应用笔记

Application Note

文档编号：**AN1138**

G32R501 迁移指南

版本：**V1.1**

1. 引言

对于嵌入式系统设计人员而言，能够在不同系列的微控制器之间轻松迁移应用程序是至关重要的。随着产品需求的不断提高，存储器大小、I/O 数量等资源的需求也相应增加，设计人员经常需要将应用程序移植到具有更高性能或更多资源的微控制器上。

本移植手册旨在帮助用户分析从现有的 Txx320F28004x 器件迁移到 G32R501 器件所需的步骤。本文档收集了最重要的信息，并列出了在迁移过程中需要注意的关键事项。迁移过程中涉及的主要方面包括硬件移植、外设移植、固件移植以及工具链的移植。

为了充分利用本手册中的信息，用户应熟悉 G32R501 系列微控制器的特性和开发环境。可以参考以下技术文档：

- G32R501 系列用户手册、数据手册和编程手册
- G32R501 系列内核相关文档，包括 Arm v8.1-M 架构参考手册等

本文将系统地介绍从 Txx320F28004x 到 G32R501 的迁移步骤，并提供实用的示例和代码，以帮助设计人员顺利完成应用程序的迁移工作，提升系统的整体性能和可靠性。本应用笔记适用的微控制器为 G32R501。

通过本移植手册，设计人员可以更好地理解和应对在迁移过程中遇到的各种挑战，确保在新的微控制器平台上实现应用程序的最佳性能。

1.1. 术语全称、缩写描述

关于本文档所使用的一些关键词及描述如表格 1 所示。

表格 1 缩写描述

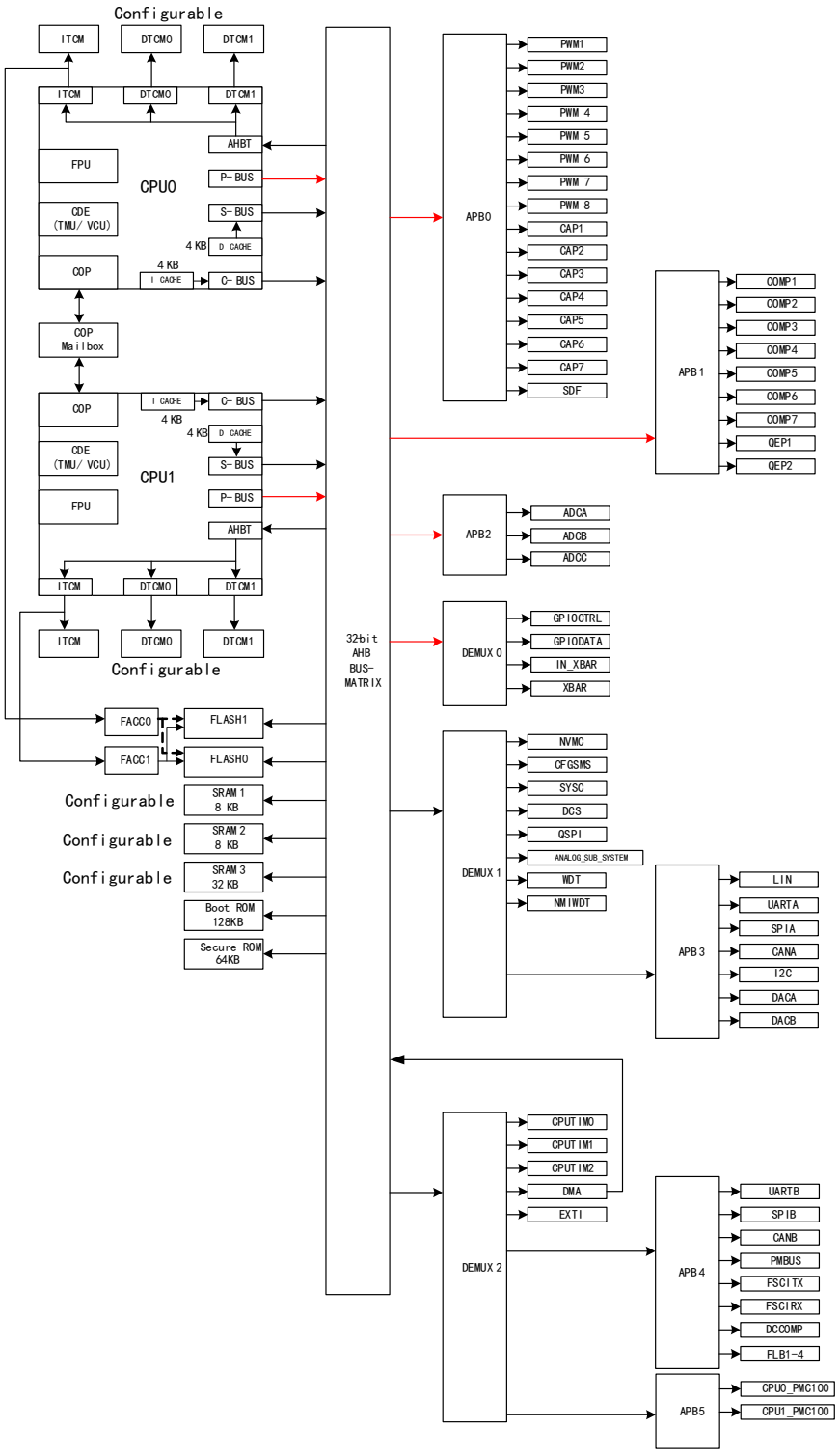
中文全称	英文全称	英文缩写
可编程静态存储子系统	Configurable Static Memory Subsystem	CFGSMS
软件	Software.	SW
硬件	Hardware.	HW
接口	Interface	IF
AHB 从接口	AHB slave interface.	ahbs_if

1.2. R5xx SoC 简介

R5xx SoC 是一款基于 Arm® Cortex®-M52 双核心的系统级芯片（SoC）。该 SoC 采用了广泛应用的 AMBA 2.0 总线来集成外设 IP 组件：需要大带宽的组件连接到 AMBA AHB 总线上，而低速组件连接到 AMBA APB 总线上。DMA 不能访问与 FLB 相关的寄存器。

R5xx SoC 的架构如图 1 所示。

图 1 R5xx SoC 架构



注意：

图上红色线为总线桥。

ITCM/DTCM/SRAM 大小通过 CFGSMS 进行配置。

目录

2. 启动模式兼容性

2.1. Boot 模式

本节介绍 G32R501 支持的 Boot 模式。BootROM 使用引导控制的 GPIO 引脚来确定引导模式配置。设备可以配置为引导到 RAM、引导到 Flash、执行引导加载程序或保持在等待模式中。

下表显示了默认的引导模式选项，用户可以自定义支持的引导模式以及引导模式选择引脚。

表格 2 G32R501 默认 Boot 模式

Boot Mode	GPIO24 (Default boot mode select pin 1)	GPIO32 (Default boot mode select pin 0)
UART / Wait boot	0	1
CAN	1	0
Flash	1	1

下表是 G32R501 支持的所有 Boot 模式，用户可以根据项目需要自定义所需引导模式，以及引导模式选择引脚。

表格 3 G32R501 所有可用的 Boot 模式

Boot Mode Number	Boot Mode
1	UART / Wait boot
2	CAN
3	Flash
4	Wait
5	RAM
6	SPI Master
7	I2C Master
10	Secure Boot

2.2. Boot 模式的 GPIO 引脚分配

用户可以根据封装及其他配置信息来选择不同的 GPIO 引脚。

表格 4 UART Boot

Option	BootMode Value	Tx	Rx
0	0x01	GPIO29	GPIO28
1	0x21	GPIO16	GPIO17

2	0x41	GPIO8	GPIO9
3	0x61	GPIO48	GPIO49
4	0x81	GPIO24	GPIO25

表格 5 CAN Boot

Option	BootMode Value	Tx	Rx
0	0x02/82	GPIO32	GPIO33
1	0x22/A2	GPIO4	GPIO5
2	0x42/C2	GPIO31	GPIO30
3	0x62/E2	GPIO37	GPIO35

表格 6 Flash Boot

Option	BootMode Value	Flash Entry Point	Flash Bank,Sector
0	0x03	0x08000000	Bank 0, Sector 0(Busmatrix IF)
1	0x23	0x00100000	Bank 0, Sector 0(ITCM IF)
2	0x43	0x08200000	Bank 1, Sector 0 (Busmatrix IF,DualBank Mode)
3	0x63	0x00120000	Bank 1, Sector 0 (ITCM IF,BualBank Mode)

表格 7 Wait Boot

Option	BootMode Value	Watchdog Status
0	0x04	Enable
1	0x24	Disable

表格 8 RAM Boot

Option	BootMode Value	RAM Entry Point	RAM IF
0	0x05	0x00000000	ITCM
1	0x25	0x20100000	SRAM0
2	0x45	0x20200000	SRAM1
3	0x65	0x20300000	SRAM2

表格 9 SPI Boot

Option	BootMode Value	MOSI	MISO	SCK	CS
1	0x26	GPIO8	GPIO10	GPIO9	GPIO10
2	0x46	GPIO54	GPIO55	GPIO56	GPIO57
3	0x66	GPIO16	GPIO17	GPIO56	GPIO57

4	0x86	GPIO8	GPIO17	GPIO9	GPIO11

表格 10 I2C Boot

Option	BootMode Value	SDA	SCL
0	0x07	GPIO32	GPIO33
2	0x47	GPIO26	GPIO27
3	0x67	GPIO42	GPIO43

表格 11 Secure Boot

Option	BootMode Value	Flash Entry Point	Flash Bank,Sector
0	0x0A	0x08000000	Bank 0, Sector 0（Busmatrix IF）
1	0x2A	0x00100000	Bank 0, Sector 0（ITCM IF）
2	0x4A	0x08200000	Bank 1, Sector 0（Busmatrix IF,DualBank Mode）
3	0x6A	0x00120000	Bank 1, Sector 0（ITCM IF,BualBank Mode）

2.2.1. 示例 1：使用 GPIO3/4/5 选择不同的引导模式

此示例展示如何通过配置 GPIO3、GPIO4、GPIO5 为引导模式选择引脚，以实现 8 种（例如：表格 12）不同的系统启动模式。通过不同的引脚电平组合，用户可以灵活选择闪存、UART、CAN 等引导方式。

表格 12 系统启动模式 1

Table	BMSP0	BMSP1	BMSP2	BootMode
0	0	0	0	闪存引导（0x03）
1	1	0	0	UART 引导（0x01）
2	0	1	0	闪存引导（0x23）
3	1	1	0	UART 引导（0x21）
4	0	0	1	CAN 引导（0x02）
5	1	0	1	SPI 引导（0x06）
6	0	1	1	RAM 引导（0x05）
7	1	1	1	I2C 引导（0x07）

以下代码中：

- EMU_BOOTPIN_CONFIG 地址为 0x50020000
- EMU BOOTDEF LOW 地址为 0x50020004
- EMU BOOTDEF HIGH 地址为 0x50020008

配置步骤：

配置 GPIO3/4/5 为 BootModePin：

```
HWREG(EMU_BOOTPIN_CONFIG) = (BOOTPIN_CONFIG_KEY  << 24) +
                            (BOOTPIN_CONFIG_BMSP2 << 16) +
                            (BOOTPIN_CONFIG_BMSP1 << 8) +
                            (BOOTPIN_CONFIG_BMSP0);
```

- 写入 0x5A 到 EMU_BOOTPIN_CONFIG 的 Key 字段以执行自定义引导过程。
- 设置 BMSP 值：
  - BOOTPIN_CONFIG_BMSP0 = 0x03
  - BOOTPIN_CONFIG_BMSP1 = 0x04
  - BOOTPIN_CONFIG_BMSP2 = 0x05

（1） 将引导模式选项 0、1、2、3 写入 EMU_BOOTDEF_LOW 寄存器：

```
HWREG(EMU_BOOTDEF_LOW) = (BOOTDEF_LOW_3 << 24) +
                         (BOOTDEF_LOW_2 << 16) +
                         (BOOTDEF_LOW_1 << 8) +
                         (BOOTDEF_LOW_0);
```

- BOOTDEF_LOW_0 = 0x03（闪存引导）
- BOOTDEF_LOW_1 = 0x01（UART 引导）
- BOOTDEF_LOW_2 = 0x23（闪存引导）
- BOOTDEF_LOW_3 = 0x21（UART 引导）

（2） 将引导模式选项 4、5、6、7 写入 EMU_BOOTDEF_HIGH 寄存器：

```
HWREG(EMU_BOOTDEF_HIGH) = (BOOTDEF_HIGH_7 << 24) +
                          (BOOTDEF_HIGH_6 << 16) +
                          (BOOTDEF_HIGH_5 << 8) +
                          (BOOTDEF_HIGH_4);
```

- BOOTDEF_HIGH_4 = 0x02（CAN 引导）
- BOOTDEF_HIGH_5 = 0x06（SPI 引导）
- BOOTDEF_HIGH_6 = 0x05（RAM 引导）
- BOOTDEF_HIGH_7 = 0x07（I2C 引导）

完成以上配置后，可在仿真状态下，通过控制 GPIO3/4/5 的外部电平选择不同的启动方式。

2.2.2. 示例 2 使用 GPIO3/4 选择不同的引导模式

在此示例中，通过配置 GPIO3 和 GPIO4 为引导模式选择引脚，可以实现 4 种（例如：表格 13）不同的系统启动模式。此方法适用于需要简化引导模式选择而使用较少引脚的场合。

表格 13 系统启动模式 2

Table	BMSP1	BMSP0	BootMode
0	0	0	闪存引导（0x03）
1	0	1	UART 引导（0x01）
2	1	0	RAM 引导（0x05）
3	1	1	CAN 引导（0x02）

在示例 1 的配置步骤基础上，

（1） 置 GPIO3/4 为 BootModePin：

- 写入 0x5A 到 EMU_BOOTPIN_CONFIG 的 Key 字段以执行自定义引导过程。
- 设置 BMSP 值：
  - BOOTPIN_CONFIG_BMSP0 = 0x03
  - BOOTPIN_CONFIG_BMSP1 = 0x04
  - BOOTPIN_CONFIG_BMSP2 = 0xFF（不使用）

（2） 将引导模式选项 0、1、2、3 写入 EMU_BOOTDEF_LOW 寄存器：

- BOOTDEF_LOW_0 = 0x03（闪存引导）
- BOOTDEF_LOW_1 = 0x01（UART 引导）
- BOOTDEF_LOW_2 = 0x05（RAM 引导）
- BOOTDEF_LOW_3 = 0x02（CAN 引导）

（3） 将无效引导模式选项写入 EMU_BOOTDEF_HIGH 寄存器：

- BOOTDEF_HIGH_4 = 0xFF
- BOOTDEF_HIGH_5 = 0xFF
- BOOTDEF_HIGH_6 = 0xFF
- BOOTDEF_HIGH_7 = 0xFF

完成以上配置后，可在仿真状态下，通过控制 GPIO3/4 的外部电平选择不同的启动方式。

2.3. BootMode 状态

BootMode 为 Boot 模式配置的运行结果。

基地址：0x2030_7F00

结构：

表格 14 当前选择的 Boot 模式

Bit	Name	Desc
[31:8]	RESERVE	保留
[7:5]	Boot Option	用于指示当前启动方式的配置
[4:3]	RESERVE	保留
[2:0]	Boot Mode	用于指示当前启动方式： 0: Parallel IO 1: UART / Wait boot 2: CAN 3: Flash 4: Wait 5: RAM 6: SPI Master 7: I2C Master 10 : Secure Boot

2.4. Boot 执行进度信息

用户可以通过该寄存器置位信息来确认 Boot 执行过程中的进度信息。

基地址：0x2030_7F08

结构：

表格 15 Boot 执行进度信息

Bit	Status	Desc
31	BOOTROM_IN_HARDFULT_HANDLER	Boot 进入 Hard Fault 处理函数
[30:28]	/	RESERVE
27	BOOTROM_GOT_A_HARD_FAULT	Boot 产生了 Hard Fault
26	BOOTROM_GOT_A_USAGE_FAULT	Boot 产生了 Usage Fault
25	BOOTROM_GOT_A_BUS_FAULT	Boot 产生了 Bus Fault
24	BOOTROM_GOT_A_MEM_FAULT	Boot 产生了 Memory Management Fault

Bit	Status	Desc
23	/	RESERVE
22	BOOTROM_GOT_A_MCLK_NMI	Boot 产生了时钟丢失 NMI
21	/	RESERVE
20	BOOTROM_GOT_A_FLASH_UNCERR_NMI	Boot 产生了 Flash ECC 不可纠正错误 NMI
[19:16]	/	RESERVE
15	BOOTROM_BOOT_COMPLETE	启动完毕
14	/	RESERVE
13	BOOTROM_HANDLED_POR	Boot 正在清除 POR 标志位
12	BOOTROM_HANDLED_XRSN	Boot 正在清除 XRSN 标志位
11	BOOTROM_RESC_HANDLED	Boot 正在处理复位标志
10	BOOTROM_POR_MEM_TEST_DONE	Boot 已完成 MBIST
9	/	RESERVE
8	BOOTROM_IN_SECURE_BOOT	Boot 进入 Secure BOOT
7	BOOTROM_IN_CAN_BOOT	Boot 进入 CAN BOOT
6	BOOTROM_IN_I2C_BOOT	Boot 进入 I2C BOOT
5	BOOTROM_IN_SPI_BOOT	Boot 进入 SPI BOOT
4	BOOTROM_IN_UART_BOOT	Boot 进入 SCI BOOT
3	BOOTROM_IN_RAM_BOOT	Boot 进入 RAM BOOT
2	BOOTROM_IN_PARALLEL_BOOT	Boot 进入 PARALLEL IO BOOT
1	BOOTROM_IN_FLASH_BOOT	Boot 进入 FLASH BOOT
0	BOOTROM_START_BOOT	开始启动流程

2.5. Boot 执行异常信息

表格 16 Boot 执行异常信息

Addr	Name	Desc
0x20307F0C	hard_fault_status	记录进入 Hard Fault 时的 SCB_HFSR 状态
0x20307F10	cbrom_fault_status	记录进入 NMI 时的 NMI_FLG
0x20307F14	bus_fault_adress	记录进入 Hard Fault 时的 SCB_BFAR 状态
0x20307F18	bus_fault_status	记录进入 Hard Fault 时的 SCB_BFSR 状态
0x20307F1C	mem_manage_fault_address	记录进入 Hard Fault 时的 SCB_MMFAR 状态
0x20307F20	mem_manage_fault_status	记录进入 Hard Fault 时的 SCB_MMFSR 状态
0x20307F24	usage_fault_status	记录进入 Hard Fault 时的 SCB_UFSR 状态

2.6. Boot Wait Point

在启动 ROM 执行期间，CPU 可能会在代码中进入等待循环状态。这种状态可能由于多种原因而发生。表格 17 列出了 CPU 程序计数器（PC）寄存器值落入的地址范围，用户可以通过读取当前 PC 地址来判断 Boot 是否进入异常。

表格 17 Boot 等待点地址信息

Entry	G32R501 Wait point	Txx320F28004x Wait point
In Wait Boot	0x1000_17DC—0x1000_17F4	0x003FAD74—0x003FB0CD
In UART Boot	0x1001_1DC4—0x1001_1E14	0x000706DC—0x000706DF
In Hard Fault Handler	0x1000_0212—0x1000_0312	—
In NMI Handler	0x1000_0316—0x1000_044E	0x003FBCD1—0x003FBD67
Failed secure Flash CMAC verification loop	0x1001_0F80—0x10010FB8	—

3. 外设移植

在进行从 Txx320F28004x 到 G32R501 的迁移过程中，了解两种微控制器在外设资源方面的差异至关重要。外设资源的选择和配置不仅直接影响系统的性能和功能，还关系到应用程序的实现和优化。本章将详细比较两种微控制器的外设资源。

3.1. 资源比较一览表

为了清晰展示 Txx320F28004x 和 G32R501 在外设资源上的差异，以下是一览表对比这两款微控制器的主要外设资源。

表格 18 G32R501 与 Txx320F28004x 外设资源差异

特性		G32R501	Txx320F28004x
处理器和加速器			
CPU0	内核	Cortex-M52	C28x
	频率	200MHz	100MHz
	FPU	支持	支持
	TMU	支持(数学计算加速指令集)	支持
	VCU	支持(数学计算加速指令集)	支持
CPU1 （xxxD 器件型号）	内核	Cortex-M52	CLA
	频率	200MHz	100MHz
	FPU	支持	内核为浮点型， 无独立 FPU
	TMU	支持(数学计算加速指令集)	不支持
	VCU	支持(数学计算加速指令集)	不支持
	6 通道 DMA	支持	支持
存储器			
Flash		640KB	256KB
RAM		128KB	100KB
代码存储区域安全保护(DCSM)		支持	支持
RAM Parity		支持	支持（除 M0, M1 外）
RAM ECC		不支持	支持（仅 M0, M1）
FLASH ECC		支持	支持
引导 ROM		支持	支持
用户可配置的 DCSM OTP		8KB	4KB
系统			
逻辑块		4 个逻辑块(FLB)	4 个逻辑块(CLB)

特性		G32R501	Txx320F28004x
32 位 CPU 计时器		3	3
看门狗计时器		1	1
非可屏蔽中断看门狗(NMIWD)计时器		1	1
晶体振荡器/外部时钟输入		1	1
引脚内部振荡器		2	2
GPIO 引脚	LQFP100	42	40
	LQFP80	44	/
	LQFP64	26	26
	VQFN56	25	25
AIO 输入	LQFP100	31	21
	LQFP80	16	/
	LQFP64	16	14
	VQFN56	14	12
	外部中断	16	5
模拟外设			
ADC	精度	12 位	12 位
	ADC 数量	3	3
	每秒百万次采样 (MSPS)	3.45	3.45
	转换时间 (ns)	290	290
ADC 通道（单端）	LQFP100	31	21
	LQFP80	16	/
	LQFP64	16	14
	QFN56	14	12
温度传感器		1	1
缓冲 DAC		2	2
COMP(增强比较器)	LQFP100	7	7
	LQFP80	7	/
	LQFP64	7	6
	QFN56	6	5
PGA		不支持	支持
控制外设			
CAP/HRCAP 模块		7 个（2 个具有 HRCAP 功能）	7 个（2 个具有 HRCAP 功能）

