SEGGER Linker

A linker for Arm and
RISC-V microcontrollers

User Guide & Reference Manual

Document: UM20005
Software Version: 4.18.0
Revision: 0
Date: November 18, 2021

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

Specifications written in this document are believed to be accurate, but are not guaranteed to be entirely free of error. The information in this manual is subject to change for functional or performance improvements without notice. Please make sure your manual is the latest edition. While the information herein is assumed to be accurate, SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions. SEGGER makes and you receive no warranties or conditions, express, implied, statutory or in any communication with you. SEGGER specifically disclaims any implied warranty of merchantability or fitness for a particular purpose.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel.	+49 2173-99312-0
Fax.	+49 2173-99312-28
E-mail:	support@segger.com*
Internet:	www.segger.com

*By sending us an email your (personal) data will automatically be processed. For further information please refer to our privacy policy which is available at https://www.segger.com/legal/privacy-policy/.

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please report it to us and we will try to assist you as soon as possible.

Contact us for further information on topics or functions that are not yet documented.

Print date: November 18, 2021

Software	Date	By	Description
4.18.0	211118	PC	Chapter "Command-line options" • Added **-I** and **--include**. • Revised **-D** and **--define-macro**. Chapter "Linker script reference" • Added "Preprocessor" section.
4.16.1	211021	PC	Updated to latest software version.
4.16.0	210924	PC	Updated to latest software version.
4.14.0	210824	PC	Updated to latest software version.
4.12.1	210728	PC	Updated to latest software version.
4.12.0	210720	PC	Updated to latest software version.
4.10.1	210324	PC	Chapter "Command-line options" • Added architecture **8.1-M.main** and **8.1-M.base**. Chapter "Linker script reference" • Added "**define access** statement".
4.10.0	210301	PC	Chapter "Command-line options" • Added **--keep-section**. • Added **--keep-init-array**. Chapter "Linker script reference" • Added "**define access** statement".
4.8.0	210115	PC	Chapter "Command-line options" • Added **--huawei-extension** and **--no-huawei-extension**. • Added **--andes-performance-extension** and **--no-andes-performance-extension**.
4.6.0	201217	PC	Chapter "Command-line options" • Added **--map-unused-inputs** and **--no-map-unused-inputs**. • Added **--map-unused-memory** and **--no-map-unused-memory**. • Added **--instruction-tables** and **--no-instruction-tables**.
4.4.1	201211	PC	Chapter "Command-line options" • Added **--outline-strand-size**. Chapter "Linker script reference" • Added **option** statement.
4.4.0	201208	PC	Chapter "Command-line options" • Added **--outline** and **--no-outline**. • Added **--tail-merge** and **--no-tail-merge**.
4.2.0	201201	PC	Chapter "Command-line options" • Added **--merge-sections** and **--no-merge-sections**.
4.0.1	201126	PC	Chapter "Command-line options" • Added **--warn-arch-mismatch** and **--no-warn-arch-mismatch**.
4.0.0	201123	PC	Chapter "Command-line options" • Added RISC-V architectures to **--cpu** option. • Added **--tp-model**. • Added **--relax** and **--no-relax**. • Added **--springboard** and **--no-springboard**. • Added **--map-listing-use-c-prefix** and **--no-map-listing-use-c-prefix**. • Added **--map-listing-use-abi-names** and **--no-map-listing-use-abi-names**.
3.20.5	201023	PC	Updated to latest software version.
3.20.4	201015	PC	Chapter "Command-line options" • Added **--warn-deprecated** and **--no-warn-deprecated**.
3.20.2	201012	PC	Chapter "Linker script reference" • Added **assert** statement.
3.20.0	201006	PC	Chapter "Linker script reference" • Added **access then size order** sorting. Chapter "Command-line options". • Added **--warn-all-double**, **--warn-unintended-double**, and **--no-warn-double**.

Software	Date	By	Description
			• Added **--lazy-load-archives** and **--no-lazy-load-archives**. • Added **--optimize-exception-table** and **--no-optimize-exception-table**. • Added **--pretty-symbol-names** and **--no-pretty-symbol-names**. • Added **--pretty-section-names** and **--no-pretty-section-names**. • Added **--map-addr-format**. • Added **--map-size-format**. • Added **--map-wrap** and **--no-map-wrap**. • Added **--map-exception-table** and **--no-map-exception-table**. • Added **--map-listing-xref** and **--no-map-listing-xref**. • Added **--little-endian** and **--big-endian**. • Option **--map-full** documents all sections enabled.
3.12.0	200911	PC	Chapter "Command-line options". • Changed map sections selected by each verbosity level. • Added **--map-listing-with-comments** and **--no-map-listing-with-comments**. • Added **--map-listing-use-adr-pseudo** and **--no-map-listing-use-adr-pseudo**. • Added **--map-listing-use-ldr-pseudo** and **--no-map-listing-use-ldr-pseudo**.
3.04	200820	PC	Chapter "Command-line options". • Added overview of map file options. • Added **--map-section-detail** and **--no-map-section-detail**. • Added **--map-module-detail** and **--no-map-module-detail**. • Added **--map-compact** and **--map-detailed**. • Prefer and accept **--map-full** as a synonym for **--map-all**. • Prefer and accept **--map-standard** as a synonym for **--map-defaults**. • Added **XXH32** integrity check algorithm.
3.02	200814	PC	Chapter "Using the linker". • Added "Conditional region selection". • Added **Adler-32** integrity check algorithm. • Changed "Initialization by symbol" to "Initialization by call".
3.00	200707	PC	Chapter "Using the linker". • Added "Firmware integrity checks". Chapter "Command-line options". • Added **--map-veneers** and **--no-map-veneers**. • Added **--map-listing-data** and **--no-map-listing-data**. • Added **--map-unused** and **--no-map-unused**. • Added **--map-text** and **--map-html**. • Added **--min-align-ram** and **--min-align-rom**. • Added **--pad-none** and **--pad-all**. • Added **--pad-code** and **--pad-data**. • Added **--pad-ram** and **--pad-rom**. • Added **--pad-ro**, **--pad-rw**, **--pad-rx**, and **--pad-zi**. • Added **cortex-a8** and **cortex-a9** CPU options. • Added **7-A** architecture option. • Changed default, now **--entry=Reset_Handler**. Chapter "Linker script reference" • Added "**fill** statement". • Added "Initialization by symbol".
2.34b	200420	PC	Chapter "Command-line options". • Option **--script** can be used more than once.
2.34a	191014	PC	Updated to latest software version.
2.34	190930	PC	Chapter "Command-line options" • Added **--wrap**. • Added **--weak-undefined-is-zero** and **--no-weak-undefined-is-zero**. Chapter "Linker script reference" • Added "Section placement selection". • Added **first symbol** and **last symbol** preference.
2.32	190410	PC	Chapter "Linker script reference" • Added **first section** and **last section** preference.
2.30a	190322	PC	Updated to latest software version.
2.30	190108	PC	Chapter "Command-line options" • Added **cortex-m23** and **cortex-m33** CPU options. • Added **8-M.base** and **8-M.main** architecture options. Chapter "Linker script reference" • Added **default region** statement. Chapter "Using the linker" • Added "Default named regions".

Software	Date	By	Description
2.26	181116	PC	Chapter "Command-line options" • Added **--dedupe-code** and **--no-dedupe-code**. • Added **--dedupe-data** and **--no-dedupe-data**. • Added **--inline** and **--no-inline**. • Added **--merge-strings** and **--no-merge-strings**.
2.24	180822	PC	Chapter "Linker script reference" • Added "Selecting sections". Chapter "Command-line options" • Added **--warn-empty-selects** and **--no-warn-empty-selects**.
2.22	180720	PC	Chapter "Linker script reference" • Added **maximum packing** sorting. • Added **minimum size order** sorting. Chapter "Command-line options" • Added **--unwind-tables** and **--no-unwind-tables**. • Added **--enable-lzss** and **--disable-lzss**. • Added **--enable-packbits** and **--disable-packbits**. • Added **--enable-zpak** and **--disable-zpak**. • Added **--min-align-code**. • Added **--min-align-data**. • Added **--min-align-rx**. • Added **--min-align-ro**. • Added **--min-align-rw**. • Added **--min-align-zi**. • Deprecated **--full-program-headers**. • Deprecated **--load-program-headers**.
2.20	180717	PC	Chapter "Linker script reference" • Added **size then alignment order** sorting. • Added **alignment then size order** sorting.
2.18	180705	PC	Chapter "Command-line options" • Added **--auto-es-block-symbols**. • Added **--auto-es-region-symbols**.
2.16	180502	PC	Upgraded to latest software version.
2.14	180411	PC	Chapter "Linker script reference" • **place at** now offers extended selection. Chapter "Command-line options" • **--auto-es-symbols** defines additional symbols.
2.12	180216	PC	Upgraded to latest software version.
2.10	180107	PC	Initial release.
1.00	170909	PC	Internal release.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Richie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual describes all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
`User Input`	Text entered at the keyboard by a user in a session transcript.
`Secret Input`	Text entered at the keyboard by a user, but not echoed (e.g. password entry), in a session transcript.
Reference	Reference to chapters, sections, tables and figures or other documents.
Emphasis	Very important sections.
SEGGER home page	A hyperlink to an external document or web site.

Table of contents

Chapter 1

About the linker

1.1 Introduction

This section presents an overview of the SEGGER Linker and its capabilities.

1.1.1 What is the SEGGER Linker?

The SEGGER Linker is a fast linker that links applications for execution on Cortex microcontrollers. It is designed to be very flexible, yet simple to use.

The linker combines the one or more ELF object files and supporting ELF object libraries to produce an executable image. This image is suitable for programming a Cortex microcontroller.

1.1.2 Linker features

The SEGGER Linker has the following features:

- Highly efficient and very fast to link.
- Flow code and data over multiple memory areas.
- Place a function or data at a specific address with ease.
- Avoid placing code and data in "keep out areas."
- Sort code and data sections for improved packing or by user preference.
- Automatically generate runtime initialization code prior to entering **`main()`**.
- Easily places code in RAM with optional flash-image compression.
- Will copy initialized data with optional flash-image compression.
- Eliminates all unused code and data for a minimum-size image.
- Accepts standard Arm and RISC-V ELF object files and libraries from any toolset.
- Generates range extension veneers as required for branch and branch-and-link instructions.
- Writes a map file that is clean and easily understood.
- Optionally generates a log file that provides additional information about the linking process.

1.1.3 Linker inputs

The SEGGER linker accepts one or more Arm or RISC-V ELF object files generated by standard-conforming toolchains such as SEGGER Embedded Studio.

In addition, the following files can be used as input to the linker:

- Arm or RISC-V object libraries created by a librarian.
- A linker control file to control placement of sections and how linking proceeds.
- An indirect file that extends command line arguments.

1.1.4 Linker outputs

The linker generates the following outputs:

- A fully-linked Arm or RISC-V ELF executable.
- An optional map file containing symbol addresses and related information.
- An optional log file providing additional information about the linking process.

Chapter 2

Using the linker

This section describes how to use the linker to link your Cortex application.

2.1 Memories

The SEGGER linker is capable of linking an application for any Cortex device. However, it has no internal knowledge of the way that memory is laid out for any device and must be told, using a linker script or command line options, the specific memory layout of the target device.

The **`define memory`** statement defines the single memory space used by Cortex devices and its size.

2.2 Regions

The **define region** statement defines a region in the available memory into which sections of code and data can be placed. The **define region** statement is much the same as a C preprocessor **#define** directive, it merely names a region of memory.

A memory region consists of one or more memory ranges, where a memory range is a contiguous sequence of bytes. Memory regions are central to the way that the linker allocates sections, and groups of sections, into the available memory.

2.2.1 Memory ranges

A memory range is specified in one of two forms:

- a *base address* and a *size*, or
- a *start address* and an *end address*.

The way that you specify a memory region does not matter, and the form that you choose is the one that naturally fits your view of the memory region.

2.2.1.1 Base address and size

The memory range specified by a base and a size has the following syntax:

[from *addr* **size** *expr* **]**

This declares a memory range that starts at the base address *addr* and extends for *expr* bytes. A typical memory range for a Cortem-M device declared in this fashion might be:

```
[from 0x20000000 size 128k]
```

Note that it is possible for the size to be zero, in which case the memory range is considered "null" and will not have anything allocated into it.

2.2.1.2 Start address and end address

The memory range specified by a start address and an end address has the following syntax:

[from *start-addr* **to** *end-addr* **]**

This declares a memory range that starts at the address *start-addr* and extends up to and including the address *end-addr*. A typical memory range for a Cortem-M device declared in this fashion might be:

```
[from 0x20000000 to 0x2001ffff]
```

2.2.1.3 Repeated ranges

It is possible to define a repeated range using **repeat** and **displacement** in the memory range:

repeat *count* [**displacement** *offset*]

If the displacement is absent it defaults to the size of the base range.

For instance, the range:

```
[from 0x20000000 size 32k repeat 4 displacement 64k]
```

results in a repeated range equivalent to the following:

```
[from 0x20000000 size 0x8000] +
[from 0x20010000 size 0x8000] +
[from 0x20020000 size 0x8000] +
[from 0x20030000 size 0x8000]
```


2.2.2 Naming a region

Regions are used to place data and code into memory, which is described later. However, it's convenient to name regions that correspond to the use they have. The **define region** statement defines a name for a region so that the region can be referred to using that name, for example:

```
define region FLASH = [0x00000000 size 2m];
define region RAM   = [0x20000000 size 192k];
```

2.2.3 Combining regions

The linker can combine ranges and regions in different ways. Every memory range is a valid memory region. The syntax for a region expression is:

(*region-expr* **)**
region-expr **+** *region-expr*
region-expr **-** *region-expr*
region-expr **&** *region-expr*
region-expr **|** *region-expr*
if *expr* **then** *region-expr* **else** *region-expr*

2.2.3.1 Region union

Two regions can be combined by using the **+** or **|** operators. In this case the result is the union of the two regions, for instance:

```
[from 0x1fff0000 size 32k] + [from 0x20000000 size 192k]
```

The resulting region is the combination of the two ranges, and as such there is a hole of 32k in the region, at `0x1fff8000` of size 32k.

It is possible to combine two contiguous ranges into a single region:

```
[from 0x1fff0000 size 64k] + [from 0x20000000 size 192k]
```

In this case the resulting region defines a contiguous range of addresses from `0x1fff000` of size 256k, but the region remains divided into two ranges.

> **Note**
>
> The fact that these two remain divided is very important for some devices where two memories are contiguous and, from a programmer perspective, would appear to be a single linear address space to allocate data into, *but from a hardware perspective are discrete*. It is common for a microcontroller to fault when reading misaligned data that spans the boundary between the two memories or when reading double-word data that spans the same boundary.

The Kinetis devices, which have such a boundary, are affected by this. Because the linker knows that there are two distinct ranges, it ensures that objects are allocated completely within each range and not over the seemingly contiguous memory area thus avoiding any latent bugs with memory accesses.

When two ranges overlap, they are combined into a single range. Therefore:

```
[from 0x1fff0000 size 64k+1] + [from 0x20000000 size 192k]
```

produces a region with a single range:

```
[from 0x1fff0000 size 256k]
```


2.2.3.2 Region subtraction

Two regions can be subtracted by using the - operator. In this case the result is generally two ranges, for instance you can punch a hole in a range by subtracting from it:

```
[from 0x1fff0000 size 256k] - [from 0x1fff8000 size 32k]
```

The resulting region is broken into two ranges:

```
[from 0x1fff0000 size 32k] + [from 0x20000000 size 192k]
```

This operator is particularly effective when you need to reserve places in flash or RAM for configuration or personalisation data or for bootloaders and other items.

Example

```
define region FLASH      = [from 0x00000000 size  2m];
define region BOOTLOADER = [from 2m-128k    size 64k];
define region CONFIG     = [from 2m-32k     size 32k];
define region APP        = FLASH - BOOTLOADER - CONFIG;
```

In this case the region APP is:

```
[0x00000000 to 0x001dffff] + [0x001f0000 to 0x001f7fff]
```

2.2.3.3 Region intersection

The intersection of two regions is calculated by the **&** operator. For instance:

```
[from 0x20000000 size 256k] & [from 0x1fff8000 size 96k]
```

results in the region:

```
[from 0x20000000 size 64k]
```

2.2.3.4 Conditional region selection

A region can be selected depending on the value of an expression. The expression is typically a symbol that can be defined on the command line.

For instance:

```
if FAST_RAM then [from 0x1fff8000 size 32k] else [from 0x20000000 size 96k]
```

results in the region…

```
[from 0x1fff8000 size 32k]
```

if `FAST_RAM` is nonzero and…

```
[from 0x20000000 size 96k]
```

if `FAST_RAM` is zero.

2.2.4 Default named regions

A default named region region is rather like a defined region except that any command ine option that specifies an identical region name will override the default region's definition. This allows you to define a generic linker script applicable to a generic Cortex-M device

and override those regions on the command line for a specific device with different region sizes or placements.

For instance:

```
default region FLASH = [0x00000000 size 2m];
default region RAM   = [0x20000000 size 192k];
```

The default region is only used when there is no other definition of the region either on the command line or previously within the linker script using **`define region`**.

2.3 A simple linker script

The linker places code and data into regions according to the user's linker script. Each required section required from the input files is mapped to a specific region using **place in** and, optionally, **place at** statements.

The specific layout of code and data will depend upon the target device. A simple linker script would define the memory required for code and data and map the appropriate sections to RAM and flash, for instance:

```
define memory with size = 4g;  ❶

define region FLASH = [from 0x00000000 size 2m];  ❷
define region RAM   = [from 0x1fff0000 size 64k] +  ❸
                      [from 0x20000000 size 192k];

place at start of FLASH { section .vectors };  ❹
place in FLASH          { readonly, readexec };  ❺
place in RAM            { readwrite, zeroinit };  ❻

keep { section .vectors };  ❼
```

❶ Define memory

The **define memory** statement covers the entire Cortex-M address space.

❷ Define flash region

This **define region** statement declares that the first two megabytes of memory are for flash. In fact, the linker does not distinguish between read-only flash and read-write RAM, all it is concerned with is mapping the user's sections from relocatable object files into the regions defined to hold them: the fact we have called this region **FLASH** is for our convenience only.

❸ Define RAM region

This **define region** statement declares that there are two ranges that are allocatable as RAM. The linker will ensure that individual sections placed into this region will be entirely contained in one of the ranges and will not span between the range.

❹ Place vectors

This **place at** statement instructs the linker to place the **.vectors** section at the start of flash memory. The vectors section typically contains the reset program counter and initial stack pointer along with interrupt and exception vectors for the target device and must, by convention, be placed at the start of memory.

❺ Place read-only code and data

This **place in** statement instructs the linker to place all read-only and read-execute sections into the FLASH region. Vectors have already been placed at the start of region, so the linker will place additional input sections into the remaining flash.

❻ Place read-write data

This **place in** statement instructs the linker to place all read-write and zero-initialized data into RAM.

❼ Keep vectors

This **keep** statement instructs the linker to keep the vectors section. The linker eliminates all code and data that is not specifically referenced by the linked application, so dead code and data are removed from the linked application and do not form part of the image. As there is usually no reference to the vectors section by the application, it would be eliminated by

the linker in normal operation. The **keep** statement instructs the linker to keep the vectors section, and anything it references, in the application.

You can use the **keep** statement to keep otherwise-unreferenced data in your application, such as configuration data or personalization data.

2.4 Selecting sections

Sections are selected by name, by symbol, or by attribute.

2.4.1 Selection by symbol

Every symbol is associated with exactly one section. You can select the section associated with a symbol using the **section** selector in any statement that uses a section selector (e.g. **place in**):

```
place in FLASH { symbol MyFunc };
```

2.4.2 Selection by section name

Every section has a section name which is distinct from the symbol name. You can select sections using section names, for instance:

```
place in RAM { section .bss };
```

This selects all sections named **.bss**, and there may well be more than one input section names **.bss**, and this selects all of them.

Selection can also be by wildcard in the section name, for instance:

```
place in RAM { section .bss, section .bss.* };
```

This selects all sections that are named **.bss** and all sections that match the wildcard **.bss.*** where * stands for a sequence of zero or more characters. These two selectors will never select a section **.bss2** because **.bss2** does not match the section name and does not match the wildcard.

The wildcard character **?** matches exactly one character, so the wildcard **.bss.?*** will match all sections that start **.bss.** and have a non-empty suffix. In this case, **.bss.Data** will be selected but **.bss.** will not be selected because there is nothing following the final period.

It is very common to select sections that are related, for instance:

```
place in FLASH { section .text, section .text.* };
```

This particular construction can be written using the ^ matcher, e.g. **.text^.***. The ^ matches the end of string or continues with matching the suffix. Hence, **.text^a** will match only the two sections with names **.text** or **.texta**.

This construction is particularly valuable when using the empty selector warning (see *--warn-empty-selects* on page 248). In this case, if the input contains no **.text** sections, the section selector will cause a warning to be issued even if **.text.*** sections are present. This can be rewritten to use a wildcard selector that will select any **.text** section and any **.text.*** section using a single wildcard covering both:

```
place in FLASH { section .text^.* };
```

This construction will not cause the linker to issue an empty selector warning (if that warning is enabled).

