one as ring */
        next = first_table;
        if( ++first_table >= &table_memory_end )
        {
            first_table = &table_memory;
        }
    }
    *next = live;
}

/* data reduction on table */

NOLOCAL
compute_coefficients()
{
    static int i;
    static struct table_entry * p;

    for( i = 0, p = first_table; i < table_values; i++ )
    {
        /*
        ...
        code to do data reduction on available data
        ...
        */
            recursive_spin_reduction(p, 0);
        if( ++p >= &table_memory_end )
        {
            p = &table_memory;
        }
    }
}

/* reduction on each entry */

static
recursive_spin_reduction(entry, item)
struct table_entry * entry;
int item;
{
    /* ... */
    if( item < entry->spins )
    {
        recursive_spin_reduction(entry, item + 1);
        /* ... */
    }
    /* ... */
}
```

TL/DD/10131–43

```
/*
        errors.c        Placed in Bank0.
*/

static int error_count = 0;

NOLOCAL
error(code)
int code;
{
    printf("Error number %i - continuing\n", code);
    error_count++;
}

NOLOCAL
fatal_error(code)
int code;
{
    static int i;

    for( i = 0; i < 15; i++ )
    {
        putchar(0x07);
    }
    printf("\n\nFATAL ERROR number %i - ABORTING PROCESSING\n\n",
        code);
    quit();
}

NOLOCAL
quit()
{
    printf("Program terminated.  %i recoverable errors\n",
        error_count);
}
```

TL/DD/10131–44

```
/*
        monitor.c       Placed in Bank1.
*/

#include "tables.h"

extern struct table_entry live;

NOLOCAL
monitor()
{
    static int predictable;

    /*
    ...
    system monitoring
    ...
    */
    while( live.spins < 3
        || live.rolls < 5 ) ;
    while( !live.result )
    {
        compute_prediction();
    }
    validate_calculation();
    capture_table();
}

compute_prediction()
{
    int i;

    /*
    ...
    complex calculations to give a SWAG
    ...
    */
    printf("Prediction: %i\n", i);
}

validate_calculation()
{
    int i, j, k;

    /*
    ...
    match latest result to what we would predict
    ...
    */
    printf("Final prediction: %i, actual: %i, accuracy: %i\n", i, j, k);
    if( k < 10 )
    {
        error(1);
    }
}
```

TL/DD/10131–45

```
/*
        main.c          Placed in Shared.
        This is the main program for the example.
*/

/*operational mode flags */
int operational = 1,
    calibrating = 1,
    predicting  = 0;

/* controlling coefficient array */
int coefficients[10];

main()
{
    initialize_inputs();
    initialize_outputs();
    initialize_table_memory();
    while( operational )
    {
        while( calibrating )
        {
            build_tables();
        }
        compute_coefficients();
        while( predicting )
        {
            monitor();
        }
    }
}
```

TL/DD/10131–46

Lit. # 100577

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408