Application Note

Document No.: AN1130

G32R5xx Keil Linker Script Application Note

Version: V1.2

1 Introduction

In the MDK-ARM development environment, linker script files (Linker Script) instruct the linker how to map code and data into the target microcontroller memory. They are crucial for G32R5xx series MCU development because they define the layout of programs and data in different memory areas, ensuring correct execution and effective use of memory resources.

This document explains commonly used linker script files for G32R5xx series MCU development and describes how to use them.

Note: These files are located under:

“device_support/g32r501/common/sct” or “device_support/g32r502/common/sct” (choose by target device)

Contents

2 Memory Allocation of G32R501

Before using linker script files, understand the memory allocation of the target chip. Some G32R501 products are dual-core and some are single-core; allocations differ, but they share the following regions:

- ITCM (Instruction Tightly Coupled Memory): Stores instructions for high-speed execution. In single-core mode, CPU0 has a larger ITCM (64KB); in dual-core mode, ITCM is 48KB (CPU0) and 8KB (CPU1).
- DTCM (Data Tightly Coupled Memory): Stores data for high-speed access. In single-core mode, CPU0 has 16KB DTCM; in dual-core mode, DTCM is 16KB (CPU0) and 8KB (CPU1).
- Flash: Non-volatile memory for program code and static data. Capacity depends on package: for example, evaluation-board G32R501Dxy can split CBUS Flash into 320KB each for CPU0/CPU1; single-core xc packages commonly have 256KB CBUS Flash. Follow the datasheet and macros in the selected ".sct"; do **not** assume "640KB for every orderable".
- SRAM: General data storage. Base address and size stay the same in single-core and dual-core modes, but in dual-core mode the regions used by different cores must not overlap.

Selecting an appropriate linker script requires understanding the target chip memory map.

2.1 Memory Allocation of G32R501 Single-core MCU

In single-core mode, CPU0 memory allocation is listed in Table 1.

Note: Table 1 is a single-core **xc** reference, aligned with "device_support/g32r501/common/sct/readme.txt" and the datasheet; verify other orderables against macros in the selected ".sct".

Table 1 Memory Allocation of G32R501 Single-core MCU

Memory area	Base address	Size
ITCM RAM	0x00000000	64KB
DTCM RAM	0x20000000	16KB
Flash (CBUS)	0x08000000 (ITCM Flash interface: 0x00100000)	256KB
SRAM1	0x20100000	8KB
SRAM2	0x20200000	8KB
SRAM3	0x20300000	32KB

2.2 Memory Allocation of G32R501 Dual-core MCU

In dual-core mode, CPU0 and CPU1 share some regions and also have dedicated regions. Dual-core memory allocation is listed in Table 2.

Table 2 Memory Allocation of G32R501 Dual-core MCU

Memory area	Base address	Size	Remarks
ITCM RAM (CPU0)	0x00000000	48KB	ITCM of CPU0
ITCM RAM (CPU1)	0x00000000	8KB	ITCM of CPU1
DTCM RAM (CPU0)	0x20000000	16KB	DTCM of CPU0
DTCM RAM (CPU1)	0x20000000	8KB	DTCM of CPU1
Flash	0x08000000 (ITCM: 0x00100000)	640KB	Shared Flash
SRAM1	0x20100000	8KB	Shared SRAM1
SRAM2	0x20200000	8KB	Shared SRAM2
SRAM3	0x20300000	32KB	Shared SRAM3

3 Memory Allocation of G32R502

G32R502 is a single-core real-time control MCU. Before selecting a ".sct", review its memory regions. Compared with G32R501, note:

- G32R502 has **no ITCM Flash interface** (when code executes from Flash it uses CBUS Flash at base "0x08000000").
- Typical evaluation / peripheral example sizes are: **ITCM RAM 16KB**, **DTCM RAM 24KB**, **CBUS Flash 256KB** (follow "device_support/g32r502/common/sct/readme.txt" and macros in the selected ".sct").
- Default "g32r502xc_cbus_flash.sct" focuses on ITCM / DTCM / CBUS Flash; **do not** copy G32R501 single-core SRAM1/SRAM2/SRAM3 rows onto G32R502.

Typical G32R502 memory allocation is listed in Table 3.

