TriCore Application Note

Setting Locator Options

Document ID: AN060-01
Date: 2002-03-25

A L T I U M B V

1 Introduction

This application note shows how the locator controls influence the locating process. This note is intended both for EDE and command line tools users.

After reading this application note the user should be able to configure the locating process and solve errors that result from the chosen settings and controls.

2 Project and Locator

EDE users:
- your project is already setup to handle the locator options as described in this application note.

Command line tools users:
- copy `c:\ctri\etc\tri.i` to your project directory and rename it to `<projectname>.i`, adapt this file to your specific memory configuration.

DO NOT MODIFY `tri.dsc`, `tri.def` and `tri.cpu`, since these files describe the TriCore architecture, which you do not change in your project.

3 Setting linker/locator options

3.1 Standard linker/locator settings

Most of the linker/locator options speak for themselves and most options have a default value that can be used. However sometimes a programmer wishes to locate a part of his program at a pre-defined location in memory. All necessary options can be defined in EDE or in the `<projectname>.i` file (command line tools users). Some option fields only require a number, others need a special syntax. See the list below which describes these options and how to use them (split up for EDE and command line tools users).

NOTE: specify all addresses in hexadecimal form (always prefixed by '0x') like: 0xa0000050.

EDE users: The linker/locator options dialog is opened by selecting: *Project -> Linker/Locator options*.

Locator tab:

Field name	data to enter in the field:
- RESET start address	<hex address>
- Interrupt table start address	<hex address>
- CrossView Pro buffer size	<size in bytes>
- CrossView Pro buffer start address	<hex address>
- *Additional options	options as listed in the Assembler Linker Utilities manual (chapt. 11.2)

* Strings used in the options need special formatting! See the Assembler Linker Utilities manual (chapt. 11).

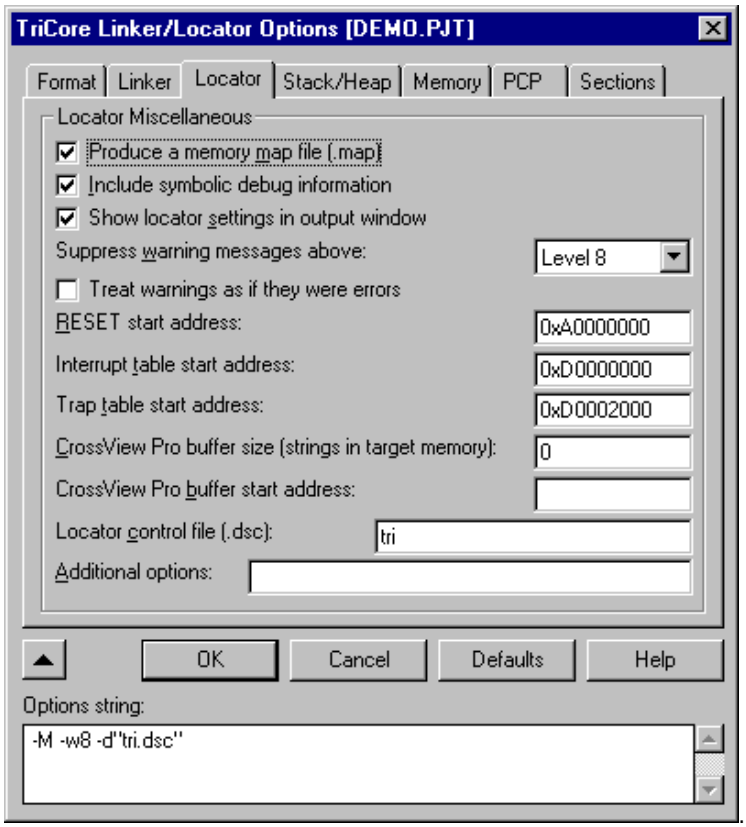

Fig.1: Linker/locator options, Locator tab.

Stack/Heap tab:

Field name	data to enter in the field
- User stack size	<size in bytes>
- User stack start address	<hex address>
- Interrupt stack size	<size in bytes>
- Interrupt stack start address	<hex address>
- Number of context blocks	<number>
- Context start address	<hex address>
- Heap size	<size in bytes>
- Heap start address	<hex address>

note 1): size may be given with suffix ‘k’, e.g. 1k for 1024.

