COPYING INITIALIZED DATA TO RAM

APPLICATION NOTE AN-132

By Ketan Deshpande and Sugavaneswaran Subramanian

INTRODUCTION

Writing ROM-able code using IDT/C™ 5.0 or IDT/C 6.0 puts restrictions on initialized data declarations. Initialized data end up in ROM space, making it impossible to change such data during program execution. This restriction is neither obvious, nor acceptable to a number of C programmers. One technique to eliminate this restriction is explained in this application note. The most effective implementation requires modification to the C compiler utilities, which may be offered in future releases of IDT/C.

OVERVIEW

IDT's C Compiler tool chains IDT/C 5.0 and 6.0 provide a means of developing embedded applications based on the IDT R30xx and R4x00 RISControllers™. IDT/C 5.0 generates ECOFF format files; IDT/C 6.0 generates ELF files. For purposes of this discussion, both output file formats will be referred to as "executable". Any differences in formats / tool chains will be noted wherever appropriate.

IDT/C organizes the executable into sections by default, as shown below:

1) .text: All instructions from all source files.
2) .rdata (ECOFF) / .rodata (ELF): All initialized data that are declared constant. (Most commonly found elements here are strings.)
3) .data: All initialized data. Data may get moved between *.rdata* and *.data* depending on what the compiler believes is constant.
4) *.bss*: All uninitialized data.
5) *.sdata*: All initialized data smaller than the size specified by the -G option.
6) *.sbss*: All uninitialized data smaller than the size specified by the -G option.

The layout of sections and determining what exactly goes into which section can be controlled using a linker script file, and by adding -T<script filename> in the linker command line.

Both IDT/C 5.0 and 6.0 allow creation of user-defined sections and embedding user-defined symbols in the executable generated, using the linker script. This flexibility is key to the technique discussed below.

PROBLEM

Initialized data in the .data section get programmed into ROM space when the PROMs are created. This is the only way that the code can "remember" the initial values of all initialized data, in an embedded environment. However, this makes it impossible for the user to modify these values. The user can get around this by not initializing the variables at the point of declaration (making them uninitialized and thus forcing them into the *.bss* section) and then initializing them in code. The drawback of this approach is that the user needs to remember where to initialize each such data structure. Another way would be to have two structures: one initialized, one uninitialized, and in the code, copy the one in *.data* to the one in *.bss*. This method has speed and space disadvantages.

This Application Note describes a three-step method to overcome this problem. Briefly, the logic can be explained as (a) build the code assuming that the *.data* section will be in the RAM space; (b) in reality, burn the *.data* section in the ROM; (c) right at the start of code execution, move the *.data* section from ROM to RAM where the code expects it to be already.

Using IDT/C, the steps would be:

1. Link the executable program in such a way that the instructions look for *.data* section in the RAM address area.
2. Build S-records using a modified version of objcopy that relocates the *.data* section to ROM area while converting the executable to S-record. This "saves" the initialized contents of the *.data* section.
3. Make the startup code copy this relocated section from ROM area to its designated place in RAM area. This is the RAM address area where the instructions will be looking for the *.data* section (as explained in step 1 above). This method has been tested and found to work with relocating *.data* from IDT/sim™; it can be extended easily to cover .rdata / .rodata too.

ADVANTAGES

1. Allows software programmers to use initialized data without restrictions.
2. Removes the necessity for additional code/data spread out all over the application for modifying initialized data.
3. Speeds up program execution, since accesses that used to go to ROM are now directed to RAM.

DISADVANTAGES

1. Increased startup time because of the code to copy the *.data* section to RAM. However, this is only a one-time effort, and hence is not a major overhead.

STEPS INVOLVED

1. Determine what section(s) of the executable are to be relocated.
2. Modify the linker script to add informational sections (for objcopy) and symbols (for the startup code) that define the source and target of relocation.
3. Modify the startup code to copy data from the relocated address (ROM) to its real address (RAM).
4. Compile and link the application using the new linker script, such that the .data section now lies in RAM.
5. Use the version of objcopy that has support for this relocation, to build PROMs.