Table 3 Typical G32R502 Memory Allocation (aligned with g32r502xccbusflash.sct macros)

Memory area	Base address	Size	Notes
ITCM RAM	0x00000000	16KB	"__ITCM_SIZE = 0x4000"
DTCM RAM	0x20000000	24KB	"__DTCM_SIZE = 0x6000"
Flash (CBUS)	0x08000000	256KB	"__CBUS_FLASH_SIZE = 0x40000"; no ITCM Flash interface

Selection notes: "device_support/g32r502/common/sct/readme.txt". If the project switches to "*_itcm_ram*.sct", the code execution region moves to ITCM; follow that script.

4 Memory Allocation of Each Linker Script File (Linker Script)

As described in Chapter 2 Memory Allocation of G32R501, G32R501 memory structures differ between single-core and dual-core modes. Create or select linker scripts for the application scenario to optimize performance and memory use.

This chapter briefly describes memory allocation and purpose of the linker scripts.

4.1 G32R501 Linker Scripts

Table 4 explains selected important G32R501 ".sct" files. For other SDK ".sct" files not listed, refer to the same style of notes.

Table 4 List of .sct Files in SDK (excerpt)

No.	File name	Description
1	g32r501dxy_cpu0_cbus_flash.sct	Dual-core CPU0; CBUS Flash; RAM vector by default (configurable)
2	g32r501dxy_cpu1_cbus_flash.sct	Dual-core CPU1; CBUS Flash
3	g32r501dxy_cpu0_itcm_ram.sct	Dual-core CPU0; code in ITCM RAM
4	g32r501dxy_cpu0_cbus_flash_noramvector.sct	Dual-core CPU0; CBUS Flash; no RAM vector table
5	g32r501dxy_cpu0_cbus_flash_secure.sct	Dual-core CPU0 secure-boot example
6	g32r501dxy_cpu1_cbus_flash_secure.sct	Dual-core CPU1 secure-boot example
7	g32r501xc_cbus_flash.sct	Single-core xc; CBUS Flash
8	g32r501xc_itcm_ram.sct	Single-core xc; ITCM RAM
9	g32r501xc_cbus_flash_noramvector.sct	Single-core xc; no RAM vector table
10	g32r501xy_cbus_flash.sct	Single-core xy; CBUS Flash
11	g32r501xy_cbus_flash_secure.sct	Single-core secure-boot example
12	g32r501xy_itcm_flash.sct	Single-core xy; ITCM Flash interface
13	g32r501xy_itcm_ram.sct	Single-core xy; ITCM RAM

Full list and selection notes: "device_support/g32r501/common/sct/readme.txt" (including dxc variants).

4.2 G32R502 Linker Scripts

G32R502 linker scripts are under “device_support/g32r502/common/sct/”. G32R502 has **no ITCM Flash interface**. Common files are listed in Table 5.

Table 5 G32R502 .sct File List

No.	File name	Description
1	g32r502xc_cbus_flash.sct	CBUS Flash; default for evaluation-board / peripheral examples
2	g32r502xc_itcm_ram.sct	Code in ITCM RAM
3	g32r502xc_cbus_flash_noramvector.sct	CBUS Flash; no RAM vector table (library evaluation, and so on)
4	g32r502xc_itcm_ram_noramvector.sct	ITCM RAM; no RAM vector table

Selection notes: “device_support/g32r502/common/sct/readme.txt”.

5 Use Linker Script File (.sct) in MDK

In MDK, using ".sct" files manages MCU memory layout. Steps to select and customize linker scripts follow.

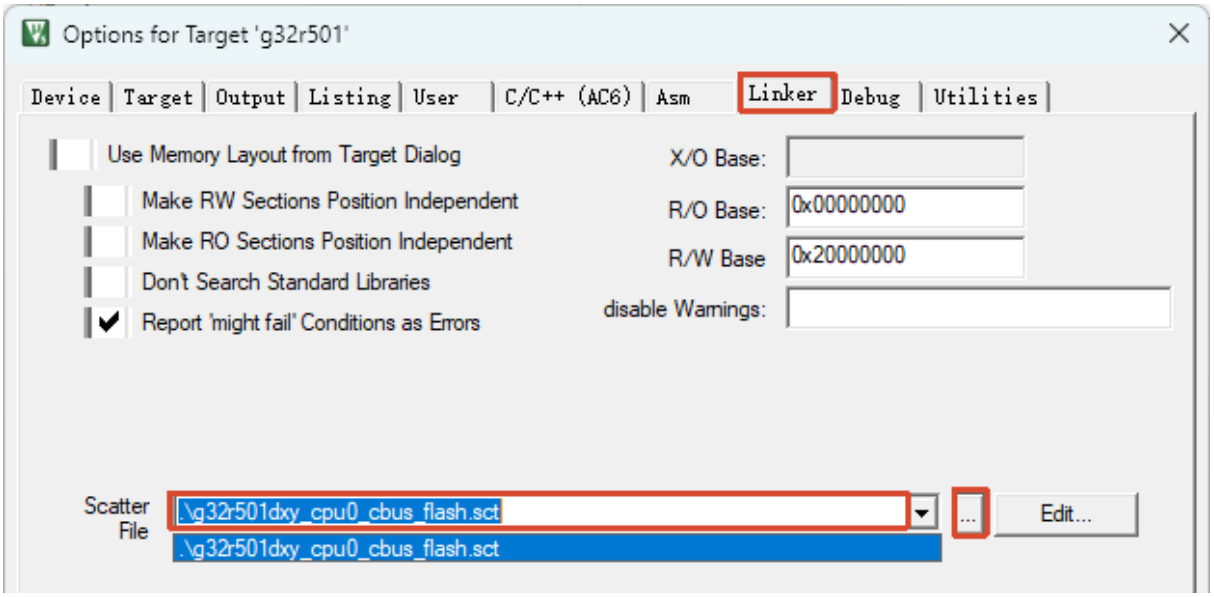
5.1 Select Linker Script Files (.sct)