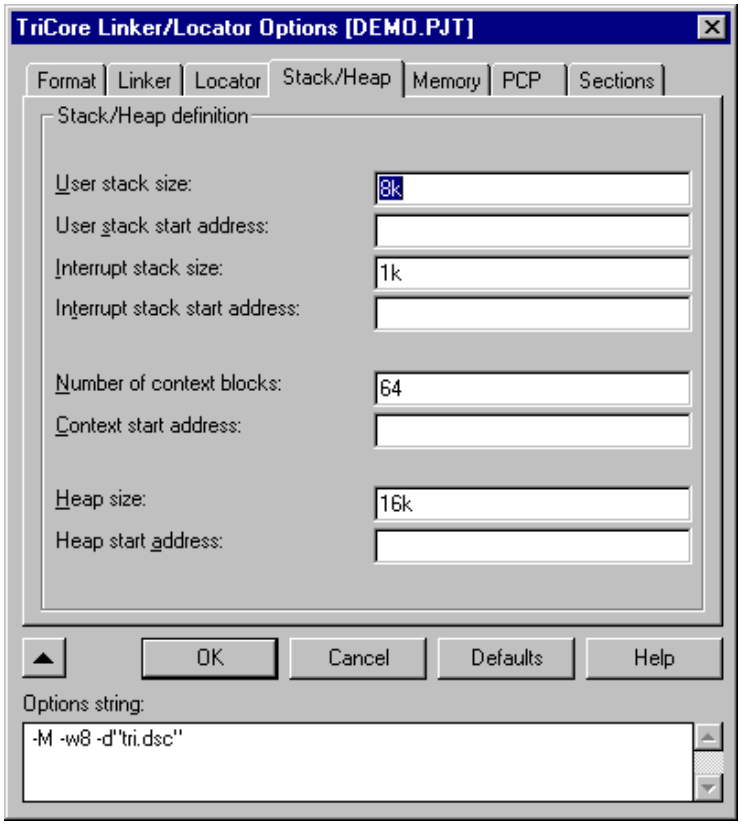

Fig. 2: Linker/locator options, Stack/Heap tab.

Memory tab:

Field name	data to enter in the field
- Available RAM memory	<memory region name>, <start addr>, <end addr> example: ext_d, 0xB0000000, 0xB007FFFF regions have to be separated using ';'
- Available ROM memory	<memory region name>, <start addr>, <end addr> regions have to be separated using ';'

On this tab you define which memory area's are available for the locator to be filled with code (ROM memory) and data (RAM memory).

PCP tab:

Field name	data to enter in the field
- PCP RAM memory	pcpram, <start addr>, <end addr> example: pcpram, 0xB0000000, 0xB007FFFF
- PCP ROM memory	pcprom, <start addr>, <end addr>

Sections tab:

Field name	data to enter in the field
- Reserved areas	<start addr>, <end addr> example: 0xB0005000, 0xB0005FFF these areas will be skipped by the locator.

Command line tools users:

Command line tools users must change the settings in their `<projectname>.i` file (a copy of the `tri.i` file). This file contains several #defines, memory area qualifiers (ROM/RAM/PCPROM/PCPRAM) and section order qualifiers (ROM_LINEAR etc.).
In this file, all the options decribed above for EDE users, can be manually adapted.

- RESET start address	#define RESET <addr>
- Interrupt table start address	#define INTTAB <addr>
- CrossView Pro buffer size	#define XVWBUF <size in bytes> addr=<address> example 32 addr=0xb000020
- User stack size	#define USTACK <size in bytes> addr=<address>
- Interrupt stack size	#define ISTACK <size in bytes> addr=<address>
- Number of context blocks	#define CSA <size> addr=<address>
- Heap size	#define HEAP <size in bytes> addr=<address> example: 8k addr=0xb0008000
- Available RAM memory	RAM(<memory region name>, <start addr>, <end addr>) example: RAM(ext_d, 0xB0000000, 0xB007FFFF)
- Available ROM memory	ROM(<memory region name>, <start addr>, <end addr>)
- PCP RAM memory	PCPRAM(pcpram, <start addr>, <end addr>)
- PCP ROM memory	PCPROM(pcprom, <start addr>, <end addr>)
- Reserved areas	RESERVED(<start addr>, <end addr>) example: RESERVED(0xB0005000, 0xB0005FFF) these areas will be skipped by the locator.

These options are quite simple, so nothing should go wrong. The only option not described yet, is more advanced and will be discussed in the next paragraph.

3.2 Advanced linker/locator settings

The most advanced option that can be used is the option that enables you to specify in which order and at which addresses you want segments to be located.

To be able to use this option, you should know globally how the compiling/locating process works.
- All compiled code is put into sections (pieces of code with names).[2)]
- Sections can have restrictions; for example: size and location.

- The locator examines all sections and their restrictions and locates the code according to those restrictions. Thus, if two sections have a size restriction, the locator can place them anywhere in memory, as long as they fit in the specified area. If you want to be sure that one section is placed before the other, an additional location restriction must be given.
- The locator is free to locate sections when all restrictions are met!