2.5 Grouping code and data

Although the previous script works, it may not produce the most compact output as initialize data (selected by **readwrite**) and initialized data (selected by **zeroinit**) can be interleaved in order to reduce the amount of padding required to align each section and, in doing so, make the linker initialization tables larger.

It is better to group all zero-initialized data together and all read-write initialized data together so that each produce a single entry in the initialization table rather than many entries.

2.5.1 Creating groups

You can collect all such data by using a block:

```
define block rwdata { readwrite };
define block zidata { zeroinit };
```

And then place these blocks into RAM:

```
place in RAM { block rwdata, block zidata };
```

Blocks can also be used to reserve memory or fix the order of sections when allocating them to memory.

2.5.2 Inline and nested blocks

Rather than define two blocks separately, it’s possible to declare blocks inline, so the above can be written as:

```
place in RAM {
  block rwdata { readwrite },
  block zidata { zeroinit  }
};
```

As the block names are now redundant, they can be omitted:

```
place in RAM {
  block { readwrite },
  block { zeroinit  }
};
```


2.6 Fixed placement of objects

With embedded systems it is essential to be able to place code and data in memory at known locations or to avoid known locations. Examples of this include placing jump tables or vectors at a specific location, avoiding flash security areas, or placing configuration data at a specific address.

2.6.1 Placement by linker script

Placing an object at a fixed address requires a single statement:

```
place at address 0x400 { section .config };
```

This places the **.config** section at address 0x400. In GNU C syntax you can define the object allocated to the section like this:

```
static unsigned char __attribute((section(".config"))) _aConfig[32];
```

If the object has **extern** linkage, like this:

```
unsigned char aConfig[32];
```

then you can use the symbol name directly bypassing the requirement to use a named section:

```
place at address 0x400 { symbol aConfig };
```

2.6.2 Placement by section name

A more convenient way to place objects at a specific address is to use a capability of the linker which interprets section names. A section name that is "**.ARM.__at_0x***addr*" is interpreted as an instruction to place the corresponding section at the address *addr*. For instance:

```
void __attribute((section(".ARM.__at_0x8000"))) MyFunc(void) {
  ...
}
```

This locates the function **MyFunc** to start at address 0x8000. The same scheme extends to variables:

```
unsigned char __attribute((section(".ARM.__at_0x400"))) aConfig[32] = {
  ...
}
```

Interpretation of section names is turned on by default. You control this capability using the **--autoat** and **--no-autoat** command line switches: see *--autoat* on page 150 and *--no-autoat* on page 151.

2.7 Placing code in RAM

The linker is capable of placing code in RAM and ensuring that the code is copied from flash to RAM before entering **main()**. In fact, the linker treats RAM-based code no different from RAM-based data that is also initialized before entering **main()**.

2.7.1 Selecting the code to place in RAM

It's possible to use the linker script with section names or symbol names to select the code that is to be placed in RAM. Continuing from the previous examples, a single function can be placed in RAM by setting its section name:

```
void __attribute((section(".ramfunc"))) MyFunc(void) {
  ...
}
```

And then placing all such sections in RAM in the linker script:

```
place in RAM       { section .ramfunc };
initialize by copy { section .ramfunc };
```

The **initialize by copy** is required as the linker assumes that all sections containing read-execute code are implicitly present at system startup, *even if they have been placed into RAM*. The **initialize by copy** overrides the default handling of this particular section and instructs the linker to make an image of the section and copy it to its final destination on system startup.

> **Note**
>
> At high optimization levels the compiler may make an inline copy of a function even though it is declared to be in a nondefault section. To ensure that the function is indeed run from RAM the compiler must be told not to inline the function:
>
> **void __attribute((section(".ramfunc"), noinline)) MyFunc(void) …**

2.7.2 Tightly-coupled memory

Some controllers have tightly-coupled instruction memory which benefits fast execution, and the above mechanism is easily extended to place code into ITCM SRAM.

2.8 Placement with preferences

The linker is capable of automatically placing code and data across multiple noncontiguous memory ranges.

2.8.1 A simple example

Kinetis devices have two separate memory ranges where execution from the lower RAM is fast, but execution from the upper RAM is slower.

A simple configuration which allocates code to run in RAM along with data would be:

```
define region RAM = [from 0x1fff0000 size  64k] +
                    [from 0x20000000 size 192k];

place in RAM { readwrite, zeroinit, section .fastrun };
```

The linker will allocate code and data, however it fits, into the RAM with no priority given to placing code in the lower fast-execute region.

A simple remedy for this is to use two separate ranges and manually distribute the code and data:

```
define region LORAM = [from 0x1fff0000 size  64k];
define region HIRAM = [from 0x20000000 size 192k];

place in LORAM         { section .fastrun    };
place in LORAM + HIRAM { readwrite, zeroinit };
```

This will allocate the code that should run quickly into the low memory, with read-write and zero-initialized data allocated to the remainder. The linker will typically allocate from low memory to high memory in a memory region, so it's highly likely that some data will be allocated to the low memory which may impact performance.

What we need to do is instruct the linker to apply a *preference order* for the data, to try to allocate objects first into high memory and, if any do not fit, to place the overflow into the low memory with the code. We specify a preference using **`then`**:

```
define region LORAM = [from 0x1fff0000 size 64k];
define region HIRAM = [from 0x20000000 size 192k];

place in LORAM            { section .fastrun    };
place in HIRAM then LORAM { readwrite, zeroinit };
```

With this script, we accomplish the best layout possible without sacrificing flexibility: the code is placed into fast-executing low memory, the data is separated into high memory that doesn't affect performance, and if data is so big it will not fit into high memory, it overflows into low memory which might affect performance but will not cause a link error.

2.8.2 A more complex example

The STM32H7 has a number of RAM regions; taking the STM32H743x as an example, it has:

- 64K of tightly-coupled instruction memory (ITCM-SRAM).
- 128K of tightly-coupled data memory (DTCM-SRAM) organized as two banks of 64K.
- 512K of RAM on the AXI bus, D1 domain (AXI-SRAM).
- Two 128K banks of RAM on the AHB bus, D2 domain (AHB-SRAM1 and AHB-SRAM2).
- An addition 32K of RAM on the AHB bus, D2 domain (AHB-SRAM3)
- 64K of RAM on the AHB bus, D3 domain (AHB-SRAM4).
- 4K of backup RAM, D3 domain.

Clearly, organizing how memories are assigned is highly important when designing a system using this processor as each domain can be accessed concurrently by processor and peripheral. In addition, the two DTCM-SRAM banks can be used in parallel.

The following is an example of how to construct a linker script for this device, placing data used for Ethernet and USB in separate memories so they can be used in parallel, and assigning data in a way that maximizes performance whilst, at the same time, relieving the developer from arduous section placement.

```
define region ITCM_SRAM   = [from 0x00000000 size  64k];  ❶
define region DTCM_SRAM   = [from 0x20000000 size 128k];
define region AXI_SRAM    = [from 0x24000000 size 128k];
define region AHB_SRAM1   = [from 0x30000000 size 128k];
define region AHB_SRAM2   = [from 0x30020000 size 128k];
define region AHB_SRAM3   = [from 0x30040000 size  32k];
define region AHB_SRAM4   = [from 0x38000000 size  64k];
define region BACKUP_SRAM = [from 0x38800000 size   4k];

place in AHB_SRAM1 { section .ETH.bss*, section .ETH.data* };  ❷
place in AHB_SRAM4 { section .USB.bss*, section .USB.data* };

place in ITCM_SRAM { section .ramcode* };  ❸
initialize by copy { section .ramcode* };

place in BACKUP_SRAM { section .backup* };  ❹
do not initialize    { section .backup* };

place in      DTCM_SRAM  ❺
         then AXI_SRAM
         then AHB_SRAM2 + AHB_SRAM3
         then AHB_SRAM1 + AHB_SRAM4
         then BACKUP_SRAM  { readwrite, zeroinit };
```

❶ Define regions

Configuring regions is straightforward, one definition per memory:

❷ Placing Ethernet and USB RAM

We separate Ethernet and USB RAM use over two regions using SRAM1 for Ethernet and SRAM4 for USB.

❸ Placing fast code

Code that needs to run quickly, such as interrupt routines, is best placed into the ITCM SRAM. The linker is told to initialize the code placed into the ITCM SRAM by copying it there (see *Placing code in RAM* on page 27).

❹ Placing backup data

Data to be preserved across power cycles must be assigned to the battery backup area, but also we must instruct the linker not to zero or otherwise initialize this data on startup.

❺ Placing application data

All that remains is to place application data with a specific preference. We should use the zero-wait-state DTCM first, followed by fast RAM areas, and any overflow should be allocated to the unused parts of the USB and Ethernet areas, and finally the battery-backup area as a last resort.

Any specific assignments of sections to memories should be made before the final catch-all placement.

2.9 Thread-local data

It is possible to use thread-local data in applications, but it requires some support from the real-time operating system and the linker script to work correctly. Typically the RTOS requires that thread-local read-write data and zero-initialized data are allocated to a contiguous block.

The following example shows how the thread-local read-write data and zero-initialized data are combined into two blocks, and those blocks are then allocated in a specific order to satisfy the requirements of the real-time operating system:

```
define block tbss  { section .tbss,  section .tbss.*  };
define block tdata { section .tdata, section .tdata.* };
define block tls   { block tbss, block tdata };
```

It is possible for the blocks to be declared inline:

```
define block tls {
  block tbss  { section .tbss,  section .tbss.*  },
  block tdata { section .tdata, section .tdata.* }
};
```

The specific arrangement for thread-local data will be documented in the real-time operating system manual. You will need to refer to this manual when using the SEGGER linker with your RTOS.

2.10 Firmware integrity checks

The SEGGER linker simplifies the calculation and inclusion of integrity checks over one or more regions of a firmware image. Integrity check algorithms supported are cyclic redundancy checks (CRCs) or a message digests (the output of a hash function).

2.10.1 The purpose of integrity checks

Integrity checks are typically added to a firmware image to detect whether it has been corrupted. A bootloader can verify that a replacement firmware image, for example, is received intact with high confidence. Alternatively, the firmware can verify its own image if flash corruption is a concern.

2.10.2 A simple CRC over a contiguous region

The linker can create sections containing calculated integrity checks using the **`integrity check of`** selector:

`integrity check of` *region-expr* [**`with`** *attr attr…*]

where *attr* is one of:

`algorithm`=*string*
`fill`=*expr*

For instance, the following creates a section that contains a CRC over the entire region named **`FLASH`**:

```
integrity check of FLASH
```

The default is to use a standard CRC-32 algorithm and assume that any *usused gaps* in the region **`FLASH`** are filled with the value `0xFF`.

The integrity check section must be placed using a **`place at`** or **`place in`** selector. Usually the integrity check is at a fixed location in flash or in the load image, or immediately follows the firmware image. Therefore, something like the following can be used:

```
define region FLASH       = [0x80000000 size 512k];  ❶
define region CRC         = [end(FLASH)-4 size 4];   ❷
define region APPLICATION = FLASH - CRC;     ❸

place in APPLICATION {  ❹
  section .text,     section .text.*,
  section .rodata.*, section .rodata.*
};

place in CRC {  ❺
  integrity check of APPLICATION with algorithm="CRC-32" fill=0xFF
};
```

❶ Define entire flash area

The definition of the **`FLASH`** region covers the entire flash region of the microcontroller.

❷ Define CRC region

The definition of the **`CRC`** region reserves four bytes of memory at the end of **`FLASH`** to contain the CRC.

❸ Define application area

The **`APPLICATION`** region is defined to be the entire **`FLASH`** region excluding the **`CRC`** region.

❹ Place the application sections

The **`place in`** statement assigns the required sections to the application area. What is placed in the application area may well be more than the sections listed here, that is entirely down to the user.

❺ Place the CRC section

The **`place in`** statement calculates the CRC over the application area and places that section into the CRC area.

2.10.3 Integrity checks over multiple regions

An integrity check doesn't need a contiguous region to work over. It is possible to create multiple integirty checks over different contiguous regions or single integrity checks over multiple regions.

For instance, you might want to create an integrity check over two flash regions: one internal flash and one external flash, like this:

```
define region INTERNAL_FLASH = [0x80000000 size 512k];
define region EXTERNAL_FLASH = [0x40000000 size 2m];
define region CRC            = [end(INTERNAL_FLASH)-4 size 4];

place in CRC {
  integrity check of INTERNAL_FLASH + EXTERNAL_FLASH - CRC;
}
```

This will calculate a single CRC over the two regions. Please note that the linker calculates the integrity check over regions *in address order*. This means that the CRC in the example above is calculated over the external flash first, as it has the lowest address, followed by the internal flash, and excludes the region where the CRC is itself held.

When verifying such an integrity check, calculate the integrity check over the regions in the same order that the linker calculates them. This is supported by SEGGER's emLib-CRC and emCrypt.

2.10.4 Supported CRC algorithms

The following CRC integrity check algorithms are suppored:

Algorithm ID	Polynomial	Reflected?	Initial	Final
CRC-7/MMC	0x09	No	0x00	0x00
CRC-8	0x07	No	0x00	0x00
CRC-8/CDMA2000	0x9B	No	0xFF	0x00
CRC-8/DARC	0x39	Yes	0x00	0x00
CRC-8/MAXIM, CRC-8/1WIRE	0x31	Yes	0x00	0x00
CRC-8/AUTOSAR	0x1D	No	0xFF	0xFF
CRC-8/BLUETOOTH	0xA7	Yes	0x00	0x00
CRC-16/CCITT:AUG	0x1021	No	0x1D0F	0x0000
CRC-16/CCITT:NOAUG	0x1021	No	0xFFFF	0x0000
CRC-16/KERMIT	0x1021	Yes	0x0000	0x0000
CRC-16/X.25	0x1021	Yes	0xFFFF	0xFFFF
CRC-16/XMODEM	0x1021	No	0x0000	0x0000
CRC-16/ARC	0x8005	Yes	0x0000	0x0000
CRC-16/UMTS	0x8005	No	0x0000	0x0000

Algorithm ID	Polynomial	Reflected?	Initial	Final
CRC-16/MODBUS	0x8005	Yes	0xFFFF	0x0000
CRC-16/USB	0x8005	Yes	0xFFFF	0xFFFF
CRC-16/CDMA2000	0xC867	No	0xFFFF	0x0000
CRC-32	0x04C11DB7	Yes	0xFFFFFFFF	0xFFFFFFFF
CRC-32/BZIP2	0x04C11DB7	No	0xFFFFFFFF	0xFFFFFFFF
CRC-32/MPEG2, **CRC-32/STM32**	0x04C11DB7	No	0xFFFFFFFF	0x00000000
CRC-32/POSIX	0x04C11DB7	No	0x00000000	0xFFFFFFFF
CRC-32/XFER	0x000000AF	No	0x00000000	0x00000000
CRC-32C	0x1EDC6F41	Yes	0xFFFFFFFF	0xFFFFFFFF
CRC-32D	0xA833982B	Yes	0xFFFFFFFF	0xFFFFFFFF
CRC-32Q	0x814141AB	No	0x00000000	0x00000000

The algorithm **CRC-16/CCITT:AUG** can be abbreviated to **CRC-CCITT:AUG** and the algorithm **CRC-16/CCITT:NOAUG** can be abbreviated to **CRC-CCITT:NOAUG**.

2.10.5 Supported message digest algorithms

The following message digest integrity check algorithms are suppored:

Algorithm ID	Digest size (bytes)
MD5	16
RIPEMD-160	20
SHA-1	20
SHA-224	28
SHA-256	32
SHA-384	48
SHA-512	64
SHA-512/224	28
SHA-512/256	32
SHA3-224	28
SHA3-256	32
SHA3-384	48
SHA3-512	64
SM3	32

2.10.6 Other supported algorithms

The following integrity check algorithms are also suppored:

Algorithm ID	Description
Adler-32	Adler-32 algorithm. The 4-byte checksum is stored in network (big-endian) byte order. See RFC 1950 for a description and example implementation of the Adler-32 algorithm.
XXH32	xxH32 algorithm. The 4-byte checksum, initialized with a seed of zero, is stored in PC (little-endian) byte order. See xxHash fast digest algorithm for a description and example implementation of the XXH32 algorithm.

2.10.7 Reference code for integrity checking

The following application uses the SEGGER CRC Library and SEGGER's emCrypt product (a cryptographic library) to verify CRCs and message digests computed by the linker.

2.10.7.1 Linker script file setting up integrity checks

```
/*********************************************************************
*                    SEGGER Microcontroller GmbH                     *
*                        The Embedded Experts                        *
**********************************************************************
*                                                                    *
*            (c) 2014 - 2020 SEGGER Microcontroller GmbH             *
*                                                                    *
*           www.segger.com     Support: support@segger.com           *
*                                                                    *
**********************************************************************
*                                                                    *
* All rights reserved.                                               *
*                                                                    *
* Redistribution and use in source and binary forms, with or         *
* without modification, are permitted provided that the following    *
* conditions are met:                                                *
*                                                                    *
* - Redistributions of source code must retain the above copyright   *
*   notice, this list of conditions and the following disclaimer.    *
*                                                                    *
* - Neither the name of SEGGER Microcontroller GmbH                  *
*   nor the names of its contributors may be used to endorse or      *
*   promote products derived from this software without specific     *
*   prior written permission.                                        *
*                                                                    *
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND             *
* CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,        *
* INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF           *
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE           *
* DISCLAIMED.                                                        *
* IN NO EVENT SHALL SEGGER Microcontroller GmbH BE LIABLE FOR        *
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR           *
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  *
* OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;    *
* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF      *
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT          *
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  *
* USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH   *
* DAMAGE.                                                            *
*                                                                    *
**********************************************************************
*/

define memory with size = 4G;

//
// Combined regions per memory type
//
define region FLASH = FLASH1;
define region RAM   = RAM1;

//
// Block definitions
//
define block vectors                  { section .vectors };
define block vectors_ram              { section .vectors_ram };
define block ctors                    { section .ctors, section .ctors.*,
                                        block with alphabetical order { init_array } };
define block dtors                    { section .dtors, section .dtors.*,
                                        block with reverse alphabetical order { fini_array } };
define block exidx                    { section .ARM.exidx, section .ARM.exidx.* };
define block tbss                     { section .tbss,      section .tbss.*  };
define block tdata                    { section .tdata,     section .tdata.* };
define block tls                      { block tbss, block tdata };
define block tdata_load               { copy of block tdata };

//
```



```
// Stack reservation
//
define block heap  with size = __HEAPSIZE__,  alignment = 8, readwrite access { };
define block stack with size = __STACKSIZE__, alignment = 8, readwrite access { };

//
// Explicit initialization settings for sections
//
do not initialize                       { section .non_init, section .non_init.*,
                                          section .*.non_init, section .*.non_init.* };
initialize by copy                      { section .data, section .data.*,
                                          section .*.data, section .*.data.* };
initialize by copy                      { section .fast, section .fast.* };

//
// Explicit placement in FLASHn
//
place in FLASH1                         { section .FLASH1, section .FLASH1.* };

//
// Define a section with known contents.
//
define root section .rodata.TestData {
  udata8 0x31, udata8 0x32, udata8 0x33,
  udata8 0x34, udata8 0x35, udata8 0x36,
  udata8 0x37, udata8 0x38, udata8 0x39,
};

//
// Wrap that section in a block so that we can access the test
// data using linker-generated symbols.
//
define block TestData {
  section .rodata.TestData
};

//
// Define a region where the test data sits.
//
define region TEST = [start(FLASH)+64k size 9];

//
// Place the test data.
//
place in TEST { block TestData };

//
// Follow the test data with a set of CRCs and hashes which
// can be verified.
//
place after TEST {
  block with fixed order {
    //
    // CRCs
    //
    integrity check of TEST with algorithm="CRC-7/MMC",
    integrity check of TEST with algorithm="CRC-8",
    integrity check of TEST with algorithm="CRC-8/CDMA2000",
    integrity check of TEST with algorithm="CRC-8/DARC",
    integrity check of TEST with algorithm="CRC-8/MAXIM",
    integrity check of TEST with algorithm="CRC-8/AUTOSAR",
    integrity check of TEST with algorithm="CRC-8/BLUETOOTH",
    integrity check of TEST with algorithm="CRC-16/CCITT:AUG",
    integrity check of TEST with algorithm="CRC-16/CCITT:NOAUG",
    integrity check of TEST with algorithm="CRC-16/KERMIT",
    integrity check of TEST with algorithm="CRC-16/X.25",
    integrity check of TEST with algorithm="CRC-16/XMODEM",
    integrity check of TEST with algorithm="CRC-16/MODBUS",
    integrity check of TEST with algorithm="CRC-16/USB",
    integrity check of TEST with algorithm="CRC-16/ARC",
    integrity check of TEST with algorithm="CRC-16/UMTS",
    integrity check of TEST with algorithm="CRC-16/CDMA2000",
    integrity check of TEST with algorithm="CRC-32",
    integrity check of TEST with algorithm="CRC-32/BZIP2",
    integrity check of TEST with algorithm="CRC-32/MPEG2",
    integrity check of TEST with algorithm="CRC-32/POSIX",
```



```
    integrity check of TEST with algorithm="CRC-32/XFER",
    integrity check of TEST with algorithm="CRC-32C",
    integrity check of TEST with algorithm="CRC-32D",
    integrity check of TEST with algorithm="CRC-32Q",
    //
    // Hashes
    //
    integrity check of TEST with algorithm="MD5",
    integrity check of TEST with algorithm="RIPEMD-160",
    integrity check of TEST with algorithm="SHA-1",
    integrity check of TEST with algorithm="SHA-224",
    integrity check of TEST with algorithm="SHA-256",
    integrity check of TEST with algorithm="SHA-384",
    integrity check of TEST with algorithm="SHA-512",
    integrity check of TEST with algorithm="SHA-512/224",
    integrity check of TEST with algorithm="SHA-512/256",
    integrity check of TEST with algorithm="SHA3-224",
    integrity check of TEST with algorithm="SHA3-256",
    integrity check of TEST with algorithm="SHA3-384",
    integrity check of TEST with algorithm="SHA3-512",
    integrity check of TEST with algorithm="SM3"
  }
};

//
// FLASH Placement
//
place at start of FLASH                { block vectors };
place in FLASH with minimum size order { section .init,   section .init.*,
                                         section .init_rodata, section .init_rodata.*,
                                         section .text,   section .text.*,
                                         section .rodata, section .rodata.*,
                                         section .segger.*,
                                         block exidx,
                                         block ctors,
                                         block dtors };
place in FLASH                         { block tdata_load };

//
// Explicit placement in RAMn
//
place in RAM1                          { section .RAM1, section .RAM1.* };

//
// RAM Placement
//
place at start of RAM                  { block vectors_ram };
place in RAM                           { section .non_init, section .non_init.*,
                                         block tls };
place in RAM with auto order           { section .fast, section .fast.*,
                                         section .data, section .data.*,
                                         section .bss, section .bss.*
                                       };
place in RAM                           { block heap };
place at end of RAM                    { block stack };
```

