应用笔记

Application Note

文档编号：**AN1127**

G32R501 SDK Examples 用户手册

版本：**V1.0**

1 引言

本使用说明提供如何使用 G32R5 系列中的 G32R5xx SDK 里的例程使用指导。

注意：本文未涉及的部分 SDK 例程在 G32R5xx SDK 的扩展包中，如您需要扩展包内容请联系极海官方进行获取。

目录

2 外设库例程

注意：这些例程位于 G32R5xx_SDK 的以下位置：

G32R5xx_SDK_VERSION：/driverlib/DEVICE_GPN/examples/CORE_IF_MULTICORE/

2.1 ADC 例程

2.1.1 ADC 软件触发

文件：adc_ex1_soc_software.c

此示例基于软件触发对 ADCA 和 ADCC 上的一些电压进行转换。由于 ADCC 不会在 ADCA 完成之前进行转换，所以 ADC 不会异步运行。但是，这样远不如允许 ADC 并行同步转换（例如，通过使用 PWM 触发）高效。

外部连接：

- A0、A1、C2 和 C3 应连接到要转换的信号

监控变量：

- myADC0Result0——引脚 A0 上电压的数字表示
- myADC0Result1——引脚 A1 上电压的数字表示
- myADC1Result0——引脚 C2 上电压的数字表示
- myADC1Result1——引脚 C3 上电压的数字表示

2.1.2 ADC PWM 触发

文件：adc_ex2_soc_pwm.c

此示例设置 PWM1 以定期触发 ADCA 的转换。

外部连接：

- A0 应连接到要转换的信号.

监控变量：

- myADC0Resulte——来自引脚 A0 的一系列模数转换样本。样本之间的时间间隔由 PWM 计时器的周期决定。

2.1.3 ADC 软件触发同步转换

文件：adc_ex4_soc_software_sync.c

此示例使用 Input XBAR 的 Input 5 作为软件强制触发，对 ADCA 和 ADCC 上的一些电压进行转

换。Input 5 是通过 GPIO0 翻转触发的，但任何空闲的 GPIO 都可以使用。这种方法将确保两个 ADC 同时开始转换。

外部连接：

- A2、A3、C2、C3 引脚应连接到要转换的信号。

监控变量：

- myADC0Result0——引脚 A2 上电压的数字表示
- myADC0Result1——引脚 A3 上电压的数字表示
- myADC1Result0——引脚 C2 上电压的数字表示
- myADC1Result1——引脚 C3 上电压的数字表示

2.1.4 ADC 连续触发

文件：adc_ex5_soc_continuous.c

此示例设置 ADC 连续转换，以实现最大采样率。

外部连接：

- A0 引脚应连接到要转换的信号。

监控变量：

- adcAResults——来自 A0 引脚的一系列模拟到数字转换样本。样本间的时间间隔是基于 ADC 速度所能达到的最小可能值。

2.1.5 ADC 连续转换通过 DMA 读数据

文件：adc_ex6_soc_continuous_dma.c

这个例子设置了两个 ADC 通道进行同时转换。结果将通过 DMA 传输到 RAM 中的一个缓冲区。

外部连接：

- A3 和 C3 引脚应连接到要转换的信号。

监控变量：

- myADC0DataBuffer：A3 引脚上电压的数字表示
- myADC1DataBuffer：C3 引脚上电压的数字表示

2.1.6 ADC PPB 偏移

文件：adc_ex7_ppb_offset.c

这个示例软件触发了 ADC。当 ADC 开始转换（SOC）时，通过后处理模块自动应用偏移调整。程序运行后，对应内存将包含 ADC 和后处理模块(PPB)的结果。

外部连接：

- A2 和 C2 引脚应连接到要转换的信号。

监控变量：

- myADC0Result：引脚 A2 上电压的数字表示
- myADC0PPBResult：引脚 A2 上电压的数字表示，减去自动添加的 100 个 LSB 偏移
- myADC1Result：引脚 C2 上电压的数字表示
- myADC1PPBResult：引脚 C2 上电压的数字表示，加上自动添加的 100 个 LSB 偏移