特性		G32R501	Txx320F28004x
PWM/HRPWM 通道		16	16
QEP 模块	LQFP100	2	2
	LQFP80	2	/
	LQFP64	2	1
	QFN56	2	1
SDF 通道	LQFP100	4	4
	LQFP80	4	/
	LQFP64	2	2
	QFN56	2	2
通信外设			
CAN		2	2
I2C		1	1
UART （UART 兼容）		2	2
SPI		2	2
LIN （UART 兼容）		1	1
PMBus		1	1
QSPI		1	0
工作电压与温度			
工作电压范围		3.1V~3.6V	3.1V~3.6V
环境工作温度(TA)		-40°C 至 105℃ （xxx7 器件型号） -40°C 至 125°C （xxx8 器件型号）	-40°C 至 105℃ （xxxS 器件型号） -40°C 至 125°C （xxxQ 器件型号）

3.2. CPU 内核

与传统的微控制器相比，G32R501 基于的 Arm® Cortex®-M52 内核额外支持高性能 DSP 指令集。Arm® Cortex®-M52 集成了 Arm v8.1-M 指令集，包括定点和浮点 DSP 扩展，使其能够高效地执行复杂的信号处理和控制算法。

注意：在程序开发的时候，高级语言开发的程序代码由编译器负责生成可执行文件，汇编语言开发的代码，则需要手动根据 Arm v8.1-M 指令集进行修改。

3.2.1. 浮点运算单元（FPU）

G32R501 支持单精度和双精度浮点运算（可选），这使得其在需要高精度计算的应用中具有显著优势。浮点运算单元（FPU）能够显著提高浮点运算的速度和精度，与没有 FPU 的内核相

比，执行浮点运算的性能提升显著。

3.2.2. Helium 指令集

G32R501 支持单指令多数据（Helium）操作，这一特性使得它能够在并行处理中提高数据处理效率。Helium 指令集可以一次性处理多个数据元素，适用于图像处理、音频处理等需要高数据吞吐量的应用。

3.2.3. 整数计算加速单元（ICAU）

在 G32R501 的扩展空间 CDE 中，Zidian 实现了一些特定的指令，以提高整数运算的性能，包括：

- Helium 要求的计算：快速傅里叶变换（FFT）、复数运算等。
- CRC 算法：提高循环冗余校验（CRC）计算的效率。

3.2.4. 浮点计算加速单元（FCAU）

在 G32R501 的扩展空间 FPCDE 中，Zidian 实现了一些特定的指令，以提高浮点运算的性能，包括：

- 三角函数计算：提高三角函数运算的效率。
- 平方根计算：提高平方根计算的速度。
- 除法运算：提高浮点除法的性能。

3.3. 中断向量表差异

G32R501 使用的中断序号最大可支持中断线为 226，所有中断都是经过嵌套向量中断控制器(NVIC)给到 CPU。外部中断控制器(EXTI)提供外部中断（GPIO/INPUT_XBAR）和外设中断（COMP）共 16 个。

表格 19 中断及异常功能对比

功能	G32R501	Txx320F28004x
中断优先级	226	19（ePIE 扩展至 224）
中断嵌套	8 级硬件嵌套	否（软件支持）
中断线数	226	19（ePIE 扩展至 224）
中断响应延时（cycle）	15	37

中断向量表上，G32R501 与 Txx320F28004x 并不相同，两者之间的差异可参考表格 20 中断向量差异表。

表格 20 中断向量差异表

Vector ID	G32R501 Name	Txx320F28004x
-15	Reset	Not used
-14	NMI	Not used
-13	HardFault	Not used
-12	MemManage	Not used
-11	BusFault	Not used
-10	UsageFault	Not used
-9	SecureFault	Not used
-5	SVCall	Not used
-4	Debug Monitor	Not used
-2	PendSV	Not used
-1	SysTick	Not used
0	Reserved	Not used
1	Reserved	Not used
2	Reserved	Not used
3	Reserved	Not used
4	Reserved	Not used
5	Reserved	Not used
6	Reserved	Not used
7	Reserved	Not used
8	Reserved	Not used
9	Reserved	Not used
10	Reserved	Not used
11	DCCOMP interrupt	Not used
12	Reserved	Not used
13	TMR1	CPU TIMER1 Interrupt
14	TMR2	CPU TIMER2 Interrupt
15	Reserved	CPU Data Logging Interrupt
16	RAM	CPU Real-Time OS Interrupt
17	COP TE0	Reserved
18	COP TE1	Non-Maskable Interrupt
19	COP TE2	Illegal Instruction (ITRAP)

Vector ID	G32R501 Name	Txx320F28004x
20	COP TE3	User-Defined Trap
21	COP RF0	User-Defined Trap
22	COP RF1	User-Defined Trap
23	COP RF2	User-Defined Trap
24	COP RF3	User-Defined Trap
25	COP GP0	User-Defined Trap
26	COP GP1	User-Defined Trap
27	COP GP2	User-Defined Trap
28	COP GP3	User-Defined Trap
29	Reserved	User-Defined Trap
30	Reserved	User-Defined Trap
31	Reserved	User-Defined Trap
32	ADCA1	ADCA1 interrupt
33	ADCB1	ADCB1 interrupt
34	ADCC1	ADCC1 interrupt
35	Reserved	XINT1 interrupt
36	Reserved	XINT2 interrupt
37	Reserved	Reserved
38	TMR0	TIMER0 interrupt
39	WWDT	WAKE WDOG interrupt
40	PWM1 TRIP ZONE	EPWM1 trip zone interrupt
41	PWM2 TRIP ZONE	EPWM2 trip zone interrupt
42	PWM3 TRIP ZONE	EPWM3 trip zone interrupt
43	PWM4 TRIP ZONE	EPWM4 trip zone interrupt
44	PWM5 TRIP ZONE	EPWM5 trip zone interrupt
45	PWM6 TRIP ZONE	EPWM6 trip zone interrupt
46	PWM7 TRIP ZONE	EPWM7 trip zone interrupt
47	PWM8 TRIP ZONE	EPWM8 trip zone interrupt
48	PWM1	EPWM1 interrupt
49	PWM2	EPWM2 interrupt
50	PWM3	EPWM3 interrupt
51	PWM4	EPWM4 interrupt

Vector ID	G32R501 Name	Txx320F28004x
52	PWM5	EPWM5 interrupt
53	PWM6	EPWM6 interrupt
54	PWM7	EPWM7 interrupt
55	PWM8	EPWM8 interrupt
56	CAP1	ECAP1 interrupt
57	CAP2	ECAP2 interrupt
58	CAP3	ECAP3 interrupt
59	CAP4	ECAP4 interrupt
60	CAP5	ECAP5 interrupt
61	CAP6	ECAP6 interrupt
62	CAP7	ECAP7 interrupt
63	Reserved	Reserved
64	QEP1	EQEP1 interrupt
65	QEP2	EQEP2 interrupt
66	Reserved	Reserved
67	Reserved	Reserved
68	FLB1	CLB1 interrupt
69	FLB2	CLB2 interrupt
70	FLB3	CLB3 interrupt
71	FLB4	CLB4 interrupt
72	SPIA_RX	SPIA_RX interrupt
73	SPIA_TX	SPIA_TX interrupt
74	SPIB_RX	SPIB_RX interrupt
75	SPIB_TX	SPIB_TX interrupt
76	Reserved	Reserved
77	Reserved	Reserved
78	Reserved	Reserved
79	Reserved	Reserved
80	DMA_CH1	DMA_CH1 interrupt
81	DMA_CH2	DMA_CH2 interrupt
82	DMA_CH3	DMA_CH3 interrupt
83	DMA_CH4	DMA_CH4 interrupt

Vector ID	G32R501 Name	Txx320F28004x
84	DMA_CH5	DMA_CH5 interrupt
85	DMA_CH6	DMA_CH6 interrupt
86	Reserved	Reserved
87	Reserved	Reserved
88	I2CA	I2CA interrupt
89	I2CA FIFO	I2CA FIFO interrupt
90	QSPI	Reserved
91	Reserved	Reserved
92	Reserved	Reserved
93	Reserved	Reserved
94	Reserved	Reserved
95	Reserved	Reserved
96	UARTA_RX	SCIA RX interrupt
97	UARTA_TX	SCIA TX interrupt
98	UARTB_RX	SCIB RX interrupt
99	UARTB_TX	SCIB TX interrupt
100	CANA INT0	CANA interrupt 0
101	CANA INT1	CANA interrupt 1
102	CANB INT0	CANB interrupt 0
103	CANA INT1	CANB interrupt 1
104	ADCA EVENT	ADCA event interrupt
105	ADCA2	ADCA2 interrupt
106	ADCA3	ADCA3 interrupt
107	ADCA4	ADCA4 interrupt
108	ADCB EVENT	ADCB event interrupt
109	ADCB2	ADCB2 interrupt
110	ADCB3	ADCB3 interrupt
111	ADCB4	ADCB4 interrupt
112	EXTI_LINE0	CLA interrupt 1
113	EXTI_LINE1	CLA interrupt 2
114	EXTI_LINE2	CLA interrupt 3
115	EXTI_LINE3	CLA interrupt 4

Vector ID	G32R501 Name	Txx320F28004x
116	EXTI_LINE4	CLA interrupt 5
117	EXTI_LINE5	CLA interrupt 6
118	EXTI_LINE6	CLA interrupt 7
119	EXTI_LINE7	CLA interrupt 8
120	EXTI_LINE8	XINT3 interrupt
121	EXTI_LINE9	XINT4 interrupt
122	EXTI_LINE10	XINT5 interrupt
123	EXTI_LINE11	Reserved
124	EXTI_LINE12	Reserved
125	EXTI_LINE13	Reserved
126	EXTI_LINE14	FPU overflow interrupt
127	EXTI_LINE15	FPU underflow interrupt
128	Reserved	Reserved
129	FPU DZC	Reserved
130	FPU IDC	Reserved
131	FPU IOC	Reserved
132	FPU OFC	Reserved
133	FPU UFC	Reserved
134	FPU IXC	Reserved
135	Reserved	Reserved
136	Reserved	Reserved
137	Reserved	Reserved
138	Reserved	Reserved
139	Reserved	Reserved
140	Reserved	Reserved
141	Reserved	Reserved
142	Reserved	Reserved
143	Reserved	Reserved
144	Reserved	Reserved
145	Reserved	Reserved
146	Reserved	Reserved
147	Reserved	Reserved

Vector ID	G32R501 Name	Txx320F28004x
148	Reserved	Reserved
149	Reserved	Reserved
150	Reserved	Reserved
151	Reserved	Reserved
152	Reserved	Reserved
153	Reserved	Reserved
154	Reserved	Reserved
155	Reserved	Reserved
156	Reserved	Reserved
157	CAP6 HRcalibration	ECAP6 HRcalibration interrupt
158	CAP7 HRcalibration	ECAP7 HRcalibration interrupt
159	Reserved	Reserved
160	SDF	SDFM1 interrupt
161	Reserved	Reserved
162	Reserved	Reserved
163	Reserved	Reserved
164	SDF DR INT1	SDFM1 DR interrupt 1
165	SDF DR INT2	SDFM1 DR interrupt 2
166	SDF DR INT3	SDFM1 DR interrupt 3
167	SDF DR INT4	SDFM1 Dr interrupt 4
168	Reserved	Reserved
169	Reserved	Reserved
170	Reserved	Reserved
171	Reserved	Reserved
172	Reserved	Reserved
173	Reserved	Reserved
174	Reserved	Reserved
175	Reserved	Reserved
176	Reserved	Reserved
177	Reserved	Reserved
178	-	FSITX_INT1
179	-	FSITX_INT2

Vector ID	G32R501 Name	Txx320F28004x
180	-	FSIRX_INT1
181	-	FSIRX_INT2
182	Reserved	CLAPROMCRC interrupt
183	Reserved	Reserved
184	LINA INT1	LINA interrupt 0
185	LINA INT2	LINA interrupt 1
186	Reserved	Reserved
187	Reserved	Reserved
188	PMBUSA interrupt	PMBUSA interrupt
189	Reserved	Reserved
190	Reserved	Reserved
191	Reserved	Reserved
192	Reserved	Reserved
193	Reserved	Reserved
194	Reserved	Reserved
195	Reserved	Reserved
196	Reserved	Reserved
197	Reserved	Reserved
198	Reserved	Reserved
199	Reserved	Reserved
200	ADCC EVENT	ADCC event interrupt
201	ADCC2	ADCC2 interrupt
202	ADCC3	ADCC3 interrupt
203	ADCC4	ADCC4 interrupt
204	Reserved	Reserved
205	Reserved	Reserved
206	Reserved	Reserved
207	Reserved	Reserved
208	Reserved	Reserved
209	Reserved	Reserved
210	Reserved	Reserved
211	Reserved	Reserved

Vector ID	G32R501 Name	Txx320F28004x
212	Reserved	Reserved
213	Reserved	Reserved
214	Reserved	Reserved
215	Reserved	Reserved
216	Reserved	Reserved
217	Reserved	RAM correctable error interrupt
218	FLASH_ECC_ERR	FLASH_ECC_ERR interrupt
219	Reserved	RAM access violation interrup
220	SYS_PLL_SLIP	PLL slip interrupt
221	Reserved	Reserved
222	Reserved	CLA overflow interrupt
223	Reserved	CLA underflow interrupt
224	Reserved	-
225	CTI INT0	-
226	CTI INT1	-

3.4. 存储器映射

G32R501 的 Flash 访问路径相较于竞品进行了优化，既支持系统总线访问方式，也支持 ITCM 接口访问。G32R501 内置两个 128bit 位宽 Flash Bank，可配置为 128bit 访问或 256bit 访问模式，提高了使用的灵活性。下表是 G32R501 与 Txx320F28004x 的 Flash 功能差异表。

表格 21 G32R501 与 Txx320F28004x Flash 功能差异表

功能	G32R501	Txx320F28004x
容量	最大 640kB	最大 256kB
存储器组	2 组	2 组
闪存等待状态	0 (SYSCLK ≤ 30) 1 (SYSCLK ≤ 60) 2 (SYSCLK ≤ 90) 3 (SYSCLK ≤120) 4 (SYSCLK ≤ 150) 5 (SYSCLK ≤ 180) 6 (SYSCLK ≤ 210) 7 (SYSCLK ≤ 240) 8 (SYSCLK ≤ 270)	0 (SYSCLK ≤ 20) 1 (SYSCLK ≤ 40) 2 (SYSCLK ≤ 60) 3 (SYSCLK ≤ 80) 4 (SYSCLK ≤ 100)

功能	G32R501	Txx320F28004x
Sleep 模式	/	支持
standby 模式	支持	支持
active 模式	支持	支持
FACC 加速	支持	支持
Busmatrix 加速	支持	支持
读位宽	256bit（单 bank 模式） /128bit（双 bank 模式）	128bit
编程分辨率	32bit	16bit
ECC	4 个 SECDED ECC	2 个 SECDED ECC
擦除	16kB（单 bank 模式）/8kB（双 bank 模式）	8kB

G32R501 支持 8 位寻址，但访问寄存器需要遵循规则如下：

偏移地址：与竞品存在差异，具体参考用户手册或参考表格 26。

访问规则：若寄存器位宽为 16bit，则 G32R501 中需要按照 16bit 访问；若寄存器位宽为 32bit，则 G32R501 中需要按照 32bit 访问。

G32R501 的存储器映射如下表所示。

表格 22 G32R501 的存储器映射

地址范围（Hex）	大小（Max）	描述
0x0000_0000 – 0x0001_FFFF	128KB	CPU0 ITCM（单核版本默认为 64KB，双核版本默认为 48KB）
0xA000_0000 – 0xA001_FFFF	128KB	CPU0 ITCM（CPU0-AHBT） （单核版本默认为 64KB，双核版本默认为 48KB）
0x0000_0000 – 0x0001_FFFF	128KB	CPU1 ITCM（单核版本默认为 0KB，双核版本默认为 8KB）
0xA100_0000 – 0xA101_FFFF	128KB	CPU1 ITCM（CPU1-AHBT） （单核版本默认为 0KB，双核版本默认为 8KB）
0x0008_0000 – 0x0008_BFFF	48KB	FLASH INFO on ITCM
0x0009_0000 – 0x0009_3FFF	16KB	FLASH INFO1 on ITCM
0x0010_0000 - 0x0019_FFFF	640KB	FLASH memory on ITCM
0x0800_0000 - 0x0809_FFFF	640KB	FLASH memory on BUSMATRIX
0x0810_0000 - 0x0810_BFFF	48KB	FLASH INFO on BUSMATRIX
0x0818_0000 - 0x0818_3FFF	16KB	FLASH INFO1 on BUSMATRIX
0x0900_0000 - 0x0901_FFFF	128KB	FLASH ECC
0x1000_0000 – 0x1001_FFFF	128KB	Boot ROM
0x1002_0000 – 0x1002_FFFF	64KB	Secure ROM
0x2000_0000 - 0x2001_FFFF	128KB	CPU0 DTCM（单核版本默认为 16KB，双核版本默认为 16KB）

地址范围（Hex）	大小（Max）	描述
0xA010_0000 - 0xA011_FFFF	128KB	CPU0 DTCM（CPU0-AHBT） （单核版本默认为 16KB，双核版本默认为 16KB）
0x2000_0000 – 0x2001_FFFF	128KB	CPU1 DTCM（单核版本默认为 0KB，双核版本默认为 8KB）
0xA110_0000 - 0xA111_FFFF	128KB	CPU1 DTCM（CPU1-AHBT） （单核版本默认为 0KB，双核版本默认为 8KB）
0x2010_0000 - 0x2011_FFFF	128KB	SRAM1（Default 8KB）
0x2020_0000 - 0x2021_FFFF	128KB	SRAM2（Default 8KB）
0x2030_0000 - 0x2031_FFFF	128KB	SRAM3（Default 32KB）

地址总线分配如下：

表格 23 地址总线分配

地址范围（Hex）	总线	功能
0x4000_0000 – 0x4000_FFFF	APB0	APB peripherals
0x4001_0000 – 0x4001_FFFF	APB1	APB peripherals
0x4002_0000 – 0x4002_FFFF	APB2	APB peripherals
0x4003_0000 – 0x4003_FFFF	DEMUX0	AHB peripherals
0x5000_0000 – 0x5000_FFFF	APB3	APB peripherals
0x5000_0000 – 0x5002_FFFF	DEMUX1	AHB peripherals
0x6000_0000 – 0x6FFF_FFFF	-	-
0x5010_0000 – 0x5010_3FFF	APB4	APB peripherals
0x5010_4000 – 0x5010_FFFF	APB5	APB peripherals
0x5010_0000 – 0x5011_FFFF	DEMUX2	AHB peripherals

3.4.1. Flash 映射

在 G32R501x 器件上最多可以使用两个 Flash 存储体（Bank0 为 128KB，Bank1 为 512KB）。Flash 存储体由两个 NVMC（非易失性存储器控制器）进行控制。CPU0 和 CPU1 都可以通过 ITCM/C-BUS 两种方式访问 FLASH MEM，两条路径访问的物理存储空间相同，但使用的地址不相同。所有 FLASH 的擦/写操作通过操作 NVMC 寄存器进行，FLASH MEM 区地址对于 CPU0/1、DMA 只可读，所有写操作将被忽视。不应对正在进行擦除/编程操作的 Flash 存储体进行任何类型的访问。

表格 24 Flash 扇区地址（按存储器 single bank 配置（256 bits 读取宽度））

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
Main	Sector0	0x0800_0000~0x0800_3FFF	0x0010_0000~0x0010_3FFF	16KByte	Bank0 &

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
memory					Bank1 mix
	Sector1	0x0800_4000~0x0800_7FFF	0x0010_4000~0x0010_7FFF	16KByte	Bank0 & Bank1 mix
	Sector2	0x0800_8000~0x0800_BFFF	0x0010_8000~0x0010_BFFF	16KByte	Bank0 & Bank1 mix
	.....	.....	.....	.....	Bank0 & Bank1 mix
	Sector14	0x0803_8000~0x0803_BFFF	0x0013_8000~0x0013_BFFF	16KByte	Bank0 & Bank1 mix
	Sector15	0x0803_C000~0x0803_FFFF	0x0013_C000~0x0013_FFFF	16KByte	Bank0 & Bank1 mix
	Sector16	0x0804_0000~0x0804_1FFF	0x0014_0000~0x0014_1FFF	8KByte	Bank1
	.....	.....	.....	.....	Bank1
	Sector63	0x0809_E000~0x0809_FFFF	0x0019_E000~0x0019_FFFF	8KByte	Bank1
IFREN	Bank0 back-up sector0	0x0810_0000~0x0810_1FFF	0x0008_0000~0x0008_1FFF	8Kbyte	Bank0
	Bank0 back-up sector1	0x0810_2000~0x0810_3FFF	0x0008_2000~0x0008_3FFF	8Kbyte	Bank0
	Bank0 back-up sector2	0x0810_4000~0x0810_5FFF	0x0008_4000~0x0008_5FFF	8KByte	Bank0
	Bank0 back-up sector3	0x0810_6000~0x0810_7FFF	0x0008_6000~0x0008_7FFF	8KByte	Bank0
	Bank1 User Option byte	0x0810_8000~0x0810_9FFF	0x0008_8000~0x0008_9FFF	8Kbyte	Bank1
	Bank1	0x0810_A000~0x0810_BFFF	0x0008_A000~0x0008_BFFF	8Kbyte	Bank1

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
	OTP				
IFREN1	FLASH trimming	0x0818_0000~0x0818_3FFF	0x0009_0000~0x0009_3FFF	16KByte	Bank0 & Bank1 mix
	ECC	0x0900_0000~0x0902_17FF	NA	128Kbyte（half-word access only）	Bank0 & Bank1 mix

表格 25 Flash 扇区地址（按存储器 dual bank 配置（128 bits 读取宽度））

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
Main memory bank0	Sector0	0x0800_0000~0x0800_1FFF	0x0010_0000~0x0010_1FFF	8KByte	Bank0
	Sector1	0x0800_2000~0x0800_3FFF	0x0010_2000~0x0010_3FFF	8KByte	Bank0
	Sector2	0x0800_4000~0x0800_5FFF	0x0010_4000~0x0010_5FFF	8KByte	Bank0
	…….	…….	…….	…….	Bank0
	Sector14	0x0801_C000~0x0801_DFFF	0x0011_C000~0x0011_DFFF	8KByte	Bank0
	Sector15	0x0801_E000~0x0801_FFFF	0x0011_E000~0x0011_FFFF	8KByte	Bank0
Main memory bank1	Sector16	0x0802_0000~0x0802_1FFF	0x0012_0000~0x0012_1FFF	8KByte	Bank1
	Sector17	0x0802_2000~0x0802_3FFF	0x0012_2000~0x0012_3FFF	8KByte	Bank1
	Sector18	0x0802_4000~0x0802_5FFF	0x0012_4000~0x0012_5FFF	8Kbyte1	Bank1
	…….	…….	…….	…….	Bank1
	Sector30	0x0803_C000~0x0803_DFFF	0x0013_C000~0x0013_DFFF	8KByte	Bank1
	Sector31	0x0803_E000~0x0803_FFFF	0x0013_E000~0x0013_FFFF	8KByte	Bank1
	Sector32	0x0804_0000~0x0804_1FFF	0x0014_0000~0x0014_1FFF	8KByte	Bank1
	…….	…….	…….	…….	Bank1
	Sector79	0x0809_E000~0x0809_FFFF	0x0019_E000~0x0019_FFFF	8KByte	Bank1
IFREN	Bank0 back-up sector0	0x0810_0000~0x0810_1FFF	0x0008_0000~0x0008_1FFF	8Kbyte	Bank0
	Bank0 back-up sector1	0x0810_2000~0x0810_3FFF	0x0008_2000~0x0008_3FFF	8Kbyte	Bank0