2.10.7.2 Integrity check complete listing

```
/*********************************************************************
*                    SEGGER Microcontroller GmbH                     *
*                        The Embedded Experts                        *
**********************************************************************
*                                                                    *
*            (c) 2014 - 2020 SEGGER Microcontroller GmbH             *
*                                                                    *
*           www.segger.com     Support: support@segger.com           *
*                                                                    *
**********************************************************************
*                                                                    *
* All rights reserved.                                               *
*                                                                    *
* Redistribution and use in source and binary forms, with or         *
* without modification, are permitted provided that the following    *
* conditions are met:                                                *
```

```
*                                                                    *
* - Redistributions of source code must retain the above copyright   *
*   notice, this list of conditions and the following disclaimer.    *
*                                                                    *
* - Neither the name of SEGGER Microcontroller GmbH                  *
*   nor the names of its contributors may be used to endorse or      *
*   promote products derived from this software without specific     *
*   prior written permission.                                        *
*                                                                    *
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND             *
* CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,        *
* INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF           *
* MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE           *
* DISCLAIMED.                                                        *
* IN NO EVENT SHALL SEGGER Microcontroller GmbH BE LIABLE FOR        *
* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR           *
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  *
* OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;    *
* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF      *
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT          *
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE  *
* USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH   *
* DAMAGE.                                                            *
*                                                                    *
**********************************************************************

-------------------------- END-OF-HEADER -----------------------------

File    : main.c
Purpose : Demonstrate verification of each SEGGER Linker integrity
          check algorithm.

*/

/*********************************************************************
*
*       #include Section
*
**********************************************************************
*/

#include "SEGGER_CRC.h"
#include "CRYPTO.h"
#include <stdio.h>

/*********************************************************************
*
*       External data
*
**********************************************************************
*/

extern unsigned char __TestData_start__[];
extern unsigned char __TestData_end__  [];

/*********************************************************************
*
*       Static code
*
**********************************************************************
*/

/*********************************************************************
*
*       _Assert()
*
*  Function description
*    Break when an assertion fails.
*/
static void _Assert(int x) {
  if (x == 0) {
    printf("Integrity check failed!\n");
    asm("bkpt");
  }
}
```



```
/*********************************************************************
*
*       _LoadU16()
*
*  Function description
*    Load 16-bit value from memory, PC byte order.
*/
static unsigned _LoadU16(const U8 *pData) {
  return pData[0] + pData[1]*0x100u;
}

/*********************************************************************
*
*       _LoadU32()
*
*  Function description
*    Load 32-bit value from memory, PC byte order.
*/
static unsigned _LoadU32(const U8 *pData) {
  return pData[0] + pData[1]*0x100u + pData[2]*0x10000u + pData[3]*0x1000000u;
}

/*********************************************************************
*
*       Public code
*
**********************************************************************
*/

/*********************************************************************
*
*       CRYPTO_X_Panic()
*
*  Function description
*    Called when the library detectes an unrecoverable error.
*/
void CRYPTO_X_Panic(void) {
  printf("CRYPTO panic!\n");
  asm("bkpt");
}

/*********************************************************************
*
*       main()
*
*  Function description
*   Application entry point for integrity verification.
*/
int main(void) {
  unsigned char *pTestData    = __TestData_start__;
  unsigned char *pCheckData   = __TestData_end__;
  unsigned       TestDataLen  = __TestData_end__ - __TestData_start__;
  U32            CalcCRC;
  static U8      aHash[512/8];
  //
  // Initialize CRYPTO.
  //
  CRYPTO_MD5_Install      (&CRYPTO_HASH_MD5_SW,        NULL);
  CRYPTO_RIPEMD160_Install(&CRYPTO_HASH_RIPEMD160_SW,  NULL);
  CRYPTO_SM3_Install      (&CRYPTO_HASH_SM3_SW,        NULL);
  CRYPTO_SHA1_Install     (&CRYPTO_HASH_SHA1_SW,       NULL);
  CRYPTO_SHA224_Install   (&CRYPTO_HASH_SHA224_SW,     NULL);
  CRYPTO_SHA256_Install   (&CRYPTO_HASH_SHA256_SW,     NULL);
  CRYPTO_SHA512_Install   (&CRYPTO_HASH_SHA512_SW,     NULL);
  CRYPTO_SHA3_224_Install (&CRYPTO_HASH_SHA3_224_SW,   NULL);
  CRYPTO_SHA3_256_Install (&CRYPTO_HASH_SHA3_256_SW,   NULL);
  CRYPTO_SHA3_384_Install (&CRYPTO_HASH_SHA3_384_SW,   NULL);
  CRYPTO_SHA3_512_Install (&CRYPTO_HASH_SHA3_512_SW,   NULL);
  //
  // Get address and length of data to check.
  //
  pTestData   = __TestData_start__;
  TestDataLen = __TestData_end__ - __TestData_start__;
  //
  // Get address of the list of corresponding integrity check
  // values.
```



```
//
pCheckData  = __TestData_end__;
//
// CRC-7/MMC.
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x00, 0x09, 7);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x00, 0x07, 8);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8/CDMA2000
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFF, 0x9B, 8);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8/DARC
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0x00, 0x9C);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8/MAXIM.
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0x00, 0x8C);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8/AUTOSAR
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFF, 0x1D, 8) ^ 0xFF;
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-8/BLUETOOTH
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0x00, 0xE5);
_Assert(CalcCRC == *pCheckData);
pCheckData += 1;
//
// CRC-16/CCITT:AUG
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x1D0F, 0x1021, 16);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/CCITT:NOAUG
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFFFF, 0x1021, 16);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/KERMIT
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0x0000, 0x8408);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/X.25
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFF, 0x8408) ^ 0xFFFF;
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/XMODEM
//
CalcCRC =  SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x0000, 0x1021, 16);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/MODBUS
```



```
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFF, 0xA001);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/USB
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFF, 0xA001) ^ 0xFFFF;
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/ARC
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0x0000, 0xA001);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/UMTS
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x0000, 0x8005, 16);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-16/CDMA2000
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFFFF, 0xC867, 16);
_Assert(CalcCRC == _LoadU16(pCheckData));
pCheckData += 2;
//
// CRC-32
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFFFFFF, 0xEDB88320) ^ 0xFFFFFFFF;
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32/BZIP2
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFFFFFFFF, 0x04C11DB7, 32) ^ 0xFFFFFFFF;
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32/MPEG2
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0xFFFFFFFF, 0x04C11DB7, 32);
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32/POSIX
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x00000000, 0x04C11DB7, 32) ^ 0xFFFFFFFF;
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32/XFER
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x00000000, 0x000000AF, 32);
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32C
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFFFFFF, 0x82F63B78) ^ 0xFFFFFFFF;
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32D
//
CalcCRC = SEGGER_CRC_Calc(pTestData, TestDataLen, 0xFFFFFFFF, 0xD419CC15) ^ 0xFFFFFFFF;
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
//
// CRC-32Q
//
CalcCRC = SEGGER_CRC_Calc_MSB(pTestData, TestDataLen, 0x00000000, 0x814141AB, 32);
_Assert(CalcCRC == _LoadU32(pCheckData));
pCheckData += 4;
```



```
  //
  // MD5.
  //
  CRYPTO_MD5_Calc(aHash, CRYPTO_MD5_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_MD5_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_MD5_DIGEST_BYTE_COUNT;
  //
  // RIPEMD-160
  //
  CRYPTO_RIPEMD160_Calc(aHash, CRYPTO_RIPEMD160_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_RIPEMD160_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_RIPEMD160_DIGEST_BYTE_COUNT;
  //
  // SHA-1
  //
  CRYPTO_SHA1_Calc(aHash, CRYPTO_SHA1_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA1_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA1_DIGEST_BYTE_COUNT;
  //
  // SHA-224
  //
  CRYPTO_SHA224_Calc(aHash, CRYPTO_SHA224_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA224_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA224_DIGEST_BYTE_COUNT;
  //
  // SHA-256
  //
  CRYPTO_SHA256_Calc(aHash, CRYPTO_SHA256_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA256_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA256_DIGEST_BYTE_COUNT;
  //
  // SHA-384
  //
  CRYPTO_SHA384_Calc(aHash, CRYPTO_SHA384_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA384_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA384_DIGEST_BYTE_COUNT;
  //
  // SHA-512
  //
  CRYPTO_SHA512_Calc(aHash, CRYPTO_SHA512_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA512_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA512_DIGEST_BYTE_COUNT;
  //
  // SHA-512/224
  //
  CRYPTO_SHA512_224_Calc(aHash, CRYPTO_SHA512_224_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA512_224_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA512_224_DIGEST_BYTE_COUNT;
  //
  // SHA-512/256
  //
  CRYPTO_SHA512_256_Calc(aHash, CRYPTO_SHA512_256_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA512_256_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA512_256_DIGEST_BYTE_COUNT;
  //
  // SHA3-224
  //
  CRYPTO_SHA3_224_Calc(aHash, CRYPTO_SHA3_224_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA3_224_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA3_224_DIGEST_BYTE_COUNT;
  //
  // SHA3-256
  //
  CRYPTO_SHA3_256_Calc(aHash, CRYPTO_SHA3_256_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA3_256_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA3_256_DIGEST_BYTE_COUNT;
  //
  // SHA3-384
  //
  CRYPTO_SHA3_384_Calc(aHash, CRYPTO_SHA3_384_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA3_384_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA3_384_DIGEST_BYTE_COUNT;
  //
  // SHA3-512
  //
  CRYPTO_SHA3_512_Calc(aHash, CRYPTO_SHA3_512_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
```



```
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SHA3_512_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SHA3_512_DIGEST_BYTE_COUNT;
  //
  // SM3
  //
  CRYPTO_SM3_Calc(aHash, CRYPTO_SM3_DIGEST_BYTE_COUNT, pTestData, TestDataLen);
  _Assert(memcmp(aHash, pCheckData, CRYPTO_SM3_DIGEST_BYTE_COUNT) == 0);
  pCheckData += CRYPTO_SM3_DIGEST_BYTE_COUNT;
  //
  printf("All integrity checks passed!\n");
  asm("bkpt");
}

/************************** End of file ***************************/
```


Chapter 3

Linker script reference

3.1 Preprocessor

The linker script is preprocessed by a C-like preprocessor built into the linker. Although the preprocessor is very similar to a standard C preprocessor, it does not provide the predefined macros found in C preprocessors as they are irrelevant when processing linker scripts.

3.1.1 Preprocessor symbols

The following preprocessor symbols are defined by the preprocessor:

Name	Description
__SEGGER_LD_VERSION__	SEGGER Linker version number in the format **Mmmpp** which corresponds to the version string **M.mm.pp**.

Note that macros predefined for C, including **__FILE__**, **__LINE__**, **__DATE__**, **__TIME__**, **__STDC__** and so on, are not defined when preprocessing a linker script.

3.1.2 Supported directives

The linker's preprocessor supports the following directives:

Directive	Description
#define *name text*	Define preprocessor symbol *name* with replacement text *text*.
#define *name*(*args*) *text*	Define function-line preprocessor symbol *name* with replacement text *text*.
#undef *name*	Remove definition of preprocessor symbol *name*.
#if...**#elif**...**#else**...**#endif**	Conditional inclusion of source based on a preprocessor expression.
#ifdef...**#elif**...**#else**...**#endif**	Conditional inclusion of source if a preprocessor symbol is defined.
#ifndef...**#elif**...**#else**...**#endif**	Conditional inclusion of source if a preprocessor symbol is not defined.
#include "*name*"	Include file *name* into linker script by searching include directories.
#error *text*	Issue a fatal error diagnostic with message *text* and abort link.

3.2 Preliminaries

3.2.1 Primary symbol expressions

A primary symbol expression has the following syntax:

symbol-primary =
start of symbol *name* |
end of symbol *name* |
size of symbol *name* |
after symbol *name* |
alignment of symbol *name*

Function and object symbols

Function and object symbols are defined by their start address in memory and any associated size. It is possible that a symbol is not assigned a size by the compiler or by the assembler, and in this case its size is zero.

Untyped symbols

Untyped symbols have no size, they are simply numbers.

Evaluation

start of symbol *name* evaluates to the address of the symbol *name* for function and object symbols, and to the value of the symbol for untyped symbols.

size of symbol *name* evaluates to the size of the symbol *name*, if the associated symbol has a size, else zero.

end of symbol *name* evaluates to:

start of symbol *name* **+ size of symbol** *name* **- 1**

after symbol *name* evaluates to:

start of symbol *name* **+ size of symbol** *name*

alignment of symbol *name* evaluates to the alignment of the section that the symbol *name* is associated with, or **1** if the symbol is not associated with a section.

3.2.2 Primary block expressions

A primary block expression has the following syntax:

block-primary =
start of block *name* |
end of block *name* |
size of block *name* |
after block *name* |
alignment of block *name*

Evaluation

start of block *name* evaluates to the origin address of the block *name*.

size of block *name* evaluates to the size of the block *name*.

end of block *name* evaluates to:

start of block *name* **+ size of block** *name* **- 1**

after block *name* evaluates to:

start of block *name* **+ size of block** *name*

alignment of block *name* evaluates to the minimum alignment of the block *name* calculated from the block specification and over the locatables contained within the block.

3.2.3 Primary region expressions

A primary region expression has the following syntax:

region-primary =
    start of region *name* |
    end of region *name* |
    size of region *name* |
    after region *name*

Evaluation

start of region *name* evaluates to the origin address of the region *name*.

size of region *name* evaluates to the size of the region *name*, spanning the lowest address to the highest address in the region irrespective of whether there are discontinuities.

end of region *name* evaluates to:

start of region *name* **+ size of region** *name* **- 1**

after region *name* evaluates to:

start of region *name* **+ size of region** *name*

3.2.4 Primary section expressions

A primary section expression has the following syntax:

section-primary =
    start of section *name* |
    end of section *name* |
    size of section *name* |
    after section *name* |
    alignment of section *name*

Evaluation

The section name *name* must be unique. If there are multiple sections with the same name, evaluation of any section expression with that name results in a special, *undefined value*.

start of section *name* evaluates to the origin address of the unique section *name*.

size of section *name* evaluates to the size of the unique section *name*.

end of section *name* evaluates to:

start of section *name* **+ size of section** *name* **- 1**

after section *name* evaluates to:

start of section *name* **+ size of section** *name*

alignment of section *name* evaluates to the alignment of the section according to the section's alignment and any additional per-section-class alignment set on the command line.

3.2.5 General expressions

A general expression has the following syntax:

expr =
    logical-or-expr [**=>** *logical-or-expr*]...
logical-or-expr =
    logical-and-expr [**||** *logical-and-expr*]...


```
logical-and-expr =
    relational-expr [ && relational-expr ]...
relational-expr =
    term [ relop term ]...
term =
    factor [ addop factor ]...
factor =
    primary [ mulop primary ]...
primary =
    number |
    name |
    ~ primary |
    ! primary |
    symbol-primary |
    block-primary |
    region-primary |
    section-primary |
    ( expr )
```

where:

```
relop =
    < | ≤ | > | ≥ | = | ≠
addop =
    + | -
mulop =
    * | / | %
```

Evaluation

3.3 Memory ranges and regions

3.3.1 define memory statement

Syntax

define memory [*name*] [**with size=***expr*] **;**

Description

Defines the memory *name* with size *expr*; if *name* is omitted, it defaults to "**mem**" and if the **size** attribute is omitted, the memory size is set to "**4G**".

There must be at exactly one memory defined by the script before any memory regions are declared.

3.3.2 define region statement

Syntax

```
define region name = region-expr ;
```

Description

Defines the named region *name* to the region expression *region-expr*.

See also

Regions on page 18

3.3.3 default region statement

Syntax

default region *name* **=** *region-expr* **;**

Description

Defines the named region *name* to the region expression *region-expr* only if no other region definition of *name* exists. A region definition will exist if it is set using the **--add-region** command line option or by a previous **define region** statement.

See also

Regions on page 18, *--add-region* on page 149

3.4 Sections and symbols

3.4.1 define block statement

Syntax

```
define block name [ with attr, attr… ] {
    section-selectors
} [ except {
    section-selectors
} ] ;
```

where *attr* is one of:

```
size=expr
alignment=expr
auto order
fixed order
size order
alignment order
alphabetical order
size then alignment order
alignment then size order
alignment then alphabetical order
reverse size order
reverse alphabetical order
access then size order
minimum size order
maximum packing [order]
[ readwrite | rw ] access
[ readexec | rx ] access
mpu ranges
```

3.4.1.1 Size

The size of the block can be set using the **size** attribute. If the block size is set with this attribute, the block size is fixed and does not expand. If the block size is not set with a **size** attribute, the block size is as large as required to contain its inputs.

3.4.1.2 Alignment

The minimum alignment of the block can be set using the **alignment** attribute. The final alignment of the block is the maximum of the alignment set by the alignment attribute (if any) and the maximum alignment of any input section.

3.4.1.3 Section ordering

The section ordering attributes are:

auto order
All inputs in the block are automatically ordered to reduce inter-object gaps. This is the default if no order is defined by the **define block** directive.

fixed order
All inputs in the block are placed in the same order as they appear in the section selector part of the **define block** directive.

size order
All inputs in the block are ordered by size, largest to smallest.

alignment order
All inputs in the block are ordered by alignment, largest to smallest alignment.

alphabetical order
All inputs in the block are ordered by section name in alphabetical order.

size then alignment order
All inputs in the block are ordered by size, largest to smallest, and equal-size blocks are ordered by alignment, largest to smallest.

alignment then size order
All inputs in the block are ordered by alignment largest to smallest and for blocks of equal alignments, then ordered by size, largest to smallest.

alignment then alphabetical order
All inputs in the block are ordered by alignment largest to smallest and for blocks of equal alignments, then ordered by name.

reverse size order
All inputs in the block are ordered by reverse size, smallest to largest.

reverse alphabetical order
All inputs in the block are ordered by section name in reverse alphabetical order.

access then size order
All inputs in the block are ordered by access type (**rx**, **ro**, **rw**, **zi**) and then by size, largest to smallest.

maximum packing [order]
All inputs in the block are ordered to reduce inter-section gaps caused by alignment and to reduce the size of the associated unwind tables (if any).

minimum size order
Identical to **maximum packing**.

3.4.1.4 Access attributes

The block section flags are set to the union of the section flags of all block inputs. The default block section flags, before being modified by additional block inputs, can be set by the following:

readwrite access
The block is given read-write access.

readexec access
The block is given read-execute access.

By default the block only has "allocation access."

3.4.1.5 Use with the MPU

The attribute **mpu ranges** instructs the linker to set the size and alignment of the block so that it can be used by the Cortex-M MPU. The linker computes the alignment and size of all input sections and then sets the block size and alignment to an appropriate power of two.

The following table shows some examples of this behavior:

Size	Align	Assigned block size and alignment
3	4	4 — block is padded to alignment
4	4	4 — exact fit for size and alignment
5	4	8 — size and alignment are increased to accommodate block

Please refer to the *ARMv7-M Architecture Reference Manual* for further information on the Cortex-M MPU.

3.4.2 define symbol statement

Syntax

define symbol *name* **=** *number* | *symbol*

Description

Defines the symbol *name* to be the value *number* or the value of the symbol *symbol*. Once defined, symbols cannot be redefined by a subsequent **define symbol** statement.