After copying the ".sct" file into the MDK project:

- Open the MDK project.
- Open Options for Target (Project menu or toolbar).
- Select the "Linker" tab.
- Use "…" to browse and select the ".sct" file.
- Use "Edit" to edit the selected file.
- Click "OK" to save.

Selection steps are shown in Figure 1.

Figure 1 How to Select or Edit ".sct" Files



Note: The following UI uses a G32R501 example (target name commonly "g32r501") as reference. For G32R502, select the matching G32R502 device in Device / Target, replace "g32r501" with "g32r502" in paths, and use "r502_dbg.ini" as the MDK debug initialization file.

5.2 Custom Configuration

After selecting a ".sct" file, customize settings in the Configuration Wizard window.

Figure 2 Select Configuration Wizard Editing Window

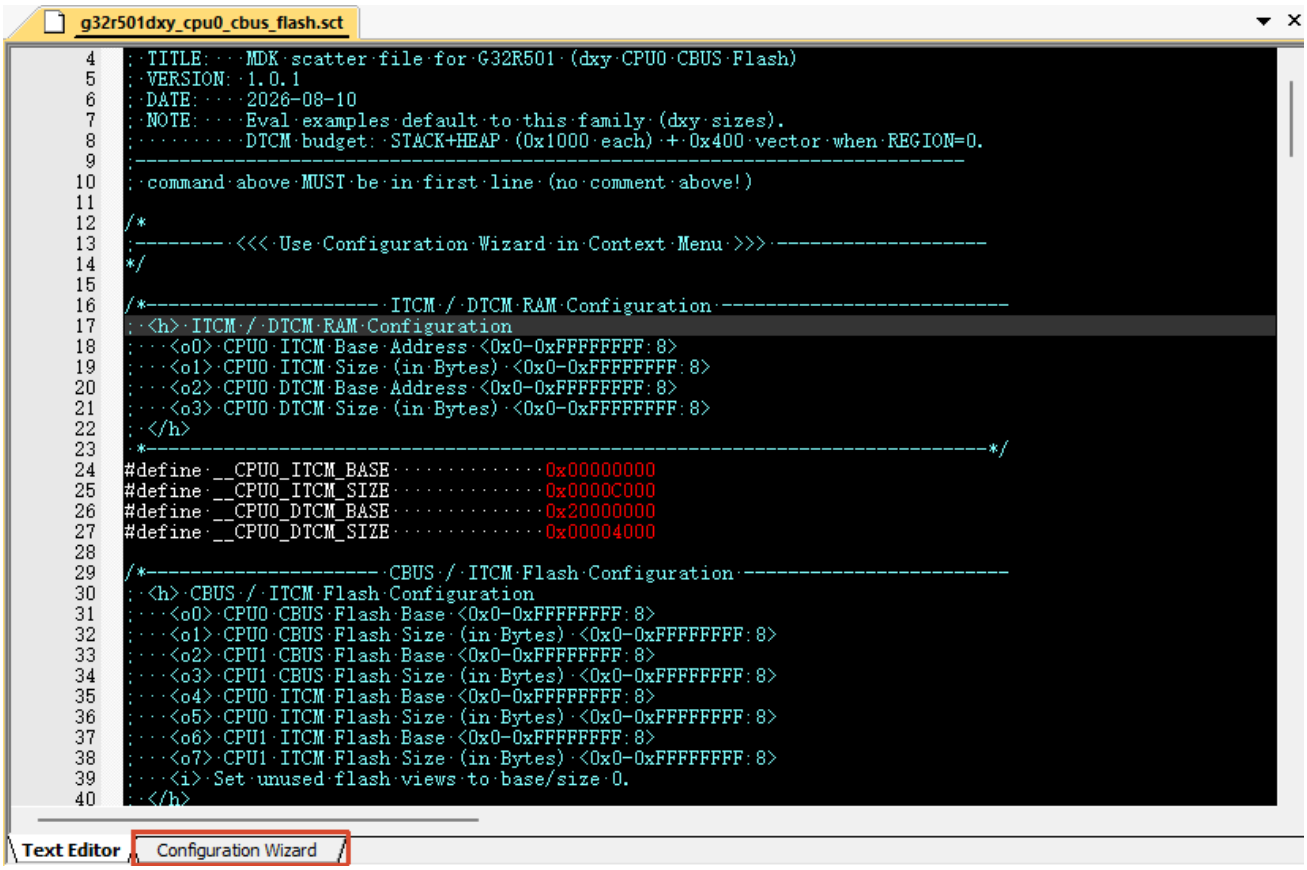
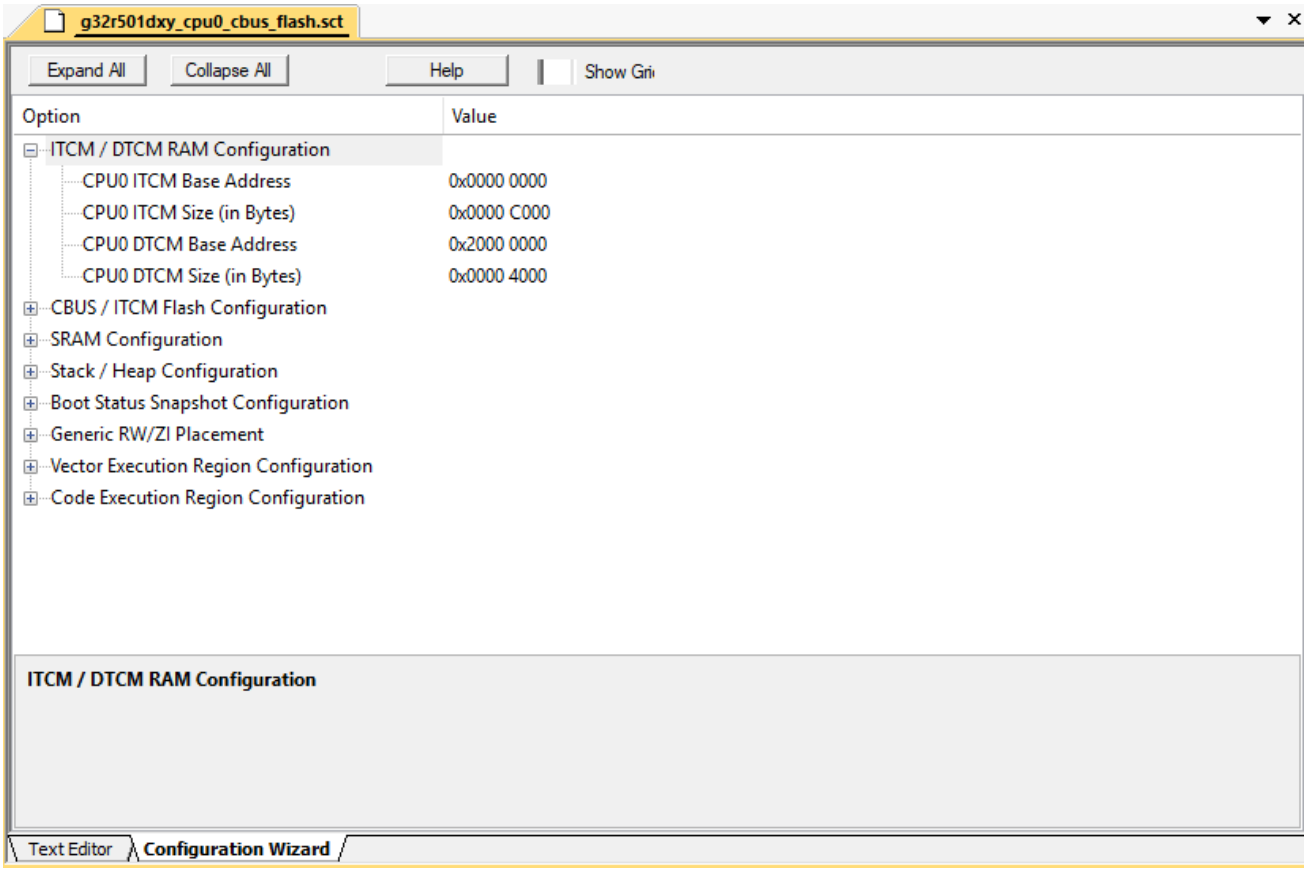