IP	Name	Sector base on C-BUS interface	Sector base on ITCM interface	Sector size	Bank number
	Bank0 back-up sector2	0x0810_4000~0x0810_5FFF	0x0008_4000~0x0008_5FFF	8KByte	Bank0
	Bank0 back-up sector3	0x0810_6000~0x0810_7FFF	0x0008_6000~0x0008_7FFF	8KByte	Bank0
	Bank1 User Option byte	0x0810_8000~0x0810_9FFF	0x0008_8000~0x0008_9FFF	8Kbyte	Bank1
	Bank1 OTP	0x0810_A000~0x0810_BFFF	0x0008_A000~0x0008_BFFF	8Kbyte	Bank1
IFREN1	FLASH trimming	0x0818_0000~0x0818_3FFF	0x0009_0000~0x0009_3FFF	16KByte	Bank0 & Bank1 mix
	ECC	0x0900_0000~0x0902_17FF	NA	128Kbyte（half-word access only）	Bank0 & Bank1 mix

3.4.2. 外设寄存器内存映射

表格 26 外设寄存器内存映射

Address	Bus	IP
0x4000_0000-0x4000_03FF	APB0	PWM1
0x4000_0400-0x4000_07FF		PWM2
0x4000_0800-0x4000_0BFF		PWM3
0x4000_0C00-0x4000_0FFF		PWM4
0x4000_1000-0x4000_13FF		PWM5
0x4000_1400-0x4000_17FF		PWM6
0x4000_1800-0x4000_1BFF		PWM7
0x4000_1C00-0x4000_1FFF		PWM8
0x4000_2000-0x4000_23FF		CAP1
0x4000_2400-0x4000_27FF		CAP2
0x4000_2800-0x4000_2BFF		CAP3
0x4000_2C00-0x4000_2FFF		CAP4
0x4000_3000-0x4000_33FF		CAP5
0x4000_3400-0x4000_37FF		CAP6
0x4000_3800-0x4000_3BFF		CAP7

Address	Bus	IP
0x4000_3C00-0x4000_3FFF		SDF
0x4000_4000-0x4000_FFFF		Reserved
0x4001_1C00-0x4001_1FFF	APB1	COMP1
0x4001_2000-0x4001_23FF		COMP2
0x4001_2400-0x4001_27FF		COMP3
0x4001_2800-0x4001_2BFF		COMP4
0x4001_2C00-0x4001_2FFF		COMP5
0x4001_3000-0x4001_33FF		COMP6
0x4001_3400-0x4001_37FF		COMP7
0x4001_3800-0x4001_3BFF		QEP1
0x4001_3C00-0x4001_3FFF		QEP2
0x4001_4000-0x4001_FFFF		Reserved
0x4002_0000-0x4002_03FF	APB2	ADCA
0x4002_0400-0x4002_07FF		ADCB
0x4002_0800-0x4002_0BFF		ADCC
0x4002_0C00-0x4002_FFFF		Reserved
0x4003_0000-0x4003_07FF	DEMUX0	GPIOCTRL
0x4003_0800-0x4003_0BFF		GPIODATA
0x4003_0C00-0x4003_0C7F		INPUTXBAR
0x4003_0C80-0x4003_0FFF		SyncSocREG
0x4003_1000-0x4003_103F		XBAR_REG
0x4003_11C0-0x4003_123F		PWM_XBAR_REG
0x4003_1240-0x4003_12BF		FLB_XBAR_REG
0x4003_12C0-0x4003_133F		OUTPUT_XBAR_REG
0x4003_1400-0x4003_FFFF		Reserved
0x5000_0000-0x5000_03FF	APB3	LIN
0x5000_0400-0x5000_0BFF		Reserved
0x5000_0C00-0x5000_0FFF		UARTA
0x5000_1000-0x5000_13FF		SPIA
0x5000_1400-0x5000_17FF		I2C
0x5000_1800-0x5000_1BFF		DACA
0x5000_1C00-0x5000_1FFF		DACB

Address	Bus	IP
0x5000_2000-0x5000_27FF		CANA
0x5000_2800-0x5000_FFFF		Reserved
0x5000_0000-0x5000_FFFF	DEMUX1	APB3
0x5001_0000-0x5001_07FF		NVMC
0x5001_0800-0x5001_0BFF		CFGSMS
0x5001_0C00-0x5001_FFFF		Reserved
0x5002_0000-0x5002_3FFF		SYSC
0x5002_4000-0x5002_5FFF		DCS
0x5002_6000-0x5002_63FF		QSPI
0x5002_6400-0x5002_64BF		WWDT
0x5002_64C0-0x5002_67FF		NMIWDT
0x5002_6800-0x5002_7FFF		Reserved
0x5002_8000-0x5002_83FF		ANALOG_SUB_SYSTEM
0x5002_8400-0x5002_FFFF		Reserved
0x6000_0000-0x6FFF_FFFF		QSPI_MEMORY
0x5010_0000-0x5010_03FF	APB4	UARTB
0x5010_0400-0x5010_07FF		SPIB
0x5010_0800-0x5010_0BFF		PMBUS
0x5010_0C00-0x5010_0FFF		Reserved
0x5010_1000-0x5010_13FF		Reserved
0x5010_1400-0x5010_17FF		DCC
0x5010_1800-0x5010_1FFF		CANB
0x5010_2000-0x5010_27FF		FLB1
0x5010_2800-0x5010_2FFF		FLB2
0x5010_3000-0x5010_37FF		FLB3
0x5010_3800-0x5010_3FFF		FLB4
0x5010_0000-0x5010_3FFF	DEMUX2	APB4
0x5010_4000-0x5010_FFFF		APB5
0x5011_0000-0x5011_03FF		TMR0
0x5011_0400-0x5011_07FF		TMR1
0x5011_0800-0x5011_0BFF		TMR2
0x5011_0C00-0x5011_0FFF		DMA

Address	Bus	IP
0x5011_1000-0x5011_13FF		EXTI
0x5011_1400-0x5011_FFFF		Reserved

3.5. 时钟树差异

G32R501 时钟使用与 Txx320F28004x 类似，支持 INTOSC1、INTOSC2、XTAL、PLL 等。其中 G32R501 内部 PLL 输出频率范围更大，最高支持 300MHz 时钟输出（PLLRAWCLK）。此外，由于是双核架构，且 CPU 以及 CPU 子系统工作频率高于外设（外设最高工作频率 125MHz），因此额外的时钟控制部分由新增的时钟控制单元模块进行控制。

时钟控制单元模块在 Txx320F28004x 的时钟控制逻辑上增加了 APB 时钟分频以及双核系统时钟控制，更多时钟结构地详细信息，请参考《G32R501 用户手册》。

表格 27 时钟结构对比

时钟	G32R501	Txx320F28004x
OSCCLK	支持	支持
PLLRAWCLK	支持	支持
WDCLK	支持	支持
PLLSYSCLK	支持	支持
SYSCLK	支持	支持
LSPCLK	支持	支持
APBCLK	支持	不支持
CPU0CLK	支持	支持
CPU1CLK	支持	不支持

3.6. 外设差异

Txx320F28004x 的 DMA 不支持 SCI/I2C/Flash 的数据传输，G32R501 的 DMA 支持所有总线外设的的数据传输。

4. 使用库进行固件移植

在 Txx320F28004x 系列微控制器迁移到 G32R501 系列微控制器的过程中，固件库的移植是一个关键步骤。固件库不仅提供了对硬件外设的抽象和封装，还简化了开发流程，提高了代码的可维护性和可移植性。不同微控制器系列的固件库在 API 设计、命名规则和实现方式上可能存在显著差异，因此在迁移过程中，需要对现有固件库进行详细分析和适配。

本章节将介绍如何将 Txx320F28004x 系列的固件库迁移到 G32R501 系列。我们将重点讨论位域库和驱动库的兼容性，以及如何在新的平台上实现现有功能。此外，还将提供一些优化和测试的建议，确保迁移后的固件库能够高效运行。

G32R501 系列微控制器所涉及的 API 情况及迁移参考更详细的内容请参考下面的表单：

- G32R501 device_support（位域库）API 迁移参考表.xlsx
- G32R501 driverlib（固件库）API 迁移参考表.xlsx

4.1. 位域库移植

位域库在嵌入式系统中用于简化对硬件寄存器的访问和操作。

G32R501 和 Txx320F28004x 在寄存器命名和位域库的支持上存在一定差异，在移植过程中，需要对照 G32R501 的用户手册，逐一修改寄存器命名，确保所有寄存器操作都能正确映射到新的硬件平台上。

本小节将详细介绍如何将 Txx320F28004x 的位域库移植到 G32R501，以及在此过程中需要注意的关键事项。我们将详细讨论寄存器命名的差异。

注意：由于篇幅关系，本小结的寄存器命名差异将不会精确至位域的命名差异，此项差异需要用户自己对照 G32R501 的用户手册。

位域库的移植一般步骤总结如下：

- 分析现有代码：
  - 浏览并理解现有代码中使用的位域库，记录所涉及的外设和寄存器。
- 对比寄存器命名：
  - 对照 G32R501 的硬件手册，找到对应的寄存器名称并记录下来。
- 修改代码：
  - 将现有代码中使用的 Txx320F28004x 寄存器名称替换为 G32R501 的寄存器名称。
  - 确保位域定义与新的寄存器布局相匹配。

4.1.1. ADC 模块

G32R501 的 ADC 模块兼容 Txx320F28004x 的 ADC 模块。在寄存器命名上，两者相同。

4.1.2. AS 模块/ ANALOGSUBSYS 模块

G32R501 的 AS 模块兼容 Txx320F28004x 的 ANALOGSUBSYS 模块。在寄存器命名上，两者并不相同。使用时请参考表格 28 的对照情况。

表格 28 AS 模块寄存器命名对照表

G32R501	Txx320F28004x
AS_REGS	**ANALOG_SUBSYS_REGS**
ANAREFPP	ANAREFPP
TSNSCTL	TSNSCTL
ANAREFCTL	ANAREFCTL
VMONCTL	VMONCTL
/	DCDCCTL
/	DCDCSTS
CMPHPMXSEL	CMPHPMXSEL
CMPLPMXSEL	CMPLPMXSEL
CMPHNMXSEL	CMPHNMXSEL
CMPLNMXSEL	CMPLNMXSEL
ADCDACLOOPBACK	/
LOCK	LOCK

4.1.3. CAN 模块

G32R501 的 CAN 模块兼容 Txx320F28004x 的 CAN 模块。在寄存器命名上，两者并不相同。使用时请参考表格 29 的对照情况。

表格 29 CAN 模块寄存器命名对照表

G32R501	Txx320F28004x
CAN_REGS	**CAN_REGS**
CAN_CTRL	CAN_CTL
CAN_EFLG	CAN_ES
CAN_ECNT	CAN_ERRC
CAN_BTIM	CAN_BTR
CAN_INT	CAN_INT
CAN_TEST	CAN_TEST
CAN_PE	CAN_PERR

G32R501	Txx320F28004x
CAN_REGS	**CAN_REGS**
CAN_RAMINIT	CAN_RAM_INIT
CAN_GLBINTEN	CAN_GLB_INT_EN
CAN_GLBINTFLG	CAN_GLB_INT_FLG
CAN_GLBINTCLR	CAN_GLB_INT_CLR
CAN_ABOTR	CAN_ABOTR
CAN_TXREQFLG	CAN_TXRQ_X
CAN_TXRQ_21	CAN_TXRQ_21
CAN_NDATAFLG	CAN_NDAT_X
CAN_NDAT_21	CAN_NDAT_21
CAN_IPENDFLG	CAN_IPEN_X
CAN_IPEN_21	CAN_IPEN_21
CAN_MVALFLG	CAN_MVAL_X
CAN_MVAL_21	CAN_MVAL_21
CAN_IP_MUX21	CAN_IP_MUX21
CAN_IF1COM	CAN_IF1CMD
CAN_IF1MASK	CAN_IF1MSK
CAN_IF1ARB	CAN_IF1ARB
CAN_IF1MCTRL	CAN_IF1MCTL
CAN_IF1DATAA	CAN_IF1DATA
CAN_IF1DATAB	CAN_IF1DATB
CAN_IF2COM	CAN_IF2CMD
CAN_IF2MASK	CAN_IF2MSK
CAN_IF2ARB	CAN_IF2ARB
CAN_IF2MCTRL	CAN_IF2MCTL
CAN_IF2DATAA	CAN_IF2DATA
CAN_IF2DATAB	CAN_IF2DATB
CAN_IF3OBS	CAN_IF3OBS
CAN_IF3MASK	CAN_IF3MSK
CAN_IF3ARB	CAN_IF3ARB
CAN_IF3MCTRL	CAN_IF3MCTL
CAN_IF3DATAA	CAN_IF3DATA

G32R501	Txx320F28004x
CAN_REGS	**CAN_REGS**
CAN_IF3DATAB	CAN_IF3DATB
CAN_IF3UPD	CAN_IF3UPD

4.1.4. CLA 模块

G32R501 无此模块。

4.1.5. FLB 模块/ CLB 模块

G32R501 的 FLB 模块兼容 Txx320F28004x 的 CLB 模块。在寄存器命名上，两者并不相同。使用时请参考表格 30、表格 31 和表格 32 的对照情况。

表格 30 FLB 模块 LOGIC_CONFIG 寄存器命名对照表

G32R501	Txx320F28004x
FLB_LOGIC_CONFIG_REGS	**CLB_LOGIC_CONFIG_REGS**
FLB_COUNT_RESET	CLB_COUNT_RESET
FLB_COUNT_MODE_1	CLB_COUNT_MODE_1
FLB_COUNT_MODE_0	CLB_COUNT_MODE_0
FLB_COUNT_EVENT	CLB_COUNT_EVENT
FLB_FSM_EXTRA_IN0	CLB_FSM_EXTRA_IN0
FLB_FSM_EXTERNAL_IN0	CLB_FSM_EXTERNAL_IN0
FLB_FSM_EXTERNAL_IN1	CLB_FSM_EXTERNAL_IN1
FLB_FSM_EXTRA_IN1	CLB_FSM_EXTRA_IN1
FLB_LUT4_IN0	CLB_LUT4_IN0
FLB_LUT4_IN1	CLB_LUT4_IN1
FLB_LUT4_IN2	CLB_LUT4_IN2
FLB_LUT4_IN3	CLB_LUT4_IN3
FLB_FSM_LUT_FN1_0	CLB_FSM_LUT_FN1_0
FLB_FSM_LUT_FN2	CLB_FSM_LUT_FN2
FLB_LUT4_FN1_0	CLB_LUT4_FN1_0
FLB_LUT4_FN2	CLB_LUT4_FN2
FLB_FSM_NEXT_STATE_0	CLB_FSM_NEXT_STATE_0
FLB_FSM_NEXT_STATE_1	CLB_FSM_NEXT_STATE_1
FLB_FSM_NEXT_STATE_2	CLB_FSM_NEXT_STATE_2

G32R501	Txx320F28004x
FLB_LOGIC_CONFIG_REGS	**CLB_LOGIC_CONFIG_REGS**
FLB_MISC_CONTROL	CLB_MISC_CONTROL
FLB_OUTPUT_LUT_0	CLB_OUTPUT_LUT_0
FLB_OUTPUT_LUT_1	CLB_OUTPUT_LUT_1
FLB_OUTPUT_LUT_2	CLB_OUTPUT_LUT_2
FLB_OUTPUT_LUT_3	CLB_OUTPUT_LUT_3
FLB_OUTPUT_LUT_4	CLB_OUTPUT_LUT_4
FLB_OUTPUT_LUT_5	CLB_OUTPUT_LUT_5
FLB_OUTPUT_LUT_6	CLB_OUTPUT_LUT_6
FLB_OUTPUT_LUT_7	CLB_OUTPUT_LUT_7
FLB_HLC_EVENT_SEL	CLB_HLC_EVENT_SEL
FLB_COUNT_MATCH_TAP_SEL	CLB_COUNT_MATCH_TAP_SEL
FLB_OUTPUT_COND_CTRL_0	CLB_OUTPUT_COND_CTRL_0
FLB_OUTPUT_COND_CTRL_1	CLB_OUTPUT_COND_CTRL_1
FLB_OUTPUT_COND_CTRL_2	CLB_OUTPUT_COND_CTRL_2
FLB_OUTPUT_COND_CTRL_3	CLB_OUTPUT_COND_CTRL_3
FLB_OUTPUT_COND_CTRL_4	CLB_OUTPUT_COND_CTRL_4
FLB_OUTPUT_COND_CTRL_5	CLB_OUTPUT_COND_CTRL_5
FLB_OUTPUT_COND_CTRL_6	CLB_OUTPUT_COND_CTRL_6
FLB_OUTPUT_COND_CTRL_7	CLB_OUTPUT_COND_CTRL_7

表格 31 FLB 模块 LOGIC_ CONTROL 寄存器命名对照表

G32R501	Txx320F28004x
FLB_LOGIC_CONTROL_REGS	**CLB_LOGIC_CONTROL_REGS**
FLB_LOAD_EN	CLB_LOAD_EN
FLB_LOAD_ADDR	CLB_LOAD_ADDR
FLB_LOAD_DATA	CLB_LOAD_DATA
FLB_INPUT_FILTER	CLB_INPUT_FILTER
FLB_IN_MUX_SEL_0	CLB_IN_MUX_SEL_0
FLB_LCL_MUX_SEL_1	CLB_LCL_MUX_SEL_1
FLB_LCL_MUX_SEL_2	CLB_LCL_MUX_SEL_2
FLB_BUF_PTR	CLB_BUF_PTR
FLB_GP_REG	CLB_GP_REG

G32R501	Txx320F28004x
FLB_LOGIC_CONTROL_REGS	**CLB_LOGIC_CONTROL_REGS**
FLB_OUT_EN	CLB_OUT_EN
FLB_GLBL_MUX_SEL_1	CLB_GLBL_MUX_SEL_1
FLB_GLBL_MUX_SEL_2	CLB_GLBL_MUX_SEL_2
FLB_PRESCALE_CTRL	CLB_PRESCALE_CTRL
FLB_INTR_TAG_REG	CLB_INTR_TAG_REG
FLB_LOCK	CLB_LOCK
FLB_DBG_OUT_2	CLB_DBG_OUT_2
FLB_DBG_R0	CLB_DBG_R0
FLB_DBG_R1	CLB_DBG_R1
FLB_DBG_R2	CLB_DBG_R2
FLB_DBG_R3	CLB_DBG_R3
FLB_DBG_C0	CLB_DBG_C0
FLB_DBG_C1	CLB_DBG_C1
FLB_DBG_C2	CLB_DBG_C2
FLB_DBG_OUT	CLB_DBG_OUT

表格 32 FLB 模块 DATA_EXCHANGE 寄存器命名对照表

G32R501	Txx320F28004x
FLB_DATA_EXCHANGE_REGS	**CLB_DATA_EXCHANGE_REGS**
FLB_PUSH[4]	CLB_PUSH[4]
FLB_PULL[4]	CLB_PULL[4]

4.1.6. FLBXBAR 模块/CLBXBAR 模块

G32R501 的 FLBXBAR 模块兼容 Txx320F28004x 的 CLBXBAR 模块。在寄存器命名上，两者相同。

4.1.7. COMP 模块/CMPSS 模块

G32R501 的 COMP 模块兼容 Txx320F28004x 的 CMPSS 模块。在寄存器命名上，两者相同。

4.1.8. TMR 模块/CPUTIMER 模块

G32R501 的 TMR 模块兼容 Txx320F28004x 的 CPUTIMER 模块。在寄存器命名上，两者相同。

4.1.9. DAC 模块

G32R501 的 DAC 模块兼容 Txx320F28004x 的 DAC 模块。在寄存器命名上，两者相同。

4.1.10. DCCOMP 模块/ DCC 模块

G32R501 的 DCCOMP 模块兼容 Txx320F28004x 的 DCC 模块。在寄存器命名上，两者并不相同。使用时请参考表格 33 的对照情况。

表格 33 DCCOMP 模块寄存器命名对照表

G32R501	Txx320F28004x
DCCOMP_REGS	**DCC_REGS**
DCCOMPCTRL	DCCGCTRL
DCCOMPREV	DCCREV
DCCOMPCNTSEED0	DCCCNTSEED0
DCCOMPVALSEED0	DCCVALIDSEED0
DCCOMPCNTSEED1	DCCCNTSEED1
DCCOMPFLG	DCCSTATUS
DCCOMPCLKSRCCNT0	DCCCNT0
DCCOMPCLKSRCVAL0	DCCVALID0
DCCOMPCLKSRCCNT1	DCCCNT1
DCCOMPCLKSRCSEL1	DCCCLKSRC1
DCCOMPCLKSRCSEL0	DCCCLKSRC0

4.1.11. DCS 模块/DCSM 模块

G32R501 的 DCS 模块兼容 Txx320F28004x 的 DCSM 模块。在寄存器命名上，两者并不相同。使用时请参考表格 34、表格 35、表格 36、表格 37 和表格 38 的对照情况。

表格 34 DCS 模块 B0Z1 寄存器命名对照表

G32R501	Txx320F28004x
DCS_B0Z1_REGS	**DCSM_BANK0_Z1_REGS**
B0Z1LP	B0_Z1_LINKPOINTER
Z1LOCK	Z1_OTPSECLOCK
Z1BOOTH	Z1_BOOTDEF_HIGH
B0Z1LPERR	B0_Z1_LINKPOINTERERR
Z1BOOTPINCFG	Z1_BOOTPIN_CONFIG
Z1GENPUR	Z1_GPREG2
Z1BOOTL	Z1_BOOTDEF_LOW
Z1CSKEY0	Z1_CSMKEY0
Z1CSKEY1	Z1_CSMKEY1
Z1CSKEY2	Z1_CSMKEY2
Z1CSKEY3	Z1_CSMKEY3
Z1CSCSTS	Z1_CR
B0Z1GRABSECT	B0_Z1_GRABSECTR
Z1GRABRAM	Z1_GRABRAMR
B0Z1EXESECT	B0_Z1_EXEONLYSECTR
Z1EXERAM	Z1_EXEONLYRAMR