3.4.3 initialize statement

Syntax

```
initialize by copy [ with attr, attr... ] {
    section-selectors
} ;

initialize by calling name {
    section-selectors
} ;
```

where *attr* is one of:

```
packing=algorithm
simple ranges
complex ranges
```

3.4.3.1 Packing

The linker is capable of compressing initialized data or code copied to RAM using different packing algorithms. The algorithm can be one of:

```
none
packbits
zpak
lzss | lz77
auto | smallest
```

Each algorithm has different compression characteristics. The packing algorithms are:

none
The initialization image is stored verbatim without any compression.

packbits
The initialization image is compressed using the PackBits algorithm.

zpak
The initialization image is compressed using the SEGGER ZPak algorithm which is good for images that have many zeros in them but are otherwise not suitable for other forms of packing (PackBits and LZSS).

lzss or **lz77**
The initialization image is compressed using a Lempel-Ziv-Storer-Szymanski scheme.

auto or **smallest**
The linker chooses the packing algorithm that minimizes the initialization image size from the active set of algorithms.

3.4.3.2 Ranges

The **simple ranges** and **complex ranges** attributes are accepted but are otherwise ignored.

3.4.3.3 Initialization by call

The **initialize by calling** statement enables custom initialization of sections before **main()** is entered. This statement instructs the linker to call the function *name* if any of the blocks or sections in the section selectors have nonzero size.

This particular feature is intended to reduce the size of application startup code by eliminating the setup of a heap, if no heap is required, and eliminating the calls to static constructors, if none are present.

Using this facility

Typically **initialize by calling** is used in the following fashion:

```
define block heap with size = __HEAPSIZE__, readwrite access { };
define block ctors { section .ctors, section .ctors.*,
                     block with alphabetical order { init_array } };

initialize by calling __SEGGER_init_heap     { block heap  };
initialize by calling __SEGGER_init_ctors    { block ctors };
```

This sequence of statements reserves space for the heap and gathers together the constructors and any pre-init arrays.

If the heap has a nonzero size (i.e. if __HEAPSIZE__ is not zero) then the linker will add a call to the function **__SEGGER_init_heap()** to the end of the initialization list.

Similarly, after gathering all static constructors that must be called before entering **main()**, if the size of the **ctors** block is nonzero then constructors exist and initialization must take place. In this case the linker will add a call to the function **__SEGGER_init_ctors()** to the end of the initialization list.

Both **__SEGGER_init_heap()** and **__SEGGER_init_ctors()** are already written and provided in the startup code.

Order of initialization

The order of initialization calls follows the order of **intialize by calling** statements in the linker script. As static constructors may well call the memory-allocation functions **malloc()**, **calloc()**, and **realloc()**, the **intialize by calling** statements are ordered to ensure the heap is initialized before calling any static constructor.

3.4.4 do not initialize statement

Syntax

```
do not initialize {
     section-selectors
} ;
```

Description

The sections selected by *section-selectors* are not added to the initialization table and will not be initialized on program entry.

This capability enables battery-backed data to be preserved across a reset or power cycle.

Example

```
do not initialize {
  section .backup,
  section .backup.*
};
```


3.4.5 place in statement

Syntax

```
[ name: ] place in region-expr [ then region-expr... ] [ with attr, attr... ] {
    [ first section section-wildcard ]
    [ first symbol symbol-name ]
    [ last section section-wildcard ]
    [ last symbol symbol-name ]
    section-selectors
} [ except {
    section-selectors
} ] ;
```

where *attr* is one of:

```
auto order
fixed order
size order
alignment order
alphabetical order
size then alignment order
alignment then size order
alignment then alphabetical order
reverse size order
reverse alphabetical order
access then size order
minimum size order
maximum packing [order]
```

3.4.5.1 Section ordering

The section ordering attributes are:

auto order
All inputs in the block are automatically ordered to reduce inter-object gaps. This is the default if no order is defined by the **place in** directive.

fixed order
All inputs in the block are placed in the same order as they appear in the section selector part of the **place in** directive.

size order
All inputs in the block are ordered by size, largest to smallest.

alignment order
All inputs in the block are ordered by alignment, largest to smallest alignment.

alphabetical order
All inputs in the block are ordered by section name in alphabetical order.

size then alignment order
All inputs in the block are ordered by size, largest to smallest, and equal-size blocks are ordered by alignment.

alignment then size order
All inputs in the block are ordered by alignment largest to smallest and for blocks of equal alignments, then ordered by size, largest to smallest.

alignment then alphabetical order
All inputs in the block are ordered by alignment largest to smallest and for blocks of equal alignments, then ordered by name.

reverse size order
All inputs in the block are ordered by reverse size, smallest to largest.

reverse alphabetical order
All inputs in the block are ordered by section name in reverse alphabetical order.

access then size order
All inputs in the block are ordered by access type (**rx**, **ro**, **rw**, **zi**) and then by size, largest to smallest.

maximum packing [order]
All inputs in the block are ordered to reduce inter-section gaps caused by alignment and to reduce the size of the associated unwind tables (if any).

minimum size order
Identical to **maximum packing**.

3.4.5.2 Section placement control

The **first section** and **last section** directives instruct the linker how to place sections relative to other sections in the placement statement.

The **first section** directive places the selected section at the lowest unused address in the region with all other selected sections in the placement statement at higher addresses.

The **last section** directive places the selected section at the lowest unused address that is beyond all other selected sections in the placement statement.

The **first symbol** and **last symbol** statements selects the first and last sections using a symbol name which may be more convenient.

3.4.5.3 Section placement selection

When selecting sections for placement, it may well be that a section can potentially match two or more selection criterial. In this case, the linker prefers the most specific placement when selecting a section. The linker uses the following criteria for section selection:

- All sections and symbols that have an exact name match are selected first.
- All sections that have a wildcard match are selected next.
- All sections that match using a mask (such as **rx**) are selected last.

If there is a unresolvable conflict, the linker will issue an error message. For instace, the section with name "abc" will be selected by the two placement statements…

```
place in RAM1 { section a* };
place in RAM2 { section *c };
```

…which cannot be resolved by the linker using section selection precedence, resulting in an error.

3.4.6 place at statement

Syntax

```
[ name: ] place at [
     address expr |
     start of region-expr |
     end of region-expr ]
[ with attr, attr... ] {
     section-selectors
} [ except {
     section-selectors
} ] ;
```

where *attr* is one of:

```
auto order
fixed order
size order
reverse size order
alignment order
alphabetical order
alignment then alphabetical order
reverse alphabetical order
minimum size order
maximum packing [order]
```

3.4.6.1 Section ordering

The section ordering attributes are:

auto order
: All inputs in the block are automatically ordered to reduce inter-object gaps. This is the default if no order is defined by the **place at** directive.

fixed order
: All inputs in the block are placed in the same order as they appear in the section selector part of the **place at** directive.

size order
: All inputs in the block are ordered by size, largest to smallest.

reverse size order
: All inputs in the block are ordered by reverse size, smallest to largest.

alignment order
: All inputs in the block are ordered by alignment, largest to smallest alignment.

alphabetical order
: All inputs in the block are ordered by section name in alphabetical order.

alignment then alphabetical order
: All inputs in the block are ordered by alignment largest to smallest and for blocks of equal alignments, then ordered by name.

reverse alphabetical order
: All inputs in the block are ordered by section name in reverse alphabetical order.

maximum packing [order]
: All inputs in the block are ordered to reduce inter-section gaps caused by alignment and to reduce the size of the associated unwind tables (if any).

minimum size order
: Identical to **maximum packing**.

3.4.6.2 Placing at an address

The **place at address** statement creates an internal block for the selected sections, sorts them according to the user preference, and attempts place the block at the specified address.

Example

```
place at address 0x001FF800 { section .bootloader };
```

3.4.6.3 Placing at the start of a range

The **`place at start of`** statement creates an internal block for the selected sections, sorts them according to the user preference, and attempts to place the block at the start of the range.

Example

```
define region FLASH = [from 0x00000000 size 2m];
place at start of FLASH { section .vectors };
```

3.4.6.4 Placing at the end of a range

The **`place at end of`** statement creates an internal block for the selected sections, sorts them according to the user preference, and attempts to place the block such that the last address used by the block is the last byte of the range.

Example

```
define region RAM = [from 0x20000000 size 192k];
place at end of RAM { section .stack };
```


3.4.7 define access statement

Syntax

```
define access access-selector, access-selector… {
    section-selectors
} ;
```

where *access-selector* is one of:

```
readonly
readwrite
readexec
not readonly
not readwrite
not readexec
```

Description

The **define access** statement modifies the section header flags of the selected sections. This may be required in order to place sections in memory regions that they would not usually reside in, or where the compiler creates sections with undesirable or incorrect section header flags.

As an example, the GCC compiler for RISC-V creates sections related to exceptions with the `SHF_WRITE` flag set, indicating that the section should be allocated to a RAM region. However, there are no expected writes to these descriptor sections and they would better reside in flash, reducing the RAM footprint and removing both the RAM initialization image and the copying operation on startup:

```
//
// The GNU compiler creates these exception-related sections as writeable.
// Override the section header flag and make them readonly so they can be
// placed into flash.
//
define access readonly { section .gcc_except_table, section .gcc_except_table.* };
define access readonly { section .eh_frame, section .eh_frame.* };
define access readonly { section .sdata.DW.* };
```


3.4.8 keep statement

Syntax

```
keep {
     section-selectors
} ;
```

Description

The **keep** statement requests that the linker keep sections in the generated output that would otherwise be discarded.

3.4.9 fill statement

Syntax

fill *region-expr* **with** *expr* **;**

Description

Instruct the linker to place *expt* in any unused bytes in the region defined by *region-expr*. This is useful when preparing program images that must be subjected to checks (such as a CRC, hash, or digital signature).

A **fill** statement can appear anywhere in the linker script; fills are executed after all sections are placed, adding linker-created sections that ensure any gaps (before, between, or after user sections) are filled.

The linker will report an error if there are conflicting **fill** statements, when the filled ranges overlap.

Example

```
define region FLASH = [from 0x00000000 size 2m];
fill FLASH with 0xFF;
```


3.5 Control options

3.5.1 assert statement

Syntax

assert *expr* [*with-clause*] **;**

assert [*with-clause*] **{**
expr, *expr*…
} ;

where *with-clause* is:

[**error** | **warning** | **remark**] *string*

Description

The **assert** statement ensures that one or more conditions are satisfied by the link process. All **assert** statements run after all sections are placed and memory layout is known, immediately before writing the output files.

Checking consistency of function inclusion

assert statements can be used to check a number of things related to symbols and memory layout. For instance, if your application has a call to **free()**, but no calls to **malloc()** or **realloc()**, then it could well be that something is wrong. Such a situation can be diagnosed with the following:

```
assert with warning "free() is required but there is no malloc() or realloc()" {
  linked symbol free ==> linked symbol malloc || linked symbol realloc
}
```

The implication operator **=>** is the standard implication operation of Boolean logic. This statement reads "if the symbol **free()** is linked into the application than either **malloc()** or **realloc()** must also be linked into the application."

Checking static size of the heap

Similarly you can check the static size of the heap. For the default implementation, a heap size of eight bytes is required, but your application may want to use more than this. The minimum size check is easily written:

```
assert with warning "when using a heap, malloc() and free() require a heap > 8 bytes" {
  (   linked symbol malloc
   || linked symbol free
   || linked symbol realloc) ==> size of block heap > 8
};
```

The implication operator is used again and this reads "if any of the symbols **malloc()**, **free()**, or **realloc()** are linked into the application then the size of the heap must be greater than eight bytes." The expression does not require parentheses, it is a matter of taste, but this makes the left-hand expression clear.

Checking well-formedness of sections

It's also possible to check some of the linker's internal workings and provide some checks as to the combination of sections. For instance, the following will check some attributes of the standard SEGGER Linker script and produce an error if something is wrong:

```
//
// Create blocks that combine constructors, destructors, and exception index
//
define block ctors  { section .ctors, section .ctors.*, init_array };
define block dtors  { section .dtors, section .dtors.*, fini_array };
define block exidx  { section .ARM.exidx, section .ARM.exidx.*     };
```



```
//
// Check blocks are well-formed.
//
assert with "constructors, destructors, or exception index is malformed" {
  size of block dtors % 4 == 0,
  size of block ctors % 4 == 0,
  size of block exidx % 8 == 0
};
```

The constructor block is simply an array of addresses, so its size must be a multiple of four. The same is true for the destructor block. The exception index is an array of two-word entries that are pointers or literal data, and therefore the size of the exception index must be a multiple of eight.

Checking correct instruction set for functions

If you are linking an application and you want to ensure that some functions are coded in Arm, for speed, rather than Thumb, you can check the value of the symbol associated with the function. Arm function symbols have the low-order bit of their value set to zero, and Thumb function symbols have it set to one. Therefore, the low-order bit of the symbol distinguishes the Arm instruction set from the from Thumb instruction set. To check that the function "FFT1024" is indeed compiled for Arm, you can use:

```
assert with "these functions are expected to be ARM code" {
  (start of symbol FFT1024 & 1) == 0
};
```

With such checking its possible to ensure that functions will execute in the intended instruction set and that execution speed is maintained and code size does not suffer inflation from unintended veneers.

Similarly, assert that a function is coded in the Thumb instruction set:

```
assert with "these functions are expected to be Thumb code" {
  (start of symbol main   & 1) == 1,
  (start of symbol printf & 1) == 1,  // And so on...
};
```

Checking size of thread-local data

It might be that you have many threads and thread-local data is an issue. Each thread typiclally has its thread-local data allocated on the stack when the task is created. As every task is allocated local copies of all thread-local data, irrespective of whether that particular task uses it or not, is can be expensive to indiscriminately use thread-local data.

You can check that things are not getting out of hand with an assert like this:

```
//
// Thread-local data blocks
//
define block tbss   { section .tbss,      section .tbss.*  };
define block tdata  { section .tdata,     section .tdata.* };
define block tls    { block tbss, block tdata };

//
// Check size of thread-local data, warning if too big.
//
assert size of block tls <= 32 with error "thread-local storage is too big";
```

Checking utilization of reserved memory areas

It is common to have configuration or personalization data maintained in flash memory assigned to a one more more flash sectors. Over time, and especially during development,

the quantity of data stored in these configuration areas increases. In this situation you might want to indicate to developers that the configuration area is becoming full and something should be done to limit its expansion.

Assuming that data are added to the area linearly, or by using a single variable to structure the block, the block's utilized size can be monitored:

```
//
// Define an empty section that acts as a high-water marker.
//
define section ConfigHighWater { };

//
// Place data in the block but make sure the sentinel is placed
// after all other sections.
//
define block Config with size=256 {
  symbol ConfigData,   // This is, e.g., a C-level variable.
  last section ConfigHighWater
};

//
// Warning if the is not at least 5% empty.
//
assert with warning "Configuration area is filling!" {
  100 * (end of block Config - start of section ConfigHighWater)
     / size of block Config >= 5;
}
```


3.5.2 option statement

Syntax

option *id*, *id*... **;**

Description

The **option** provides additional command-line options as if they were specified on the linker command line. These options are processed after all options given on the command line.

Example

```
option "--map-listing";              // Always produce an absolute listing
option "--relax", "--springboard";   // Always enable these optimizations
```


Chapter 4

Command-line options

Command line option naming is generally compatible with the following toolsets:

- GNU linker **`ld`**
- Arm linker **`armlink`**
- IAR linker **`ilink`**

Equivalence of command line options

The following sections describe the syntax of command line options. The SEGGER linker accepts command line options with either all hyphens between words (e.g. **`--no-unwind-tables`**) or with all underscores between words (e.g. **`--no_unwind_tables`**).

This manual shows all options using hyphens rather than underscores.

4.1 Input file options

4.1.1 --script

Synopsis

Add linker control script.

Syntax

--script *filename*
--script=*filename*
-T*filename*

Description

This option sets the linker script file name to *filename*. This option can be used more than once to supply multiple linker script files to the linker. The linker concatenates individual linker scripts into one combined script for processing.

4.1.2 --via

Synopsis

Read additional options and input files from file.

Syntax

--via *filename*
--via=*filename*
-f *filename*
@*filename*

Description

This option reads the file *filename* for additional options and input files. Options are separated by spaces or newlines, and file names which contain special characters, such as spaces, must be enclosed in double quotation marks.

Notes

This option can only be provided on the command line and cannot appear in an indirect file.

4.2 Output file options

4.2.1 --bare

Summary

Generate minimal output file.

Syntax

```
--bare
--no-sections
```

Description

This option eliminates all debug information, the symbol table, and the string table from the ELF file, and also removes the section table.

Note

This option is equivalent to specifying **`--no-debug`** and **`--no-symbols`**.

See also

--no-debug on page 76, *--no-symbols* on page 88

4.2.2 --block-section-headers

Summary

Produce minimal section table with block information.

Syntax

```
--block-section-headers
```

Description

This option produces an ELF file containing minimal section information (section header string table and string table) together with sections covering any blocks created by the linker script.

See also

--bare on page 73, *--full-section-headers* on page 82, *--minimal-section-headers* on page 84

4.2.3 --debug

Summary

Include debugging information.

Syntax

--debug

Description

This option instructs the linker to include any relevant debug input sections from the input object files and libraries and also includes both the symbol table and string table in the ELF file.

See also

--no-debug on page 76

4.2.4 --no-debug

Summary

Discard debugging information.

Syntax

--no-debug

Description

This option excludes debug information from the output file: all input debug sections are excluded and both the symbol table and string table are removed. With debug information removed, the resulting Arm ELF file is smaller, but debugging at source level is impossible.

Note

Discarding debug information only affects the image size as loaded into the debugger—it has *no effect* on the size of the executable image that is loaded into the target.

See also

--debug on page 75

4.2.5 --entry

Summary

Set target core or architecture.

Syntax

--entry *name*
--entry=*name*
-e*name*

Description

Sets the entry point to the symbol *name* and *name* is automatically kept.

The default is **--entry=Reset_Handler**.

See also

--no-entry on page 78

4.2.6 --no-entry

Summary

Set the entry point to zero.

Syntax

```
--no-entry
```

Description

The entry point in the ELF file is set to zero. To link code into the final application, there must be at least one root symbol--see *--keep-symbol* on page 171.

Note

The entry point may be used by debuggers to configure applications ready for execution once downloaded. Removing the entry point may, therefore, require manual setup or scripting specific to the debugger to correctly configure the application for execution.

See also

--entry on page 77

4.2.7 --force-output

Summary

Write ELF file regardless of link errors.

Syntax

```
--force-output
```

Description

This option instructs the linker to write an ELF file even if there are link errors.

4.2.8 --no-force-output

Summary

Do not write ELF file when link errors.

Syntax

```
--no-force-output
```

Description

This option instructs the linker not to write an ELF file in the presence of errors.

4.2.9 --full-program-headers

Summary

Produce program headers for all load regions.

Syntax

`--full-program-headers`

Description

This option produces an ELF file containing program headers for all sections including regions that are initialized by the runtime startup using the initialization table.

> **Note**
>
> This option is now deprecated and is ignored

See also

--load-program-headers on page 83

4.2.10 --full-section-headers

Summary

Produce a section table with per-object sections.

Syntax

`--full-section-headers`

Description

This option produces an ELF file containing a detailed per-object section table.

See also

--bare on page 73, *--block-section-headers* on page 74, *--minimal-section-headers* on page 84

4.2.11 --load-program-headers

Summary

Produce program headers for implicit load regions.

Syntax

`--load-program-headers`

Description

This option produces an ELF file containing program headers only for load regions that are implicitly loaded. Any region that is initialized by an **`initialize`** statement or is precluded from initialization by a **`do not initialize`** statement will be excluded from the ELF program headers.

> **Note**
>
> This option is now deprecated and is ignored

See also

--full-program-headers on page 81

4.2.12 --minimal-section-headers

Summary

Produce minimal section header table.

Syntax

```
--minimal-section-headers
```

Description

This option produces an ELF file containing minimal section information (section header string table and string table if symbols are required).

This is the default option for section headers.

See also

--bare on page 73, *--block-section-headers* on page 74, *--full-section-headers* on page 82

4.2.13 --output

Summary

Set output file name.

Syntax

--output *filename*
-o *filename*

--output=*filename*
-o=*filename*

Description

This option sets the ELF output filename, typically with extension "**elf**".

4.2.14 --strip

Summary

Remove debug information and symbols.

Syntax

```
--strip
```

Description

This option eliminates all debug information, the symbol table, and the string table from the ELF file, but does no remove the section table.

This option is equivalent to specifying **--no-debug** and **--no-symbols**.

See also

--no-debug on page 76, *--no-symbols* on page 88

4.2.15 --symbols

Summary

Include symbol table.

Syntax

`--symbols`

Description

This option includes the symbol table in the ELF file.

See also

--no-symbols on page 88

4.2.16 --no-symbols

Summary

Discard symbol table.

Syntax

```
--no-symbols
```

Description

This option removes the symbol table from the ELF file.

See also

--symbols on page 87

4.3 Map file options

A map file contains the following sections which can be individually controlled:

Name	Controlling option
Linker command line and script	*--map-script* on page 130
Section placement	*--map-placement* on page 128
Module summary	*--map-modules* on page 124
Module detail	*--map-module-detail* on page 126
Section detail	*--map-section-detail* on page 132
Unused memory summary	*--map-unused-memory* on page 140
Initialization table	*--map-init-table* on page 106
Linker-created veneers	*--map-veneers* on page 142
Exception tables	*--map-exception-table* on page 104
Symbol list	*--map-symbols* on page 136
Unused input summary	*--map-unused-inputs* on page 138
Absolute listing	*--map-listing* on page 108
Link summary	*--map-summary* on page 134

Rather than select each option individually, increasing level of detail can be selected by:

```
--map-none
--map-compact
--map-standard
--map-detailed
--map-full
```

The following table shows which sections are enabled for each detail level.