2.1.7 ADC PPB 限制

文件：adc_ex8_ppb_limits.c

这个示例设置了 PWM 以定期触发 ADC。如果结果超出定义的范围，后处理模块将产生中断。

默认限制是 1000 LSB 和 3000 LSB。当 VREFHI 设置为 3.3V 时，如果输入电压超过约 2.4V 或低于约 0.8V，PPB 将产生中断。

外部连接：

- A0 引脚应连接到要转换的信号。

监控变量：

- 无

2.1.8 ADC PPB 延迟捕获

文件：adc_ex9_ppb_delay.c

这个例子演示了使用后处理块进行延迟捕获。

设置了两个异步 ADC 触发器：

- PWM1，周期为 2048，触发 SOC0 在引脚 A0 上进行转换
- PWM2，周期为 9999，触发 SOC1 在引脚 A2 上进行转换

每次转换在转换结束时生成一个 ISR。在 SOC0 的 ISR 中，转换计数器增加，并检查 PPB 以确定样本是否被延迟。

程序运行后，内存将包含：

- conversion：SOC0 的转换延迟序列。

监控变量：

- delay：每个延迟转换对应的延迟时间。

2.1.9 ADC PWM 触发多个 SOC

文件：adc_ex10_multiple_soc_pwm.c

这个例子设置了 PWM1 以周期性地触发 ADCA 和 ADCC 上的一组转换。这个例子演示了多个 ADC 协同工作，利用多个 ADC 之间的并行性来处理一批转换。

外部连接：

- A0，A1，A2 和 C2，C3，C4 引脚应连接到要转换的信号。

监控变量：

- adcAResult0——引脚 A0 上电压的数字表示
- adcAResult1——引脚 A1 上电压的数字表示
- adcAResult2——引脚 A2 上电压的数字表示
- adcCResult0——引脚 C2 上电压的数字表示
- adcCResult1——引脚 C3 上电压的数字表示
- adcCResult2——引脚 C4 上电压的数字表示

2.1.10 ADC 突发模式

文件：adc_ex11_burst_mode_pwm.c

这个例子设置了 PWM1 以周期性地使用突发模式（burst mode）触发 ADCA。这允许每次突发时对不同的通道进行采样。

每次突发触发 3 次转换。A0 和 A1 是每次突发的一部分，而第三次转换在 A2、A3 和 A4 之间轮换。这允许对高优先级信号进行高速采样，而对低优先级信号可以以较低的速率进行采样。

使用 ADCA 中断 ISR 来读取 ADCA 的结果。

外部连接：

- A0，A1，A2，A3，A4 引脚应连接到要转换的信号。

监控变量：

- adcAResult0——引脚 A0 上电压的数字表示
- adcAResult1——引脚 A1 上电压的数字表示
- adcAResult2——引脚 A2 上电压的数字表示

- adcAResult3——引脚 A3 上电压的数字表示
- adcAResult4——引脚 A4 上电压的数字表示

2.1.11 ADC 突发模式过采样

文件：adc_ex12_burst_mode_oversampling.c

这个例子是一个使用软件实现的 ADC 过采样例子。ADC SOC 被配置在突发模式（burst mode）下，由 PWM 的 SOCA 事件触发。

外部连接：

- A2 引脚应连接到要转换的信号。

监控变量：

- lv_results——在引脚 A2 上测量的数字值数组（过采样量由 Oversampling_Amount 配置）

2.1.12 ADC SOC 过采样

文件：adc_ex13_soc_oversampling.c

这个例子设置了 PWM1 以周期性地触发 ADCA 上的一组转换，包括多个 SOC，这些 SOC 都对 A2 进行转换，以实现 A2 的过采样。

使用 ADCA 中断 ISR 来读取 ADCA 的结果。

外部连接：

- A0，A1，A2 引脚应连接到要转换的信号。

监控变量：

- adcAResult0——引脚 A0 上电压的数字表示
- adcAResult1——引脚 A1 上电压的数字表示
- adcAResult2——引脚 A2 上电压的数字表示

2.1.13 ADC PPB PWM 触发

文件：adc_ex14_ppb_pwm_trip.c

这个例子演示了通过 ADC 限幅检测 PPB 块触发 PWM。ADCAINT1 被配置为在初始软件强制触发后周期性地触发 ADCA 通道 2。配置了限幅检测后处理块（PPB），如果 ADC 结果超出了定义的范围，后处理块将生成 ADCxEVTy 事件。这个事件通过配置 PWM XBAR 和相应的 PWM 的跳闸区和数字比较子模块，被配置为 PWM 的触发源。这个例子展示了：