表格 35 DCS 模块 B0Z2 寄存器命名对照表

G32R501	Txx320F28004x
DCS_B0Z2_REGS	**DCSM_BANK0_Z2_REGS**
B0Z2LP	B0_Z2_LINKPOINTER
Z2LOCK	Z2_OTPSECLOCK
B0Z2LPERR	B0_Z2_LINKPOINTERERR
Z2CSKEY0	Z2_CSMKEY0
Z2CSKEY1	Z2_CSMKEY1
Z2CSKEY2	Z2_CSMKEY2
Z2CSKEY3	Z2_CSMKEY3
Z2CSCSTS	Z2_CR
B0Z2GRABSECT	B0_Z2_GRABSECTR
Z2GRABRAM	Z2_GRABRAMR

G32R501	Txx320F28004x
DCS_B0Z2_REGS	**DCSM_BANK0_Z2_REGS**
B0Z2EXESECT	B0_Z2_EXEONLYSECTR
Z2EXERAM	Z2_EXEONLYRAMR

表格 36 DCS 模块 COMMON 寄存器命名对照表

G32R501	Txx320F28004x
DCS_COMMON_REGS	**DCSM_COMMON_REGS**
FLSE	FLSEM
B0SECT	B0_SECTSTAT
RAM	RAMSTAT
B1SECT	B1_SECTSTAT
SEERR	SECERRSTAT
SEERRCLR	SECERRCLR
SEERRSET	SECERRFRC

表格 37 DCS 模块 B1Z1 寄存器命名对照表

G32R501	Txx320F28004x
DCS_B1Z1_REGS	**DCSM_BANK1_Z1_REGS**
B1Z1LP	B1_Z1_LINKPOINTER
B1Z1LPERR	B1_Z1_LINKPOINTERERR
B1Z1GRABSECT	B1_Z1_GRABSECTR
B1Z1EXESECT	B1_Z1_EXEONLYSECTR

表格 38 DCS 模块 B1Z2 寄存器命名对照表

G32R501	Txx320F28004x
DCS_B1Z2_REGS	**DCSM_BANK1_Z2_REGS**
B1Z2LP	B1_Z2_LINKPOINTER
B1Z2LPERR	B1_Z2_LINKPOINTERERR
B2Z2GRABSECT	B1_Z2_GRABSECTR
B1Z2EXESECT	B1_Z2_EXEONLYSECTR

4.1.12. DMA 模块

G32R501 的 DMA 模块兼容 Txx320F28004x 的 DMA 模块。在寄存器命名上，两者相同。

4.1.13. CAP 模块/ECAP 模块

G32R501 的 CAP 模块兼容 Txx320F28004x 的 ECAP 模块。在寄存器命名上，两者并不相同。使用时请参考表格 39 的对照情况。

表格 39 CAP 模块寄存器命名对照表

G32R501	Txx320F28004x
CAP_REGS	**ECAP_REGS**
CCTL0	ECCTL0
CCTL1	ECCTL1
CCTL2	ECCTL2
CEINT	ECEINT
CFLG	ECFLG
CCLR	ECCLR
CFRC	ECFRC

4.1.14. PWM 模块/EPWM 模块

G32R501 的 PWM 模块兼容 Txx320F28004x 的 EPWM 模块。在寄存器命名上，两者并不相同。使用时请参考表格 40 的对照情况。在仿真环境中，如果需要在 PWM 中断函数中打断点，需要设置断点时自动停止计数，让计数器在仿真停止时不进行计数，否者会导致新的中断“阻塞”，导致以为中断不响应。在初始化 PWM 时为外设启用仿真停止信号。

表格 40 PWM 模块寄存器命名对照表

G32R501	Txx320F28004x
PWM_REGS	**EPWM_REGS**
PWMXLINK	EPWMXLINK
PWMLOCK	EPWMLOCK

4.1.15. PWM_XBAR 模块/EPWM_XBAR 模块

G32R501 的 PWM_XBAR 模块兼容 Txx320F28004x 的 EPWM_XBAR 模块。在寄存器命名上，两者相同。

4.1.16. QEP 模块/EQEP 模块

G32R501 的 QEP 模块兼容 Txx320F28004x 的 EQEP 模块。在寄存器命名上，两者相同。

4.1.17. ERAD 模块

G32R501 无此模块。

4.1.18. FLASH 模块

G32R501 的 FLASH 模块和 Txx320F28004x 的 FLASH 模块存在较大的差异。其具体用法请参考相关手册。

4.1.19. GPIO 模块

G32R501 的 GPIO 模块兼容 Txx320F28004x 的 GPIO 模块。在寄存器命名上，两者相同。

但 G32R501 的 GPIO 模块多了几个寄存器，其具体用法请参考相关手册和表格 41 的对照情况。

表格 41 GPIO 模块寄存器差异

G32R501	Txx320F28004x
GPIO_CTRL_REGS	**GPIO_CTRL_REGS**
GPADSEL1	/
GPADSEL2	/
GPAFEN	/
GPBDSEL1	/
GPBDSEL2	/
GPBFEN	/

4.1.20. I2C 模块

G32R501 的 I2C 模块兼容 Txx320F28004x 的 I2C 模块。在寄存器命名上，两者并不相同。使用时请参考表格 42 的对照情况。

表格 42 I2C 模块寄存器命名对照表

G32R501	Txx320F28004x
I2C_REGS	**I2C_REGS**
I2COADDR	I2COAR
I2CIREN	I2CIER
I2CSTS	I2CSTR
I2CCLKL	I2CCLKL
I2CCLKH	I2CCLKH

G32R501	Txx320F28004x
I2C_REGS	**I2C_REGS**
I2CCNT	I2CCNT
I2CRXD	I2CDRR
I2CSADDR	I2CSAR
I2CTXD	I2CDXR
I2CCTRL	I2CMDR
I2CISRC	I2CISRC
I2CEXT	I2CEMDR
I2CPSC	I2CPSC
I2CTXFIFO	I2CFFTX
I2CRXFIFO	I2CFFRX

4.1.21. INTPUT_XBAR 模块

G32R501 的 INTPUT_XBAR 模块兼容 Txx320F28004x 的 INTPUT_XBAR 模块。在寄存器命名上，两者相同。

4.1.22. LIN 模块

G32R501 的 LIN 模块兼容 Txx320F28004x 的 LIN 模块。在寄存器命名上，两者并不相同。使用时请参考表格 43 的对照情况。

表格 43 LIN 模块寄存器命名对照表

G32R501	Txx320F28004x
LIN_REGS	**LIN_REGS**
LINGCTRL0	SCIGCR0
LINGCTRL1	SCIGCR1
LINGCTRL2	SCIGCR2
LINIEN	SCISETINT
LINICLR	SCICLEARINT
LINILEN	SCISETINTLVL
LINILCLR	SCICLEARINTLVL
LINFLG	SCIFLR
LINIVO0	SCIINTVECT0
LINIVO1	SCIINTVECT1

G32R501	Txx320F28004x
LIN_REGS	**LIN_REGS**
LINLCFG	SCIFORMAT
LINBR	BRSR
LINRXED	SCIED
LINRXD	SCIRD
LINTXD	SCITD
LINPCTRL0	SCIPIO0
LINPCTRL2	SCIPIO2
LINCOMP	LINCOMP
LINRXB0	LINRD0
LINRXB1	LINRD1
LINIDMASK	LINMASK
LINID	LINID
LINTXB0	LINTD0
LINTXB1	LINTD1
LINMBRPSC	MBRSR
LINEET	IODFTCTRL
LINGIEN	LIN_GLB_INT_EN
LINGIFLG	LIN_GLB_INT_FLG
LINGICLR	LIN_GLB_INT_CLR

4.1.23. NMI 模块/ NMIINTRUPT 模块

G32R501 的 NMI 模块兼容 Txx320F28004x 的 NMIINTRUPT 模块。在寄存器命名上，两者相同。

4.1.24. OUTPUT_XBAR 模块

G32R501 的 OUTPUT_XBAR 模块兼容 Txx320F28004x 的 OUTPUT_XBAR 模块。在寄存器命名上，两者相同。

4.1.25. PGA 模块

G32R501 无此模块。

4.1.26. PIECTRL 模块

G32R501 无此模块。

4.1.27. PMBUS 模块

G32R501 的 PMBUS 模块兼容 Txx320F28004x 的 PMBUS 模块。在寄存器命名上，两者并不相同。使用时请参考表格 44 的对照情况。

表格 44 PMBUS 模块寄存器命名对照表

G32R501	Txx320F28004x
PMBUS_REGS	**PMBUS_REGS**
PMBUS_MCTRL	PMBMC
PMBUS_TXB	PMBTXBUF
PMBUS_RXB	PMBRXBUF
PMBUS_ACK	PMBACK
PMBUS_STS	PMBSTS
PMBUS_IMASK	PMBINTM
PMBUS_SCTRL	PMBSC
PMBUS_HSADDR	PMBHSA
PMBUS_CTRL	PMBCTRL
PMBUS_TIMCTRL	PMBTIMCTL
PMBUS_CLKTIM	PMBTIMCLK
PMBUS_STATIM	PMBTIMSTSETUP
PMBUS_BUSIDTIM	PMBTIMBIDLE
PMBUS_CLKLTOTIIM	PMBTIMLOWTIMOUT
PMBUS_CLKHTOTIIM	PMBTIMHIGHTIMOUT

4.1.28. UART 模块/SCI 模块

G32R501 的 UART 模块兼容 Txx320F28004x 的 SCI 模块。在寄存器命名上，两者并不相同。使用时请参考表格 45 的对照情况。在 UART 初始化的过程中，如果 RX 对应的 IO 引脚处于悬空状态，初始化完成后可能会接收到无效数据，这是由于 GPIO 没有配置上下拉，导致从 IO 管脚给到内部的数字电路值是无法预测的（与模拟的电气特性相关），因此在初始化 RX 对应的 GPIO 时，配置为“上拉”可避免该问题。

表格 45 UART 模块寄存器命名对照表

G32R501	Txx320F28004x
UART_REGS	**SCI_REGS**
UARTCCR	SCICCR
UARTCTL1	SCICTL1
UARTHBAUD	SCIHBAUD
UARTLBAUD	SCILBAUD
UARTCTL2	SCICTL2
UARTRXST	SCIRXST
UARTRXEMU	SCIRXEMU
UARTRXBUF	SCIRXBUF
UARTTXBUF	SCITXBUF
UARTFFTX	SCIFFTX
UARTFFRX	SCIFFRX
UARTFFCT	SCIFFCT
UARTPRI	SCIPRI

4.1.29. SDF 模块/ SDFM 模块

G32R501 的 SDF 模块兼容 Txx320F28004x 的 SDFM 模块。在寄存器命名上，两者相同。

4.1.30. SPI 模块

G32R501 的 SPI 模块兼容 Txx320F28004x 的 SPI 模块。在寄存器命名上，两者并不相同。使用时请参考表格 46 的对照情况。

表格 46 SPI 模块寄存器命名对照表

G32R501	Txx320F28004x
SPI_REGS	**SPI_REGS**
SPICFG	SPICCR
SPICTRL	SPICTL
SPISTS	SPISTS
SPIBR	SPIBRR
SPIRXEMU	SPIRXEMU
SPIRXBUF	SPIRXBUF
SPITXBUF	SPITXBUF

SPIDAT	SPIDAT
SPITXFIFO	SPIFFTX
SPIRXFIFO	SPIFFRX
SPIFFCTRL	SPIFFCT
SPIPRICTRL	SPIPRI

4.1.31. SYSCTRL 模块

G32R501 的 SYSCTRL 模块兼容 Txx320F28004x 的 SYSCTRL 模块。在寄存器命名上，两者并不相同。寄存器也存在有差异。使用时请参考相关文档，以及表格 47、表格 48、

表格 49 和表格 50 的对照情况。

表格 47 SYSCTRL 模块 DEV 寄存器命名对照表

G32R501	**Txx320F28004x**
DEV_REGS	**DEV_CFG_REGS**
PARTIDL	PARTIDL
PARTIDH	PARTIDH
REVID	REVID
/	DC21
/	FUSEERR
/	SOFTPRES0
SOFTPRES2	SOFTPRES2
SOFTPRES3	SOFTPRES3
SOFTPRES4	SOFTPRES4
SOFTPRES6	SOFTPRES6
SOFTPRES7	SOFTPRES7
SOFTPRES8	SOFTPRES8
SOFTPRES9	SOFTPRES9
SOFTPRES10	SOFTPRES10
SOFTPRES11	/
SOFTPRES13	SOFTPRES13
SOFTPRES14	SOFTPRES14
/	SOFTPRES15
SOFTPRES16	SOFTPRES16

SOFTPRES17	SOFTPRES17
SOFTPRES19	SOFTPRES19
SOFTPRES20	SOFTPRES20
/	SOFTPRES40
PACKSEL	TAP_STATUS

表格 48 SYSCTRL 模块 CPU 寄存器命名对照表

G32R501	**Txx320F28004x**
CPU_REGS	**CPU_SYS_REGS**
CPUSYSLOCK1	CPUSYSLOCK1
/	PIEVERRADDR
PCLKCR0	PCLKCR0
PCLKCR1	/
PCLKCR2	PCLKCR2
PCLKCR3	PCLKCR3
PCLKCR4	PCLKCR4
PCLKCR6	PCLKCR6
PCLKCR7	PCLKCR7
PCLKCR8	PCLKCR8
PCLKCR9	PCLKCR9
PCLKCR10	PCLKCR10
PCLKCR13	PCLKCR13
PCLKCR14	PCLKCR14
/	PCLKCR15
PCLKCR16	PCLKCR16
PCLKCR17	PCLKCR17
PCLKCR18	PCLKCR18
PCLKCR19	PCLKCR19
PCLKCR20	PCLKCR20
PCLKCR21	PCLKCR21
LPMCR	LPMCR
GPIOLPMSEL0	GPIOLPMSEL0
GPIOLPMSEL1	GPIOLPMSEL1
TMR2CLKCTL	TMR2CLKCTL

RESCCLR	RESCCLR
RESC	RESC

表格 49 SYSCTRL 模块 PERIPH 寄存器命名对照表

G32R501	Txx320F28004x
PERIPH_REGS	**PERIPH_AC_REGS**
ADCA_AC	ADCA_AC
ADCB_AC	ADCB_AC
ADCC_AC	ADCC_AC
COMP1AC	CMPSS1_AC
COMP2AC	CMPSS2_AC
COMP3AC	CMPSS3_AC
COMP4AC	CMPSS4_AC
COMP5AC	CMPSS5_AC
COMP6AC	CMPSS6_AC
COMP7AC	CMPSS7_AC
DACA_AC	DACA_AC
DACB_AC	DACB_AC
/	PGA1_AC
/	PGA2_AC
/	PGA3_AC
/	PGA4_AC
/	PGA5_AC
/	PGA6_AC
/	PGA7_AC
PWM1_AC	EPWM1_AC
PWM2_AC	EPWM2_AC
PWM3_AC	EPWM3_AC
PWM4_AC	EPWM4_AC
PWM5_AC	EPWM5_AC
PWM6_AC	EPWM6_AC
PWM7_AC	EPWM7_AC

G32R501	Txx320F28004x
PERIPH_REGS	**PERIPH_AC_REGS**
PWM8_AC	EPWM8_AC
QEP1_AC	EQEP1_AC
QEP2_AC	EQEP2_AC
CAP1_AC	ECAP1_AC
CAP2_AC	ECAP2_AC
CAP3_AC	ECAP3_AC
CAP4_AC	ECAP4_AC
CAP5_AC	ECAP5_AC
CAP6_AC	ECAP6_AC
CAP7_AC	ECAP7_AC
SDF1_AC	SDFM1_AC
FLB1_AC	CLB1_AC
FLB2_AC	CLB2_AC
FLB3_AC	CLB3_AC
FLB4_AC	CLB4_AC
/	CLA1PROMCRC_AC
SPIA_AC	SPIA_AC
SPIB_AC	SPIB_AC
PMBUS_A_AC	PMBUS_A_AC
LIN_A_AC	LIN_A_AC
CANA_AC	DCANA_AC
CANB_AC	DCANB_AC
/	FSIATX_AC
/	FSIARX_AC
HRPWM_A_AC	HRPWM_A_AC
PERIPH_AC_LOCK	PERIPH_AC_LOCK

表格 50 SYSCTRL 模块 SPEC 寄存器命名对照表

G32R501	Txx320F28004x
SPEC_REGS	/
EMUBOOTPINCFG	/
EMUBOOTDEFL	/

G32R501	Txx320F28004x
EMUBOOTDEFH	/
BOOTADDR1	/
BOOTCTRL	/
CFGSMSSEL	/
APBCLKDIVCFG	/
VOS	/

4.1.32. XBAR 模块

G32R501 的 XBAR 模块兼容 Txx320F28004x 的 XBAR 模块。在寄存器命名上，两者相同。

4.1.33. XINT 模块

G32R501 无此模块。

4.2. 固件库移植

本小节将详细介绍如何将 Txx320F28004x 的固件库移植到 G32R501，以及在此过程中需要注意的关键事项。我们将详细讨论固件库差异。

针对以下方面，详细进行差异点对比：

- 驱动库函数 API 命名/API 数量/传参类型/传参个数的差异
- 宏定义命名的差异
- 枚举体/枚举变量命名的差异

4.2.1. ADC 模块

G32R501 的 ADC 模块的驱动库兼容 Txx320F28004x 的 ADC 模块的驱动库。但部分宏定义、枚举变量的命名有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异

- 差异请参考表格 51

- 驱动库枚举体差异
  - 差异请参考表格 52

表格 51 ADC 模块驱动库宏定义命名对照表

G32R501	Txx320F28004x
GEEHY_OTP_DEV_KEY	TI_OTP_DEV_KEY
GEEHY_OTP_DEV_PRG_KEY	TI_OTP_DEV_PRG_KEY

表格 52 ADC 模块驱动库枚举体差异表

G32R501	Txx320F28004x
ADC_Trigger 枚举变量	ADC_Trigger 枚举变量

4.2.2. ASYSCTL 模块

G32R501 的 ASYSCTL 模块的驱动库兼容 Txx320F28004x 的 ASYSCTL 模块的驱动库。但部分 API 命名、API 数量、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 53
- 驱动库 API 差异
  - 数量差异请参考表格 54
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 53
- 驱动库枚举体差异
  - 无差异

表格 53 ASYSCTL 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
AS_xxxxx	ASYSCTL_xxxxx

表格 54 ASYSCTL 模块驱动库 API 数量差异表

G32R501	Txx320F28004x
API 函数名	**API 函数名**
/	ASysCtl_enableDCDC
/	ASysCtl_disableDCDC
/	ASysCtl_getInductorFaultStatus
/	ASysCtl_getSwitchSequenceStatus
/	ASysCtl_lockDCDC

4.2.3. CAN 模块

G32R501 的 CAN 模块的驱动库兼容 Txx320F28004x 的 CAN 模块的驱动库。但部分 API 的传参类型、宏定义命名有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量无差异
  - 传参类型差异请参考表格 55
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 56
- 驱动库枚举体差异
  - 无差异

表格 55 CAN 模块驱动库 API 的传参类型差异表

G32R501	Txx320F28004x
API 传参类型	**API 传参类型**
CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen, const uint8_t *msgData)	CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen, const uint16_t *msgData)
CAN_sendMessage_updateDLC(uint32_t base,	CAN_sendMessage_updateDLC(uint32_t base,

G32R501	Txx320F28004x
API 传参类型	**API 传参类型**
uint32_t objID, uint16_t msgLen, const uint8_t *msgData)	uint32_t objID, uint16_t msgLen, const uint16_t *msgData)
CAN_readMessage(uint32_t base, uint32_t objID, uint8_t *msgData)	CAN_readMessage(uint32_t base, uint32_t objID, uint16_t *msgData)
bool CAN_readMessageWithID(uint32_t base, uint32_t objID, CAN_MsgFrameType *frameType, uint32_t *msgID, uint8_t *msgData)	bool CAN_readMessageWithID(uint32_t base, uint32_t objID, CAN_MsgFrameType *frameType, uint32_t *msgID, uint16_t *msgData)
CAN_writeDataReg(const uint8_t *const data, uint32_t address, uint32_t size)	CAN_writeDataReg(const uint16_t *const data, uint32_t address, uint32_t size)
CAN_readDataReg(uint8_t *data, const uint32_t address, uint32_t size)	CAN_readDataReg(uint16_t *data, const uint32_t address, uint32_t size)

表格 56 CAN 模块驱动库宏定义命名表

G32R501	Txx320F28004x
CAN_STATUS_EWFLG	CAN_STATUS_EWARN
CAN_STATUS_EPFLG	CAN_STATUS_EPASS

4.2.4. CLA 模块

G32R501 无此模块。

4.2.5. FLB 模块/CLB 模块

G32R501 的 FLB 模块的驱动库兼容 Txx320F28004x 的 CLB 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 57
- 驱动库 API 差异

- 数量无差异
- 传参类型无差异
- 传参个数无差异

- 驱动库宏定义命名差异
  - 差异请参考表格 57
- 驱动库枚举体差异
  - 差异请参考表格 57 和表格 58

表格 57 FLB 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
FLB_xxxxx	CLB_xxxxx

表格 58 FLB 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
FLB_LocalInputMux 枚举体变量	CLB_LocalInputMux 枚举体变量
FLB_GlobalInputMux 枚举体变量	CLB_GlobalInputMux 枚举体变量

4.2.6. COMP 模块/CMPSS 模块

G32R501 的 COMP 模块的驱动库兼容 Txx320F28004x 的 CMPSS 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 59
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 59
- 驱动库枚举体差异

- 差异请参考表格 59

表格 59 COMP 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	命名前缀
COMP_xxxxx	CMPSS_xxxxx

4.2.7. cpu.h

G32R501 的 cpu.h 的驱动库兼容 Txx320F28004x 的 cpu.h 的驱动库。但部分宏定义命名有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 60
- 驱动库枚举体差异
  - 无差异

