

Application Note

Document No.: AN1133

G32R5xx Interrupt Application Note

Version: V1.1

1 **Introduction**

This application note introduces G32R5xx interrupts and application design.

Contents

1	Introduction	1
2	NVIC	3
2.1	Interrupt Vector Table and Interrupt Priority	3
2.2	Interrupt Service Function	5
3	EXTI	6
3.1	Hardware Interrupt Selection	6
3.2	Hardware Event Selection	6
3.3	Software Interrupt/Event Selection	6
4	Software Configuration Process	7
5	Revision	9

2 NVIC

NVIC (Nested Vectored Interrupt Controller) controls the interrupt-related functions of the entire chip and is integrated into the core. The core of G32R5xx is Arm® Cortex®-M52. By programming the NVIC registers, interrupts can be enabled or interrupt priorities can be configured.

For both G32R501 and G32R502, the number of external interrupt channels in the current CMSIS device headers is **227** (IRQ numbers 0–226; the last item is for example “CTI_INT1_IRQn = 226”). Besides external IRQs, the vector table also includes Cortex-M system exception slots (Reset, NMI, Fault, SVCall, PendSV, SysTick, and so on; headers often express the total length as “NVIC_RAM_VECTOR_SIZE = 16 + 227”, meaning system slots + external IRQs). When writing interrupt code, open the matching device header and verify vector names; do not mix 501/502 peripheral-specific interrupt names.

2.1 Interrupt Vector Table and Interrupt Priority

If a peripheral interrupt is used when programming a G32R5xx device, the interrupt vector table in ROM must be copied to RAM, and the RAM vector table must be remapped through the core VTOR register so that users can register new interrupt response functions.

Table 1 is an excerpt from the interrupt vector table:

Table 1 Interrupt Vector Table (excerpt)

Exception type	Vector No.	Priority	Description
-	-	-	Reserved
Reset	-	-15	Reset
NMI	-	-14	Non-maskable interrupt
HardFault	-	-13	Various hardware faults
MemManage	-	-12	Memory fault
BusFault	-	-11	Bus fault
UsageFault	-	-10	Instruction execution fault
SecureFault	-	-9	Data security fault
SVCall	-	-5	System service called by general SWIU instruction
Debug Monitor	-	-4	Debugging function exception
PendSV	-	-2	Pending system service
SysTick	-	-1	System tick timer
...	...	...	...
DCCOMP interrupt	11	Configurable	DCCOMP interrupt

Exception type	Vector No.	Priority	Description
...	...	...	...
CTI INT0	225	Configurable	CTI interrupt 0
CTI INT1	226	Configurable	CTI interrupt 1

Arm® Cortex®-M52 allows users to set priority for each exception/interrupt so that interrupt responses can be managed more flexibly.

2.1.1 Preempt priority and subpriority

Think of “interrupt priority” as two labels. First call “Interrupt_setPriorityGroup()” to define how many bits belong to preempt priority versus subpriority, then write concrete values with “Interrupt_setPriority(irq, preempt, subpriority)”.

- **Preempt priority:** Controls whether an interrupt **can preempt another**. A **smaller** value means **higher** priority. If interrupt A has a higher preempt priority (smaller value) than running interrupt B, A can preempt B (nesting). If both have the same preempt priority, A **cannot** preempt B.
- **Subpriority:** Used only when preempt priorities are equal and multiple interrupts are pending, to decide **which is serviced first**. It **cannot** preempt an already running interrupt that has the same preempt priority.

Example (aligned with the SDK):

1. Call “Interrupt_setPriorityGroup(INTERRUPT_PRIGROUP_PREEMPT_3_2_SUB_1_0);” so that higher bits of the priority field are preempt and lower bits are subpriority (exact widths are documented on the macro in “interrupt.h”). Current examples such as “interrupt_ex1_external” use this grouping.
2. Then set “Interrupt_setPriority(INT_XINT1, 1, 0);” and “Interrupt_setPriority(INT_XINT2, 1, 0);”: both have preempt priority 1 and subpriority 0, so **neither can preempt the other**; if both are pending, hardware arbitration and pending order decide which runs first.
3. If XINT2 is changed to “Interrupt_setPriority(INT_XINT2, 0, 0);” (preempt priority 0 is higher than 1), XINT2 can preempt into the XINT1 ISR.

G32R5xx driverlib provides several grouping macros in “interrupt.h”, for example:

- “INTERRUPT_PRIGROUP_PREEMPT_3_0_SUB_NO”
- “INTERRUPT_PRIGROUP_PREEMPT_3_1_SUB_0”
- “INTERRUPT_PRIGROUP_PREEMPT_3_2_SUB_1_0” (commonly used by examples)

- “INTERRUPT_PRIGROUP_PREEMPT_3_SUB_2_0”
- “INTERRUPT_PRIGROUP_PREEMPT_NO_SUB_3_0”

Each configuration defines how preempt priority and subpriority bits are allocated. Choose one grouping for the project, then fill the last two parameters of “Interrupt_setPriority” consistently under that grouping to avoid expecting nesting when preempt levels are actually equal.

2.2 Interrupt Service Function

The four states of an interrupt: Active, Pending, Inactive, Active and pending.