SECTIONS TO BE RELOCATED

Let us assume that only the *.data* section needs to be relocated.

MODIFYING THE LINKER SCRIPT

Linker scripts for IDT/C 5.0 and 6.0 are slightly different; the modifications done are very similar.

The following information needs to be inserted into the linker script to enable both objcopy and the startup code to perform the relocation and data movement.

a) Sections *.start*, *.endsect*:

This is done by inserting section lines in the linker script.

The program "objcopy" relocates all sections between these two sections to the address defined by _src_start.

b) Symbols _src_start, _src_end:

This is done by inserting symbol lines in the linker script.

The startup code copies data from _src_start to _tgt_start, until _src_end is reached.

c) Symbol _tgt_start:

This is done by inserting a symbol line in the linker script.

The startup code copies all data that was relocated, to this RAM address.

The modified linker scripts are listed on the following pages, with the changes highlighted.

Sample Linker Script for IDT/C 5.0:

```
OUTPUT_FORMAT("ecoff-bigmips")
ENTRY(start)
SECTIONS
{
  .text  0xbfc00000 : {
     _ftext = . ;
    *(.init)
     eprol  =  .;
    *(.text)
    *(.fini)
     etext  =  .;
     _etext  =  .;
  }
  .rdata  . : {
    *(.rdata)
  }

/* Relocate the sections between .start and
  .endsect, to begin from the current ad-
  dress */
  .start . : {}
  _src_start = . ;
  _tgt_start = 0xa0000200 ;
/* _tgt_start should be equal to the start
  of the .data section below */

  .data  0xa0000200 : {
    _fdata = .;
    *(.data)
    CONSTRUCTORS
   edata  =  .;
    _edata  =  .;
  }

/* OK, this is all we wanted to relocate */
  .endsect . : {}
  _src_end = . ;

  .reginfo . : {}
  .scommon . : {}
  .bss  . : {
   _fbss = .;
    *(.bss)
    *(COMMON)
  }
   end = .;
   _end = .;
}
```

Sample Linker script file for IDT/C 6.0:

```
OUTPUT_FORMAT("elf32-bigmips")
OUTPUT_ARCH(mips)
_DYNAMIC_LINK = 0;
SECTIONS
{
  /* Read-only sections, merged into text
  segment: */
  .text 0xbfc00000  :
  {
    _ftext = . ;
    *(.text)
    CREATE_OBJECT_SYMBOLS
    _etext = .;
  }
  .init    ALIGN(8)  : { *(.init)    } =0
  .fini    ALIGN(8)  : { *(.fini)    } =0
  .ctors   ALIGN(8)  : { *(.ctors)   }
  .dtors   ALIGN(8)  : { *(.dtors)   }
  .rodata  ALIGN(8)  : { *(.rodata)  }
  .rodata1 ALIGN(8)  :
    {
      *(.rodata1)
      . = ALIGN(8);
    }
  .reginfo . : { *(.reginfo) }

/* Relocate the sections between .start and
  .endsect, to begin from the current ad-
  dress */
  .start . : {}
  _src_start = . ;
  _tgt_start = 0xa0000200 ;
/* _tgt_start should be equal to the start
  of the .data section below */

  .data  0xa0000200 :
  {
    _fdata = . ;
    *(.data)
```

```
    CONSTRUCTORS
  }
  .data1 ALIGN(8)  : { *(.data1) }
  _gp = . + 0x8000;
  .lit8 . : { *(.lit8) }
  .lit4 . : { *(.lit4) }
  .sdata   ALIGN(8)  : { *(.sdata) }
  _edata  =  .;

/* OK, this is all we wanted to relocate */
  .endsect . : {}
  _src_end = . ;

  __bss_start = . ;
  .sbss    ALIGN(8)  : { *(.sbss)
 *(.scommon) }
  .bss     . :
  {
  _fbss = .;
   *(.bss)
   *(COMMON)
   _end = . ;
   end = . ;
  }

  .line                0 : { *(.line)
 }
  .debug               0 : { *(.debug)
 }
  .debug_sfnames       0 : {
 *(.debug_sfnames)        }
  .debug_srcinfo       0 : {
 *(.debug_srcinfo)        }
  .debug_macinfo       0 : {
 *(.debug_macinfo)        }
  .debug_pubnames      0 : {
 *(.debug_pubnames)       }
  .debug_aranges       0 : {
 *(.debug_aranges)        }
}
```