Name	Compact	Standard	Detailed	Full
Linker command line and script				●
Section placement			●	●
Module summary	●	●	●	●
Module detail			●	●
Section detail		●	●	●
Unused memory summary				●
Initialization table			●	●
Exception tables			●	●
Linker-created veneers				●
Symbol list	●	●	●	●
Unused inputs				●
Absolute listing				●
Link summary	●	●	●	●

4.3.1 --map-file

Summary

Generate a linker map file.

Syntax

--map-file *filename*
--Map *filename*

--map-file=*filename*
--Map=*filename*

Description

Generates a linker map file to the given filename.

4.3.2 --map-none

Summary

Exclude all map file sections in the generated map file.

Syntax

--map-none

Description

This can be used to easily select a small subset of map sections, for example to include only symbols and a summary use:

```
--map-none --map-symbols --map-summary
```

See also

--map-detailed on page 94, *--map-full* on page 95, *--map-standard* on page 93

4.3.3 --map-compact

Summary

Set compact map file options.

Syntax

--map-compact

Description

This option enables the following map section options:

```
--map-modules
--map-symbols
--map-summary
```

The following sections are disabled:

```
--no-map-script
--no-map-placement
--no-map-module-detail
--no-map-section-detail
--no-map-init-table
--no-map-unused
--no-map-veneers
--no-map-exceptions
--no-map-listing
```

See also

--map-none on page 91, *--map-standard* on page 93, *--map-detailed* on page 94, *--map-full* on page 95

4.3.4 --map-standard

Summary

Set standard map file options.

Syntax

```
--map-standard
--map-defaults
-M
```

Description

This option enables the following map section options:

```
--map-modules
--map-section-detail
--map-symbols
--map-summary
```

The following sections are disabled:

```
--no-map-script
--no-map-placement
--no-map-module-detail
--no-map-unused
--no-map-init-table
--no-map-veneers
--no-map-exceptions
--no-map-listing
```

See also

--map-none on page 91, *--map-compact* on page 92, *--map-detailed* on page 94, *--map-full* on page 95

4.3.5 --map-detailed

Summary

Set detailed map file options.

Syntax

--map-compact

Description

This option enables the following map section options:

```
--map-placement
--map-modules
--map-module-detail
--map-section-detail
--map-init-table
--map-exceptions
--map-symbols
--map-summary
```

The following sections are disabled:

```
--no-map-script
--no-map-veneers
--no-map-listing
```

See also

--map-none on page 91, *--map-standard* on page 93, *--map-detailed* on page 94, *--map-full* on page 95

4.3.6 --map-full

Summary

Include all map file sections in the generated map file.

Syntax

--map-full
--map-all

Description

This option enables the following map section options:

```
--map-placement
--map-modules
--map-module-detail
--map-section-detail
--map-summary
--map-script
--map-init-table
--map-exceptions
--map-veneers
--map-symbols
--map-listing
```

See also

--map-none on page 91, *--map-compact* on page 92, *--map-standard* on page 93, *--map-detailed* on page 94

4.3.7 --map-html

Summary

Generate linker map file in HTML format.

Syntax

`--map-html`

Description

This option formats the map file as an HTML document.

See also

--map-text on page 97

4.3.8 --map-text

Summary

Generate linker map file in plain text format.

Syntax

`--map-html`

Description

This option formats the map file as a plain text document.

See also

--map-html on page 96

4.3.9 --map-narrow

Summary

Enable narrow name fields in map file.

Syntax

```
--map-narrow
```

See also

--map-wide on page 99

4.3.10 --map-wide

Summary

Enable wide name fields in map file.

Syntax

`--map-wide`

See also

--map-narrow on page 98

4.3.11 --map-wrap

Summary

Wrap when text exceeds column with.

Syntax

--map-wrap

Description

When text within a column exceeds the column width, the linker wraps following columns to a new line and maintains column alignment.

See also

--no-map-wrap on page 101

4.3.12 --no-map-wrap

Summary

Wrap when text exceeds column with.

Syntax

```
--no-map-wrap
```

Description

When text within a column exceeds the column width, the text is truncated to fix and "…" replaces the truncated part.

See also

--map-wrap on page 100

4.3.13 --map-addr-format

Summary

Set the format for addresses.

Syntax

--map-addr-format=*value*

Description

This option defines how addresses are formatted in the map file and log file.

The following table shows the presentation for all formats and a selection of addresses.

Format	0x0000001a	0x00001a2b	0x001a2b3c	0x1a2b3c4d
0	1a	1a2b	1a2b3c	1a2b3c4d
1	1A	1A2B	1A2B3C	1A2B3C4D
2	1a	1a2b	1a’2b3c	1a2b’3c4d
3	1A	1A2B	1A’2B3C	1A2B’3C4D
4	0x1a	0x1a2b	0x1a2b3c	0x1a2b3c4d
5	0x1A	0x1A2B	0x1A2B3C	0x1A2B3C4D
6	0x1a	0x1a2b	0x1a’2b3c	0x1a2b’3c4d
7	0x1A	0x1A2B	0x1A’2B3C	0x1A2B’3C4D
8	0000001a	00001a2b	001a2b3c	1a2b3c4d
9	0000001A	00001A2B	001A2B3C	1A2B3C4D
10	0000’001a	0000’1a2b	001a’2b3c	1a2b’3c4d
11	0000’001A	0000’1A2B	001A’2B3C	1A2B’3C4D
12	0x0000001a	0x00001a2b	0x001a2b3c	0x1a2b3c4d
13	0x0000001A	0x00001A2B	0x001A2B3C	0x1A2B3C4D
14	0x0000’001a	0x0000’1a2b	0x001a’2b3c	0x1a2b’3c4d
15	0x0000’001A	0x0000’1A2B	0x001A’2B3C	0x1A2B’3C4D

The default setting is 13.

4.3.14 --map-size-format

Summary

Set the format for sizes.

Syntax

--map-size-format=*value*

Description

This option defines how sizes are formatted in the map file and log file.

The following table shows the presentation for all formats and a selection of sizes.

Format	90	23148	5926013	1517058956
0	0x5a	0x5a6c	0x5a6c7d	0x5a6c7b8c
1	0x5a	0x5a6c	0x5a’6c7d	0x5a6c’7b8c
2	0x5A	0x5A6C	0x5A6C7D	0x5A6C7B8C
3	0x5A	0x5A6C	0x5A’6C7D	0x5A6C’7B8C
4	90	23148	5926013	1517058956
5	90	23 148	5 926 013	1 517 058 956

The default setting is 5.

4.3.15 --map-exception-table

Summary

Include an exception table summary in the generated map file.

Syntax

--map-exception-table

Description

The linker will gather exception tables and exception indexes together for C++ programs. It will ensure the correct ordering of sections along with the index is maintained and will synthesize no-unwind exception table entires for C functions without exception support.

Example

```
*******************************************************************************
***                                                                         ***
***                              EXCEPTION TABLE                            ***
***                                                                         ***
*******************************************************************************

Sections with section ordering imposed:

  Index       Execution Range    Unwind   Execution Range
  Address     Covered            Type     Symbol or [section] name  Details
  ----------  -----------------  -------  ------------------------  -------------------------
  0x00006e18  00000040-0000012f  Generic  f1()                      Unwind data at 0x00006D50
  0x00006e20  00000130-0000016d  PR0      __SEGGER_RTL_X_assert     sp += 28; <84>; sp += 4
  0x00006e28  0000016e-0000019b  PR0      A::A(int)                 sp += 12; <84>; sp += 4
  0x00006e30  0000019c-000001cd  Stop     A::A(A const&)            [ Compiler created ]
  0x00006e38  000001ce-000001ef  PR0      A::~A()                   sp += 12; <84>; sp += 4
  0x00006e40  000001f0-000001fb  Stop     __clang_call_terminate    [ Compiler created ]
  0x00006e48  000001fc-00000359  Generic  f2()                      Unwind data at 0x00006CEC
  0x00006e50  0000035a-00000379  Stop     B::B(B const&)            [ Compiler created ]
  [--------]  0000037a-00000391  Stop     B::~B()                   [ Removed by index optimization ]
  0x00006e58  00000392-00000477  Generic  main                      Unwind data at 0x00006D90
  0x00006e60  00000478-00000483  Stop     fprintf                   [ Compiler created ]

...more exception index entries...

Summary:

  Description                        Containing             Size
  -----------------------------  ---------------  ---------------
  Index table, unoptimized:          120 sections        960 bytes
  Optimization removed:               -41 sections       -328 bytes
  -----------------------------  ---------------  ---------------
  Subtotal:                           79 sections        632 bytes
  -----------------------------  ---------------  ---------------
  Unwind data:                        14 sections       2419 bytes
  =============================  ===============  ===============
  Total required:                     93 sections       3051 bytes
  =============================  ===============  ===============
```

See also

--no-map-exception-table on page 105, *--optimize-exception-index* on page 198, *--no-optimize-exception-index* on page 199

4.3.16 --no-map-exception-table

Summary

Include the exception table summary from the generated map file.

Syntax

`--no-map-exception-table`

See also

--map-exception-table on page 104

4.3.17 --map-init-table

Summary

Include an initializaton table map section in the generated map file.

Syntax

--map-init-table

Example

```
*******************************************************************************
***                                                                         ***
***                          INITIALIZATION TABLE                           ***
***                                                                         ***
*******************************************************************************

Zero (__SEGGER_init_zero)
    1 destination range, total size 0xa5
          [0x1fff0444 to 0x1fff04e8]

Copy packing=copy (__SEGGER_init_copy)
    1 source range, total size 0x44 (100% of destination)
          [0x00000fd0 to 0x00001013]
    1 destination range, total size 0x44
          [0x1fff0400 to 0x1fff0443]

Copy packing=copy (__SEGGER_init_copy)
    1 source range, total size 0x10 (100% of destination)
          [0x00001014 to 0x00001023]
    1 destination range, total size 0x10
          [0x1fff04ea to 0x1fff04f9]

Totals
    Table size:     0x30 bytes
    Image size:     0x54 bytes
```

See also

--no-map-init-table on page 107

4.3.18 --no-map-init-table

Summary

Exclude an initializaton table map section from the generated map file.

Syntax

--no-map-init-table

See also

--map-init-table on page 106

4.3.19 --map-listing

Summary

Include an absolute program listing in the generated map file.

Syntax

--map-listing

Example

```
*******************************************************************************
***                                                                         ***
***                          ABSOLUTE LISTING                               ***
***                                                                         ***
*******************************************************************************

;===============================================================================
; Section .init from thumb_crt0.o, alignment 4
;
_start:
  0x0000051C  4924         LDR      R1, [PC, #0x90]
  0x0000051E  4825         LDR      R0, [PC, #0x94]
  0x00000520  1A0A         SUBS     R2, R1, R0
  0x00000522  D002         BEQ      0x0000052A
  0x00000524  2207         MOVS     R2, #7
  0x00000526  4391         BICS     R1, R2
  0x00000528  468D         MOV      SP, R1
  0x0000052A  4923         LDR      R1, [PC, #0x8C]
  0x0000052C  4823         LDR      R0, [PC, #0x8C]
  0x0000052E  1A0A         SUBS     R2, R1, R0
  0x00000530  D006         BEQ      0x00000540
  0x00000532  2207         MOVS     R2, #7
  0x00000534  4391         BICS     R1, R2
```

See also

--no-map-listing on page 109

4.3.20 --no-map-listing

Summary

Exclude an absolute program listing from the generated map file.

Syntax

```
--no-map-listing
```

See also

--map-listing on page 108

4.3.21 --map-listing-with-comments

Summary

Include information comments in absolute listing.

Syntax

```
--map-listing-with-comments
```

Description

When this option is selected, a comment field is appended to the instruction that describes, for instance, the target address of a branch, the effective address of an instruction, and any data loaded by the instruction if known.

This is the default.

See also

--no-map-listing-with-comments on page 111

4.3.22 --no-map-listing-with-comments

Summary

Exclude informational comments from absolute listing.

Syntax

```
--no-map-listing-with-comments
```

Description

When this option is selected, no informational comments are appended to the instruction.

See also

--map-listing-with-comments on page 110

4.3.23 --map-listing-with-data

Summary

Include data sections in the absolute listing.

Syntax

--map-listing-with-data

Description

The default absolute program listing selected by **--map-listing** will only include executable code. The **--map-listing-with-data** option will include data sections (as data) in the absolute listing.

Example

```
*******************************************************************************
***                                                                         ***
***                            ABSOLUTE LISTING                             ***
***                                                                         ***
*******************************************************************************

;*******************************************************************************
; Section .rodata.MainTask.str1.4 from Start_Zynq7007s_emPower.o, size=16, align=4
;

.LC1:
  0xFC019BF0  4C           DC8      0x4C
  0xFC019BF1  50           DC8      0x50
  0xFC019BF2  20           DC8      0x20
  0xFC019BF3  54           DC8      0x54
  0xFC019BF4  61           DC8      0x61
  0xFC019BF5  73           DC8      0x73
  0xFC019BF6  6B           DC8      0x6B
  0xFC019BF7  00           DC8      0x00

;*******************************************************************************
; Section .rodata.OS_L2CACHE_XilinxZynq7000 from Start_Zynq7007s_emPower.o, size=24, align=4
;

OS_L2CACHE_XilinxZynq7000:
  0xFC019BF8  6D8601FC     DC32     0xFC01866D
  0xFC019BFC  A18701FC     DC32     0xFC0187A1
  0xFC019C00  2D8601FC     DC32     0xFC01862D
  0xFC019C04  4D8601FC     DC32     0xFC01864D
  0xFC019C08  598701FC     DC32     0xFC018759
  0xFC019C0C  058701FC     DC32     0xFC018705
```

See also

--no-map-listing-with-data on page 113

4.3.24 --no-map-listing-with-data

Summary

Exclude data section from the absolute listing.

Syntax

--no-map-listing-with-data

Description

This option excludes any read-write, read-only, or zero-initialized data sections from the absolute listing selected by **--map-listing**. Only sections marked to contain executable code will be included in the absolute listing.

See also

--map-listing-with-data on page 112

4.3.25 --map-listing-use-abi-names (RISC-V)

Summary

Use RISC-V ABI register names rather than hard register names.

Syntax

--map-listing-use-abi-names

Description

When this option is selected, instructions are disassembled using ABI register names (e.g. **a1**) in preference to hard register names (e.g. **x11**).

This is the default.

See also

--no-map-listing-use-abi-names (RISC-V) on page 115

4.3.26 --no-map-listing-use-abi-names (RISC-V)

Summary

Use hard register names rather than RISC-V ABI register names.

Syntax

--no-map-listing-use-abi-names

Description

When this option is selected, instructions are disassembled using hard register names (e.g. **x11**) in preference to ABI register names (e.g. **a1**).

See also

--map-listing-use-abi-names (RISC-V) on page 114

4.3.27 --map-listing-use-adr-pseudo (Arm)

Summary

Prefer **ADR** to **ADD R*d*, PC, #*imm*** in absolute listing.

Syntax

--map-listing-use-adr

Description

When this option is selected, the instruction **ADD R*d*, PC, #*imm*** is shown using the standard Arm assembler pseudo-instruction **ADR R*d*, *addr***.

This is the default.

See also

--no-map-listing-use-adr-pseudo (Arm) on page 117

4.3.28 --no-map-listing-use-adr-pseudo (Arm)

Summary

Prefer **ADD R*d*, PC, #*imm*** to **ADR** in absolute listing.

Syntax

--no-map-listing-use-adr-pseudo

Description

When this option is selected, the instruction **ADD R*d*, PC, #*imm*** is shown exactly as it appears in the Arm Architecture Reference Manual.

See also

--map-listing-use-adr-pseudo (Arm) on page 116

4.3.29 --map-listing-use-c-prefix (RISC-V)

Summary

Use a **C.** compact prefix for compact instructions.

Syntax

--map-listing-use-c-prefix

Description

When this option is selected, compact instructions are disassembled using the compact prefix **C.**.

See also

--no-map-listing-use-c-prefix (RISC-V) on page 119

4.3.30 --no-map-listing-use-c-prefix (RISC-V)

Summary

Do not use a **c.** compact prefix for compact instructions.

Syntax

--no-map-listing-use-c-prefix

Description

When this option is selected, compact instructions are disassembled without the **c.** compact prefix which is a more natural way of presenting RISC-V instructions.

This is the default.

See also

--map-listing-use-c-prefix (RISC-V) on page 118

4.3.31 --map-listing-use-ldr-pseudo (Arm)

Summary

Prefer **LDR R*d*, =*data*** to **LDR R*d*, [PC, #*offs*]** in absolute listing.

Syntax

--map-listing-use-ldr-pseudo

Description

When this option is selected, the instruction **LDR R*d*, [PC, #*offs*]** is shown using the standard Arm assembler pseudo-instruction **LDR R*d*, =*data*** whenever possible. The value of ***data*** is the value that will be loaded into **R*d*** by the instruction.

This is the default.

See also

--no-map-listing-use-ldr-pseudo (Arm) on page 121

4.3.32 --no-map-listing-use-ldr-pseudo (Arm)

Summary

Prefer **LDR R*d*, [PC, #*offs*]** to **LDR R*d*, =*data*** in absolute listing.

Syntax

--no-map-listing-use-ldr-pseudo

Description

When this option is selected, the instruction **LDR R*d*, [PC, #*offs*]** is is shown exactly as it appears in the Arm Architecture Reference Manual.

See also

--map-listing-use-ldr-pseudo (Arm) on page 120

4.3.33 --map-listing-xref

Summary

Include a section cross-refernce in the absolute listing.

Syntax

--map-listing-xref

Description

The default absolute program listing selected by **--map-listing** will only include executable code. The **--map-listing-with-data** option will include data sections (as data) in the absolute listing.

Example

```
*******************************************************************************************
***                                                                                     ***
***                                    ABSOLUTE LISTING                                 ***
***                                                                                     ***
*******************************************************************************************

;=========================================================================================
; .text.log
;=========================================================================================
;  Module:     floatops.o (x-libc_v7m_t_le_eabi_small_swc_ienum.a)
;  Attributes: read-only, executable (SHF_EXECINSTR), allocatable (SHF_ALLOC), %progbits
;  Size:       400 (0x190) bytes
;  Align:      4 bytes
;
;  Uses:
;    0x00002818  __aeabi_dadd
;    0x00002C6C  __aeabi_dmul
;    0x00002A56  __aeabi_ddiv
;    0x000031FC  __aeabi_i2d
;    0x000026F4  ldexp
;    0x00003222  frexp
;    0x0000311E  __SEGGER_RTL_float64_PolyEvalP
;    0x0000307C  __SEGGER_RTL_float64_PolyEvalQ
;    0x00000C00  __SEGGER_RTL_float64_Log
;
;  Used by:
;    0x00002618  log1p
;    0x00001CA4  asinh
;    0x00001E68  acosh
;
log:
  0x00000DE4  E92D 4FFE    PUSH.W     {R1-R11, LR}
  0x00000DE8  F04F 32FF    MOV.W      R2, #0xFFFFFFFF
  0x00000DEC  F240 73FE    MOVW       R3, #0x07FE
  0x00000DF0  EB02 5211    ADD.W      R2, R2, R1, LSR #20
  ...
```

See also

--no-map-listing-xref on page 123

4.3.34 --no-map-listing-xref

Summary

Exclude section cross-reference from the absolute listing.

Syntax

```
--no-map-listing-xref
```

Description

This option excludes the section cross-reference from the listing.

See also

--map-listing-xref on page 122

4.3.35 --map-modules

Summary

Include a module breakdown in the generated map file.

Syntax

--map-modules

Example

```
*******************************************************************************************************
***                                                                                                 ***
***                                        MODULE SUMMARY                                           ***
***                                                                                                 ***
*******************************************************************************************************

Memory use by input file:

  Object File                                    RX Code     RO Data     RW Data     ZI Data
  ---------------------------------------------  ----------  ----------  ----------  ----------
  Cortex_M_Startup.o                                 128
  SEGGER_Linker_Support.o                             38
  SEGGER_THUMB_Startup.o                              20
  bench-fp-math-speed-accuracy.o                   1 296     354 759                     96
  ---------------------------------------------  ----------  ----------  ----------  ----------
  Subtotal (objects):                              1 482     354 759                     96
  ---------------------------------------------  ----------  ----------  ----------  ----------
  x-libc_v7m_t_le_eabi_small_swc_ienum.a           6 880         176
  x-libio_v7m_t_le_eabi_small_swc_ienum.a             80           9
  ---------------------------------------------  ----------  ----------  ----------  ----------
  Subtotal (archives):                             6 960         185
  ---------------------------------------------  ----------  ----------  ----------  ----------
  Linker created (init + veneers + exceptions):                     36                  4 096
  =============================================  ==========  ==========  ==========  ==========
  Grand total:                                     8 442     354 980                  4 192
  =============================================  ==========  ==========  ==========  ==========

Detailed memory use by individual object file:

  Object File                                    RX Code     RO Data     RW Data     ZI Data
  ---------------------------------------------  ----------  ----------  ----------  ----------
  bench-fp-math-speed-accuracy.o                   1 296     354 759                     96
  SEGGER_Linker_Support.o                             38
  SEGGER_THUMB_Startup.o                              20
  Cortex_M_Startup.o                                 128
  floatasmops_arm.o (x-libc_v7m_t_le_eabi_small_swc_ienum.a)
                                                   2 488
  floatops.o (x-libc_v7m_t_le_eabi_small_swc_ienum.a)
                                                   4 392         176
  support-io.o (x-libio_v7m_t_le_eabi_small_swc_ienum.a)
                                                      58           9
  SEGGER_SEMIHOST.o (x-libio_v7m_t_le_eabi_small_swc_ienum.a)
                                                      18
  SEGGER_SEMIHOST_Generic.o (x-libio_v7m_t_le_eabi_small_swc_ienum.a)
                                                       4
  [ Linker created ]                                                36                  4 096
  =============================================  ==========  ==========  ==========  ==========
  Total:                                           8 442     354 980                  4 192
  =============================================  ==========  ==========  ==========  ==========
```

See also

--no-map-modules on page 125

4.3.36 --no-map-modules

Summary

Exclude a module breakdown from the generated map file.