- 一次性触发

- 周期性触发
- 通过数字比较子模块直接触发 PWM

默认的限制是 0 LSB 和 3600 LSB。当 VREFHI 设置为 3.3V 时，如果输入电压超过约 2.9V，PPB 将生成一个触发事件。

外部连接：

- A2 应连接到一个要转换的信号
- 使用示波器观察以下信号：
  - PWM1 (GPIO0——GPIO1)
  - PWM2 (GPIO2——GPIO3)
  - PWM3 (GPIO4——GPIO5)

监控变量：

- adcA2Results——引脚 A2 上电压的数字表示

2.1.14 ADC 开路和短路检测

文件：adc_ex15_open_shorts_detection.c

这个例子演示了 ADC 开路和短路检测（ADCOSDETECT）的电路配置，用于检测系统中的引脚故障。该例子启用了开路和短路检测电路以及必要的 ADC 配置，并在开始正常 ADC 转换之前诊断 ADCAA0 输入引脚状态。

要启用 ADC OSDetect 电路：

1. 配置 ADC 进行转换（例如，通道、SOC、ACQPS、预分频器、触发器等）。OSDetect 功能仅在 12 位模式下可用。
2. 设置 ADCOSDETECT 寄存器以进行所需的分压连接。有关可用的 OSDetect 配置的详细信息，请参阅 G32R5xx 用户手册。
3. 启动转换并检查转换结果。注意：结果必须根据输入端的驱动情况和 Rs 与 Cp 的值进行解释。如果 Vs 信号可以与输入引脚断开，则该电路可用于检测开路和短路输入引脚。在示例中，配置了 ADCAA0 通道，并使用以下算法检查 A0 引脚状态：

- 步骤 1：配置全量程 OSDETECT 模式并捕获 ADC 结果（resultHi）
- 步骤 2：配置零量程 OSDETECT 模式并捕获 ADC 结果（resultLo）
- 步骤 3：禁用 OSDETECT 模式并捕获 ADC 结果（resultNormal）
- 步骤 4：确定 ADC 引脚的状态。
  - 如果引脚开路，resultLo 将等于 Vreflo，resultHi 将等于 Vrefhi。

- 如果引脚短路到 Vrefhi，resultLo 应约等于 Vrefhi，resultHi 应等于 Vrefhi。
- 如果引脚短路到 Vreflo，resultLo 应等于 Vreflo，resultHi 应约等于 Vreflo。
- 如果引脚连接到有效信号，resultLo 应大于 osdLoLimit 但小于 resultNormal，而 resultHi 应小于 osdHiLimit 但大于 resultNormal。

表格 1 引脚状态判断

输入	全量程输出	零量程输出	引脚状态
未知	VREFHI	VREFLO	开路
VREFHI	VREFHI	约等于 VREFHI	短路到 VREFHI
VREFLO	约等于 VREFLO	VREFLO	短路到 VREFLO
Vn	Vn < resultHi < VREFHI	VREFLO < resultLo < Vn	良好

- 步骤 5：osDetectStatusVal 大于 4 表示引脚没有故障。
  - 如果 osDetectStatusVal == 1，表示引脚 A0 开路。
  - 如果 osDetectStatusVal == 2，表示引脚 A0 短路到 VREFLO。
  - 如果 osDetectStatusVal == 4，表示引脚 A0 短路到 VREFHI。
  - 如果 osDetectStatusVal == 8，表示引脚 A0 状态良好/有效。
  - 任何 osDetectStatusVal > 4 的值表示引脚 A0 状态有效。

配置 ADC 进入 OSDETECT 模式时应注意以下几点：

1. 分压电阻的公差可能变化较大，因此不应使用此功能来检查转换精度。
2. 有关模拟输入通道的实现和可用性，请参阅芯片数据手册。
3. 由于高驱动阻抗，需要比 ADC 最小采样保持持续时间更长的 S+H 持续时间。