Figure 3 Configuration Wizard Editing Window



5.2.1 Current Configuration Wizard Fields (G32R501 scripts)

Use current scripts under “device_support/g32r501/common/sct/” as the source of truth for these fields. Field meanings are as shown in Table 6.

Table 6 Configuration Wizard Field Descriptions

Field	Values	Description
“__VECTOR_REGION”	0: DTCM; 1: ITCM; 2: None	Selects the RAM vector-table region. Value 2 creates no LR_VEC and matches “*_noramvector*” purpose.

Field	Values	Description
"__RWZI_BANK"	0: DTCM; 1: ITCM; 2: DTCM then ITCM	Selects the general RW/ZI region. Keep 0 in "itcm_ram" master scripts.
"__BOOT_STATUS_SNAPSHOT_ENABLE"	0: off; 1: on	When enabled, reserves the last 0x20 bytes of DTCM and moves the stack top down.

With a "*_noramvector*" script, the vector-region setting is absent and the project must define "LINKER_VECTOR_REGION_NONE". These scripts create no LR_VEC and use DTCM from "0x20000000".

Common ".sct" script selection is as shown in Table 7.

Table 7 Common .sct Script Selection

Script	Typical use
"g32r501dxy_cpu0_cbus_flash.sct"	Default for evaluation-board and product examples
"*_noramvector*.sct"	When "Interrupt_initVectorTable" is not called or the startup vector table must remain; define "LINKER_VECTOR_REGION_NONE"
"g32r501dxy_cpu0_itcm_ram.sct"	dxy IR execution; RW/ZI fixed to DTCM
"g32r502xc_cbus_flash.sct"	Default G32R502 evaluation / peripheral examples

5.2.2 SRAM configuration

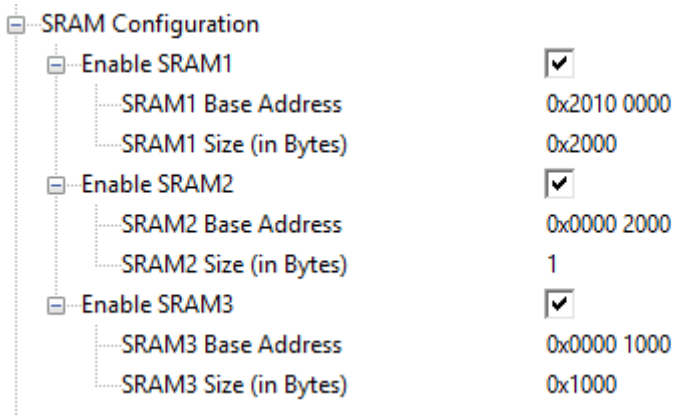
Different SRAM areas can be enabled or disabled:

- SRAM1: Typical size 8KB.
- SRAM2: Typical size 8KB.

- SRAM3: Typical size 32KB.

Note: In dual-core mode, ensure SRAM regions used by different cores do not overlap. G32R502 default CBUS Flash scripts focus on ITCM/DTCM/Flash; do not assume the same SRAM1/2/3 Wizard items as G32R501.

Figure 4 Enable or Disable Different SRAM Areas



5.2.3 Stack/Heap Configuration

Set stack and heap size and location (DTCM, SRAM1, SRAM2, or SRAM3).

- Default stack location: DTCM.