表格 60 CPU.H 驱动库宏定义命名差异表

G32R501	Txx320F28004x
WRPRT_DISABLE	EALLOW
WRPRT_ENABLE	EDIS
IDLE_ASM	IDLE

4.2.8. TMR 模块/CPUTIMER 模块

G32R501 的 TMR 模块的驱动库兼容 Txx320F28004x 的 CPUTIMER 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异

- 差异请参考表格 61

- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 61
- 驱动库枚举体差异
  - 差异请参考表格 61 和表格 62

表格 61 TMR 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
TMR_xxxxx	CPUTIMER_xxxxx
TMR_xxxxx	CPUTimer_xxxxx

表格 62 TMR 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
TMR_ClockSource 枚举体变量	CPUTimer_ClockSource 枚举体变量

4.2.9. DAC 模块

G32R501 的 DAC 模块的驱动库兼容 Txx320F28004x 的 DAC 模块的驱动库。驱动库无差异。

4.2.10. DCCOMP 模块/DCC 模块

G32R501 的 DCCOMP 模块的驱动库兼容 Txx320F28004x 的 DCC 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 63
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异

- 传参个数无差异

- 驱动库宏定义命名差异
  - 有前缀差异，差异请参考表格 63
- 驱动库枚举体差异
  - 有前缀差异，差异请参考表格 63

表格 63 DCCOMP 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	命名前缀
DCCOMP_xxxxx	DCC_xxxxx

4.2.11. DCS 模块/DCSM 模块

G32R501 的 DCS 模块的驱动库兼容 Txx320F28004x 的 DCSM 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 64
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 64 和表格 65
- 驱动库枚举体差异
  - 差异请参考表格 64 和表格 66

表格 64 DCS 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	命名前缀
DCS_xxxxx	DCSM_xxxxx

表格 65 DCS 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
DCS_LOCKFLG	DCSM_UNSECURE
DCS_READCSPS	DCSM_ARMED

表格 66 DCS 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
DCS_SemaphoreZone 枚举体变量	DCSM_SemaphoreZone 枚举体变量

4.2.12. DMA 模块

G32R501 的 DMA 模块的驱动库兼容 Txx320F28004x 的 DMA 模块的驱动库。但部分枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 无差异
- 驱动库枚举体差异
  - 差异请参考表格 67

表格 67 DMA 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
DMA_Trigger 枚举体变量	DMA_Trigger 枚举体变量

4.2.13. driver_inclusive_terminology_mapping.h

G32R501 无此文件。

4.2.14. CAP 模块/ECAP 模块

G32R501 的 CAP 模块的驱动库兼容 Txx320F28004x 的 ECAP 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 68
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 68
- 驱动库枚举体差异
  - 差异请参考表格 68 和表格 69

表格 68 CAP 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
CAP_xxxxx	ECAP_xxxxx

表格 69 CAP 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
CAP_InputCaptureSignals 枚举体变量	ECAP_InputCaptureSignals 枚举体变量

4.2.15. PWM 模块/EPWM 模块

G32R501 的 PWM 模块的驱动库兼容 Txx320F28004x 的 EPWM 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名和结构体成员变量命名有所差异。在仿真环境中，如果需要在 PWM 中断函数中打断点，需要设置断点时自动停止计数，让计数器在仿真停止时不进行计数，否则会导致新的中断“阻塞”，导致以为中断不响应。在初始化 PWM 时加上下面的 API 函数，例如：

DBGMCU_enableDebugHaltSignal(DBGMCU_CORE_CPU0, DBGMCU_DEBUG_HALT_PWM1)

- 驱动库 API 命名差异

- 差异请参考表格 70

- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 70
- 驱动库枚举体差异
  - 差异请参考表格 70
- 驱动库结构体差异
  - 差异请参考表格 70 和表格 71

表格 70 PWM 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
PWM_xxxxx	EPWM_xxxxx

表格 71 PWM 模块驱动库结构体差异表

G32R501	Txx320F28004x
结构体成员变量	**结构体成员变量**
PWM_SignalParams 结构体成员 apbClkInHz	EPWM_SignalParams 结构体成员 sysClkInHz

4.2.16. QEP 模块/EQEP 模块

G32R501 的 QEP 模块的驱动库兼容 Txx320F28004x 的 EQEP 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 72
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异

- 差异请参考表格 72

- 驱动库枚举体差异
  - 差异请参考表格 72

表格 72 QEP 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	命名前缀
QEP_xxxxx	EQEP_xxxxx

4.2.17. ERAD 模块

G32R501 无此模块。

4.2.18. FLASH 模块

G32R501 的 FLASH 模块的驱动库和 Txx320F28004x 的 FLASH 模块的驱动库存在较大差异，具体使用方法请参考相关文档。其中 API 数量、部分枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量差异请参考表格 73
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 无差异
- 驱动库枚举体差异
  - 差异请参考表格 74

表格 73 FLASH 模块驱动库 API 数量差异表

G32R501	Txx320F28004x
API	API
/	Flash_setPumpPowerMode
/	Flash_setPumpActiveGracePeriod
/	Flash_setPumpWakeupTime

/	Flash_isPumpReady
/	Flash_selectLowECCBlock
/	Flash_selectHighECCBlock
Flash_selectLowECCBlock	/
Flash_selectHighECCBlock	/
Flash_flushCache	/
Flash_flushBuffer	/
Flash_getBankMode	/

表格 74 FLASH 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
Flash_BankPowerMode 枚举体变量	Flash_BankPowerMode 枚举体变量
/	Flash_PumpPowerMode 枚举体变量
Flash_BankMode 枚举体变量	/

4.2.19. GPIO 模块

G32R501 的 GPIO 模块的驱动库兼容 Txx320F28004x 的 GPIO 模块的驱动库。但部分 API 命名、API 数量、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 75
- 驱动库 API 差异
  - 数量差异请参考表格 75
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 无差异
- 驱动库枚举体差异
  - 差异请参考表格 76

表格 75 GPIO 模块驱动库 API 命名/数量差异表

G32R501	Txx320F28004x

API	API
GPIO_setMasterCore	GPIO_setControllerCore
GPIO_setMaxOutputFreqEnable	/
GPIO_setDrivingCapability	/

表格 76 GPIO 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
GPIO_IntType 枚举体变量	GPIO_IntType 枚举体变量
GPIO_DRIVING_CAPABILITY 枚举体变量	/
GPIO_CoreSelect 枚举体变量	GPIO_CoreSelect 枚举体变量
GPIO_ExternalIntNum 枚举体变量	GPIO_ExternalIntNum 枚举体变量

4.2.20. HRCAP 模块

G32R501 的 HRCAP 模块的驱动库兼容 Txx320F28004x 的 HRCAP 模块的驱动库。但部分 API 数量和传参个数有所差异。

- 驱动库 API 命名差异
  - 无差异
- 驱动库 API 差异
  - 数量差异请参考表格 77
  - 传参类型无差异
  - 传参个数差异请参考表格 77
- 驱动库宏定义命名差异
  - 无差异
- 驱动库枚举体差异
  - 无差异

表格 77 HRCAP 模块驱动库 API 数量和传参个数差异表

G32R501	Txx320F28004x
API	**API**
/	HRCAP_configCalibrationPeriod
HRCAP_convertEventTimeStampNanoseconds(	HRCAP_convertEventTimeStampNanoseconds(

uint32_t timeStamp, float32_t scaleFactor, uint32_t APBClock)	uint32_t timeStamp, float32_t scaleFactor)

4.2.21. HRPWM 模块

G32R501 的 HRPWM 模块的驱动库兼容 Txx320F28004x 的 HRPWM 模块的驱动库。驱动库无差异。

4.2.22. I2C 模块

G32R501 的 I2C 模块的驱动库兼容 Txx320F28004x 的 I2C 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 78
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 79
- 驱动库枚举体差异
  - 差异请参考表格 80

表格 78 I2C 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
API	**API**
I2C_initMaster	I2C_initController
I2C_setSlaveAddress	I2C_setTargetAddress
I2C_setOwnSlaveAddress	I2C_setOwnAddress

表格 79 I2C 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
I2C_MASTER_SEND_MODE	I2C_CONTROLLER_SEND_MODE
I2C_MASTER_RECEIVE_MODE	I2C_CONTROLLER_RECEIVE_MODE

I2C_SLAVE_SEND_MODE	I2C_TARGET_SEND_MODE
I2C_SLAVE_RECEIVE_MODE	I2C_TARGET_RECEIVE_MODE
I2C_INT_ADDR_SLAVE	I2C_INT_ADDR_TARGET
I2C_STS_INTMASK	I2C_STR_INTMASK
I2C_STS_ADDR_SLAVE	I2C_STS_ADDR_TARGET
I2C_STS_SLAVE_DIR	I2C_STS_TARGET_DIR

表格 80 I2C 模块驱动库枚举体差异表

G32R501	Txx320F28004x
I2C_InterruptSource 枚举变量 I2C_INTSRC_ADDR_ SLAVE	I2C_InterruptSource 枚举变量 I2C_INTSRC_ADDR_TARGET

4.2.23. INTERRUPT 模块

G32R501 的 INTERRUPT 模块的驱动库和 Txx320F28004x 的 INTERRUPT 模块（PIE）的驱动库存在较大差异，具体使用方法请参考 7.2 章节。

4.2.24. LIN 模块

G32R501 的 LIN 模块的驱动库兼容 Txx320F28004x 的 LIN 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 81
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 82
- 驱动库枚举体差异
  - 差异请参考表格 83

表格 81 LIN 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
API	**API**

LIN_setIDSlaveTask	LIN_setIDResponderTask
xxxUARTxxx	xxxSCIxxx

表格 82 LIN 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
LIN_UART_ALL_ERRORS	LIN_SCI_ALL_ERRORS
LIN_UART_FRAME_ERROR	LIN_SCI_FRAME_ERROR
LIN_UART_PARITY_ERROR	LIN_SCI_PARITY_ERROR
LIN_UART_BREAK_ERROR	LIN_SCI_BREAK_ERROR
LIN_UART_INT_BREAK	LIN_SCI_INT_BREAK
LIN_UART_INT_WAKEUP	LIN_SCI_INT_WAKEUP
LIN_UART_INT_TX	LIN_SCI_INT_TX
LIN_UART_INT_RX	LIN_SCI_INT_RX
LIN_UART_INT_TX_DMA	LIN_SCI_INT_TX_DMA
LIN_UART_INT_RX_DMA	LIN_SCI_INT_RX_DMA
LIN_UART_INT_RX_DMA_ALL	LIN_SCI_INT_RX_DMA_ALL
LIN_UART_INT_PARITY	LIN_SCI_INT_PARITY
LIN_UART_INT_OVERRUN	LIN_SCI_INT_OVERRUN
LIN_UART_INT_FRAME	LIN_SCI_INT_FRAME
LIN_UART_INT_ALL	LIN_SCI_INT_ALL

表格 83 LIN 模块驱动库枚举体差异表

G32R501	Txx320F28004x
LIN_UARTCommMode 枚举变量	LIN_SCICommMode 枚举变量
LIN_LINMode 枚举变量	LIN_LINMode 枚举变量
LIN_MessageFilter 枚举变量	LIN_MessageFilter 枚举变量
LIN_UARTParityType 枚举变量	LIN_SCIParityType 枚举变量
LIN_UARTStopBits 枚举变量	LIN_SCIStopBits 枚举变量

4.2.25. MEMCFG 模块

G32R501 无此模块。

4.2.26. PGA 模块

G32R501 无此模块。

4.2.27. pin_map_legacy.h / pin_map.h

由于 G32R501 与 Txx320F28004x 的引脚兼容性存在差异，所以 G32R501 的驱动文件 pin_map_legacy.h / pin_map.h 和 Txx320F28004x 的驱动文件 pin_map_legacy.h / pin_map.h 存在差异。该两文件内容是 GPIO 的复用宏定义，存在差异较多，详细请查看文件。

注意：引脚兼容性可参考《G325Rx1 与 Txx320F28004x 差异手册》的第 2 章第 2 节内容。

4.2.28. PMBUS 模块

G32R501 的 PMBUS 模块的驱动库兼容 Txx320F28004x 的 PMBUS 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 84
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 85
- 驱动库枚举体差异
  - 差异请参考表格 86

表格 84 PMBUS 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
API	**API**
PMBus_initSlaveMode	PMBus_initTargetMode
PMBus_configSlave	PMBus_configTarget
PMBus_putSlaveData	PMBus_putTargetData
PMBus_initMasterMode	PMBus_initControllerMode
PMBus_putMasterData	PMBus_putControllerData
PMBus_configMaster	PMBus_configController
PMBus_setSlaveAddress	PMBus_setTargetAddress

表格 85 PMBUS 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
PMBUS_MASTER_ENABLE_TXPROMEN	PMBUS_CONTROLLER_ENABLE_PRC_CALL
PMBUS_MASTER_ENABLE_GCOMEN	PMBUS_CONTROLLER_ENABLE_GRP_CMD
PMBUS_MASTER_ENABLE_PECBYTEEN	PMBUS_CONTROLLER_ENABLE_PEC
PMBUS_MASTER_ENABLE_BYTECFG	PMBUS_CONTROLLER_ENABLE_EXT_CMD
PMBUS_MASTER_ENABLE_COMCEN	PMBUS_CONTROLLER_ENABLE_CMD
PMBUS_MASTER_ENABLE_READ	PMBUS_CONTROLLER_ENABLE_READ
PMBUS_INT_BUS_IDLE	PMBUS_INT_BUS_FREE
PMBUS_INT_SLAVE_ADDR_READY	PMBUS_INT_TARGET_ADDR_READY
PMBUS_INTSRC_AVAFLG	PMBUS_INTSRC_BUS_FREE
PMBUS_INTSRC_CLKLTOFLG	PMBUS_INTSRC_CLK_LOW_TIMEOUT
PMBUS_INTSRC_READDATA	PMBUS_INTSRC_DATA_READY
PMBUS_INTSRC_ADITDATA	PMBUS_INTSRC_DATA_REQUEST
PMBUS_INTSRC_RDYRSADDR	PMBUS_INTSRC_TARGET_ADDR_READY
PMBUS_INTSRC_ENDFLG	PMBUS_INTSRC_EOM
PMBUS_INTSRC_ALERTTRAN	PMBUS_INTSRC_ALERT
PMBUS_INTSRC_CTRLTRAN	PMBUS_INTSRC_CONTROL
PMBUS_INTSRC_LOSTCTRL	PMBUS_INTSRC_LOST_ARB
PMBUS_INTSRC_CLKHFLG	PMBUS_INTSRC_CLK_HIGH_DETECT
PMBUS_SLAVE_ENABLE_SADDRACKEN	PMBUS_TARGET_ENABLE_MANUAL_ACK
PMBUS_SLAVE_ENABLE_PEC_PROCESSING	PMBUS_TARGET_ENABLE_PEC_PROCESSING
PMBUS_SLAVE_TXPEC	PMBUS_TARGET_TRANSMIT_PEC
PMBUS_SLAVE_ENABLE_COMC	PMBUS_TARGET_ENABLE_MANUAL_CMD_ACK
PMBUS_SLAVE_DISABLE_ADDRESS_MASK	PMBUS_TARGET_DISABLE_ADDRESS_MASK
PMBUS_SLAVE_AUTOACK_1_BYTES	PMBUS_TARGET_AUTO_ACK_1_BYTES
PMBUS_SLAVE_AUTOACK_2_BYTES	PMBUS_TARGET_AUTO_ACK_2_BYTES
PMBUS_SLAVE_AUTOACK_3_BYTES	PMBUS_TARGET_AUTO_ACK_3_BYTES
PMBUS_SLAVE_AUTOACK_4_BYTES	PMBUS_TARGET_AUTO_ACK_4_BYTES
PMBUS_STSMRSTEN	PMBUS_RESET
PMBUS_ENABLE_SLAVE_ALERTLOW	PMBUS_ENABLE_TARGET_ALERT
PMBUS_SET_CLKLTOCFG	PMBUS_SET_BUS_LO_INT_FALLING_EDGE
PMBUS_CTRLCFG	PMBUS_SET_CNTL_INT_RISING_EDGE
PMBUS_ENABLE_THRUENA	PMBUS_ENABLE_IBIAS_A_EN
PMBUS_ENABLE_THRUENB	PMBUS_ENABLE_IBIAS_B_EN
PMBUS_DISABLE_DISCLKLTO	PMBUS_DISABLE_CLK_LOW_TIMEOUT
PMBUS_ENABLE_SLAVE_MODE	PMBUS_ENABLE_TARGET_MODE

G32R501	Txx320F28004x
PMBUS_ENABLE_MASTER_MODE	PMBUS_ENABLE_CONTROLLER_MODE
PMBUS_APB_FREQ_MIN	PMBUS_SYS_FREQ_MIN
PMBUS_APB_FREQ_MAX	PMBUS_SYS_FREQ_MAX

表格 86 PMBUS 模块驱动库枚举体差异表

G32R501	Txx320F28004x
PMBUS_IMASKEdge 枚举变量	PMBus_intEdge 枚举变量

4.2.29. UART 模块/SCI 模块

G32R501 的 UART 模块的驱动库兼容 Txx320F28004x 的 SCI 模块的驱动库。但部分 API 命名、API 传参类型、宏定义命名、枚举变量命名有所差异。在 UART 初始化的过程中，如果 RX 对应的 IO 引脚处于悬空状态，初始化完成后可能会接收到无效数据，这是由于 GPIO 没有配置上下拉，导致从 IO 管脚给到内部的数字电路值是无法预测的（与模拟的电气特性相关），因此在初始化 RX 对应的 GPIO 时，配置为“上拉”可避免该问题。

- 驱动库 API 命名差异
  - 差异请参考表格 87
- 驱动库 API 差异
  - 数量无差异
  - 传参类型差异请参考表格 88
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 87
- 驱动库枚举体差异
  - 差异请参考表格 87

表格 87 UART 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
UART_xxxxx	SCI_xxxxx

表格 88 UART 模块驱动库 API 传参类型差异表

G32R501	Txx320F28004x
API	**API**

UART_writeCharBlockingFIFO(uint32_t base, uint8_t data)	SCI_writeCharBlockingFIFO(uint32_t base, uint16_t data)
UART_writeCharBlockingNonFIFO(uint32_t base, uint8_t data)	SCI_writeCharBlockingNonFIFO(uint32_t base, uint16_t data)
UART_writeCharNonBlocking(uint32_t base, uint8_t data)	SCI_writeCharNonBlocking(uint32_t base, uint16_t data)
static inline uint8_t UART_readCharNonBlocking(uint32_t base)	static inline uint16_t SCI_readCharNonBlocking(uint32_t base)
UART_writeCharArray(uint32_t base, const uint8_t * const array, uint16_t length)	SCI_writeCharArray(uint32_t base, const uint16_t * const array, uint16_t length)
UART_readCharArray(uint32_t base, uint8_t * const array, uint16_t length)	SCI_readCharArray(uint32_t base, uint16_t * const array, uint16_t length)

表格 89 UART 模块驱动库枚举体差异表

G32R501	**Txx320F28004x**
枚举体/枚举变量	**枚举体/枚举变量**
UART_xxxxx	SCI_xxxxx

4.2.30. SDF 模块/ SDFM 模块

G32R501 的 SDF 模块的驱动库兼容 Txx320F28004x 的 SDFM 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 90 和表格 91
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 90 和表格 92

- 驱动库枚举体差异
  - 差异请参考表格 90

表格 90 SDF 模块驱动库命名前缀差异表

G32R501	Txx320F28004x
命名前缀	**命名前缀**
SDF_xxxxx	SDFM_xxxxx

表格 91 SDF 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
SDF_enableMasterInterrupt	SDFM_enableMainInterrupt
SDF_disableMasterInterrupt	SDFM_disableMainInterrupt
SDF_enableMasterFilter	SDFM_enableMainFilter
SDF_disableMasterFilter	SDFM_disableMainFilter

表格 92 SDF 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
SDF_MASTER_INTERRUPT_FLAG	SDFM_MAIN_INTERRUPT_FLAG

4.2.31. SPI 模块

G32R501 的 SPI 模块的驱动库兼容 Txx320F28004x 的 SPI 模块的驱动库。但部分 API 命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 93
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 无差异
- 驱动库枚举体差异
  - 差异请参考表格 94

表格 93 SPI 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
API	**API**
SPI_setSTESignalPolarity	SPI_setPTESignalPolarity

表格 94 SPI 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
SPI_Mode 枚举变量	SPI_Mode 枚举变量
SPI_STEPolarity 枚举变量	SPI_PTEPolarity 枚举变量

4.2.32. SYSCTL 模块

G32R501 的 SYSCTL 模块的驱动库兼容 Txx320F28004x 的 SYSCTL 模块的驱动库。但部分 API 命名、API 返回值、API 数量、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 95
- 驱动库 API 差异
  - 返回值差异请参考表格 95
  - 数量差异请参考表格 95
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 96
- 驱动库枚举体差异
  - 差异请参考表格 97

表格 95 SYSCTL 模块驱动库 API 差异表

G32R501	Txx320F28004x
API	**API**
SysCtl_delay	SYSCTL_DELAY (宏)
static void SysCtl_pollX1Counter(void)	static bool SysCtl_pollX1Counter(void)

SysCtl_getAPBClock	/
SysCtl_setAPBClk	/
SysCtl_enableVOS	/
SysCtl_disableVOS	/
SysCtl_setLDOVoltage	/
SysCtl_enableBootRomPrefetch	/
SysCtl_disableBootRomPrefetch	/
SysCtl_enableSecureRomPrefetch	/
SysCtl_disableSecureRomPrefetch	/
SysCtl_setBootRomLatency	/
SysCtl_setSecureRomLatency	/
/	SysCtl_isWatchdogEnabled
/	SysCtl_getEfuseError
/	SysCtl_getPIEVErrAddr
SysCtl_setCPU1BootAddress	/
SysCtl_enableBootCPU1	/

表格 96 SYSCTL 模块驱动库宏定义差异表

G32R501	Txx320F28004x
SYSCTL_WDT_CHKBITS	SYSCTL_WD_CHKBITS
SYSCTL_WDT_ENRSTKEY	SYSCTL_WD_ENRSTKEY
SYSCTL_WDT_RSTKEY	SYSCTL_WD_RSTKEY
/	SYSCTL_DEVICECAL_CONTEXT_SAVE
/	SYSCTL_DEVICECAL_CONTEXT_RESTORE
/	SYSCTL_NMI_RAMUNCERR
/	SYSCTL_NMI_PIEVECTERR
SYSCTL_NMI_FLBNMI	SYSCTL_NMI_CLBNMI