Modifying the startup code

Typically, embedded applications have code that performs CPU control register initialization, cache flushing, memory sizing, initializing .bss etc. With the *.data* section in its new positions in ROM, the code will still look to RAM addresses for initialized data. Before any such references are attempted, the .data section should be copied out into it's real place. A good place to do this is usually after *.bss* initialization. The code segment below demonstrates how this can be done. The same code can be used for IDTC/5.0 and 6.0; though for the R4x00 processors, the user may want to use double-word loads and stores for faster execution.

```
      la  t0, _src_start
      la  t1, _tgt_start
      la  t2, _src_end
2:    lw  t3, 0(t0)
      nop
      sw    t3, 0(t1)
      addu  t0, 4
      addu  t1, 4
      blt   t1, t2, 2b
      nop
```

Modification to OBJCOPY

The binary utility "objcopy" needs to be modified to make it intelligent enough to recognize the sections that the linker script was asked to create, and to move the appropriate sections to their temporary PROM addresses. Most of the code modifications needed to perform this movement are in the function setup_section() in the file objcopy.c (the main source code file for the objcopy utility), and are shown on the next page, in boldface. Some adjacent code is shown for reference. Initialization of the variables may not be shown explicitly; it is mentioned wherever appropriate.

```
  setup_section(......)
  {
  ...../* Original variable declarations here
*/
    int sec_addr;
    static int new_data_addr = 0;
    static int move_section = FALSE;
  ......
  ....../* Original code here */
    if (!bfd_set_section_size (obfd,
                  osection,
                    bfd_section_size (ibfd,
isection)))
      {
        err = "size";
        goto loser;
      }

   /* start_address = bfd_get_start_address
(ibfd);
       in copy_object() */
      if  (!new_data_addr)  new_data_addr  =
start_address;
   new_data_addr += bfd_section_size (ibfd,
isection);

  /* Got section .start? Now remember current
address
     and keep track of new relocation address
*/
     if  (!strcmp(bfd_get_section_name(ibfd,
isection),
        ".start"))
      move_section = TRUE;
   /* Got section .endsect? Stop relocation
*/
  else if (!strcmp(bfd_get_section_name(ibfd,
```

```
              isection), ".endsect"))
            move_section = FALSE;

   if (move_section) sec_addr = new_data_addr;
     else sec_addr = bfd_section_vma (ibfd,
isection);

    /* Actually do the relocation */
    if (bfd_set_section_vma (obfd, osection,
sec_addr)
        == false)
      {
        err = "vma";
        goto loser;
      }

    if (bfd_set_section_alignment (obfd,
                   osection,
                       bfd_section_alignment
(ibfd, isection))
        == false)
      {
        err = "alignment";
        goto loser;
      }

  ......
```

Compile, link the application and build PROMs

This can be done in the usual manner. The scripts shown above setup the *.data* section to reside in RAM area. The new version of objcopy with this option may be available in future releases of IDT/C.

SUMMARY

This application note described a technique that relocated certain sections to ROM and then copied them to their designated locations in RAM. This method has been demonstrated on the *.data* section; it can very easily be extended to include other sections too.

The advantages are: provide C programmers with the ability to use initialized variables much more freely, removal of the need for extra code or data, faster access without requiring any extra ROM space.

IMPORTANT NOTICE AND DISCLAIMER

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

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.