外部连接：

- A0 引脚应连接到一个要转换的信号

监控变量：

- osDetectStatusVal：引脚 A0 上电压的开路/短路检测状态
- adcAResult0：引脚 A0 上电压的数字表示

2.2 Boot 例程说明

2.2.1 使用 DCS OTP 配置的 Boot 错误状态 Pin 例程

文件：boot_ex1_errorstatuspin.c

本示例演示了如何配置启动模式、启动模式选择引脚和错误状态引脚。

注意：DCS OTP（一次性可编程）内存用于配置启动模式、启动模式选择引脚和错误状态引脚。一旦 DCS OTP 部分被编程，它们无法被擦除和再次编程。

本示例旨在展示如何配置启动控制以及错误状态引脚的功能。一旦错误状态引脚启用，软件将触发 NMI，错误状态引脚将变为高电平。然后，NMI ISR 将清除错误状态并将错误状态引脚驱动为低电平，并返回主循环，在主循环中 NMI 将再次被触发。更多详情请参见用户手册的“ROM 代码和外设启动”章节。

外部连接：

- 根据编程到 GPREG2 中的内容，监测使用的错误状态引脚。错误状态引脚可能是 GPIO 24、GPIO 28 或 GPIO 29。

监控变量：

- 无

2.2.2 自定义启动配置例程

文件：boot_ex2_customBootConfig.c

该示例根据所选的项目构建配置实现自定义启动配置：0 个启动模式选择引脚（ZERO_BMSPS），1 个启动模式选择引脚（ONE_BMSP），或者 3 个启动模式选择引脚（THREE_BMSPS）。

选择构建配置并运行程序。要测试特定的启动模式：暂停程序，必要时对 BMSP 应用适当的电压，然后执行复位。如果需要测试多个特定的启动模式，请在执行上述步骤前复位调试器并运行程序；这确保设备不会执行来自先前启动模式的“不可调试”代码，并且仿真等效的 Boot ROM 寄存器包含适当的值。不要通过断电来执行复位；如果设备断电，仿真等效的 Boot ROM 寄存器中的值可能会发生变化。

默认情况下，本示例将配置启动模式选择引脚和启动模式选项，以执行仿真启动过程。

以下常量可以在头文件中更改：

- STANDALONE_BOOT——确定设备复位时（仿真器连接时）是模拟独立启动还是仿真启动过程。警告：独立启动过程需要用户可配置的 DCS OTP（一次性可编程）寄存器被编程。在选择模拟独立启动过程之前，请确保 DCS OTP 寄存器已被编程。OTP 寄存器只能编程一次，且无法擦除。
- BOOTPIN_CONFIG_BMSP2——启动模式选择引脚 2，默认为 GPIO2
- BOOTPIN_CONFIG_BMSP1——启动模式选择引脚 1，默认为 GPIO1
- BOOTPIN_CONFIG_BMSP0——启动模式选择引脚 0，默认为 GPIO0

BOOTDEF 选项也可以根据需要更改。示例的默认选项符合下面描述的构建配置。

表格 2 BOOTDEF 选项配置

Options	Boot Mode Number	BMSP2	BMSP1	BMSP0	Boot Mode
ZERO_BMSPS	0	N/A	N/A	N/A	Flash Boot
ONE_BMSP	0	N/A	N/A	0	Flash Boot
	1	N/A	N/A	1	UART Boot
THREE_BMSPS	0	0	0	0	Flash Boot
	1	0	0	1	UART Boot
	2	0	1	0	Flash BootALT.1
	3	0	1	1	UART Boot ALT.1
	4	1	0	0	CAN Boot
	5	1	0	1	SPI Boot
	6	1	1	0	RAM Boot
	7	1	1	1	I2C Boot

在运行仿真启动配置时，在内存浏览器中监视以下地址：

- 0x50020000 和 0x50020003 用于 EMU BOOTPIN CONFIG
- 0x50020004 和 0x50020007 用于 EMU BOOTDEF LOW
- 0x50020008 和 0x5002000B 用于 EMU BOOTDEF HIGH