表格 97 SYSCTL 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
SysCtl_PeripheralPCLOCKCR 枚举变量	SysCtl_PeripheralPCLOCKCR 枚举变量
SysCtl_PeripheralSWRST 枚举变量	SysCtl_PeripheralSOFTPRES 枚举变量
SysCtl_WDPredivider 枚举变量	SysCtl_WDPredivider 枚举变量

SysCtl_WDPrescaler 枚举变量	SysCtl_WDPrescaler 枚举变量
SysCtl_WDMode 枚举变量	SysCtl_WDMode 枚举变量
SysCtl_APBCLKPrescaler 枚举变量	/
SysCtl_AccessPeripheral 枚举变量	SysCtl_AccessPeripheral 枚举变量
SysCtl_AccessMaster 枚举变量	SysCtl_AccessController 枚举变量
SysCtl_ClockOut 枚举变量	SysCtl_ClockOut 枚举变量
SysCtl_SyncInput 枚举变量	SysCtl_SyncInput 枚举变量
SysCtl_SyncInputSource 枚举变量	SysCtl_SyncInputSource 枚举变量
SysCtl_SyncOutputSource 枚举变量	SysCtl_SyncOutputSource 枚举变量
SysCtl_Timer2ClkDivider 枚举体名	SysCtl_Cputimer2ClkDivider 枚举体名
SysCtl_Timer2ClkSource 枚举变量	SysCtl_Cputimer2ClkSource 枚举变量
SysCtl_LDOVolScaler 枚举变量	/

4.2.33. VERSION 模块

G32R501 的 VERSION 模块的驱动库兼容 Txx320F28004x 的 VERSION 模块的驱动库。驱动库无差异。

4.2.34. XBAR 模块

G32R501 的 XBAR 模块的驱动库兼容 Txx320F28004x 的 XBAR 模块的驱动库。但部分 API 命名、宏定义命名、枚举变量命名有所差异。

- 驱动库 API 命名差异
  - 差异请参考表格 98
- 驱动库 API 差异
  - 数量无差异
  - 传参类型无差异
  - 传参个数无差异
- 驱动库宏定义命名差异
  - 差异请参考表格 99
- 驱动库枚举体差异
  - 差异请参考表格 100

表格 98 XBAR 模块驱动库 API 命名差异表

G32R501	Txx320F28004x
API	**API**
XBAR_setPWMMuxConfig	XBAR_setEPWMMuxConfig
XBAR_setFLBMuxConfig	XBAR_setCLBMuxConfig
XBAR_enablePWMMux	XBAR_enableEPWMMux
XBAR_disablePWMMux	XBAR_disableEPWMMux
XBAR_invertPWMSignal	XBAR_invertEPWMSignal
XBAR_lockPWM	XBAR_lockEPWM
XBAR_enableFLBMux	XBAR_enableCLBMux
XBAR_disableFLBMux	XBAR_disableCLBMux
XBAR_invertFLBSignal	XBAR_invertCLBSignal

表格 99 XBAR 模块驱动库宏定义命名差异表

G32R501	Txx320F28004x
XBAR_PWM_CFG_REG_BASE	XBAR_EPWM_CFG_REG_BASE
XBAR_PWM_EN_REG_BASE	XBAR_EPWM_EN_REG_BASE
XBAR_FLB_CFG_REG_BASE	XBAR_CLB_CFG_REG_BASE
XBAR_FLB_EN_REG_BASE	XBAR_CLB_EN_REG_BASE

表格 100 XBAR 模块驱动库枚举体差异表

G32R501	Txx320F28004x
枚举变量	**枚举变量**
XBAR_OutputMuxConfig 枚举变量	XBAR_OutputMuxConfig 枚举变量
XBAR_PWMMuxConfig 枚举变量	XBAR_EPWMMuxConfig 枚举变量
XBAR_FLBMuxConfig 枚举变量	XBAR_CLBMuxConfig 枚举变量
XBAR_InputFlag 枚举变量	XBAR_InputFlag 枚举变量

4.3. 常见问题与解决方案

4.3.1. 位域库命名差异

由于部分兼容模块的寄存器命名上不一致，下面的示例一为 G32R501 UART 模块的寄存器兼容 Txx320F28004x SCI 模块的寄存器样例，它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
UartaRegs.UARTCCR.all = 0x0007;
UartaRegs.UARTCTL1.all = 0x0003;
UartaRegs.UARTCTL2.all = 0x0003;
UartaRegs.UARTCTL2.bit.TXINTENA = 1;
UartaRegs.UARTCTL2.bit.RXBKINTENA = 1;
```

等效于 Txx320F28004x 中的代码：

```
SciaRegs.SCICCR.all = 0x0007;
SciaRegs.SCICTL1.all = 0x0003;
SciaRegs.SCICTL2.all = 0x0003;
SciaRegs.SCICTL2.bit.TXINTENA = 1;
SciaRegs.SCICTL2.bit.RXBKINTENA = 1;
```

即解决方案为：修改兼容模块对应的寄存器命名，以及对应的位域命名。

4.3.2. 固件库 API 命名差异

由于部分兼容模块的固件库 API 命名上不一致，下面的示例一为 G32R501 TMR 模块的固件库 API 兼容 Txx320F28004x CPUTIMER 模块的固件库 API 样例，它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
TMR_startTimer(TMR0_BASE);
TMR_startTimer(TMR1_BASE);
TMR_startTimer(TMR2_BASE);
```

等效于 Txx320F28004x 中的代码：

```
CPUTimer_startTimer(CPUTIMER0_BASE);
CPUTimer_startTimer(CPUTIMER1_BASE);
CPUTimer_startTimer(CPUTIMER2_BASE);
```

即解决方案为：修改兼容模块固件库 API 命名。

4.3.3. 固件库 API 传参类型差异

由于部分兼容模块的固件库 API 传参类型上不一致，下面的示例一为 G32R501 CAN 模块的固件库 API 兼容 Txx320F28004x CAN 模块的固件库 API 样例。示例二为 G32R501 UART 模块的固件库 API 兼容 Txx320F28004x SCI 模块的固件库 API 样例。它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
extern void CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen,
const uint8_t *msgData);

uint8_t txMsgData[2];
//
// Send CAN message data from
// message object 1
//
CAN_sendMessage(myCAN0_BASE, 1, MSG_DATA_LENGTH, txMsgData);
```

等效于 Txx320F28004x 中的代码：

```
extern void CAN_sendMessage(uint32_t base, uint32_t objID, uint16_t msgLen,
const uint16_t *msgData);

uint16_t txMsgData[2];
//
// Send CAN message data from
// message object 1
//
CAN_sendMessage(myCAN0_BASE, 1, MSG_DATA_LENGTH, txMsgData);
```

即解决方案为：修改实参的声明类型，以便调用函数时，符合兼容模块固件库的 API 传参类型。

示例二：

G32R501 的以下代码：

```
extern void UART_writeCharArray(uint32_t base, const uint8_t * const array,
uint16_t length);

//
// Send starting message.
//
msg = "\r\n\n\nHello World!\0";
UART_writeCharArray(UARTA_BASE, (uint8_t *)msg, 17);
```

等效于 Txx320F28004x 中的代码：

```
extern void SCI_writeCharArray(uint32_t base, const uint16_t * const array,
uint16_t length);
```

```
//
// Send starting message.
//
msg = "\r\n\n\nHello World!\0";
SCI_writeCharArray(SCIA_BASE, (uint16_t*)msg, 17);
```

即解决方案为：调用函数时，修改兼容模块固件库 API 传参类型。

4.3.4. 固件库 API 传参个数差异

由于部分兼容模块的固件库 API 传参个数上不一致，下面的示例一为 G32R501 HRCAP 模块的固件库 API 兼容 Txx320F28004x HRCAP 模块的固件库 API 样例，它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
static inline float32_t HRCAP_convertEventTimeStampNanoseconds(
uint32_t timeStamp, float32_t scaleFactor, uint32_t APBClock)

//
// Convert counts to nanoseconds
// using the scale factor
//
onTime1 = HRCAP_convertEventTimeStampNanoseconds(
absCountOn1, hrcapCalResult.scaleFactor, APBClock);
```

等效于 Txx320F28004x 中的代码：

```
static inline float32_t HRCAP_convertEventTimeStampNanoseconds(
uint32_t timeStamp, float32_t scaleFactor)

//
// Convert counts to nanoseconds
// using the scale factor
//
onTime1 = HRCAP_convertEventTimeStampNanoseconds(
absCountOn1, hrcapCalResult.scaleFactor);
```

即解决方案为：调用函数时，添加兼容模块固件库 API 的传参个数。

4.3.5. 宏定义差异

由于部分兼容模块的固件库宏定义命名上不一致，下面的示例一为 G32R501 的固件库宏定义兼容 Txx320F28004x 的固件库宏定义样例，它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
//
// Define to disable writes to protected registers
//
#ifndef WRPRT_DISABLE
#define WRPRT_DISABLE   __wrprt_disable()
#endif
//
// Define to allow writes to protected registers
//
#ifndef WRPRT_ENABLE
#define WRPRT_ENABLE __wrprt_enable()
#endif


//
// code
//
WRPRT_DISABLE;
...
WRPRT_ENABLE;
```

等效于 Txx320F28004x 中的代码：

```
//
// Define to allow writes to protected registers
//
#ifndef EALLOW
#ifndef __TMS320C28XX_CLA__
#define EALLOW __eallow()
#else
#define EALLOW __meallow()
#endif
```

```
#endif

//
// Define to disable writes to protected registers
//
#ifndef EDIS
#ifndef __TMS320C28XX_CLA__
#define EDIS   __edis()
#else
#define EDIS   __medis()
#endif
#endif


//
// code
//
EALLOW;
...
EDIS;
```

即解决方案为：修改兼容模块固件库宏定义命名。

4.3.6. 固件库枚举变量差异

由于部分兼容模块的固件库枚举变量命名上不一致，下面的示例一为 G32R501 ADC 模块的固件库枚举变量兼容 Txx320F28004x ADC 模块的固件库枚举变量样例，它们实现的功能一致，可在实际应用代码中进行直接替换。

示例一：

G32R501 的以下代码：

```
typedef enum
{
    ...
    ADC_TRIGGER_PWM1_SOCA  = 5U,
    ADC_TRIGGER_PWM1_SOCB  = 6U,
    ADC_TRIGGER_PWM2_SOCA  = 7U,
    ADC_TRIGGER_PWM2_SOCB  = 8U,
    ...
} ADC_Trigger;
```

```
//
// code
//
ADC_setupSOC(myADC0_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_PWM1_SOCA,
ADC_CH_ADCIN0, 8U);
```

等效于 Txx320F28004x 中的代码：

```
typedef enum
{
    ...
    ADC_TRIGGER_EPWM1_SOCA  = 5U,
    ADC_TRIGGER_EPWM1_SOCB  = 6U,
    ADC_TRIGGER_EPWM2_SOCA  = 7U,
    ADC_TRIGGER_EPWM2_SOCB  = 8U,
    ...
} ADC_Trigger;
```

```
//
// code
//
ADC_setupSOC(myADC0_BASE, ADC_SOC_NUMBER0, ADC_TRIGGER_EPWM1_SOCA,
ADC_CH_ADCIN0, 8U);
```

即解决方案为：调用函数时，修改兼容模块固件库枚举变量命名。

5. 校准库移植

5.1. SFO 库软件移植

与 Txx320F28004x 类似，G32R5xx 为校准 PWM 提供了一个 C 可调用库 SFO 库（SFO.lib），其中包含一个 SFO 函数，该函数利用硬件并确定最佳 HRP 比例系数。在 HRPWM 运行时，该 SFO 函数有助于动态确定每个 PWMCLK 周期的 HRP 步进。

通过该程序驱动 HRP 校准模块运行 SFO 诊断，并在任何给定时间为器件确定适当的 HRP 比例系数，即每个 PWMCLK 粗步进的 HRP 步进。假设 PWMCLK =TBCLK = 100MHz 且 HRP 步进为 150ps，在 100MHz 时典型的比例系数值为每个 10ns（TBCLK 单元）有 66 HRP 步进。

在 HRP 校准模块上，在任何时候都可以调用 SFO()运行 SFO 诊断。

如果 PWM 通道以 HRPWM 模式运行，则可以在后台随时调用该函数。其作用是对 HRP 校准模块中的诊断逻辑进行利用，然后将当时得到的比例系数结果可以应用于所有运行在 HRPWM 模式下的 PWM 通道。

这个函数返回值来表示不同情况：

- 该程序返回 00：表示校准仍在运行
- 该程序返回 01：表示校准结束并计算出新的比例系数
- 该程序返回 10：表示出现错误，此时 HRP 比例系数大于每个粗步进

SFO.lib 的用法：

- 在工程中添加 SFO.lib 和“Include”文件
- 变量声明：

  例如：int MEP_ScaleFactor; // SFO 库使用的全局变量
- MEP_ScaleFactor 初始化:

  应调用 SFO()函数来计算初始 MEP_ScaleFactor 值。
- 应用程序代码：

  在应用程序运行时，SFO 函数应定期作为较慢的后台循环的一部分重新运行。

6. 数学库移植

在嵌入式系统中，数学库的使用对于实现复杂的数学运算和算法至关重要。从 Txx320F28004x 系列迁移到 G32R501 系列微控制器时，必须对数学库进行适配和优化，以充分利用目标平台的硬件特性。

本章节将重点介绍各种数学库的移植方法，我们将重点讨论数学库的兼容性及自研的 Zidian 扩展单元如何结合数学库进行计算加速。R501 数学库包括：定点运算库（Fixed_Point）、浮点运算单元（FPU）、矢量计算单元（VCU）、定点数学库（Fix32math，对标 IQmath）、快速浮点运算库（FPUfastRTS）和 Zidian 扩展单元。

数学库的移植一般步骤如下：

- 分析现有代码：
  - 浏览并理解现有代码中使用的数学库，记录使用的数学库函数。
- 对比函数命名：
  - 对照以下章节给出的各个数学库的函数命名表，找到相应的数学库函数。
- 修改代码：
  - 将现有代码中使用的数学库函数替换为 G32R501 的数学库函数。

此外，在移植过程中，为了使计算加快，程序计算需要用到的数据（如 FFT buffers）需要存放在特定的内存区域。Txx320F28004x SDK 通过 cmd 文件将数组放在了相应的内存段中。在 R501 中，通过链接文件，用户通过如下示例代码即可将相应数据存放在指定的内存区域。

```
// Create the FFT buffers, and stored in DCTM
SECTION_DTCM_BSS

int32_t ipcb[2*FFT_SIZE];
```

6.1. Fixed_Point

定点运算在嵌入式系统中广泛应用于控制和信号处理任务。本小节将介绍如何将定点运算库从 Txx320F28004x 迁移到 G32R501。

在 TI Fixed_Point 库中存在 ISA_C2800, ISA_C2800_EABI, ISA_C28FPU32 和 ISA_C28FPU32_EABI 四种不同的构建配置，分别对应四个不同的 lib 库文件，c28x_fixedpoint_dsp_library.lib，c28x_fixedpoint_dsp_library_eabi.lib，c28x_fixedpoint_dsp_library_fpu32.lib 和 c28x_fixedpoint_dsp_library_fpu32_eabi.lib。在 Txx320F28004x 使用到的是 c28x_fixedpoint_dsp_library_fpu32_eabi.lib，G32R501 的 Fixed_Point 库完全兼容 Txx320F28004x 所使用的 Fixed_Point 库，下表为 R501 实现函数与 Txx320F28004x 实现函数一览表：

表格 101 R501 Fixed_Point 实现函数与竞品实现函数一览表

G32R501		Txx320F28004x	
函数名	函数名	函数名	函数原型
Bit Reverse Modules			
CFFT32_brev	void CFFT32_brev(int32_t *src, int32_t *dst, uint16_t size)	CFFT32_brev	void CFFT32_brev(int32 *src, int32 *dst, int16 size);
RFFT32_brev	void RFFT32_brev(int32_t *src, int32_t *dst, uint16_t size)	RFFT32_brev	void RFFT32_brev(int32 *src, int32 *dst, int16 size);
RFFT32_brev_RT	void RFFT32_brev_RT(void *handle)	RFFT32_brev_RT	void RFFT32_brev_RT(void *);
FFT Modules			
FFT32_calc	void FFT32_calc(void *handle)	FFT32_calc	void FFT32_calc(void *);
FFT32_init	void FFT32_init(void *handle)	FFT32_init	void FFT32_init(void *);
FFT32_izero	void FFT32_izero(void *handle)	FFT32_izero	void FFT32_izero(void *);
CFFT32_mag	void CFFT32_mag(void *handle)	CFFT32_mag	void CFFT32_mag(void *);
CFFT32_win	void CFFT32_win(void *handle)	CFFT32_win	void CFFT32_win(void *);
RFFT32_split	void RFFT32_split(void *handle)	RFFT32_split	void RFFT32_split(void *);
RFFT32_mag	void RFFT32_mag(void *handle)	RFFT32_mag	void RFFT32_mag(void *);
RFFT32_win	void RFFT32_win(void *handle)	RFFT32_win	void RFFT32_win(void *);
FIR Modules			
FIR16_init	void FIR16_init(void *handle)	FIR16_init	void FIR16_init(void *);
FIR16_calc	void FIR16_calc(void *handle)	FIR16_calc	void FIR16_calc(void *);
FIR16_alt_init	void FIR16_alt_init(void *handle)	FIR16_Alt_init	void FIR16_Alt_init(void *);
FIR16_alt_calc	void FIR16_alt_calc(void *handle)	FIR16_Alt_calc	void FIR16_Alt_calc(void *);
FIR32_init	void FIR32_init(void *handle)	FIR32_init	void FIR32_init(void *);
FIR32_calc	void FIR32_calc(void *handle)	FIR32_calc	void FIR32_calc(void *);
FIR32_alt_init	void FIR32_alt_init(void *handle)	FIR32_Alt_init	void FIR32_Alt_init(void *);
FIR32_alt_calc	void FIR32_alt_calc(void *handle)	FIR32_Alt_calc	void FIR32_Alt_calc(void *);
IIR Modules			
IIR5BIQ16_init	void IIR5BIQ16_init(IIR5BIQ16 *IIR5BIQ16_Handle)	IIR5BIQ16_init	void IIR5BIQ16_init(void *);
IIR5BIQ16_calc	void IIR5BIQ16_calc(IIR5BIQ16 *IIR5BIQ16_Handle)	IIR5BIQ16_calc	void IIR5BIQ16_calc(void *);
IIR5BIQ32_init	void IIR5BIQ32_init(IIR5BIQ32 *IIR5BIQ32_Handle)	IIR5BIQ32_init	void IIR5BIQ32_init(void *);
IIR5BIQ32_calc	void IIR5BIQ32_calc(IIR5BIQ32 *IIR5BIQ32_Handle)	IIR5BIQ32_calc	void IIR5BIQ32_calc(void *)

在文件 fft.h 中进行了 Fixed_Point 库的实现函数的声明及相关结构体参数的定义，用户可以根据函数一览表，将程序移植在 G32R501 上。

示例如下：

```
// Declare and initialize the structure object.
// Use the CFFT32_<n>P_DEFUALTS in the FFT header file if
// unsure as to what values to program the object with.
CFFT32 cfft = CFFT32_128P_DEFAULTS;

//Calling Functions
GET_DWT_CYCLE_COUNT(dwtCycleCounts[0], CFFT32_brev(ipcbsrc, ipcb,
FFT_SIZE));  // Real part bit-reversing