Note 2) For instance all code from file `test.c` is placed in section ***code.tes***t, unless the ***#pragma section*** is used somewhere in this file. In assembly ***.sdecl*** and ***.sect*** are used to respectively define and use sections. If you want to place a part of the C-code in another section from within C, use:

```
….. // default sections created by the compiler

#pragma section code=”my_section” [data=”my_data_section”]
…. Some C code, placed in section my_section,
    data declared here will be placed in data section my_data_section
#pragma section

….. // return to default sections
```

To set the section restrictions for each section EDE users must select the ‘Sections’ tab through the *Project -> Linker/Locator* dialog.
In the lower part of the window, (‘Define section order / address’), you can specify the order in which certain sections must be located and/or bind sections to specified addresses. Four fields are available (see fig 3): code linear, code abs24, data linear, data abs18. (The abs24 and abs18 attributes specify that sections placed in these fields will be placed in direct addressable code or data memory.)

For example: consider a project consisting of three files `test1.c`, `test2.c` and `test3.c`. You want `test2.c` to be located before `test1.c`. Also `test2.c` should be located at address 0xC0000010. The locator may locate `test3.c` anywere in the available code memory.
The field ‘code linear’ must be filled with: **code.test2 addr=0xC0000010; code.test1**

Command line tools users can set the section restrictions for each section by adapting their `<projectname>.i` file. The specification of the section ordering can be done with: ROM_LINEAR, ROM_ABS24, etc.

Using the above given example, this means adding to your `<projectname>.i` file:

```
ROM_LINEAR(code.test1 addr=0xC0000200)
ROM_LINEAR(code.test2)
```

NOTE: if an address is specified for a group of sections, the first section must be assigned to the address, otherwise unexpected results will occur! (for both EDE and command line tools users!).

For example:

```
ROM_LINEAR(code.test1)
ROM_LINEAR(code.test2)
```

The locator will place **code.test2** after section **code.test1** somewhere in code memory (to be determined by the locator)

Further:

```
ROM_LINEAR(code.test1 addr=0xC0000200)
ROM_LINEAR(code.test2)
```

Will locate section **code.test1** at address 0xC0000200 and section **code.test2** will be placed somewhere after section **code.test1**, however not necessarily consecutive in memory! The locator may choose to put other sections between **code.test1** and **code.test2**.

However:

```
ROM_LINEAR(code.test2)
ROM_LINEAR(code.test1 addr=0xC0000200)
```

will give unexpected results for section **code2.test**.

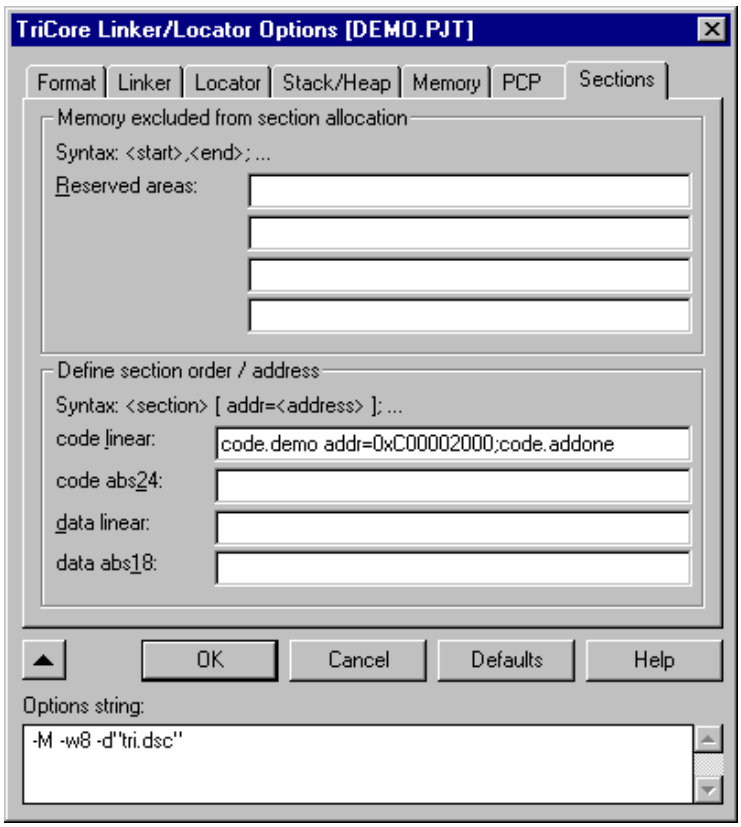

Fig 3. Linker/locator options, 'Sections' tab.

3.3 Copying code from ROM to RAM

If you want to run a code section in RAM memory (for instance because it is faster), the code section needs to be copied from ROM to RAM by the startup code. This can be achieved by declaring this code section (`code.test`) with the attributes initialize(**i**), writable(**w**) and not read-only (**-r**).

The EDE-user can do this by putting the string "**code.test attr=iw-r**" in the 'code linear'-field on the 'Sections' tab of the *Project->linker/locator* dialog.

The commandline tools user can do this by adding the following line to the `<projectname>.i` file:

```
ROM_LINEAR(code.test attr=iw-r)
```