当 STANDALONE_BOOT 常量设置为非零值时，在内存浏览器中监视以下地址：

- 0x50020000 用于 EMU BOOTPIN CONFIG

外部连接：

- 根据需要将 3V 或 GND 连接到 BMSP2
- 根据需要将 3V 或 GND 连接到 BMSP1
- 根据需要将 3V 或 GND 连接到 BMSP0

监控变量：

- 无

2.2.3 CPU0 安全 Flash 启动

文件：boot_ex3_cpu0_secure_flash.c

本示例演示了如何使用 CPU0 的安全闪存启动模式。

安全闪存启动在设备启动时对闪存的入口扇区执行 CMAC 认证。如果认证通过，应用程序将开始执行。更多关于安全闪存启动模式的信息请参考用户手册。

本项目展示了如何使用 Geehy CMAC 工具生成基于用户 CMAC 密钥的 CMAC 标签并将该值嵌入到闪存应用程序中。此外，示例还详细说明了如何从用户应用程序调用 CMAC API 以计算应用程序入口闪存扇区之外的其他闪存扇区的 CMAC 值。

在未连接调试器的情况下确定通过/失败：

- LED2 和 LED3 关闭 = 安全启动失败
- LED2 或 LED3 闪烁 = 安全启动通过，全闪存 CMAC 失败
- LED2 和 LED3 闪烁 = 安全启动通过且全闪存 CMAC 通过。

外部连接：

- 无

监控变量：

- applicationCMACStatus：0x00：CPU0 的完整闪存 CMAC 认证通过，0xFFFFFFFF：CPU0 的完整闪存 CMAC 认证失败，其他值：其他错误。

2.3 CAN 例程说明

2.3.1 CAN 外部回环

文件：can_ex1_loopback.c

本示例展示了 CAN 的基本设置，以便在 CAN 总线上发送和接收消息。CAN 外设被配置为使用特定的 CAN ID 发送消息。消息以每秒一次的频率发送，使用一个简单的延迟循环来计时。发送的消息是一个包含递增模式的 2 字节消息。

本示例将 CAN 控制器设置为外部回环测试模式。发送的数据在 CANTXA 引脚上可见，并会内部接收回 CAN 内核。请参考 G32R5xx 用户手册中 CAN 章节关于外部回环测试模式的详细信息。

外部连接：

- 无。

监控变量：

- msgCount——成功接收消息的计数器
- txMsgData——存储发送数据的数组
- rxMsgData——存储接收数据的数组

2.3.2 CAN 外部回环使用中断

文件：can_ex2_loopback_interrupts.c

本示例展示了 CAN 的基本设置，以便在 CAN 总线上发送和接收消息。CAN 外设被配置为使用特定的 CAN ID 发送消息。消息以每秒一次的频率发送，使用一个简单的延迟循环来计时。发送的消息是一个包含递增模式的 4 字节消息。使用 CAN 中断处理程序来确认消息的传输并统计已发送消息的数量。

本示例将 CAN 控制器设置为外部回环测试模式。发送的数据在 CANTXA 引脚上可见，并会内部接收回 CAN 内核。请参考 G32R5xx 用户手册中 CAN 章节关于外部回环测试模式的详细信息。

外部连接：

- 无。

监控变量：

- txMsgCount——发送消息的计数器
- rxMsgCount——接收消息的计数器
- txMsgData——存储发送数据的数组
- rxMsgData——存储接收数据的数组
- errorFlag——表示发生错误的标志

2.3.3 CANA 到 CANB 的外部传输

文件：can_ex3_external_transmit.c

本示例初始化 CAN 模块 A 和 CAN 模块 B 以进行外部通信。CANA 模块被设置为传输递增数据给 CANB 模块，传输次数为"n"，其中"n"是 TXCOUNT 的值。CANB 模块被设置为在接收到数据时触发中断服务程序 (ISR)。如果传输的数据与接收的数据不匹配，将设置一个错误标志。

设备上的两个 CAN 模块需要通过 CAN 收发器连接在一起。所需硬件：

- 一块带有两个 CAN 收发器的 G32R5xx 开发板。

外部连接：

- Eval 的 CANA 位于 DEVICE_GPIO_PIN_CANTXA(CANTXA) 和 DEVICE_GPIO_PIN_CANRXA（CANRXA）
- Eval 的 CANB 位于 DEVICE_GPIO_PIN_CANTXB (CANTXB) 和 DEVICE_GPIO_PIN_CANRXB (CANRXB)

监控变量：

- TXCOUNT——调节以设置要发送的消息数量

- txMsgCount——发送消息的计数器
- rxMsgCount——接收消息的计数器
- txMsgData——存储发送数据的数组
- rxMsgData——存储接收数据的数组
- errorFlag——表示发生错误的标志