Syntax

`--no-map-modules`

See also

--map-modules on page 124

4.3.37 --map-module-detail

Summary

Include a detailed section breakdown by module in the generated map file.

Syntax

--map-module-detail

Example

```
*******************************************************************************
***                                                                         ***
***                              MODULE DETAIL                              ***
***                                                                         ***
*******************************************************************************

Module BSP.o:

  Section                                     Code   RO Data   RW Data   ZI Data
  ------------------------------------------  -----  --------  --------  --------
  .text.BSP_Init                               122
  .text.BSP_SetLED                             110
  .BootHeader                                          2240
  .bss._LEDState                                                            4
  ==========================================  =====  ========  ========  ========
  Total:                                       232      2240                 4
  ==========================================  =====  ========  ========  ========

Module BSP_FPGA.o:

  Section                                     Code   RO Data   RW Data   ZI Data
  ------------------------------------------  -----  --------  --------  --------
  .text.BSP_FPGA_Init                          442
  .text._SendFPGAConfigData                    250
  .rodata._aFPGAData                                  94978
  .bss._PLCfgByteCnt                                                        4
  .bss._FPGAConfigNumBytesInBuf                                             4
  .bss._acPLCfgBuff                                                        64
  ==========================================  =====  ========  ========  ========
  Total:                                       692     94978                72
  ==========================================  =====  ========  ========  ========

...more modules...

All modules:

                                              Code   RO Data   RW Data   ZI Data
  ==========================================  =====  ========  ========  ========
  Grand total:                                9292     99181         4     21523
  ==========================================  =====  ========  ========  ========
```

See also

--no-map-module-detail on page 127

4.3.38 --no-map-module-detail

Summary

Exclude a detailed section breakdown by module from the generated map file.

Syntax

`--no-map-module-detail`

See also

--map-module-detail on page 126

4.3.39 --map-placement

Summary

Include a section placement section in the generated map file.

Syntax

--map-placement

Example

```
*******************************************************************************
***                                                                         ***
***                           PLACEMENT SUMMARY                             ***
***                                                                         ***
*******************************************************************************

place at 0x00000000

  Section                  Type  Address       Size  Object
  -------                  ----  -------       ----  ------
  .vectors                 Code  00000000     0x410  MK66F18_Vectors.o

"RAM": place in [0x1fff0000 to 0x1fffffff] | [0x20000000 to 0x2002ffff]

  Section                  Type  Address       Size  Object
  -------                  ----  -------       ----  ------
  .data.OS_Global          Init  1fff0400      0x40  OS_Global.o (OS.a)
  .data.x                  Init  1fff0440       0x4  Main.o
  .bss..L_MergedGlobals    Zero  1fff0444       0xc  BSP_IP.o
  .bss.OS_pTickHookRoot    Zero  1fff0450       0x4  OS_Global.o (OS.a)
  .bss.OS_pMainContext     Zero  1fff0454       0x4  OS_Global.o (OS.a)

...more sections...

"FLASH": place in [0x00000000 to 0x001fffff]

  Section                  Type  Address       Size  Object
  -------                  ----  -------       ----  ------
  .init                    Code  00000410      0x44  Kinetis_K60_Startup.o
  .init                    Code  00000454      0xc8  MK66F18_Vectors.o
  .init                    Code  0000051c      0xcc  thumb_crt0.o
  .rodata.OS_JLINKMEM_BufferSize
                           Cnst  000005e8       0x4  RTOSInit_K66F_CMSIS.o
  .text                    Code  000005ec     0x198  RTOS.o (OS.a)
  .text.libc._execute_at_exit_fns
                           Code  000007b4      0x28  libc2.o (libc_v7em_fpv4_sp...
  .text.libc.memcpy        Code  000007dc      0x54  libc2_asm.o (libc_v7em_fpv...
  .text.main               Code  00000830      0x1c  Main.o

...more sections...
```

See also

--no-map-placement on page 129

4.3.40 --no-map-placement

Summary

Exclude a section placement section from the generated map file.

Syntax

```
--no-map-placement
```

See also

--map-placement on page 128

4.3.41 --map-script

Summary

Include a listing of all input scripts in the generated map file and include the invocation command line.

Syntax

```
--map-script
```

See also

--no-map-script on page 131

4.3.42 --no-map-script

Summary

Exclude a listing of all input scripts in the generated map file and exclude the invocation command line.

Syntax

```
--no-map-script
```

See also

--map-script on page 130

4.3.43 --map-section-detail

Summary

Include a detailed section breakdown in the generated map file.

Syntax

--map-section-detail

Example

```
*******************************************************************************
***                                                                         ***
***                              SECTION DETAIL                             ***
***                                                                         ***
*******************************************************************************

Sections by address:

  Section                  Range                  Size  Align  Type  Object File
  -----------------------  -----------------  --------  -----  ----  -----------
  .non_init                00010000-00013fff    0x4000  16384  Zero  RTOSInit_XC7Z007S.o
  .no_init                 00014000-0001417f     0x180      4  Zero  RTOSInit_XC7Z007S.o
  .no_init                 00014180-00014187       0x8      4  Zero  OS_ARMv7_MSA.o
  stack_fiq                00014188-00014287     0x100      8  Cnst  - Linker created -
  stack_irq                00014288-00014387     0x100      8  Cnst  - Linker created -
  .data.libc.__SEGGER_RTL_pHeap
                           00014388-0001438b       0x4      4  Data  heapops.o
 (libc_v7a_a_le_eabi_small.a)
  .bss.OS_SysTimer_Settings
                           0001438c-000143ab      0x20      4  Zero  OS_Global.o
  .bss._LEDState           000143ac-000143af       0x4      4  Zero  BSP.o
  .bss._PLCfgByteCnt       000143b0-000143b3       0x4      4  Zero  BSP_FPGA.o
  .bss._FPGAConfigNumBytesInBuf
                           000143b4-000143b7       0x4      4  Zero  BSP_FPGA.o
  .bss._SpuriousIrqCnt     000143b8-000143bb       0x4      4  Zero  RTOSInit_XC7Z007S.o
  .bss.OS_COM_pTask        000143bc-000143bf       0x4      4  Zero  OS_Com.o
  .bss._acPLCfgBuff        000143c0-000143ff      0x40     64  Zero  BSP_FPGA.o
  .bss.OS_Global           00014400-0001443f      0x40      8  Zero  OS_Global.o
```

See also

--no-map-section-detail on page 133

4.3.44 --no-map-section-detail

Summary

Exclude a detailed section breakdown from the generated map file.

Syntax

```
--no-map-section-detail
```

See also

--map-section-detail on page 132

4.3.45 --map-summary

Summary

Include a link summary in the generated map file.

Syntax

--map-summary

Example

```
*******************************************************************************
***                                                                         ***
***                                LINK SUMMARY                             ***
***                                                                         ***
*******************************************************************************

Load summary:

  Name     Range                 Size            Used           Unused  Alignment Loss
  -------  -----------------  --------  --------------  ---------------  --------------
  FLASH    00000000-000fffff   1048576     160   0.02%  1048416  99.98%       0   0.00%
  RAM      20000000-2000ffff     65536    2048   3.13%    63488  96.88%       0   0.00%

Link complete: 0 errors, 0 warnings, 0 remarks
```

See also

--no-map-summary on page 135

4.3.46 --no-map-summary

Summary

Exclude a link summary from the generated map file.

Syntax

```
--no-map-summary
```

See also

--map-summary on page 134

4.3.47 --map-symbols

Summary

Include a symbol map section in the generated map file.

Syntax

--map-symbols

Example

```
*******************************************************************************
***                                                                         ***
***                              SYMBOL LIST                                ***
***                                                                         ***
*******************************************************************************

Symbols by value

  Symbol                   Value         Size  Type      Object
  ------                   -----         ----  ----      ------
  _vectors                 00000000     0x410  Code  Gb  MK66F18_Vectors.o
  __FLASH_segment_used_start__
                           00000000           ----  Gb  - Linker created -
  Reset_Handler            00000411      0x44  Code  Gb  Kinetis_K60_Startup.o
  NMI_Handler              00000455      0xc8  Code  Wk  MK66F18_Vectors.o
  HardFault_Handler        00000457      0xc8  Code  Wk  MK66F18_Vectors.o

...more symbols...

Symbols by name

  Symbol                   Value         Size  Type      Object
  ------                   -----         ----  ----      ------
  ADC0_IRQHandler          000004ad      0xc8  Code  Wk  MK66F18_Vectors.o
  main                     00000831      0x1c  Code  Gb  Main.o

...more symbols...
```


4.3.48 --no-map-symbols

Summary

Exclude a symbol map section from the generated map file.

Syntax

--no-map-symbols

See also

--map-symbols on page 136

4.3.49 --map-unused-inputs

Summary

Include a summary of unused object files in the generated map file.

Syntax

```
--map-unused-inputs
```

Description

This option will instruct the linker to list the object files that were not used when linking the application. This list can be used to reduce the number of object files to only those requires to successfully link an application.

See also

--no-map-unused-inputs on page 139

4.3.50 --no-map-unused-inputs

Summary

Exclude a summary of unused object files in the generated map file.

Syntax

`--no-map-unused-inputs`

See also

--map-unused-inputs on page 138

4.3.51 --map-unused-memory

Summary

Include a summary of unused memory in the generated map file.

Syntax

--map-unused-memory

Description

The linker will automatically generate a listing of all unused memory regions and the reason they are unused.

Example

```
*******************************************************************************
***                                                                         ***
***                            UNUSED MEMORY SUMMARY                        ***
***                                                                         ***
*******************************************************************************

Detail:

  Range                  Size  Reason
  -----------------  --------  ----------------------------------------------------
  00011873-00011877         5  Unused memory between sections '.bss.Heap_IsInited'
                               and 'heap'
  00011978-00013fff      9864  Unused memory between sections 'heap' and '.non_init'
  00018000-0003fbff    162816  Unused memory between sections '.non_init' and 'stack'
  fc017c12-fc017c13         2  Filler between sections '.rodata._aFPGAData' and
                               '.text.BSP_SetLED' as align=4
                               NOTE: section '.rodata._aFPGAData' has size 94978 that
                               is not a multiple of its alignment 16
  fc01a819-fc01a81b         3  Filler between sections '.rodata.str1.4' and
                               '.rodata.OS_DebugInfo' as align=4
                               NOTE: section '.rodata.str1.4' has size 21 that is not
                               a multiple of its alignment 4
  fc01a836-fc01a837         2  Filler between sections '.rodata.OS_DebugInfo' and
                               '.rodata.main.str1.4' as align=4
                               NOTE: section '.rodata.OS_DebugInfo' has size 26 that
                               is not a multiple of its alignment 4
  fc01a881-fc01a883         3  Unused memory between sections '.rodata.libc.__aeabi_uidiv'
                               and '.segger.init.table'
```

See also

--no-map-unused-memory on page 141

4.3.52 --no-map-unused-memory

Summary

Exclude summary of unused memory from the generated map file.

Syntax

```
--no-map-unused-memory
```

See also

--map-unused-memory on page 140

4.3.53 --map-veneers

Summary

Include a summary of linker-created veneers in the generated map file.

Syntax

--map-veneers

Description

The linker will automatically generate veneers when required. The linker constructs two types of veneer:

- a *range extension* veneer when a branch or call cannot reach the target address.
- a *mode switch* veneer when a branch or call requires a mode switch from one instruction set to another.

A range extension veneer is required when execution flows over two distant memory regions, such as flash and RAM, and these regions are far apart. Time-critical functions are usually placed in RAM to eliminate the stalls inherent with flash memory and flash accelerators and, therefore, execute code with known timing. If the call from flash to RAM is too far, the linker inserts a small amount of code between functions, close to the call, that will *trampoline* the call to the target with a small code and execution penalty.

A mode switch veneer is required when the calling function is written in a different instruction set to the called function. For instance, it could be that compute-intensive code is written in the Arm instruction set but the remainder of the application is written in the Thumb instruction set to reduce code size. When the Arm code calls the Thumb code, a mode switch veneer may be required if there is no instruction set support for the mode switch.

The **--map-veneers** option enables a listing of all the constructed veneers for inspection. With this information, the user is able to check that functions are coded in the correct instruction set and that they reside in the correct memory range in order to avoid undue veneer construction by the linker.

Example

```
*******************************************************************************
***                                                                         ***
***                          LINKER-CREATED VENEERS                         ***
***                                                                         ***
*******************************************************************************

*** Arm-Arm range extension veneers:

              Target                          Source
  Veneer Size Address   Symbol                Address   Section
  ----------- --------  --------------------  --------  --------------------
            0 Total
  ----------- ------------------------------

*** Thumb-Thumb range extension veneers:

              Target                          Source
  Veneer Size Address   Symbol                Address   Section
  ----------- --------  --------------------  --------  --------------------
            8 00000475  _EraseSector          fc2010c8  .text.FLASH_WriteEx+148
            8 00000475  _EraseSector          fc201114  .text.FLASH_EraseSector+64
            8 000004e1  _Program              fc2010c0  .text.FLASH_WriteEx+140
            8 0000055d  _GetUID                fc201148  .text.FLASH_GetUID+48
  ----------- ------------------------------
           32 Total
  ----------- ------------------------------

*** Arm-Thumb mode switch veneers:

              Target                          Source
```



```
  Veneer Size  Address   Symbol                Address   Section
  -----------  --------  --------------------  --------  --------------------
           12  fc201e45  Undefined_Handler     fc27d78c  .text+336
  -----------  ------------------------------
           12  Total
  -----------  ------------------------------

*** Thumb-Arm mode switch veneers:

               Target                          Source
  Veneer Size  Address   Symbol                Address   Section
  -----------  --------  --------------------  --------  --------------------
           12  fc200800  HW_Delay_us           fc201284  .text.HW_ActivateTargetPower+52
           12  fc200800  HW_Delay_us           fc203110  .text.HW_S8_Stop+48
           12  fc200a54  OS_EnableInt          fc202130  .text.OS_InitHW+320
           12  fc200a54  OS_EnableInt          fc2228c0  .text.USB_OS_DecRI+56
           12  fc200a54  OS_EnableInt          fc228b70  .text.IP_OS_EnableInterrupt+56
           12  fc200a54  OS_EnableInt          fc2305d8  .text.OS_EVENT_GetBlocked+236
           12  fc200a54  OS_EnableInt          fc230a68  .text.OS_EnableProfiling+128
           12  fc200a54  OS_EnableInt          fc230af8  .text.OS_TASK_LeaveRegion+120
           12  fc200a54  OS_EnableInt          fc230b30  .text.OS_INT_EnableConditional+52
           12  fc200a54  OS_EnableInt          fc230f24  .text.OS_MakeTaskReady+172
           12  fc200a54  OS_EnableInt          fc230f74  .text.OS_ClearWaitObj+76
           12  fc200a54  OS_EnableInt          fc231488  .text.OS_Deactivated+88
           12  fc200a54  OS_EnableInt          fc231628  .text._PutMail+112
           12  fc200a54  OS_EnableInt          fc2316c0  .text._GetMail+148
           12  fc200a54  OS_EnableInt          fc232c18  .text._CleanRange+96
           12  fc200a54  OS_EnableInt          fc232c74  .text._Clean+88
           12  fc200a54  OS_EnableInt          fc232cd0  .text._Sync+88
           12  fc200a54  OS_EnableInt          fc2374b8  .text._AddToConnectCnt+100
  -----------  ------------------------------
          216  Total
  ===========  ==============================
          260  Grand Total
  -----------  ------------------------------
```

The linker will automatically reuse veneers when it can:

```
*******************************************************************************
***                                                                         ***
***                          LINKER-CREATED VENEERS                         ***
***                                                                         ***
*******************************************************************************

*** Arm-Arm range extension veneers:

               Target                          Source
  Veneer Size  Address   Symbol                Address   Section
  -----------  --------  --------------------  --------  --------------------
            8  00014690  LowArmLeaf            fc017f54  .text.HighArmAux+88
            8  00014690  LowArmLeaf            fc0180c4  .text.HighArm+340
     (reused)  00014690  LowArmLeaf            fc017f98  .text.HighArm+40
     (reused)  00014690  LowArmLeaf            fc018070  .text.HighArm+256
     (reused)  00014690  LowArmLeaf            fc0180b0  .text.HighArm+320
            8  fc0008c0  HighArmLeaf           00014624  .fast.LowArmAux+88
            8  fc0008c0  HighArmLeaf           000144d4  .fast.LowArm+332
     (reused)  fc0008c0  HighArmLeaf           000143a4  .fast.LowArm+28
     (reused)  fc0008c0  HighArmLeaf           00014458  .fast.LowArm+208
     (reused)  fc0008c0  HighArmLeaf           00014498  .fast.LowArm+272
  -----------  ------------------------------
           32  Total
  -----------  ------------------------------
```

See also

--no-map-veneers on page 144

4.3.54 --no-map-veneers

Summary

Exclude a summary of linker-created veneers.

Syntax

--no-map-veneers

See also

--map-veneers on page 142

4.4 Log file options

4.4.1 --log-file

Summary

Generate a linker log file.

Syntax

--log-file *filename*
--log-file=*filename*

Description

Generates a linker log file to the given filename.

4.5 Preprocessor options

4.5.1 --define-macro

Summary

Define preprocessor macro.

Syntax

-define-macro *name*[**=***text*]
-D*name*[**=***text*]

Description

Define the preprocessor macro **name** as *text*. Preprocessor macros are replaced in the linker script and are not related to Elf symbols defined with **--define-symbol**.

4.5.2 --include

Summary

Add directory to include path.

Syntax

--include *dir*
-I*dir*

Description

Add the directory *dir* to the end of the paths to search for included files.

4.6 Symbol and section options

4.6.1 --add-region

Summary

Add memory region.

Syntax

--add-region *region-name=size@addr*
--add-region:*region-name=size@addr*
--add-region=*region-name=size@addr*

Description

Add a region in addition to those defined in the linker script. This option allows you to write a generic linker script file for your selected toolset and to parameterize the target's memory regions (typically RAM and flash) from the command line.

It is possible to define a non-continguous memory region from the command line using two **--add-region** options, for example:

```
--add-region:RAM=64k@0x1fff0000 --add-region:RAM=192k@0x20000000
```

is equivalent to the linker script fragment:

```
define region RAM = [0x1fff0000 size 64k] | [0x20000000 size 192k];
```

See also

define region statement on page 50

4.6.2 --autoat

Summary

Enable automatic placement of sections.

Syntax

`--autoat`

Description

When enabled, the linker places sections that conform to the “auto-at” naming convention at the designated position in the linked image.

See also

--no-autoat on page 151

4.6.3 --no-autoat

Summary

Disable automatic placement of sections.

Syntax

`--no-autoat`

Description

Turns off automatic placement of sections which use the “auto-at” naming convention.

See also

--autoat on page 150

4.6.4 --autokeep

Summary

Enable automatic retention of sections.

Syntax

--autokeep

Description

When enabled, the linker retains all initializer and finalizer sections in all input files so they can be selected and placed as appropriate for the runtime system. The linker defaults to autokeep enabled.

Enabling autokeep is identical to placing the following in the linker script:

```
keep { init_array, fini_array };
```

See also

--no-autokeep on page 153

4.6.5 --no-autokeep

Summary

Disable automatic retention of sections.

Syntax

--no-autokeep

Description

The linker does not automatically keep initialization and finalization sections and it is the user's responsibility to keep and select them in the linker script.

See also

--autokeep on page 152

4.6.6 --auto-arm-symbols

Summary

Enable automatic generation of Armlink symbols.

Syntax

--auto-arm-symbols

Description

Turns on automatic generation of Armlink symbols. The symbols defined in this mode are:

***B*$$Start**
***B*$$Limit**
***B*$$Length**

where *B* is the name of a block declared using **define block**.

See also

--no-auto-arm-symbols on page 155

4.6.7 --no-auto-arm-symbols

Summary

Disable automatic generation of Armlink symbols.

Syntax

```
--no-auto-arm-symbols
```

Description

Turns off automatic generation of Armlink symbols.