GET_DWT_CYCLE_COUNT(dwtCycleCounts[1], CFFT32_brev(&ipcbsrc[1], &ipcb[1],
FFT_SIZE)); // Imaginary part bit-reversing
```

6.2. FPU

浮点运算单元（FPU）在处理复杂数学运算时显著提升了性能。Txx320F28004x 和 G32R501 均支持 FPU，但前者仅支持单精度浮点运算，而后者则同时支持单精度和双精度浮点运算。本小节将介绍如何在 G32R501 上利用 FPU 进行浮点运算的优化。

R501 FPU 库的 lib 文件名为 g32r501_FPU.lib ，FPU 库完全兼容 Txx320F28004x 所使用的 FPU 库，下表为 R501 实现函数与 Txx320F28004x 实现函数一览表：

表格 102 R501 FPU 实现函数与竞品实现函数一览表

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
FFT Modules			
CFFT_f32	void CFFT_f32(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32	void CFFT_f32(CFFT_F32_STRUCT *);
CFFT_f32t	void CFFT_f32t(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32t	void CFFT_f32t(CFFT_F32_STRUCT *);
CFFT_f32i	void CFFT_f32i(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32i	void CFFT_f32i(CFFT_F32_STRUCT *);
CFFT_f32it	void CFFT_f32it(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32it	void CFFT_f32it(CFFT_F32_STRUCT *);

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
CFFT_f32u	void CFFT_f32u(CFFT_F32_STRUCT_Handle hndCFFT_F32);	CFFT_f32u	void CFFT_f32u(CFFT_F32_STRUCT *);
CFFT_f32ut	void CFFT_f32ut(CFFT_F32_STRUCT_Handle hndCFFT_F32);	CFFT_f32ut	void CFFT_f32ut(CFFT_F32_STRUCT *);
CFFT_f32_mag	void CFFT_f32_mag(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_mag	void CFFT_f32_mag(CFFT_F32_STRUCT *);
CFFT_f32_mag_TMU0	void CFFT_f32_mag_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_mag_TMU0	void CFFT_f32_mag_TMU0(CFFT_F32_STRUCT *);
CFFT_f32s_mag	void CFFT_f32s_mag(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32s_mag	void CFFT_f32s_mag(CFFT_F32_STRUCT *);
CFFT_f32s_mag_TMU0	void CFFT_f32s_mag_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32s_mag_TMU0	void CFFT_f32s_mag_TMU0(CFFT_F32_STRUCT *);
CFFT_f32_pack	void CFFT_f32_pack(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_pack	void CFFT_f32_pack(CFFT_F32_STRUCT *);
CFFT_f32_phase	void CFFT_f32_phase(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_phase	void CFFT_f32_phase(CFFT_F32_STRUCT *);
CFFT_f32_phase_TMU0	void CFFT_f32_phase_TMU0(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_phase_TMU0	void CFFT_f32_phase_TMU0(CFFT_F32_STRUCT *);
CFFT_f32_sincostable	void CFFT_f32_sincostable(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_sincostable	void CFFT_f32_sincostable(CFFT_F32_STRUCT *);
CFFT_f32_unpack	void CFFT_f32_unpack(CFFT_F32_STRUCT_Handle hndCFFT_F32)	CFFT_f32_unpack	void CFFT_f32_unpack(CFFT_F32_STRUCT *);
void CFFT_f32_win	void CFFT_f32_win(float *pBuffer, const float *pWindow, const uint16_t size)	CFFT32_f32_win	void CFFT32_f32_win(float *, float *, uint16_t);
CFFT_f32_win_dual	void CFFT_f32_win_dual(float *pBuffer, const float *pWindow, const uint16_t size)	CFFT32_f32_win_dual	void CFFT32_f32_win_dual(float *, float *, uint16_t);

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
ICFFT_f32	void ICFFT_f32(CFFT_F32_STRUCT_Handle hndCFFT_F32)	ICFFT_f32	void ICFFT_f32(CFFT_F32_STRUCT *);
ICFFT_f32t	void ICFFT_f32t(CFFT_F32_STRUCT_Handle hndCFFT_F32)	ICFFT_f32t	void ICFFT_f32t(CFFT_F32_STRUCT *);
RFFT_f32	void RFFT_f32(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32	void RFFT_f32(RFFT_F32_STRUCT *);
RFFT_f32u	void RFFT_f32u(RFFT_F32_STRUCT_Handle hndRFFT_F32);	RFFT_f32u	void RFFT_f32u(RFFT_F32_STRUCT *);
RFFT_adc_f32	void RFFT_adc_f32(RFFT_ADC_F32_STRUCT_Handle hndRFFT_ADC_F32)	RFFT_adc_f32	void RFFT_adc_f32(RFFT_ADC_F32_STRUCT *);
RFFT_adc_f32u	void RFFT_adc_f32u(RFFT_ADC_F32_STRUCT_Handle hndRFFT_ADC_F32);	RFFT_adc_f32u	void RFFT_adc_f32u(RFFT_ADC_F32_STRUCT *);
RFFT_adc_f32_win	void RFFT_adc_f32_win(uint16_t *pBuffer, const uint16_t *pWindow,const uint16_t size)	RFFT_adc_f32_win	void RFFT_adc_f32_win(RFFT_ADC_F32_STRUCT *);
RFFT_f32_mag	void RFFT_f32_mag(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_mag	void RFFT_f32_mag(RFFT_F32_STRUCT *);
RFFT_f32_mag_TMU0	void RFFT_f32_mag_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_mag_TMU0	void RFFT_f32_mag_TMU0(RFFT_F32_STRUCT *);
RFFT_f32s_mag	void RFFT_f32s_mag(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32s_mag	void RFFT_f32s_mag(RFFT_F32_STRUCT *);
RFFT_f32s_mag_TMU0	void RFFT_f32s_mag_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32s_mag_TMU0	void RFFT_f32s_mag_TMU0(RFFT_F32_STRUCT *);
RFFT_f32_phase	void RFFT_f32_phase(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_phase	void RFFT_f32_phase(RFFT_F32_STRUCT *);
RFFT_f32_phase_TMU0	void RFFT_f32_phase_TMU0(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_phase_TMU0	void RFFT_f32_phase_TMU0(RFFT_F32_STRUCT *);

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
RFFT_f32_sincostable	void RFFT_f32_sincostable(RFFT_F32_STRUCT_Handle hndRFFT_F32)	RFFT_f32_sincostable	void RFFT_f32_sincostable(RFFT_F32_STRUCT *);
RFFT_f32_win	void RFFT_f32_win(float *pBuffer, const float *pWindow, const uint16_t size)	RFFT_f32_win	void RFFT_f32_win(float *, float *, uint16_t)
Filter Modules			
FIR_f32_calc	void FIR_f32_calc(FIR_f32_Handle hndFIR_f32)	FIR_f32_calc	void FIR_f32_calc(FIR_f32_handle);
IIR_f32_calc	void IIR_f32_calc(IIR_f32_Handle hndIIR_f32)	IIR_f32_calc	void IIR_f32_calc(IIR_f32_handle);
Vector Modules			
abs_SP_CV	void abs_SP_CV(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV	void abs_SP_CV(float32 *, const complex_float *, const Uint16);
abs_SP_CV_2	void abs_SP_CV_2(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV_2	void abs_SP_CV_2(float32 *, const complex_float *, const Uint16);
abs_SP_CV_TMU0	void abs_SP_CV_TMU0(float *y, const complex_float *x, const uint16_t N)	abs_SP_CV_TMU0	void abs_SP_CV_TMU0(float32 *, const complex_float *, const Uint16);
add_SP_CSxCV	void add_SP_CSxCV(complex_float *y, const complex_float *x, const complex_float c, const uint16_t N)	add_SP_CSxCV	void add_SP_CSxCV(complex_float *, const complex_float *, const complex_float, const Uint16);
add_SP_CVxCV	void add_SP_CVxCV(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	add_SP_CVxCV	void add_SP_CVxCV(complex_float *, const complex_float *, const complex_float *, const Uint16);
iabs_SP_CV	void iabs_SP_CV(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV	void iabs_SP_CV(float32 *, const complex_float *, const Uint16);
iabs_SP_CV_2	void iabs_SP_CV_2(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV_2	void iabs_SP_CV_2(float32 *, const complex_float *, const Uint16);
iabs_SP_CV_TMU0	void iabs_SP_CV_TMU0(float *y, const complex_float *x, const uint16_t N)	iabs_SP_CV_TMU0	void iabs_SP_CV_TMU0(float32 *, const complex_float *, const Uint16);

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
mac_SP_CVxCV	complex_float mac_SP_CVxCV(const complex_float *w,const complex_float *x, const uint16_t N)	mac_SP_CVxCV	complex_float mac_SP_RVxCV(const complex_float *, const complex_float *, const uint16_t);
mac_SP_i16RVxCV	complex_float mac_SP_i16RVxCV(const complex_float *w, const int16_t *x,const uint16_t N)	mac_SP_RVxCV	complex_float mac_SP_RVxCV(const complex_float *, const float *, const uint16_t);
mac_SP_RVxCV	complex_float mac_SP_RVxCV(const complex_float *w,const float *x, const uint16_t N)	mac_SP_i16RVxCV	complex_float mac_SP_i16RVxCV(const complex_float *, const int16_t *, const uint16_t);
maxidx_SP_RV_2	uint16_t maxidx_SP_RV_2(const float *x, const uint16_t N)	maxidx_SP_RV_2	Uint16 maxidx_SP_RV_2(float32 *, Uint16);
mean_SP_CV_2	complex_float mean_SP_CV_2(const complex_float *x, const uint16_t N)	mean_SP_CV_2	complex_float mean_SP_CV_2(const complex_float *, const Uint16);
median_noreorder_SP_RV	float median_noreorder_SP_RV(const float *x, const uint16_t N)	median_noreorder_SP_RV	float32 median_noreorder_SP_RV(const float32 *, Uint16);
median_SP_RV	float median_SP_RV(float *x, const uint16_t N)	median_SP_RV	float32 median_SP_RV(float32 *, Uint16);
mpy_SP_CSxCS	complex_float mpy_SP_CSxCS(const complex_float w, const complex_float x)	mpy_SP_CSxCS	complex_float mpy_SP_CSxCS(complex_float, complex_float);
mpy_SP_CVxCV	void mpy_SP_CVxCV(complex_float *y, const complex_float *w, const complex_float *x, const uint16_t N)	mpy_SP_CVxCV	void mpy_SP_CVxCV(complex_float *, const complex_float *, const complex_float *, const Uint16);
mpy_SP_CVxCVC	void mpy_SP_CVxCVC(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	mpy_SP_CVxCVC	void mpy_SP_CVxCVC(complex_float *, const complex_float *, const complex_float *, const Uint16);
mpy_SP_RSxRV_2	void mpy_SP_RSxRV_2(float *y, const float *x, const float c,const uint16_t N)	mpy_SP_RSxRV_2	void mpy_SP_RSxRV_2(float32 *, const float32 *, const float32, const Uint16);

G32R501		Txx320F28004x	
函数名	函数原型	函数名	函数原型
mpy_SP_RSxRVxRV_2	void mpy_SP_RSxRVxRV_2(float *y, const float *w, const float *x,const float c, const uint16_t N)	mpy_SP_RSxRVxRV_2	void mpy_SP_RSxRVxRV_2(float32 *, const float32 *, const float32 *, const float32, const Uint16);
mpy_SP_RVxCV	void mpy_SP_RVxCV(complex_float *y, const complex_float *w,const float *x, const uint16_t N)	mpy_SP_RVxCV	void mpy_SP_RVxCV(complex_float *, const complex_float *, const float32 *, const Uint16);
mpy_SP_RVxRV_2	void mpy_SP_RVxRV_2(float *y, const float *w, const float *x,const uint16_t N)	mpy_SP_RVxRV_2	void mpy_SP_RVxRV_2(float32 *, const float32 *, const float32 *, const Uint16);
qsort_SP_RV	void qsort_SP_RV(void *x, const uint16_t N)	qsort_SP_RV	void qsort_SP_RV(void *, Uint16);
rnd_SP_RS	float rnd_SP_RS(const float x)	rnd_SP_RS	float32 rnd_SP_RS(float32);
sub_SP_CSxCV	void sub_SP_CSxCV(complex_float *y, const complex_float *x,const complex_float c, const uint16_t N)	sub_SP_CSxCV	void sub_SP_CSxCV(complex_float *, const complex_float *, const complex_float, const Uint16);
sub_SP_CVxCV	void sub_SP_CVxCV(complex_float *y, const complex_float *w,const complex_float *x, const uint16_t N)	sub_SP_CVxCV	void sub_SP_CVxCV(complex_float *, const complex_float *, const complex_float *, const Uint16);

在 fpu_fft.h 文件中包含了 FPU 库的实现函数的声明及相关结构体参数的定义，用户可以根据实现函数一览表将 Txx320F28004x 上所使用到的 FPU 库函数的使用实例移植在 G32R501 上。

6.3. VCU

矢量计算单元（VCU）常用于高效处理矢量运算。G32R501 通过 DSP 指令集和 Helium 支持实现类似功能。

6.3.1. VCU0 libraries

R501 VCU 库的 lib 文件名为 g32r501_VCU.lib ，VCU 库兼容 Txx320F28004x 所使用的 FPU 库，下表为 R501 实现函数与 Txx320F28004x 实现函数一览表：

表格 103 R501 VCU 实现函数与 Txx320F28004x 实现函数一览表

G32R501	Txx320F28004x

函数名	函数名	函数名	函数原型
Fast Fourier Transform			
cfft16_128p_calc	void cfft16_128p_calc(cfft16_t *cfft16_handle_s)	cfft16_128p_calc	void cfft16_128p_calc (cfft16_t *cfft16_handle_s)
cfft16_256p_calc	void cfft16_256p_calc(cfft16_t *cfft16_handle_s)	cfft16_256p_calc	void cfft16_256p_calc (cfft16_t *cfft16_handle_s)
cfft16_64p_calc	void cfft16_64p_calc(cfft16_t *cfft16_handle_s)	cfft16_64p_calc	void cfft16_64p_calc (cfft16_t *cfft16_handle_s)
cfft16_brev	void cfft16_brev(cfft16_t *cfft16_handle_s)	cfft16_brev	void cfft16_brev (cfft16_t *cfft16_handle_s)
cfft16_flip_re_img	void cfft16_flip_re_img(cfft16_t *cfft16_handle_s)	cfft16_flip_re_img	void cfft16_flip_re_img (cfft16_t *cfft16_handle_s)
cfft16_flip_re_img_conj	void cfft16_flip_re_img_conj(cfft16_t *cfft16_handle_s)	cfft16_flip_re_img_conj	void cfft16_flip_re_img_conj (cfft16_t *cfft16_handle_s)
cfft16_init	void cfft16_init(cfft16_t *cfft16_handle_s)	cfft16_init	void cfft16_init (cfft16_t *cfft16_handle_s)
cfft16_unpack	void cfft16_unpack(cfft16_t *cfft16_handle_s)	cfft16_unpack_asm	void cfft16_unpack_asm (cfft16_t *cfft16_handle_s)
cifft16_pack	void cifft16_pack(cfft16_t *cfft16_handle_s)	cifft16_pack_asm	void cifft16_pack_asm (cfft16_t *cfft16_handle_s)
Cyclic Redundancy Check			
CRC_reset	void CRC_reset(void)	CRC_reset	void CRC_reset (void)

-	-	flipInputBuf_cpu	void flipInputBuf_cpu (uint16 *dst, uint16 *src, uint16 rxLen)
genCRC16P1Table	void genCRC16P1Table(void)	genCRC16P1Table	void genCRC16P1Table ()
genCRC16P2Table	void genCRC16P2Table(void)	genCRC16P2Table	void genCRC16P2Table ()
genCRC32Table	void genCRC32Table(void)	genCRC32Table	void genCRC32Table ()
genCRC8Table	void genCRC8Table(void)	genCRC8Table	void genCRC8Table ()
getCRC16P1_cpu	uint16 getCRC16P1_cpu (uint16 input_crc16_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC16P1_cpu	uint16 getCRC16P1_cpu (uint16 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P1_vcu	uint16 getCRC16P1_vcu(uint32 input_crc16_accum, uint16 * msg, CRC_parity_e parity, uint16 rxLen)	getCRC16P1_vcu	uint16 getCRC16P1_vcu (uint32 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P2_cpu	uint16 getCRC16P2_cpu (uint16 input_crc16_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC16P2_cpu	uint16 getCRC16P2_cpu (uint16 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC16P2_vcu	uint16 getCRC16P2_vcu(uint32 input_crc16_accum, uint16 * msg, CRC_parity_e parity, uint16 rxLen)	getCRC16P2_vcu	uint16 getCRC16P2_vcu (uint32 input_crc16_accum, uint16 *msg, CRC_parity_e parity,uint16 rxLen)
getCRC32_cpu	uint32 getCRC32_cpu (uint32 input_crc32_accum, uint16 * msg, CRC_parity_e parity,uint16 rxLen)	getCRC32_cpu	uint32 getCRC32_cpu (uint32 input_crc32_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)

getCRC32_vcu	uint32 getCRC32_vcu(uint32 input_crc32_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)	getCRC32_vcu	uint32 getCRC32_vcu (uint32 input_crc32_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
-	-	getCRC32_vcu_hilo_order_swap	uint32 getCRC32_vcu_hilo_order_swap (uint32 input_crc32_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
getCRC8_cpu	uint16 getCRC8_cpu (uint16 input_crc8_accum, uint16 * msg, CRC_parity_e parity, uint16 rxLen)	getCRC8_cpu	uint16 getCRC8_cpu (uint16 input_crc8_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
getCRC8_vcu	uint8 getCRC8_vcu(uint32 input_crc8_accum, uint16 *msg, CRC_parity_e parity, uint16 rxLen)	getCRC8_vcu	uint16 getCRC8_vcu (uint32 input_crc8_accum, uint16 ∗msg, CRC_parity_e parity, uint16 rxLen)
Viterbi Decoding (VCU0)			
cnvDec	void cnvDec(int16 nBits, int16 *in_p, int16 *out_p, int16 flag)	cnvDec_asm	void cnvDec_asm (int nBits, int ∗in_p, int ∗out_p, int flag)
cnvDecInit	void cnvDecInit(int16 nTranBits)	cnvDecInit_asm	void cnvDecInit_asm (int nTranBits)
cnvDecMetricRescale	void cnvDecMetricRescale(void)	cnvDecMetricRescale_asm	void cnvDecMetricRescale_asm ()

G32R501 与 Txx320F28004x 函数名不同的函数与未实现的函数说明：

- 函数 cfft16_unpack、cifft16_pack、cnvDec、cnvDecInit 与 cnvDecMetricRescale 只是函数名称改变，函数的参数与作用并未改变。
- 函数：flipInputBuf_cpu

  描述：此函数用于翻转一个 16 位输入缓冲区。

若用户在 FFT 计算时需要用到位翻转，R501 VCU0 提供了函数 void cfft16_brev(cfft16_t *cfft16_handle_s)，cfft16_brev 函数实现了一个 16 点复数 FFT 的位反转操作。通过该操作，可以将输入缓冲区中的数据重新排列，以便于 FFT 算法的后续计算。

- 函数：getCRC32_vcu_hilo_order_swap

  描述：此函数计算消息缓冲区的 32 位 CRC。

  R501 VCU0 库中 getCRC32_vcu 函数实现了使用 VCU 指令计算 32 位 CRC 的功能。

在 vcu0_fft.h 文件中包含了 VCU0 库的实现函数的声明及相关结构体参数的定义，与 Txx320F28004x VCU0 库的相关文件比较，R501 可以基本完全兼容 Txx320F28004x 的 VCU0 库。

6.4. Fix32math（对标 IQmath）

Fix32math 库对标 Txx320F28004x IQmath 库，用于高效实现定点数学运算。

R501 Fix32math 库的 lib 文件名为 g32r501_Fix32math.lib ，完全兼容 Txx320F28004x 所使用的 IQmath 库，下表为 R501 实现函数与 Txx320F28004x 实现函数一览表：

表格 104 R501 IQmath 库实现函数与 Txx320F28004x 实现函数一览表

G32R501	Txx320F28004x
函数名	函数名
Conversion Utilities	
IQN	IQN
IQNtoF	IQNtoF
atoIQN	atoIQN
IQNtoa	IQNtoa
IQNint	IQNint
IQNfrac	IQNfrac
IQtoIQN	IQtoIQN
IQNtoIQ	IQNtoIQ
IQtoQN	IQtoQN
QNtoIQ	QNtoIQ
Shift to Multiply or Divide by Powers of 2	
IQmpy2,4,8…64	IQmpy2,4,8…64
IQdiv2,4,8…64	IQdiv2,4,8…64
Arithmetic Operations	
IQNmpy	IQNmpy
IQNrmpy	IQNrmpy
IQnrsmpy	IQnrsmpy
IQNmpyI32	IQNmpyI32
IQNmpyIQX	IQNmpyIQX

G32R501	Txx320F28004x
函数名	**函数名**
IQNdiv	IQNdiv
Trigonometric Functions	
IQNasin	IQNasin
IQNsin	IQNsin
IQNsinPU	IQNsinPU
IQNacos	IQNacos
IQNcos	IQNcos
IQNcosPU	IQNcosPU
IQNatan2	IQNatan2
IQNatan2PU	IQNatan2PU
IQNatan	IQNatan
Mathematical Utilities	
IQNexp	IQNexp
IQNlog	IQNlog
IQNsqrt	IQNsqrt
IQNisqrt	IQNisqrt
IQNmag	IQNmag
Miscellaneous Utilities	
IQNabs	IQNabs
IQsat	IQsat

由于 Fix32math 库的每个函数都存在 IQ0~IQ30，30 个 IQ 格式。以 IQN 为例，在全局 IQ 函数（IQ 格式 = GLOBAL_Q，GLOBAL_Q 在 Fix32mathLib.h 文件中定义），函数的调用方式为_iq _IQ(float F)；IQ 格式特定的 IQ 函数（IQ 格式=IQ1 到 IQ29）调用方式为_iqN _IQN(float F)

在 Fix32mathLib.h 文件中进行了 IQ 函数的声明，与 Txx320F28004x IQmath 库的相关文件相比较，R501 完全兼容 Txx320F28004x 的 IQmath 库，Fix32math 库完全兼容 Txx320F28004x 的 IQmath 库。

6.5. FPUfastRTS

FPUfastRTS 库用于快速实现浮点运算。

R501 FPUfastRTS 库的 lib 文件名为 g32r501_FPUfastRTS.lib ，完全兼容 Txx320F28004x 所使用的 FPUfastRTS 库，下表为 R501 实现函数与 Txx320F28004x 实现函数一览表：

表格 105 R501 FPUfastRTS 实现函数与 Txx320F28004x 实现函数一览表

G32R501		Txx320F28004x	
函数名	函数名	函数名	函数原型
Arithmetic and Trigonometric			
atan2f	float32_t atan2f_1(float32_t Y, float32_t X);	atan2f	float32_t atan2f (float32_t Y, float32_t X);
atanf	float32_t atanf_1(float32_t X);	atanf	float32_t atanf (float32_t X);
cosf	float32_t cosf_1(float32_t X);	cosf	float32_t cosf (float32_t X);
expf	float32_t expf_1 (float32_t X);	expf	float32_t expf (float32_t X);
-	-	FS$$DIV	float32_t FS $$DIV (float32_t X, float32_t Y);
isqrtf	float32_t isqrtf_1(float32_t X);	isqrtf	float32_t isqrtf (float32_t X);
logf	float32_t logf_1(float32_t X);	logf	float32_t logf (float32_t X);
powf	float32_t powf_1(float32_t X, float32_t Y);	powf	float32_t powf (float32_t X, float32_t Y);
sincosf	void sincosf_1(float32_t radian, float32_t *PtrSin, float32_t *PtrCos);	sincosf	void sincosf (float32_t radian, float32_t *PtrSin, float32_t *PtrCos);
sinf	float32_t sinf_1(float32_t X);	sinf	float32_t sinf (float32_t X);
sqrtf	float32_t sqrtf_1 (float32_t X);	sqrtf	float32_t sqrtf (float32_t X);
acosf	float32_t acosf_1(float32_t X);	acosf	float32_t acosf (float32_t X)
asinf	float32_t asinf_1(float32_t X);	asinf	float32_t asinf (float32_t X)

6.6. Zidian

为扩展 R501 的功能，R501 添加 Zidian 模块，Zidian 模块在某些特定操作上提供了硬件加速，在 Txx320F28004x 中没有直接对应的模块。R501 通过定制指令集和硬件加速单元，实现加快执行常见三角函数和算术运算的能力。

Zidian 中的指令包括 ICAU 和 FACU 两类：

- ICAU：提高整数运算的性能，包括需要 Helium 计算的操作：FFT、复数运算等；CRC 算法。

- FACU：提高浮点运算的性能，包括三角函数的计算、平方根和除法。

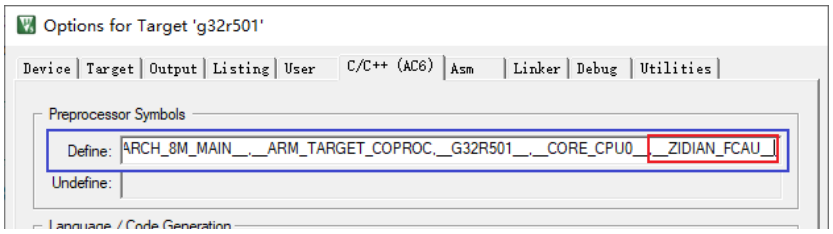
在 Txx320F28004x 中，通过**--tmu_support=tmu0** 来选择是否启用 TMU 功能，来使用 FPU、FPUfastRTS 库中的某些函数的 TMU 变体进行计算。

Txx320F28004x TMU 中的所有算术指令可以被 ICAU 和 FCAU 指令或 Arm® Cortex®-M52 处理器的基本指令和 DSP 指令替代，为了替换一个 TMU 指令，可能需要多于一个 ARM 指令（CDE（Custom Datapath Extension）或 ARM 原生指令）。

注意：CDE（Custom Datapath Extension）是 Arm® Cortex®-M52 处理器中的一项特性，允许设计人员在标准处理器架构上添加自定义的硬件加速功能。这种扩展功能使得开发者可以针对特定应用优化处理器性能，实现更高效的处理能力。

若用户需要使用类似于 Txx320F28004x 中 TMU 变体函数，只需如图 2 在 option for target 中的 C/C++选项卡中的 Define 添加__ZIDIAN_FCAU__，以启用 Zidian 功能，然后调用需要的计算函数即可。以下示例将展示如何适用 Zidian 使用 G32R501 库的中 TMU 变体函数。

图 2 在 keil 配置界面中添加__ZIDIAN_FCAU__宏定义



示例 1：

```
#ifdef __ZIDIAN_FCAU__
#define TMU
#endif
......