Figure 5 Setting the Size and Location of the Stack and Heap

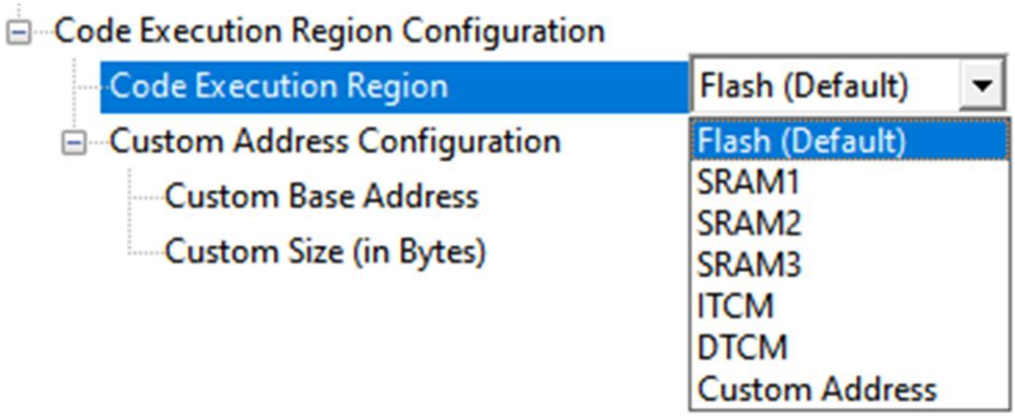
Note: DMA-accessed data should not be placed in DTCM; place it in a separate section such as “dma_data” in SRAM1/SRAM2/SRAM3.

5.2.4 Code execution area configuration

Select the code execution area (Flash, SRAM, ITCM, DTCM, or custom address). For a custom address, verify the region is valid.

- Flash: Default code storage and execution area.
- SRAM: SRAM1/SRAM2/SRAM3 for faster execution where applicable.
- ITCM: Critical code for faster execution.
- DTCM: Fast data access.
- Custom address: Manual execution region.

Figure 6 Selecting Execution Area of the Code



Note: Save after modification; changes take effect on the next build.

6 Create Linker Script Files (.sct)

Users can author project-specific ".sct" files based on the MDK-ARM scatter-file format. See MDK-ARM scatter-file format requirements and Chapter 2 Memory Allocation of G32R501.

6.1 Syntax Rules of Scatter Files

A scatter file contains one or more load regions. Each load region can contain one or more execution regions. Example:

```
; Load Region1
LOAD_ROM_1 0x0000 {
  ; Execution Region1
  EXEC_ROM_1 0x0000 {
    ; Input Section1
    program1.o (+RO)
  }
  ; Execution Region2
  DRAM 0x18000 0x8000 {
    ; Input Section2
    program1.o (+RW,+ZI)
  }
}
```

- Load region standard form (items in brackets optional):

```
LoadRegionName (BaseAddress | ("+" AddressOffset(+<offset>))) [AttributeList] [MaxSize]
{
  ExecutionRegion
}
```

- LoadRegionName: Identifies load regions for the linker.
- BaseAddress: Start address of code/data in the load region. Address offset is optional and must meet alignment. "<offset>" is bytes after the end of the previous load region (or from zero for the first region).
- Attributes (common):
  - ABSOLUTE — absolute address (default)
  - ALIGN "<alignment>" — address alignment
  - NOCOMPRESS — do not compress this load region in the image

- MaxSize: Maximum size; exceeding it causes an armlink error.
- Execution region standard form:

```
ExecutionRegionName (BaseAddress | ("+"AddressOffset (+<offset>))) [Attributes] [MaxSize |
<length>]
{
  InputSections
}
```

- EMPTY [-]“<length>” reserves an empty block (often for stack). Section selectors cannot be placed in EMPTY regions. Negative length means downward-growing stack with “<base_address>” as end address.
- Input sections commonly use Module Selector (+attributes), Module Selector (symbol style), or Module Selector (section style). Wildcards “” and “.ANY” are supported (“.ANY” has lowest priority).

Attribute descriptions are as shown in Table 8.

Table 8 Attribute Descriptions

Attribute Descriptor	Description
RO-CODE/CODE	Read-only code segment
RO-DATA/CONST	Read-only data segment
RO/TEXT	RO-CODE + RO-DATA
RW-CODE	Readable and writable code segment
RW-DATA	Readable and writable data segment
RW/DATA	RW-CODE + RW-DATA
XO	Executable-only region
ZI/BSS	Zero-initialized readable and writable data segment
ENTRY	Section entry

Additionally, FIRST / LAST pseudo-attributes are recognized. Symbol style uses the “:gdef:” prefix. Section styles such as “InRoot$$Sections” and “RESET” place startup/root sections.

6.2 Application Scenario Modification Examples

6.2.1 Using Custom Linker Macros

In “g32r501.h” / “g32r502.h” (paths “device_support/g32r501/common/device/Geehy/” and “device_support/g32r502/common/device/Geehy/”), macros control the memory layout of SDK code, as shown in Figure 7. For G32R502, use the corresponding device header and “common/sct/readme.txt”; do not apply G32R501 SRAM1/2/3 capacity descriptions directly.