See also

--auto-arm-symbols on page 154

4.6.8 --auto-es-symbols

Summary

Enable automatic generation of Embedded Studio symbols.

Syntax

--auto-es-symbols

Description

Turns on automatic generation of Embedded Studio symbols. This option is equivalent to specifying **--auto-es-block-symbols** and **--auto-es-region-symbols**.

The symbols defined in this mode are:

__*B*_start__
__*B*_end__
__*B*_size__
__*B*_start
__*B*_end
__*B*_size

where ***B*** is the name of a block declared using **define block**, and:

__*R*_segment_start__
__*R*_segment_end__
__*R*_segment_size__
__*R*_segment_used_start__
__*R*_segment_used_end__
__*R*_segment_used_size__

where ***R*** is the name of a memory region created by a **define region** script command or by the **--add-region** command line option.

See also

--auto-es-block-symbols on page 158, *--auto-es-region-symbols* on page 160, *--no-auto-es-symbols* on page 157

4.6.9 --no-auto-es-symbols

Summary

Disable automatic generation of Embedded Studio symbols.

Syntax

`--no-auto-es-symbols`

Description

Turns off automatic generation of Embedded Studio symbols.

See also

--auto-es-symbols on page 156

4.6.10 --auto-es-block-symbols

Summary

Enable automatic generation of Embedded Studio block symbols.

Syntax

--auto-es-block-symbols

Description

The symbols defined in this mode are:

__*B*_start__
__*B*_end__
__*B*_size__
__*B*_start
__*B*_end
__*B*_size

where *B* is the name of a block declared using **define block**.

See also

--no-auto-es-block-symbols on page 159

4.6.11 --no-auto-es-block-symbols

Summary

Disable automatic generation of Embedded Studio block symbols.

Syntax

--no-auto-es-block-symbols

Description

Turns off automatic generation of Embedded Studio block symbols.

See also

--auto-es-block-symbols on page 158

4.6.12 --auto-es-region-symbols

Summary

Enable automatic generation of Embedded Studio region symbols.

Syntax

--auto-es-region-symbols

Description

Turns on automatic generation of Embedded Studio region symbols. The symbols defined in this mode are:

__*R*_segment_start__
__*R*_segment_end__
__*R*_segment_size__
__*R*_segment_used_start__
__*R*_segment_used_end__
__*R*_segment_used_size__

where *R* is the name of a memory region created by a **define region** script command or by the **--add-region** command line option.

See also

--no-auto-es-region-symbols on page 161

4.6.13 --no-auto-es-region-symbols

Summary

Disable automatic generation of Embedded Studio region symbols.

Syntax

--no-auto-es-region-symbols

Description

Turns off automatic generation of Embedded Studio region symbols.

See also

--auto-es-region-symbols on page 160

4.6.14 --define-symbol

Summary

Define linker symbol.

Syntax

--define-symbol *name=number* | *name*

--defsym:*name=number* | *name*
--defsym=*name=number* | *name*

-defsym:*name=number* | *name*
-defsym=*name=number* | *name*

Description

Define the symbol *name* as either a number or the value of an existing symbol.

4.6.15 --enable-lzss

Summary

Enable LZSS algorithm in auto packing selection.

Syntax

`--enable-lzss`

Description

When this option is specified, the LZSS compression algorithm is a a candidate algorithm when selecting a packing algorithm using **`packing=auto`**. This is the default.

See also

--disable-lzss on page 164

4.6.16 --disable-lzss

Summary

Disable LZSS algorithm in auto packing selection.

Syntax

--disable-lzss

Description

When this option is specified, the LZSS compression algorithm is not a candidate algorithm when selecting a packing algorithm using **packing=auto**.

Note that disabling LZSS compression in this manner does not prevent it from being selected as a packing algorithm using **with packing=lzss** in an **initialize by copy** statement.

See also

--enable-lzss on page 163

4.6.17 --enable-packbits

Summary

Enable PackBits algorithm in auto packing selection.

Syntax

`--enable-packbits`

Description

When this option is specified, the PackBits compression algorithm is a a candidate algorithm when selecting a packing algorithm using **`packing=auto`**. This is the default.

See also

--disable-packbits on page 166

4.6.18 --disable-packbits

Summary

Disable PackBits algorithm in auto packing selection.

Syntax

--enable-packbits

Description

When this option is specified, the PackBits compression algorithm is not a candidate algorithm when selecting a packing algorithm using **packing=auto**.

Note that disabling PackBits compression in this manner does not prevent it from being selected as a packing algorithm using **with packing=packbits** in an **initialize by copy** statement.

See also

--enable-packbits on page 165

4.6.19 --enable-zpak

Summary

Enable ZPak algorithm in auto packing selection.

Syntax

`--enable-zpak`

Description

When this option is specified, the SEGGER ZPak compression algorithm is a a candidate algorithm when selecting a packing algorithm using **`packing=auto`**. This is the default.

See also

--disable-zpak on page 168

4.6.20 --disable-zpak

Summary

Disable ZPak algorithm in auto packing selection.

Syntax

`--disable-zpak`

Description

When this option is specified, the SEGGER ZPak compression algorithm is not a candidate algorithm when selecting a packing algorithm using **`packing=auto`**.

Note that disabling ZPak compression in this manner does not prevent it from being selected as a packing algorithm using **`with packing=zpak`** in an **`initialize by copy`** statement.

See also

--enable-zpak on page 167

4.6.21 --keep-init-array

Summary

Keep section.

Syntax

--keep-init-array=(**none** | **if-referenced** | **for-objects** | **all**)

Description

This option controls how the linker includes sections that are initialization arrays. Initialization array sections are typically generated by static constructors in C++ code.

The options are:

none
: All initialization arrays are discarded.

if-referenced
: An object file's initialization array section is included in the link only if some other section in the object file is also required when linking.

for-objects
: Any object file provided on the command line always inclues the initialization array irrespective of whether it is required or not, which matches the GNU linker's behavior. Initialization arrays from object files in libraries are processed using **if-referenced** semantics, i.e. they are only included if a section from the archive's object file is required.

all
: Initialization arrays encountered in object files, whether from the command linke or from a library, are always included in the link.

The default for this option is **for-objects**.

4.6.22 --keep-section

Summary

Keep section.

Syntax

--keep-section *name*
--keep-section=*name*

Description

The linker keeps code and data reachable from root symbols, such as the entry point symbol, and discards all other sections. If an application image must contain some code or data (such as configuration or personalization data) that is not directly reachable from the root symbols, use this option to instruct the linker to treat additional section names as roots.

Note that the name can be a wildcard, such as **.text.keep.***.

4.6.23 --keep-symbol

Summary

Keep symbol.

Syntax

--keep-symbol *name*
--keep-symbol=*name*
--keep *name*
--keep=*name*

Description

The linker keeps code and data reachable from root symbols, such as the entry point symbol, and discards all other sections. If an application image must contain some code or data (such as configuration or personalization data) that is not directly reachable from the root symbols, use this option to instruct the linker to treat additional symbols as roots.

4.6.24 --min-align-code

Summary

Set minimum code section alignment.

Description

Please see *--min-align-rx* on page 178.

4.6.25 --min-align-data

Summary

Set minimum data section alignment.

Syntax

--min-align-data=*value*

Description

This option overrides the alignment of all data section (read-only, read-write, zero-initialized) from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

This options can be useful to force data that would be aligned on an odd byte address to an even or word-aligned address. Some example applications from silicon vendors have made assumptions regarding alignment of data and will fail to work correctly, usually failing with a misaligned load or store resulting in a hard fault. This is not a fault of the linker, which honors all alignment requirements, but increasing the alignment of all data to at least four bytes can quickly identify if a hard fault is caused by alignment assumptions.

See also

--min-align-ro on page 175, *--min-align-rw* on page 177, *--min-align-rx* on page 178

4.6.26 --min-align-ram

Summary

Set minimum read-write and zero-initialized section alignment.

Syntax

--min-align-ram=*value*

Description

This option overrides the alignment of read-write and zero-initialized sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

See also

--min-align-rw on page 177, *--min-align-zi* on page 179

4.6.27 --min-align-ro

Summary

Set minimum read-only data section alignment.

Syntax

--min-align-ro=*value*

Description

This option overrides the alignment of all read-only data sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

This options can be useful to force data that would be aligned on an odd byte address to an even or word-aligned address. Some example applications from silicon vendors have made assumptions regarding alignment of data and will fail to work correctly, usually failing with a misaligned load or store resulting in a hard fault. This is not a fault of the linker, which honors all alignment requirements, but increasing the alignment of all data to at least four bytes can quickly identify if a hard fault is caused by alignment assumptions.

See also

--min-align-data on page 173, *--min-align-rw* on page 177, *--min-align-zi* on page 179

4.6.28 --min-align-rom

Summary

Set minimum read-execute and read-only section alignment.

Syntax

--min-align-rom=*value*

Description

This option overrides the alignment of read-execute and read-only sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024). Read-execute and read-only sections are typically allocated to flash memory on an embedded system.

See also

--min-align-rw on page 177, *--min-align-zi* on page 179

4.6.29 --min-align-rw

Summary

Set minimum read-write data section alignment.

Syntax

--min-align-rw=*value*

Description

This option overrides the alignment of all read-write data sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

This options can be useful to force data that would be aligned on an odd byte address to an even or word-aligned address. Some example applications from silicon vendors have made assumptions regarding alignment of data and will fail to work correctly, usually failing with a misaligned load or store resulting in a hard fault. This is not a fault of the linker, which honors all alignment requirements, but increasing the alignment of all data to at least four bytes can quickly identify if a hard fault is caused by alignment assumptions.

See also

--min-align-data on page 173, *--min-align-ro* on page 175, *--min-align-zi* on page 179

4.6.30 --min-align-rx

Summary

Set minimum code section alignment.

Syntax

--min-align-rx=*value*
--min-align-code=*value*

Description

This option overrides the alignment of read-execute sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

This options can be useful to force functions that would have two-byte alignment to be aligned on four-byte boundaries which may increase the effectiveness of flash accelerators and thereby increase execution speed (but at the expense of some wasted flash storage for the alignment).

It can also be used to increase the alignment of all functions to 16-byte boundaries that some flash accelerators benefit from.

4.6.31 --min-align-zi

Summary

Set minimum zero-initialized data section alignment.

Syntax

--min-align-zi=*value*

Description

This option overrides the alignment of all zero-initialized data sections from input object files and libraries such that each section has a minimum alignment requirement of *value* bytes (which is limited to 1024).

This options can be useful to force data that would be aligned on an odd byte address to an even or word-aligned address. Some example applications from silicon vendors have made assumptions regarding alignment of data and will fail to work correctly, usually failing with a misaligned load or store resulting in a hard fault. This is not a fault of the linker, which honors all alignment requirements, but increasing the alignment of all data to at least four bytes can quickly identify if a hard fault is caused by alignment assumptions.

See also

--min-align-data on page 173, *--min-align-ro* on page 175, *--min-align-rw* on page 177

4.6.32 --pad-all

Summary

Pad all sections.

Syntax

--pad-all

Description

Pad all sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any code or read-only data section and `0x00` bytes to the end of any read-write or zero-initialized section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying specifying **--pad-ro**, **--pad-rw**, **--pad-rx**, **--pad-zi**.

See also

--pad-ro on page 185, *--pad-rw* on page 187, *--pad-rx* on page 188, *--pad-zi* on page 189

4.6.33 --pad-code

Summary

Pad code sections.

Syntax

`--pad-code`

Description

Pad all code sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any code section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying **`--pad-rx`**.

See also

--pad-rx on page 188

4.6.34 --pad-data

Summary

Pad data sections.

Syntax

--pad-data

Description

Pad all data sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any read-only data section and `0x00` bytes to the end of any read-write or zero-initialized section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying **--pad-ro**, **--pad-rw**, **--pad-zi**.

See also

--pad-ro on page 185, *--pad-rw* on page 187, *--pad-zi* on page 189

4.6.35 --pad-none

Summary

Do not pad sections.

Syntax

```
--pad-none
```

Description

Turn of padding for all sections, i.e. no section is padded.

4.6.36 --pad-ram

Summary

Pad RAM-based sections.

Syntax

`--pad-ram`

Description

Pad all RAM-based sections to a multiple of their alignment. The linker adds `0x00` bytes to the end of any read-write or zero-initialized when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying **`--pad-rw`**, **`--pad-zi`**.

See also

--pad-rw on page 187, *--pad-zi* on page 189

4.6.37 --pad-ro

Summary

Pad read-only data sections.

Syntax

--pad-ro

Description

Pad all read-only data sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any read-only data section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

4.6.38 --pad-rom

Summary

Pad ROM-based sections.

Syntax

--pad-rom

Description

Pad all ROM-based sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any code or read-only data section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying **--pad-ro**, **--pad-rx**.

See also

--pad-ro on page 185, *--pad-rx* on page 188

4.6.39 --pad-rw

Summary

Pad read-write sections.

Syntax

`--pad-rw`

Description

Pad all read-write sections to a multiple of their alignment. The linker adds `0x00` bytes to the end of any read-write section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

See also

--pad-data on page 182, *--pad-zi* on page 189

4.6.40 --pad-rx

Summary

Pad code sections.

Syntax

`--pad-rx`

Description

Pad all code sections to a multiple of their alignment. The linker adds `0xFF` bytes to the end of any code section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

Note

This option is equivalent to specifying **`--pad-code`**.

See also

--pad-code on page 181

4.6.41 --pad-zi

Summary

Pad read-write sections.

Syntax

`--pad-zi`

Description

Pad all zero-initialized sections to a multiple of their alignment. The linker adds `0x00` bytes to the end of any zero-initialized section when padding. A maximum alignment of 16 is enforced for padding such that a maximum of 15 bytes are added to the end of the section in order to limit the maximum expansion of a section.

See also

--pad-data on page 182, *--pad-rw* on page 187

4.6.42 --pretty-section-names

Summary

Demangle Elf section names to human-readable C++ section names

Syntax

--pretty-section-names

Description

The names of Elf sections containing C++ functions and objects are encoded by the compiler to produce an acceptable ELF section name in a process generally known as "mangling". Unfortunately this name is incomprehensible by the average user as it doesn't directly relate to the C++ souce code.

With this option, the linker recognizes these encoded ELF section names and "demangles" them to a human-readable section names referring to the C++ object or function name.

This is the default.

See also

--no-pretty-section-names on page 191

4.6.43 --no-pretty-section-names

Summary

Do not demangle Elf section names to human-readable C++ section names

Syntax

```
--no-pretty-section-names
```

Description

The names of C++ functions and objects are encoded by the compiler to produce an acceptable ELF section name in a process generally known as "mangling". Unfortunately this name is incomprehensible by the average user as it doesn't directly relate to the C++ souce code.

With this option, the linker leaves section names in their native state, useful for tool developers and integrators.

See also

--pretty-section-names on page 190

4.6.44 --pretty-symbol-names

Summary

Demangle Elf symbols to human-readable C++ symbols

Syntax

--pretty-symbol-names

Description

The names of C++ functions and objects are encoded by the compiler to produce an acceptable ELF symbol name in a process generally known as "mangling". Unfortunately this name is incomprehensible by the average user as it doesn't directly relate to the C++ souce code.

With this option, the linker recognizes these encoded ELF symbols and "demangles" them to a human-readable C++ object or function name.

This is the default.

See also

--no-pretty-symbol-names on page 193

4.6.45 --no-pretty-symbol-names

Summary

Do not demangle Elf symbols to human-readable C++ symbols

Syntax

```
--no-pretty-symbol-names
```

Description

The names of C++ functions and objects are encoded by the compiler to produce an acceptable ELF symbol name in a process generally known as "mangling". Unfortunately this name is incomprehensible by the average user as it doesn't directly relate to the C++ souce code.

With this option, the linker leaves symbol names in their native state, useful for tool developers and integrators.

See also

--pretty-symbol-names on page 192

4.6.46 --undefined-weak-is-zero

Summary

Undefined weak references are replaced with zero.

Syntax

--undefined-weak-is-zero

Description

All references to an undefined weak symbol are resolved as if the weak symbol had the value zero.

Note

The Arm EABI requires that direct calls to undefined weak symbols are replaced with no-operation instructions rather than a call to location zero. This behavior is implemented irrespective of the **--no-undefined-weak-is-zero** and **--undefined-weak-is-zero** setting.

See also

--undefined-weak-is-zero on page 194

4.6.47 --no-undefined-weak-is-zero

Summary

Report undefined weak references as errors.

Syntax

--no-undefined-weak-is-zero

Description

A reference to an undefined weak symbol is reported as an error. This is the default setting.

Note

The Arm EABI requires that direct calls to undefined weak symbols are replaced with no-operation instructions. This behavior is implemented irrespective of the **--no-undefined-weak-is-zero** and **--undefined-weak-is-zero** setting.

See also

--undefined-weak-is-zero on page 194

4.6.48 --unwind-tables

Summary

Include and generate exception unwinding tables.

Syntax

```
--unwind-tables
```

Description

Include all references exception unwinding entries and create an unwinding table for stack unwinding according to the Arm EAEBI. This is the default.

Notes

If there are no sections that have associated unwinding information, the table is empty and takes no space in the application even with this option selected.

See also

--no-unwind-tables on page 197

4.6.49 --no-unwind-tables

Summary

Remove exception unwinding tables.

Syntax

--no-unwind-tables

Description

Remove all references to exception unwinding entries and do not create an unwiding table for stack unwinding. In this case it is not possible for C or C++ code to throw exceptions or unwind the stack.

See also

--unwind-tables on page 196

4.6.50 --optimize-exception-index

Summary

Optimize exception index table.

Syntax

--optimize-exception-index

Description

Directs the linker to optimize the exception index by collapsing adjacent entries with identical unwind sequences.

Collapsing one or more adjacent entries with the same unwind sequence reduces the size of the exception index and also reduces the time required to search for an entry at runtime when an exception is thrown.

This is the default.

See also

--no-optimize-exception-index on page 199

4.6.51 --no-optimize-exception-index

Summary

Do not optimize exception index table.

Syntax

`--no-optimize-exception-index`

Description

Directs the linker to retain individual exception index entries and not collapse them.

See also

--optimize-exception-index on page 198

4.6.52 --wrap

Summary

Wrap symbol.

Syntax

--wrap *name*

Description

Use a wrapper function for the symbol *name*. Any undefined reference to the symbol *name* will be resolved to **__wrap_***name*. The existing definition of the symbol *name* will be renamed to **__real_***name*. This can be used to provide a wrapper for library functions, e.g. to provide a trace or monitoring capability.

Example

In this example the symbol **sin** is wrapped so that its inputs and output can be monitoed. As the function is typically provided in a object library, it cannot easily be changen, but symbol wrapping provides a way to modify its behavior.

```
#include <stdio.h>
#include <math.h>

double __real_sin(double);

double __wrap_sin(double Arg) {
  double Result;
  //
  printf("About to call sin(%g)...\n", Arg);
  Result = __real_sin(Arg);
  printf("...and sin(%g) returned %g\n", Arg, Result);
  return Result;
}

void main(void) {
  double x;
  //
  x = sin(2.7);
}
```

When this program is executed, the output is:

```
About to call sin(2.7)...
...and sin(2.7) returned 0.42738
```

Note

Wrapping is performed by the linker where references to the wrapped function are replaced. The linker will not be able to wrap references that are resolved internally by the assembler (which do not lead to a name resolution), or when link-time optimization is selected.

4.7 Program transformation options

4.7.1 --dedupe-code

Summary

Deduplicate code sections.

Syntax

```
--dedupe-code
--dedupe-rx
```

Description

This option instructs the linker to try to find readonly code sections that are identical and discard duplicates, retaining a single “leader” instance.

This optimization potentially reduces readonly code size with no detrimental effect on runtime performance (and may even enhance performance for cached systems).

See also

--no-dedupe-code on page 202

4.7.2 --no-dedupe-code

Summary

Do not deduplicate code sections.

Syntax

```
--no-dedupe-code
--no-dedupe-rx
```

Description

This option instructs the linker not to deduplicate readonly code sections.

See also

--dedupe-code on page 201

4.7.3 --dedupe-data

Summary

Deduplicate data sections.

Syntax

```
--dedupe-data
--dedupe-ro
```

Description

This option instructs the linker to try to find readonly data sections that are identical and discard duplicates, retaining a single “leader” instance.

This optimization potentially reduces readonly data size with no detrimental effect on runtime performance.

See also

--no-dedupe-data on page 204

4.7.4 --no-dedupe-data

Summary

Do not deduplicate data sections.

Syntax

```
--no-dedupe-data
--no-dedupe-ro
```

Description

This option instructs the linker not to deduplicate readonly data sections.

See also

--dedupe-data on page 203

4.7.5 --inline

Summary

Inline small functions.

Syntax

```
--inline
```

Description

This option instructs the linker to inline small functions at the call site rather than calling the function. Functions that take one or two halfwords, excluding function return, can be inlined in many cases.

If all instances of calls to the function are inlined, it becomes possible for the called function to be eliminated entirely as long as it is not used as a function pointer.

This optimization always increases runtime performance and may reduce overall code size.

See also

--no-inline on page 206

4.7.6 --no-inline

Summary

Do not inline small functions.

Syntax

`--no-inline`

Description

This option instructs the linker not to inline small functions.