#ifdef TMU    // when defined __ZIDIAN_FCAU__ in the project options
    // Run the calculation function and measure the DWT cycle count
simultaneously
    // Calculate magnitude, result stored in CurrentOutPtr
    GET_DWT_CYCLE_COUNT(dwtCycleCounts[3],CFFT_f32_mag_TMU0(hnd_cfft));
#else
    // Run the calculation function and measure the DWT cycle count
simultaneously
    // Calculate magnitude, result stored in CurrentOutPtr
    GET_DWT_CYCLE_COUNT(dwtCycleCounts[3],CFFT_f32_mag(hnd_cfft));
#endif
```

此示例代码展示了在 FPU 库中通过添加宏__ZIDIAN_FCAU__，来使用 CFFT_f32_mag_TMU0 函数。

示例 2：

```
GET_DWT_CYCLE_COUNT(dwtCycleCounts[0], out.f32  = atanf(in.f32));
```

此示例代码展示了在 FPUfastRTS 库中通过添加宏__ZIDIAN_FCAU__，直接调用相应函数，来实现 Zidian 加速运算。

更多有关 Zidian 的使用请查看《G32R5xx Zidian 应用笔记》。

注意：FPUfastRTS 库函数表中，expf、logf、powf 并未实现 Zidian 加速功能。

7. 编程模式移植

不同的微控制器架构在指令集、中断处理机制、寄存器保护等方面存在显著差异，因此在迁移过程中需要特别关注这些编程模式的兼容性和适配问题。

7.1. 双核情况

双核版本的 G32R501 集成了两颗 Arm® Cortex®-M52 内核，两颗内核可以并行工作，在执行高性能计算任务的同时，兼顾实时控制任务。

7.1.1. 启动 CPU1

CPU0 为默认上电时启动，用户需要启动 CPU1 时，需要在 CPU0 在程序中设置 BOOTCTRL1 寄存器为 0x2 来使能 CPU1 的时钟，以启动 CPU1。

7.1.2. 双核通信

CPU0 和 CPU1 可以通过内部处理器间通信单元（IPC）和共享的 RAM 存储区，进行数据交换和通信。

7.1.3. 资源竞争

CPU0 与 CPU1 共享同一片 **Flash 存储空间**。因此，在设计双核应用程序时，需合理分配 CPU0 与 CPU1 所占用的 Flash 存储空间。在 G32R5xx SDK 中提供的双核示例中，Flash 存储空间被均等划分给两个 CPU 核，如果用户希望增加 CPU0 或 CPU1 所占用的 Flash 存储空间，可以通过修改 SDK 示例中提供的链接文件来实现。

此外，**所有外设资源**和 **SRAM1-3 存储区**，均可被 CPU0 与 CPU1 访问。由于 G32R501 没有硬件互斥机制，若两个 CPU 核使用了同一个外设或同一块 SRAM 存储区，将会出现资源竞争的情况。

以下是两个 CPU 同时运行“GPIO 翻转”程序下的示例和说明，可在实际应用中进行参考：

双 Bank 模式下，CPU0 在 Flash Bank0 执行程序，CPU1 也在 Flash Bank0 中执行程序，在 CPU0 程序执行时，CPU1 GPIO 翻转时间变长。

- CPU0 和 CPU1 同时翻转 IO 时，双核会同时访问 GPIO 外设，这会导致总线上存在竞争，由于 CPU0 优先级更高，CPU1 翻转 IO 的周期会变长。
- 在双 Bank 模式下，双核同时访问一块 FLASH；或者单 Bank 模式下，双核同时访问 FLASH 时，会存在总线竞争，由于 CPU0 优先级更高，CPU0 从 FLASH 取数据，CPU1 从 FLASH 取指会受总线竞争影响。

在 SDK 提供的双核示例 ipc_ex4_resource_share 中，展示了如何使用 IPC 模块的 GP（通用）中断来实现资源竞争情况下的同步与互斥访问。有关详细介绍，请参考文档《AN1138_G32R501 IPC 应用笔记》。

7.1.4. CLA 功能

在 Txx320F28004x 中，CLA（控制律加速器）是一个协处理器，设计用于处理实时控制任务。它能够执行独立于主 CPU 的独立运算任务，主要用于加速控制算法如 PID 控制器，滤波器等，这可以显著提高系统的实时性能。CLA 能够直接访问片上 RAM 和外设寄存器，执行浮点和定点运算，适合需要快速响应的控制应用。

如果在应用中使用到 Txx320F28004x 的 CLA，那么可以考虑在 G32R501 中使用 CPU1 来承担相应的角色。具体而言：

- 计算任务：CPU1 能够独立进行计算工作，执行包括函数运算和三角函数运算在内的各种任务。
- 双核通信：通过 IPC（进程间通信）或共享内存区域，可以实现双核之间的高效通信。

7.2. 中断处理

在 Txx320F28004x 和 G32R501 微控制器中，中断控制器和中断处理方式存在显著差异。Txx320F28004x 采用 PIE（Peripheral Interrupt Expansion）机制，而 G32R501 则集成了 Nested Vectored Interrupt Controller（NVIC）。在迁移过程中，需要重新配置中断向量表和中断优先级，以适应新的中断处理机制。

关于 G32R501 微控制器的中断处理除本章节内容外，可参考文档《Arm China Cortex®-M52 Processor Technical Reference Manual》。

7.2.1. 中断嵌套

G32R501 采用 NVIC（Nested Vectored Interrupt Controller）机制，NVIC 支持中断嵌套，允许高优先级的中断打断低优先级的中断服务程序（ISR）。它的特性有：

- G32R501 提供了多达 226 个中断向量，每个中断都有独立的优先级。
- 中断优先级分为抢占优先级和子优先级，灵活性较高，这使得开发者可以精细控制中断的响应顺序。
- G32R501 具有可编程的中断优先级，优先级级别取决于具体的实现方案（通常是 4 位）。开发者可以在应用程序中配置中断优先级，以控制中断的抢占行为。

Txx320F28004x 采用 PIE（Peripheral Interrupt Expansion）机制，其中断优先级设置逻辑是通过分组和通道来实现的，不支持中断嵌套。

7.2.2. 中断优先级

Txx320F28004x 的中断控制流程如下：

- 中断分组：PIE 将中断分为 12 个组，每组包含 8 个通道。例如，PIE 组 1 负责处理外设 1 到外设 8 的中断请求。

- 优先级设置：在每个分组内，可以设置每个通道的优先级。优先级的设置通过 PIEIER（PIE 中断使能寄存器）和 PIEIFR（PIE 中断标志寄存器）来实现。

  ```
  PieCtrlRegs.PIEIERx.bit.INTy = 1;
  ```

G32R501 中断优先级设置需要先配置优先级分组，然后为每个中断源设置具体的预占优先级和子优先级。

- **优先级分组**：在设置具体中断的优先级之前，需要先配置整个芯片的优先级分组。优先级分组定义了预占优先级（Preempt Priority）和子优先级（Sub Priority）的比例。

  可以通过以下函数设置优先级分组：

  Interrupt_setPriorityGroup(uint32_t priorityGroup);

  例如，将优先级分组设置为 7 位预占优先级和 1 位子优先级：

  Interrupt_setPriorityGroup(NVIC_PRIORITYGROUP_7_1);

- **优先级设置**：通过 Interrupt_setPriority 函数，可以为每个中断源设置优先级，包括预占优先级和子优先级。用户可以灵活配置当前中断的优先级至哪个优先级组和优先级层次。

  ```
  Interrupt_setPriority(int32_t interruptNumber, uint32_t preemptPriority,
                        uint32_t subPriority)
  ```

  例如，设置某个外设中断的优先级：

  ```
  Interrupt_setPriority(INT_XINT1, 0, 0);
  ```

7.2.3. 中断使能

在 Txx320F28004x 中，中断控制器 PIE（Peripheral Interrupt Expansion）需要通过使能位 ENPIE 进行配置：

```
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;
```

在 G32R501 中，中断使能通过 NVIC 进行配置。可以使用以下函数来使能特定中断：

```
Interrupt_enable(int32_t interruptNumber);
```

例如，使能某个外设中断：

```
Interrupt_enable(INT_XINT1);
```

7.2.4. 中断退出

在 Txx320F28004x 中，中断服务程序结束时需要显式清除 ACK（Acknowledge）标志位，以通知 PIE 控制器中断处理已完成：

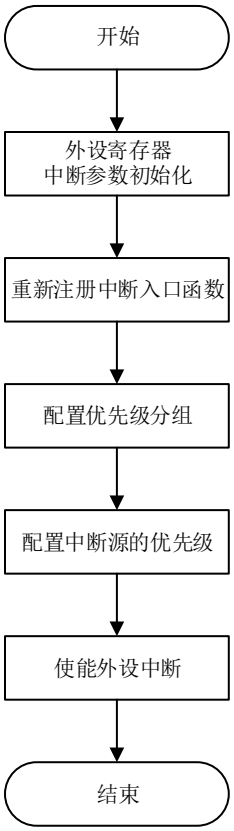
```
PieCtrlRegs.PIEACK.all = PIEACK_GROUPx;
```

*在 G32R501 中，中断服务程序结束时无需显式清除中断挂起状态，NVIC 会自动处理。*一旦中断服务程序执行完毕，返回操作会自动清除中断挂起状态。

7.2.5. 中断初始化流程

在 G32R501 中，若需要使用外设中断功能，流程配置可参考图 3。

图 3 中断初始化流程



代码参考如下：

```
//
// Initializes NVIC and clears NVIC registers. Disables CPU interrupts.
// and clear all CPU interrupt flags.
//
Interrupt_initModule();

//
// Initializes the NVIC vector table with pointers to the shell Interrupt
// Service Routines (ISR).
//
Interrupt_initVectorTable();
```

```
    //
    // Set the priority group to indicate the PREEMPT and SUB priority bits.
    //
    Interrupt_setPriorityGroup(INTERRUPT_PRIGROUP_PREEMPT_7_6_SUB_5_0);

    //
    // Set the global and group interrupt priority to allow CPU interrupts
    // with higher priority
    Interrupt_setPriority(INT_XINT1, 0, 0);

    //
    // Board Initialization
    //
    Board_init();

    GPIO_setInterruptType(GPIO_INT_XINT1, GPIO_INT_TYPE_FALLING_EDGE);
    GPIO_setInterruptPin(myGPIOInputInterrupt0, GPIO_INT_XINT1);
    GPIO_enableInterrupt(GPIO_INT_XINT1);

    Interrupt_register(INT_XINT1, &gpioInterruptHandler);
    Interrupt_enable(INT_XINT1);

    //
    // Enables CPU interrupts
    //
    Interrupt_enableMaster();
```

7.3. RPT 指令

在 Txx320F28004x 的指令集中，有一种名为 RPT（Repeat）的指令，常用来实现延时功能。例如：

```
asm(" RPT #5 || NOP ");
```

这条指令表示重复执行 5 次 NOP 操作，以实现精确的短时间延时。

在 G32R501 系列芯片中，使用的是 Arm® Cortex®-M52 内核。Arm® Cortex®-M52 内核并不直接支持类似于 RPT 的指令。因此，建议使用以下方法来实现类似的延时功能。

*使用 NOP 指令进行控制：*在 Arm® Cortex®-M52 内核中，可以通过编写自定义的延时函数来实现类似的延时效果。以下是一个名为 SysCtl_delay 的函数示例，通过汇编代码实现了循环延时：

```
void SysCtl_delay(uint32_t count)
{
```

```
    __asm volatile
    (
                                        "MOV R1, LR \n"
                                        "MOV LR, R0 \n"
          "delay_loop: \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "SUBS LR, LR, #1 \n"
                                        "BNE delay_loop \n"
           "delay_end: \n"
                                        "NOP \n"
                                        "NOP \n"
                                        "MOV LR, R1 \n"
    );
}
```

虽然 G32R501 系列芯片的 Arm® Cortex®-M52 内核不支持直接的 RPT 指令，但通过编写自定义的延时函数，如 SysCtl_delay 函数，仍然可以实现类似的延时效果。用户可以根据实际需求调整延时的周期数，以达到所需的延时长度。

7.4. 寄存器访问

G32R501 系列芯片的寄存器访问需要遵循特定规则，包括偏移地址和访问位宽，同时某些寄存器具有写保护机制。在进行写操作时，需要先解除写保护，操作完成后重新启用写保护。用户在开发过程中应仔细参考用户手册，以确保正确访问和操作寄存器。

G32R501 支持 8 位寻址，但在访问寄存器时需遵循以下规则：

偏移地址：

- G32R501 的寄存器偏移地址与 Txx320F28004x 存在差异，具体信息请参考用户手册。

访问规则：

- 如果寄存器位宽为 16 位（16bit），则在 G32R501 中必须按照 16 位（16bit）进行访问。
- 如果寄存器位宽为 32 位（32bit），则在 G32R501 中必须按照 32 位（32bit）进行访问。

写寄存器保护（WRPRT）：

- G32R501 的 WRPRT 保护机制是为了防止 CPU 对系统中的一些寄存器进行错误的写操作，该机制使用特殊的协处理器指令（“WRPRT”和“WRPRTDIS”）来启用和禁用对受保护寄存器的访问。

参考用户手册，得知 ADC 模块的 ADCCTL1 寄存器是具有写保护机制，该寄存器进行写操作

时，需要先解除写保护，操作完成后重新启用写保护。示例代码如下：

```
//
// Set the position of the pulse.
//
WRPRT_ENABLE;
HWREGH(ADCA_BASE + ADC_O_CTL1) =
(HWREGH(ADCA_BASE + ADC_O_CTL1) & ~ADC_CTL1_INTPULSEPOS) |
(uint16_t)ADC_PULSE_END_OF_ACQ_WIN;
WRPRT_DISABLE;
```

8. 仿真支持

在嵌入式开发中，仿真器的使用至关重要。不同芯片在仿真模式下的行为可能会有所不同，特别是在中断处理和断点设置方面。以下是对 Txx320F28004x 系列和 G32R501 系列 MCU 在仿真支持兼容性方面的详细讨论和建议。

8.1. 硬件支持

相较于 Txx320F28004x，G32R501 不仅支持 JTAG 接口，也支持 SWD 接口进行仿真。

G32R501 仿真器支持情况：

- Geehy-Link（WinUSB）、DAP Link（固件版本为 CMSIS-DAP V2 及以上）
- J-Link V12（J-Link V7.94g 及以上）
- Ulink Pro

8.2. 中断响应行为

在仿真调试过程中，中断处理的行为对软件开发和调试有着重要影响。Txx320F28004x 系列和 G32R501 系列在这方面存在显著差异。

Txx320F28004x 系列

行为：支持在 main 函数中设置断点时，中断依然可以正常响应。这种行为使得开发者在调试时能够利用中断来检测和处理事件，而不影响主程序的执行。

G32R501 系列

行为：在 main 函数中设置断点时，通常会停止所有的 CPU 活动，包括中断响应。这种行为导致在断点停留期间，中断无法触发和处理。

G32R501 系列对此差异的解决措施建议如下：

- 了解并接受其在断点状态下中断无法响应的局限性，调整调试策略。

8.3. 双核仿真

与 Txx320F28004x 不同，G32R501 系列 MCU 具有双核系统。在进行双核仿真时，需要特别注意两个 CPU 核心的调试和同步问题。

实现对 CPU1 的调试流程：

- 启动 CPU0：首先启动 CPU0 并执行 BootROM 代码。
- 跳转到应用代码：CPU0 在执行完 BootROM 后跳转到应用代码。
- 使能 CPU1 的时钟：在应用代码中，设置 BOOTCTRL1 寄存器的值为 0x2，以使能 CPU1

的时钟。

- 连接并调试 CPU1：完成以上步骤后，CPU1 即可正常连接调试。

8.3.1. 双核调试系统（DCDS）

G32R501 系列 MCU 所使用的仿真系统为双核调试系统。关于如何进行双核仿真，请参考文档《AN1128_G32R501_双核仿真指导手册》。

图 4 结构框图

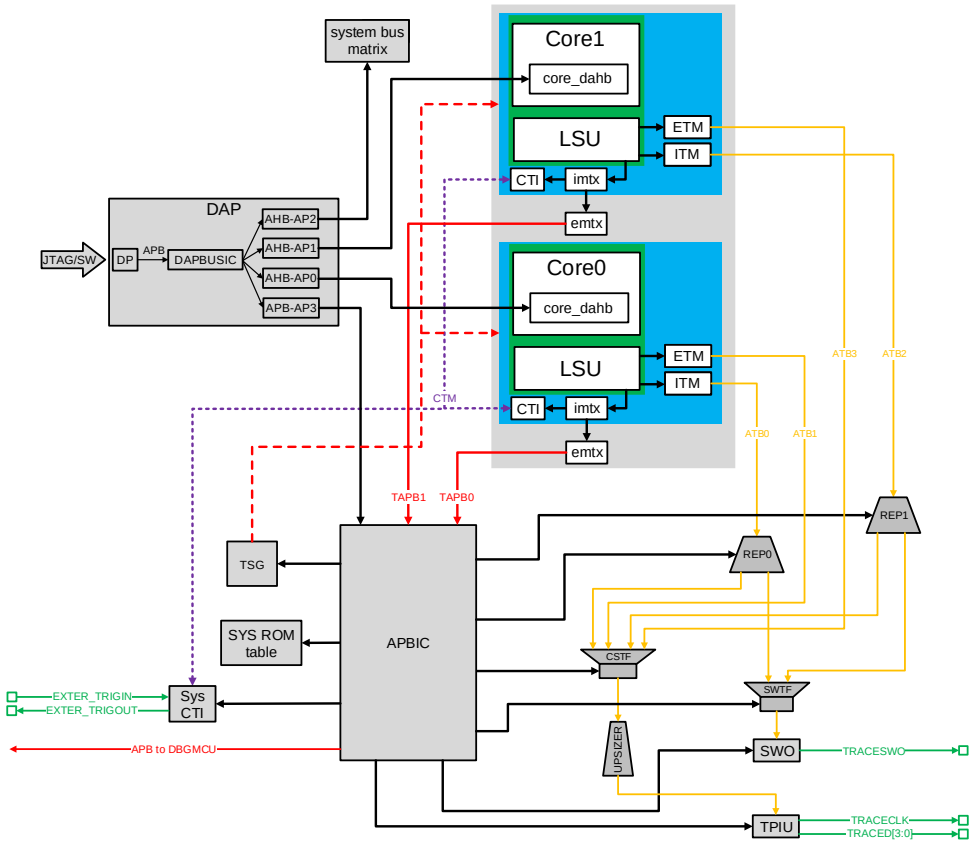
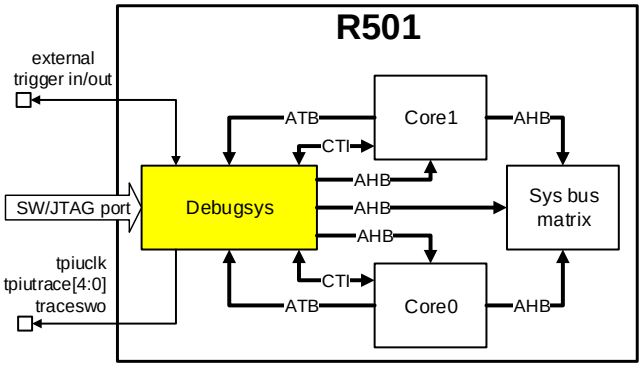


图 5 调试系统的典型集成



主要特征：

- 支持系统中每个 CPU 内核的独立断点调试
- 支持代码执行跟踪
- 支持 JTAG 调试端口
- 支持串行线调试端口
- 支持软件仪器化
- 支持交叉触发
- 支持触发器输入和触发器输出
- 支持访问系统总线矩阵
- 支持串行线（SWO）跟踪端口
- 支持 TPIU 跟踪端口

9. Keil MDK-ARM 开发工具链

在从 Txx320F28004x 系列微控制器迁移到 G32R501 系列微控制器的过程中，开发工具的迁移是一个重要环节。Code Composer Studio（CCStudio）集成开发环境（IDE）和 Keil MDK-ARM 都是嵌入式开发中的常用工具。由于这两种 IDE 在工程创建、文件格式、编译器选项、链接脚本以及调试环境上存在显著差异，因此在迁移过程中需要特别注意这些方面的兼容性问题。

关于该章节的内容请参考文档：《AN1126_G32R501 IDE 与工具链使用说明》。

10. 版本历史

表格 106 文件版本历史

日期	版本	变更历史
2025.01	1.0	新建
2025.04	1.1	修正部分描述

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“™”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，也不应被视为极海对第三方产品、服务或知识产权提供任何形式的保证，包括但不限于任何第三方知识产权的非侵权保证，除非极海在销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及/或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

极海产品并非设计、授权或担保适合用于军事、生命保障系统、污染控制或有害物质管理系统中的关键部件，亦非设计、授权或担保适合用于在产品失效或故障时可导致人员受伤、死亡、财产或环境损害的应用。

如果产品未标明“汽车级”，则表示不适用于汽车应用。如果用户对产品的应用超出极海提供的规格、应用领域、规范，极海不承担任何责任。

用户应该确保对产品的应用符合相应标准以及功能安全、信息安全、环境标准等要求。用户对极海产品的选择和使用负全部的责任。对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册及产品的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册及产品而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失），这涵盖了可能导致的人身安全、财产或环境损害等情况，对于这些损害极海概不承担责任。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。