Figure 7 Custom Linker Macros

```
// Define macros to place functions or variables in specific memory sections
// Usage:
// - Use these macros before function declarations or variable definitions to place them in specific sections.
// - Ensure that the linker script (MDK) or ICF file (IAR), or LD script (GCC) includes the corresponding sections.

#if defined (__CC_ARM) || defined (__ICCARM__) || defined (__GNUC__) || defined (__ARMCC_VERSION)
  #define SECTION_ITCM_INSTRUCTION  __attribute__((section("itcm.instruction")))
  #define SECTION_ITCM_RAMFUNC      __attribute__((section("itcm.ramfunc")))
  #define SECTION_DTCM_DATA         __attribute__((section("dtcm.data")))
  #define SECTION_DTCM_BSS          __attribute__((section("dtcm.bss")))
  #define SECTION_SRAM1_SHARE_DATA  __attribute__((section("sram1.share_data")))
  #define SECTION_SRAM2_SHARE_DATA  __attribute__((section("sram2.share_data")))
  #define SECTION_SRAM3_SHARE_DATA  __attribute__((section("sram3.share_data")))
#else
  #error "Unsupported compiler: no section macros can be defined."
#endif
```

Custom linker section base addresses are as shown in Table 9.

Table 9 Custom Linker Section Base Addresses

Specific section name	Link base address
itcm.instruction	ITCM (0x00000000)
itcm.ramfunc	ITCM (0x00000000)
dtcm.data	DTCM (0x20000000)
dtcm.bss	DTCM (0x20000000)
sram1.share_data	SRAM1 (0x20100000)
sram2.share_data	SRAM2 (0x20200000)
sram3.share_data	SRAM3 (0x20300000)

Users can place functions and variables in ITCM, DTCM, and SRAM with macros in the device headers above. If macros are insufficient, write a custom “.sct” and add custom sections (G32R502 projects use the corresponding G32R502 headers/scripts).

6.2.2 Specifying Function and Variable Link Regions

Example: place variable “Data_Ex” in DTCM.

- Target region: “0x20000000”–“0x20004000” (DTCM, 16KB).

- Add an input section in the DTCM execution region, for example “data.ex”:

```
RW_RAM_CPU0_DTCM __CPU0_DTCM_BASE __CPU0_DTCM_SIZE {
  .ANY (data.ex)
  .ANY (+RW +ZI)
}
```

- Control placement with:

```
__attribute__((section("data.ex")))
uint32_t Data_Ex;
```

6.2.3 Specifying Function and Variable Link Addresses

Example: place “Data_Ex” at “0x20300000”.

Method 1 (“const” read-only only):

```
__attribute__((__used__, section(".ARM.__at_0x20300000")))
const uint32_t Data_Ex;
```

Note: This method applies only to read-only variables qualified with “const”; applying it to RW variables generates a warning.

Method 2: modify the “.sct” to define a custom execution region and section name, then use “__attribute__((section("xxx")))”.

- First modify the “.sct” file: define a custom execution region at the target address, then define a custom input section name.

```
RW_RAM 0x20001000 0x400 {
  ;
  ; newly added input section
  .ANY (data.ex)
}
```

- In the definition, use “__attribute__((section("xxx")))” to link the variable to the new execution region above.

```
__attribute__((section("data.ex")))
uint32_t Data_Ex;
```

This links the defined variable to the specified address.

6.2.4 Application Partitioning Regions

Scatter-loading can partition the application for better memory management and performance optimization. Example:

```
# define __RO_BASE    __CPU0_ROM_BASE

# define __RO_SIZE    __CPU0_ROM_SIZE

# define __CPU0_ROM_BASE               __CPU0_CBUS_FLASH_BASE

# define __CPU0_ROM_SIZE               __CPU0_CBUS_FLASH_SIZE

# define __CPU0_CBUS_FLASH_BASE        0x08000000

# define __CPU0_CBUS_FLASH_SIZE        0x00050000
// 320KB

# define __SRAM3_BASE                   0x20300000

# define __SRAM3_SIZE                   0x00008000
// __SRAM3_SIZE,32KB

# define __SRAM1_BASE                   0x20100000

# define __SRAM1_SIZE                   0x00002000
// __SRAM1_SIZE,8KB

# define __CPU0_DTCM_BASE              0x20000000

# define __CPU0_DTCM_SIZE              0x00004000
// 16KB LR_ROM __RO_BASE __RO_SIZE {
ER_ROM __RO_BASE __RO_SIZE {
  *.o (RESET, +First) *(InRoot$$Sections)
  .ANY (+RO)
  .ANY (+XO)
}
RW_RAM_CPU0_DTCM __CPU0_DTCM_BASE __CPU0_DTCM_SIZE {
  .ANY (dtcm.data)
  .ANY (dtcm.bss)
  .ANY (+RW +ZI)
```

```
}
RW_RAM_SRAM1 __SRAM1_BASE __SRAM1_SIZE {
  .ANY (sram1.share_data)
  .ANY (+RW +ZI)
}
}
LR_ROM_SRAM3 __SRAM3_BASE __SRAM3_SIZE {
  ER_ROM_SRAM3 __SRAM3_BASE __SRAM3_SIZE {
    .ANY (+CONST)
    .ANY (share.data)
  }
}
```