2.3.4 CAN 外部回环使用 DMA

文件：can_ex4_loopback_dma.c

本示例设置 CAN 模块，以便在 CAN 总线上发送和接收消息。CAN 模块被设置为内部发送一个 4 字节的消息。使用中断来断言 DMA 请求线，然后触发 DMA 将接收到的数据从 CAN 接口寄存器传输到接收缓冲区数组。一旦传输完成，将进行数据检查。

本示例将 CAN 控制器设置为外部回环测试模式。发送的数据在 CANTXA 引脚上可见，并会内部接收回 CAN 内核。请参考 G32R5xx 用户手册中 CAN 章节关于外部回环测试模式的详细信息。

外部连接：

- 无。

监控变量：

- txMsgCount——发送消息的计数器
- rxMsgCount——接收消息的计数器
- txMsgData——存储发送数据的数组
- rxMsgData——存储接收数据的数组

2.3.5 CAN 发送和接收配置

文件：can_ex5_transmit_receive.c

本示例展示了 CAN 的基本设置，以便使用特定消息 ID 在 CAN 总线上发送或接收消息。CAN 控制器根据定义的选择进行配置。当选择 TRANSMIT 定义时，CAN 控制器充当发送器，将数据发送到外部连接的第二个 CAN 控制器。如果未定义 TRANSMIT，则 CAN 控制器充当接收器，等待外部 CAN 控制器发送消息。

设备上的 CAN 模块需要通过 CAN 收发器进行连接。所需硬件：

- 一块带有 CAN 收发器的 G32R5xx 开发板。

外部连接：

- Eval 的 CANA 位于 DEVICE_GPIO_PIN_CANTXA（CANTXA）和

DEVICE_GPIO_PIN_CANRXA（CANRXA）

监控变量：

- MSGCOUNT——调节以设置消息数量
- txMsgCount——发送消息的计数器
- txMsgData——存储发送数据的数组
- errorFlag——表示发生错误的标志
- rxMsgCount——初始值为要接收的消息数量，并在每接收一条消息时递减

2.3.6 CAN 错误生成示例

文件：can_ex6_error_generation.c

本示例演示了处理 CAN 错误条件的方法。它生成 CAN 数据包并通过 GPIO 发送。数据包在外部回环并在 CAN 模块中接收。CAN 中断服务程序读取错误状态，并演示如何检测不同的错误条件。

更改 ERR_CFG 定义为不同的错误场景并运行示例。相应的错误标志将在 canaISR()例程的状态变量中设置。使用一个定时器（定时器 0）进行 CANBITRATE 微秒的周期性定时器中断。在定时器中断时，它发送具有指定错误条件的所需 CAN 帧类型。设备上的 CAN 模块需要通过 CAN 收发器进行连接。

外部连接：

- Eval 的 GPIOTX_PIN 应连接到 DEVICE_GPIO_PIN_CANRXA（CANRXA）

监控变量：

- status – canaISR 函数中的变量，用于检查错误状态。

2.3.7 CAN 远程请求回环

文件：can_ex7_loopback_tx_rx_remote_frame.c

本示例展示了 CAN 的基本设置，以便传输远程帧并获取远程帧的响应，并将其存储在接收对象中。CAN 外设被配置为使用特定的 CAN ID 传输远程请求帧和远程应答帧消息。消息对象 3 被配置为传输远程请求。消息对象 2 被配置为远程应答对象，具有过滤掩码，以便接受具有任何消息 ID 的远程帧，并以消息 ID 7 和数据长度 8 传输远程应答。消息对象 1 被配置为接收对象，具有消息 ID 7 的过滤消息，以便存储由消息对象 2 传输的远程应答数据。

本示例将 CAN 控制器设置为外部回环测试模式。发送的数据在 CANTXA 引脚上可见，并会内部接收回 CAN 内核。请参考 G32R5xx 用户手册中 CAN 章节关于外部回环测试模式的详细信息。