See also

--inline on page 205

4.7.7 --instruction-tables (RISC-V)

Summary

Compress application using AndeStar CoDense extension.

Syntax

`--instruction-tables`

Description

This option instructs the linker to optimize the size of the application using the AndeStar CoDense instruction table capability.

The linker finds the most-frequently-used 32-bit RISC-V instructions that the application uses and replaces them with 16-bit **`EXEC.IT`** instructions using the Andes CoDense extension.

The linker creates the CoDense instruction table and places it into the section `.text.exec.it` along with a symbol `__SEGGER_execit_table__` which designates the start of the table. This symbol can be used to inititalize the instruction table base address (**`uitb`**) in the startup code.

See also

--no-instruction-tables (RISC-V) on page 208

4.7.8 --no-instruction-tables (RISC-V)

Summary

Do not compress application using AndeStar CoDense extension.

Syntax

```
--no-instruction-tables
```

Description

This option probibits the linker from compressing the application using the AndeStar CoDense extension.

See also

--instruction-tables (RISC-V) on page 207

4.7.9 --merge-sections

Summary

Merge compatible sections.

Syntax

```
--merge-sections
```

Description

This option instructs the linker to merge duplicate entries in compatible sections (sections with the **SHF_MERGE** section flag set). This optimization reduces data size with no detrimental effect on runtime performance.

This is the default.

Note that section merging is related to section deduplication, but works with finer-grained section content.

See also

--no-merge-sections on page 210

4.7.10 --no-merge-sections

Summary

Do not merge compatible sections.

Syntax

--no-merge-strings

Description

This option instructs the linker not to perform section merging. Note that section merging is related to section deduplication, but works with finer-grained section content.

See also

--merge-strings on page 211

4.7.11 --merge-strings

Summary

Merge string constants.

Syntax

```
--merge-strings
```

Description

This option instructs the linker to merge duplicate string constants so that the duplicates are reduced to one shared constant thus reducing memory footprint. In addition, a string constant matches the end of another, a single string constant suffices.

This optimization reduces data size with no detrimental effect on runtime performance.

See also

--no-merge-strings on page 212

4.7.12 --no-merge-strings

Summary

Do not merge string constants.

Syntax

```
--no-merge-strings
```

Description

This option instructs the linker to keep all duplicate string constants separate and not merge them to a single, shared string.

See also

--merge-strings on page 211

4.7.13 --outline (RISC-V)

Summary

Enable outlining optimization.

Syntax

`--outline`

Description

This option instructs the linker to try to reduce code size by extracting small strands of common code into subroutines. This reduces code size at the expense of adding one additional subroutine call per outline fragment at runtime.

Outlining is not enabled by default.

See also

--no-outline (RISC-V) on page 214

4.7.14 --no-outline (RISC-V)

Summary

Disable outlining optimization.

Syntax

`--no-outline`

Description

This option instructs the linker not to run the outlining pass.

Outlining is not enabled by default.

See also

--outline (RISC-V) on page 213

4.7.15 --outline-strand-size (RISC-V)

Summary

Parameterize outlining optimization.

Syntax

--outline-strand-size=[*min-size*]**-**[*max-size*]

Description

This option parameterizes the size of the strands that are extracted for outlining and tail merging optimizations. If *min-size* is not specified, it defaults to 6; if *max-size* is not specified it defaults to 64; if **max-size** is less than or equal to **min-size**+6, it is set to **min-size**+6 as outlining is only beneficial for strands with a minimum of 6 instructions bytes (three RISC-V instruction packets).

Increasing the minimum strand enables the trade-off between code size and execution speed to be moved to favor execution speed at the expense of code size. For maximum compression of the program image, use a minimum size of 6 (and a fair maximum size of 64).

4.7.16 --relax (RISC-V)

Summary

Enable instruction relaxation.

Syntax

```
--relax
-mrelax
```

Description

This option instructs the linker to try to reduce code size by replacing data access sequences and function calling sequences with smaller, faster sequences.

For relaxation to be effective, ELF files and archives should be compiled with relaxation enabled before they are provided to the linker.

Instriction relaxation is enabled by default.

See also

--no-relax (RISC-V) on page 217

4.7.17 --no-relax (RISC-V)

Summary

Disable instruction relaxation.

Syntax

```
--no-relax
-mno-relax
```

Description

This option instructs the linker not to run the relaxation pass and to leave data access sequences and function calling sequences as they appear after compilation.

This option can reduce linking time at the expense of larger application code size.

See also

--relax (RISC-V) on page 216

4.7.18 --springboard (RISC-V)

Summary

Enable call springboarding.

Syntax

--springboard

Description

This option instructs the linker to reduce code size further than standard relaxation by *springboarding* calls. Only function call sites that are marked as relaxable are eligible for springboarding.

Springboarding adds a set of "springboards" to the end of the function. For each call in a function, the linker determines whether it would be advantageous to replace that call with a call through a *springboard* appened to the function instead. As such, a set of long calls can be replaced by compact calls to a springboard, saving code space.

Whilst reducing code size, it also adds an execution penalty of one branch instruction for each call that is springboarded.

Springboarding is enabled by default.

See also

--no-springboard (RISC-V) on page 219

4.7.19 --no-springboard (RISC-V)

Summary

Disable call springboarding.

Syntax

--no-springboard

Description

This option instructs the linker not to run the springboarding optimization.

See also

--springboard (RISC-V) on page 218

4.7.20 --tail-merge (RISC-V)

Summary

Enable tail merging optimization.

Syntax

```
--tail-merge
```

Description

This option instructs the linker to try to reduce code size by extracting small strands of common code immediately before a function return. This reduces code size at the expense of adding one additional jump per merged fragment at runtime.

Tail merging is not enabled by default.

See also

--no-tail-merge (RISC-V) on page 221

4.7.21 --no-tail-merge (RISC-V)

Summary

Disable tail merging optimization.

Syntax

```
--no-tail-merge
```

Description

This option instructs the linker not to run the tail merging pass.

Tail merging is not enabled by default.

See also

--tail-merge (RISC-V) on page 220

4.7.22 --tp-model (RISC-V)

Summary

Set the thread-pointer model.

Syntax

--tp=model=*model*

Description

This option controls how the thread pointer register (**tp**) is used in the linked application:

none
: The linker considers the thread pointer register untouchable—it is an error if thread-local data is linked into in the application.

tls
: The application uses the thread pointer register as the the base of the executing thread's thread-local data.

base
: The linker is allowed to use the thread pointer register as a secondary base pointer, in addition to the global pointer (**gp**), in order to reduce program size and increase execution speed.

auto
: The linker chooses between **tls** and **base** depending upon whether the application requires thread-local data.

The default for this option is **auto**.

4.8 Control options

4.8.1 --andes-performance-extension

Synopsis

Enable support of AndeStar V5 Performance Extension.

Syntax

```
--andes-performance-extension
```

Description

Instruct the linker to support AndeStar V5 Performance Extension instructions and relocations.

It is not possible to enable both the AndeStar and Huawei custom extensions.

See also

--no-andes-performance-extension on page 224

4.8.2 --no-andes-performance-extension

Synopsis

Disable support of AndeStar V5 Performance Extension.

Syntax

`--no-andes-performance-extension`

Description

Instruct the linker not to support AndeStar V5 Performance Extension instructions and relocations. It is a reported error if any object module contains relocations associated with the AndeStar V5 Custom Extension.

This is the default.

See also

--andes-performance-extension on page 223

4.8.3 --cpu (Arm)

Summary

Set target core or architecture.

Syntax

--cpu=*name*
-cpu=*name*
-mcpu=*name*

Description

This option selects the target processor for the application and controls the construction of appropriate veneers when required.

The core names accepted are:

- **cortex-m0**
- **cortex-m0plus**
- **cortex-m1**
- **cortex-m3**
- **cortex-m4**
- **cortex-m7**
- **cortex-m23**
- **cortex-m33**
- **cortex-a8**
- **cortex-a9**

The architecture names accepted are:

- **6-M**
- **7-M**
- **7E-M**
- **7-A**
- **8-M.base**
- **8-M.main**
- **8.1-M.base**
- **8.1-M.main**

The default is **--cpu=cortex-m0**.

4.8.4 --cpu (RISC-V)

Summary

Set target core or architecture.

Syntax

--cpu=*name*
-cpu=*name*
-mcpu=*name*

Description

This option selects the target processor for the application and controls the construction of appropriate veneers when required.

The architecture names accepted are:

- **rv32e**
- **rv32ema**
- **rv32emac**
- **rv32i**
- **rv32ima**
- **rv32imac**
- **rv32imaf**
- **rv32imafc**
- **rv32g**
- **rv32gc**

The default is **--cpu=rv32imac**.

4.8.5 --big-endian

Synopsis

Big-endian byte ordering.

Syntax

```
--big-endian
-EB
```

Description

Specify that all input ELF files must be in big-endian byte order and the linked output must be in big-endian byte order.

See also

--little-endian on page 228

4.8.6 --little-endian

Synopsis

Little-endian byte ordering.

Syntax

```
--little-endian
-EL
```

Description

Specify that all input ELF files must be in little-endian byte order and the linked output must be in little-endian byte order.

This is the default.

See also

--big-endian on page 227

4.8.7 --huawei-extension

Synopsis

Enable support of Huawei Custom Extension.

Syntax

--huawei-extension

Description

Instruct the linker to support Huawei Custom Extension instructions and relocations. It is not possible to mix RV32D object modules with object modules compiled for the Huawei Custom Extension.

It is not possible to enable both the AndeStar and Huawei custom extensions.

See also

--no-huawei-extension on page 230

4.8.8 --no-huawei-extension

Synopsis

Disable support of Huawei Custom Extension.

Syntax

--no-huawei-extension

Description

Instruct the linker not to support Huawei Custom Extension instructions and relocations. It is a reported error if any object module contains relocations associated with the Huawei Custom Extension.

This is the default.

See also

--huawei-extension on page 229

4.8.9 --lazy-load-archives

Summary

Defer loading object modules from archives until required.

Syntax

`--lazy-load-archives`

Description

Delays reading the content of a required ELF object module until one of its symbols is required. This option can improve link performance if there are archives which contain many object files that are not ignored when linking.

For small applications and small archives, the performance gain is negligible.

For this option to work correctly, archives must contain an index to the contents, usually created by the archiver or by `ranlib`. If there is no table of contents, the archive may still be linked but lazy loading must be disabled using **`--no-lazy-load-archives`**.

See also

--no-lazy-load-archives on page 232

4.8.10 --no-lazy-load-archives

Summary

Defer loading object modules from archives until required.

Syntax

```
--no-lazy-load-archives
```

Description

Loads all object modules from an archive before linking, irrespective of whether its content is required or not. This option can improve link performance for archives that most always provide required object modules. For small applications and small archives, the performance of lazy loading and non-lazy loading is negligible.

This is the default.

See also

--lazy-load-archives on page 231

4.8.11 --list-all-undefineds

Summary

Issue one error for each undefined symbol.

Syntax

`--list-all-undefineds`

Description

This option instructs the linker to issue one error per undefined symbol which is the preferred option when running the linker inside an integrated development environment.

See also

--no-list-all-undefineds on page 234

4.8.12 --no-list-all-undefineds

Summary

Issue one error covering all undefined symbols.

Syntax

```
--no-list-all-undefineds
```

Description

This option instructs the linker to issue a list of undefined symbols and a single error message indicating that there are undefined symbols. This is the default option for undefined symbols and is intended to produce clear output when the linker is run interactively from the command line.

See also

--list-all-undefineds on page 233

4.8.13 --remarks

Summary

Issue remarks.

Syntax

```
--remarks
```

Description

This option instructs the linker to issue remarks for potential issues during linking. This is the default.

See also

--no-remarks on page 238, *--remarks-are-warnings* on page 237, *--remarks-are-errors* on page 236

4.8.14 --remarks-are-errors

Summary

Elevate remarks to errors.

Syntax

```
--remarks-are-errors
```

Description

This option elevates all remark diagnostics issued by the linker to errors.

See also

--no-remarks on page 238, *--remarks* on page 235, , *--remarks-are-warnings* on page 237

4.8.15 --remarks-are-warnings

Summary

Elevate remarks to warnings.

Syntax

```
--remarks-are-warnings
```

Description

This option elevates all remark diagnostics issued by the linker to warnings.

See also

--no-remarks on page 238, *--remarks* on page 235, , *--remarks-are-errors* on page 236

4.8.16 --no-remarks

Summary

Suppress remarks.

Syntax

--no-remarks

Description

This option disables all remark diagnostics issued by the linker. Although remarks are suppressed, the total number of remarks that are suppressed by the linker is shown at the end of linking:

```
C:> segger-ld --via=app.ind
Copyright (c) 2017-2018 SEGGER Microcontroller GmbH    www.segger.com
SEGGER Linker 2.14 compiled Apr 11 2018 10:50:34

Link complete: 0 errors, 0 warnings, 2 remarks suppressed

C:> _
```

See also

--remarks-are-warnings on page 237, *--remarks-are-errors* on page 236

4.8.17 --silent

Summary

Do not show output.

Syntax

```
--silent
-q
```

Description

This option inhibits all linker status messages; only diagnostic messages are shown.

See also

--verbose on page 240

4.8.18 --verbose

Summary

Increase verbosity.

Syntax

```
--verbose
-b
```

Description

This option increase the verbosity of the linker by one level.

See also

--silent on page 239

4.8.19 --warn-any-double

Summary

Warn when application requires any **double** support.

Syntax

--warn-any-double

Description

This option instructs the linker to issue warnings when the application requires any runtime support for the **double** datatype. This is useful when your application uses the **float** datatype but it unintentionally promotes **float** to **double**.

For instance, the following piece of code seems to do exactly what is intended, see if a floating value is less than 1.1:

```
static int SensorFired(float Reading) {
  return Reading < 1.1;
}
```

Unfortunately the comparison is performed using **double** as **1.1** is a **double**, not **float** constant.

With this option enabled, such use is diagnosed at link time:

```
warning: 'double' runtime function '__aeabi_dcmplt' is required [--warn-any-double]
warning: 'double' runtime function '__aeabi_f2d' is required [--warn-any-double]
```

This can be fixed by comparing to the **float** constant **1.1f**:

```
static int SensorFired(float Reading) {
  return Reading < 1.1f;
}
```

Notes

This warning can be promoted to an error using **--warnings-are-errors**.

See also

--warn-unintended-double on page 242, *--no-warn-double* on page 243, *--warnings-are-errors* on page 251

4.8.20 --warn-unintended-double

Summary

Warn when application requires unintended **double** support.

Syntax

--warn-unintended-double

Description

This option instructs the linker to issue warnings when the application requires any runtime support for the **double** datatype except for conversion from **float** to **double**. This is useful when your application uses the **float** datatype but must also convert it to **double** for use in variadic argument lists, for instance when using the SEGGER Semihosting library for output.

With this option enabled, the following code will not illicit a warning becuase promotion from **float** to **double** is intended: in this case as all **float** arguments are promoted to **double** when passing through a variable argument list:

```
void DebugPrint(const char *sFormat, ...);

static int _PrintSensorValue(float Reading) {
  DebugPrint("Reading (mA): %f\n", Reading);
}
```

However, an innocent-looking modification to change units…

```
static int _PrintSensorValue(float Reading) {
  DebugPrint("Reading (uA): %f\n", Reading * 1000.0);
}
```

…is rejected because the multiplication is performed using **double** arithmetic as the constant **1000.0** is a **double**:

```
warning: 'double' runtime function '__aeabi_dmul' is required [--warn-unintended-double]
```

This can be fixed by multiplying by the **float** constant **1000.0f**:

```
static int _PrintSensorValue(float Reading) {
  DebugPrint("Reading (uA): %f\n", Reading * 1000.0f);
}
```

Notes

With this option, conversion of **float** to **double** is permitted so that a floating argument can be passed through a variadic argument list. Conversion from **double** to **float** is permitted so that the incoming **double** argument can be converted back to **float** and maniplulated using **float**-only arithmetic.

This warning can be promoted to an error using **--warnings-are-errors**.

See also

--warn-any-double on page 241, *--no-warn-double* on page 243, *--warnings-are-errors* on page 251

4.8.21 --no-warn-double

Summary

Do not warn when application requires **`double`** support.

Syntax

`--no-warn-double`

Description

This option instructs the linker inhibit warnings for applications thay require **`double`** runtime support.

This is the default.

See also

--warn-any-double on page 241, *--warn-unintended-double* on page 242

4.8.22 --warn-arch-mismatch (RISC-V)

Summary

Issue warnings for architecture mismatches.

Syntax

`--warn-arch-mismatch`

Description

This option instructs the linker to issue a warning when it detects linkage of ELF files with mismatching architecture flags in the ELF header.

This is the default.

See also

--no-warn-arch-mismatch (RISC-V) on page 245

4.8.23 --no-warn-arch-mismatch (RISC-V)

Summary

Do not issue warnings for architecture mismatches.

Syntax

```
--no-warn-arch-mismatch
```

Description

This option instructs the linker not to issue a warning when it detects linkage of ELF files with mismatching architecture flags in the ELF header. Although matching architecture flags for all inputs is ideal, it may be that linking object files with incorrect flags is unavoidable, in which case specifying this option will silence the warning.

See also

--warn-arch-mismatch (RISC-V) on page 244

4.8.24 --warn-deprecated

Summary

Issue warnings for deprecated features.

Syntax

--warn-deprecated

Description

This option instructs the linker to issue a diagnostic remark for features that are deprecated and due to be removed. This can be elevated to a warning by using **--remarks-are-warnings**.

See also

--no-warn-deprecated on page 247, *--remarks-are-warnings* on page 237

4.8.25 --no-warn-deprecated

Summary

Do not issue warnings for deprecated features.

Syntax

```
--no-warn-deprecated
```

Description

This option instructs the linker not to issue diagnostic remarks for features that are deprecated and due to be removed.

This is the default.

See also

--warn-deprecated on page 246

4.8.26 --warn-empty-selects

Summary

Issue warnings for potential section selection errors.

Syntax

--warn-empty-selects

Description

This option instructs the linker to issue warnings for section selections that do not select any sections.

Consider a section "**.IDLABEL**" that is intended to be selected into a region **IDREGION** using the section selection statement:

```
place in IDREGION { section IDLABEL };
```

Because the section name in the ELF file (**.IDLABEL**) does not exactly match the section name in the section selector (**IDLABEL**, no period), the section is not selected and therefore not placed into **IDREGION**.

This option instructs the linker to issue a warning when an exact section selector, in this case **section IDLABEL**, does not select any section and as such could be considered an error.

See also

--no-warn-empty-selects on page 249

4.8.27 --no-warn-empty-selects

Summary

Do not issue warnings for potential section selection errors.

Syntax

```
--no-warn-empty-selects
```

Description

This option instructs the linker inhibit warnings for section selections that do not select any sections.

See also

--warn-empty-selects on page 248

4.8.28 --warnings

Summary

Issue warnings.

Syntax

```
--warnings
```

Description

This option instructs the linker to issue warnings for dubious use or inputs. This is the default.

See also

--no-warnings on page 252, *--warnings-are-errors* on page 251

4.8.29 --warnings-are-errors

Summary

Elevate warnings to errors.

Syntax

```
--warnings-are-errors
--fatal-warnings
```

Description

This option elevates all warning diagnostics issued by the linker to errors.

See also

--no-warnings on page 252, *--warnings* on page 250

4.8.30 --no-warnings

Summary

Suppress warnings.

Syntax

--no-warnings

Description

This option disables all warning diagnostics issued by the linker. Although warnings are suppressed, the total number of warnings that are suppressed by the linker is shown at the end of linking:

```
C:> segger-ld --via=app.ind
Copyright (c) 2017-2018 SEGGER Microcontroller GmbH    www.segger.com
SEGGER Linker 2.14 compiled Apr 11 2018 10:50:34

Link complete: 0 errors, 1 warnings suppressed, 0 remarks

C:> _
```

See also

--warnings on page 250, *--warnings-are-errors* on page 251

4.9 Compatibility options

4.9.1 --begin-group

Synopsis

Start input file group.

Syntax

```
--begin-group
-(
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored. The SEGGER Linker will automatically resolve references from object files and libraries and does not require library grouping to do so.

4.9.2 --end-group

Synopsis

End input file group.

Syntax

```
--end-group
-)
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

The SEGGER Linker will automatically resolve references from object files and libraries and does not require library grouping to do so.

4.9.3 --emit-relocs

Synopsis

Emit relocations.

Syntax

```
--emit-relocs
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

4.9.4 --allow-multiple-definition

Synopsis

Allow multiple definitions of the same symbol.

Syntax

--allow-multiple-definition

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

The SEGGER Linker does not support multiple definitions of identical symbols: a symbol is required to have exactly one strong definition.

4.9.5 --gc-sections

Synopsis

Garbage collect sections.

Syntax

```
--gc-sections
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

The SEGGER Linker only includes sections that are reachable from root symbols and, therefore, this option is redundant.

4.9.6 --omagic

Synopsis

Set NMAGIC flag.

Syntax

```
--omagic
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

4.9.7 --discard-locals

Synopsis

Discard local symbols.

Syntax

```
--discard-locals
-X
```

Description

This option is accepted for GNU **ld** compatibility and is otherwise ignored.

Chapter 5

Indexes

5.1 Subject index