This linker script defines two load regions: one for specific read-only data and one for read-only code.

- "LR_ROM_SRAM3" starts at "__SRAM3_BASE" with size "__SRAM3_SIZE", typically for specific read-only data.
- "LR_ROM" starts at "__RO_BASE" with size "__RO_SIZE" for read-only code.

6.2.5 Dual-core Example Link Files

Dual-core example ".sct" files are under "SDK\device_support\g32r501\common\sct\":

- "g32r501dxy_cpu0_cbus_flash.sct"
- "g32r501dxy_cpu1_cbus_flash.sct"

Flash is split into two parts. To change the split, edit the macros in the example files. Example: CPU0 Flash 384KB and CPU1 Flash 256KB:

```
# define __CPU0_CBUS_FLASH_BASE       0x08000000

# define __CPU0_CBUS_FLASH_SIZE        0x00060000   // 384KB

# define __CPU1_CBUS_FLASH_BASE       0x08060000

# define __CPU1_CBUS_FLASH_SIZE        0x00040000   // 256KB
```

6.2.6 Secure Boot Example Link Files

When CPU-read Flash data must not be encrypted, separate read-only data in the secure-boot

".sct" so the non-encryptable region is explicit. Refer to "g32r501dxy_cpu0_cbus_flash_secure.sct". The default non-encryptable region starts at "0x08050000"; change the macros if a different window is required.

For example, change the non-encryptable region start address and size from the SDK default ("0x08050000", 64KB) to "0x08020000" and 128KB:

```
// Define the size of ITCM and DTCM RAM for CPU0

# define __CPU0_ITCM_BASE              0x00000000

# define __CPU0_ITCM_SIZE               0x0000C000
// 48KB

# define __CPU0_DTCM_BASE             0x20000000

# define __CPU0_DTCM_SIZE              0x00004000
// 16KB
// Define the size of CBUS and ITCM secure Flash for CPU0,
// This area allows users to be set with encryption permissions.

# define __CPU0_CBUS_FLASH_SECURE_BASE         0x08000000

# define __CPU0_CBUS_FLASH_SECURE_SIZE          0x00020000
// 128KB

# define __CPU0_ITCM_FLASH_SECURE_BASE          0x0

# define __CPU0_ITCM_FLASH_SECURE_SIZE           0x0
// Define the size of CBUS and ITCM unsecure Flash for CPU0,
// This area does not allow users to be set with encryption permissions.

# define __CPU0_CBUS_FLASH_UNSECURE_BASE       0x08020000
// non-encryptable region moved to 0x08020000

# define __CPU0_CBUS_FLASH_UNSECURE_SIZE        0x00020000
// 128KB

# define __CPU0_ITCM_FLASH_UNSECURE_BASE        0x0

# define __CPU0_ITCM_FLASH_UNSECURE_SIZE         0x0
// Define the size of CBUS Flash for CPU1
```

```
# define __CPU1_CBUS_FLASH_BASE                0x08060000

# define __CPU1_CBUS_FLASH_SIZE                 0x00040000
// 256KB
```

7 Revision

Table 10 Document revision

Date	Version	Description
2025.1	1.0	● Initial release
2025.4	1.1	● Added scatter-file syntax rules
2026.8	1.2	● Applicable products listed as G32R501 and G32R502 ● Linker script paths described per device tree ● Refreshed file inventory and Flash size notes against current “common/sct” ● Xc Flash noted as 256KB ● Added G32R502 four-script section and G32R502 memory allocation chapter/table ● Updated linker/SRAM screenshot notes

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “TM” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy

hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY

DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS

"AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.