- Active:
 - Being processed.
 - Not yet finished; the handler was preempted by a higher-priority exception/interrupt handler.
- Pending: An exception/interrupt has been generated but not yet activated.
- Inactive: No exception/interrupt has been generated yet.
- Active and pending:
 - One instance of an exception/interrupt is active, and a second instance is pending.
 - Only asynchronous exceptions/interrupts can be both active and pending at the same time. Synchronous exceptions/interrupts are inactive, pending, or active.

When an exception/interrupt is Active, the program responds with the interrupt service function: the processor finds the matching handler from the vector table and jumps to it.

In the ISR, the program executes code for the related event. After handling, it usually clears the corresponding interrupt flag so the system is ready for the next interrupt.

After the ISR returns, the system restores the previously saved context and continues the interrupted program.

3 EXTI

When using an EXTI interrupt line, configure and enable the line before an interrupt can be generated. Set the two trigger registers for the required edge detection, and write “1” to the corresponding bit of the interrupt mask register to enable the interrupt request. If the interrupt is enabled, the selected edge on the EXTI line generates an interrupt request and sets the corresponding flag. Writing “1” to the corresponding bit of the pending register clears that interrupt flag.

To generate an event, first configure and enable the event line. Set the two trigger registers for the required edge detection, and write “1” to the corresponding bit of the event mask register to allow the event request. When the selected edge appears on the event line, an event pulse is generated and the corresponding pending bit is not set to 1.

All EXTI lines can also generate software interrupt/event requests by writing “1” to the software interrupt/event register in software.

3.1 Hardware Interrupt Selection

To configure a line as an interrupt source:

1. Configure the mask bit of the interrupt line.
2. Configure the trigger selection bit for the interrupt line.
3. Configure the enable and mask bits of the NVIC IRQ channel mapped to EXTI so that requests on any interrupt line can be handled correctly.

3.2 Hardware Event Selection

To configure a line as an event source:

1. Configure the mask bit of the event line.
2. Configure the trigger selection bit for the event line.

3.3 Software Interrupt/Event Selection

Any configurable line can be configured as a software interrupt/event line. To generate a software interrupt:

1. Configure the mask bit for the interrupt/event line.
2. Clear the corresponding request bit in the software interrupt register.
3. Set the corresponding request bit in the software interrupt register.
4. Clear the corresponding request bit in the software interrupt register.

4 Software Configuration Process

This section takes the I2CA FIFO interrupt as an example of the G32R5xx interrupt software configuration flow:

1. "Interrupt_initModule();" disables global interrupts and disables all interrupts.
2. "Interrupt_initVectorTable();" initializes the NVIC vector table.
3. "Interrupt_setPriorityGroup(INTERRUPT_PRIGROUP_PREEMPT_3_2_SUB_1_0);" sets the priority group (aligned with current SDK examples such as "interrupt_ex1_external"; the older "INTERRUPT_PRIGROUP_PREEMPT_7_6_SUB_5_0" is no longer used).
4. "Interrupt_register(INT_I2CA_FIFO, &INT_I2CA_FIFO_IRQHandler);" writes the ISR entry address into the RAM vector table for the interrupt vector number. The two parameters mean:
 - "INT_I2CA_FIFO": I2CA FIFO interrupt vector number (index in the vector table).
 - "INT_I2CA_FIFO_IRQHandler": ISR whose entry address is written to that vector-table slot.
5. "Interrupt_setPriority(INT_I2CA_FIFO, 1, 0);" sets interrupt priority (second parameter = preempt priority, third = subpriority; see the previous section).
6. "Interrupt_enable(INT_I2CA_FIFO);" enables the NVIC interrupt.

Example code (priority-group macro aligned with the current SDK):

```
//
// Initialize NVIC and clears NVIC registers. Disables CPU interrupts.
//
Interrupt_initModule();
//
// Initialize the NVIC vector table with pointers to the shell Interrupt
// Service Routines (ISR).
//
Interrupt_initVectorTable();
//
// Interrupts that are used in this example are re-mapped to ISR functions
// found within this file.
//
Interrupt_register(INT_I2CA_FIFO, &INT_I2CA_FIFO_IRQHandler);
//
// Set the priority group to indicate the PREEMPT and SUB priority bits.
//
```

```
Interrupt_setPriorityGroup(INTERRUPT_PRIGROUP_PREEMPT_3_2_SUB_1_0);  
//  
// Set the global and group priority to allow CPU interrupts  
// with higher priority  
//  
Interrupt_setPriority(INT_I2CA_FIFO,1,0);  
//  
// Enable interrupts required for this example  
//  
Interrupt_enable(INT_I2CA_FIFO);  
//  
// Enable Global Interrupt (INTM) and realtime interrupt (DBGM)  
//  
EINT;  
ERTM;
```

5 Revision

Table 2 Document revision

Date	Version	Description
2025.1	1.0	● Initial release
2026.8	1.1	● Applicable products listed as G32R501 and G32R502 ● Clarified external interrupt count per device ● Added preempt/subpriority concepts and examples ● Aligned priority-group macros with current“interrupt_ex1_external”

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.

© 2026 Geehy Semiconductor Co., Ltd. - All Rights Reserved