外部连接：

- 无。

监控变量：

- txMsgData——存储发送数据的数组
- rxMsgData——存储接收数据的数组

2.3.8 演示掩码寄存器使用的 CAN 示例

文件：can_ex8_mask.c

本示例初始化 CAN 模块 A 以用于接收。当接收到与过滤条件匹配的帧时，数据将被复制到邮箱 1，并且 LED 会闪烁几次，然后代码准备接收下一帧。如果接收到其他 MSGID 的消息，将提供一个 ACK。接收的完成由轮询 CAN_NDAT_21 寄存器来确定。不使用中断。所需硬件：

- 一个向 G32R5xx MCU 上的 CAN——A 发送消息的外部 CAN 节点。

外部连接：

- 无。

监控变量：

- rxMsgCount——接收消息的计数器
- rxMsgData——存储接收数据的数组

2.4 CAP 例程

2.4.1 CAP APWM 应用

文件：cap_ex1_apwm.c

这个程序在 APWM 模式下设置 CAP 模块。PWM 波形将在 GPIO5 上输出。使用影子寄存器加载下一个周期/比较值，PWM 的频率配置为在 5Hz 和 10Hz 之间变化。

2.4.2 CAP 捕获 PWM 应用

文件：cap_ex2_capture_pwm.c

这个例子配置了 PWM3A 为：

- 正计数模式
- 周期从 500 开始，增加到 8000
- 在 PRD 上切换输出

CAP1 被配置为捕获 PWM3A 输出的上升沿和下降沿之间的时间。

外部连接

- CAP1 在 GPIO16 上
- PWM3A 在 GPIO4 上
- 将 GPIO4 连接到 GPIO16。

监控变量

- cap1PassCount——成功捕获次数
- cap1IntCount——中断计数。

2.4.3 CAP APWM 移相位应用

文件：cap_ex3_apwm_phase_shift.c

该程序设置 CAP1 和 CAP2 模块为 APWM 模式，产生两个相位移的 PWM 输出，具有相同的占空比和频率值。频率、占空比和相位值可以通过更新定义的宏来进行选择。默认使用 10KHz 频率、50%占空比和 30%相位移值。CAP2 输出比 CAP1 输出领先 30%。GPIO5 和 GPIO6 被用作 CAP1/2 输出，可以使用分析仪/示波器进行探测以观察波形。

2.4.4 CAP 软件同步应用

文件：cap_ex4_sw_sync.c

这个例子配置了 PWM3A 如下：

- 向上计数模式
- 周期从 500 开始，增加到 8000
- 周期结束时切换输出

CAP1、CAP2 和 CAP3 被配置为捕获 PWM3A 输出的上升沿和下降沿之间的时间。

外部连接：

- CAP1、CAP2、CAP3 在 GPIO16
- PWM3A 在 GPIO4
- 连接 GPIO4 到 GPIO16。

观察变量：

- capPassCount——捕获次数
- cap3IntCount——中断计数。

2.5 COMP 例程

2.5.1 COMP 异步配置触发

文件：comp_ex1_asynch.c

这个示例启用了 COMP1 COMPH 比较器，并将异步的 CTRIPOUTH 信号馈送到 GPIO14/OUTPUTXBAR3 引脚，将 CTRIPH 馈送到 GPIO15/PWM8B。

COMP 被配置为生成触发 PWM 信号的触发信号。CMPIN1P 用于提供正输入，内部 DAC 被配置为提供负输入。内部 DAC 被配置为在 VDD/2 处提供信号。在 GPIO15 生成 PWM 信号，并配置为由 CTRIPOUTH 触发。

当向 CMPIN1P 提供低输入（VSS）时：

- 触发信号（GPIO14）输出低
- PWM8B（GPIO15）提供 PWM 信号

当向 CMPIN1P 提供高输入（高于 VDD/2）时，

- 触发信号（GPIO14）输出高
- PWM8B（GPIO15）被触发并输出高

外部连接

- 在 CMPIN1P 上提供输入（该引脚与 ADCINB6 共用）
- 可以使用示波器在 GPIO14 和 GPIO15 上观察输出

监视变量

- 无

2.5.2 COMP 数字滤波器配置

文件：comp_ex2_digital_filter.c

这个例子启用了 COMP1 COMPH 比较器，并通过数字滤波器将输出馈送到 GPIO14/OUTPUTXBAR3 引脚。

CMPIN1P 用于提供正输入，内部 DAC 被配置为提供负输入。内部 DAC 被配置为在 VDD/2 处提供信号。

当向 CMPIN1P 提供低输入（VSS）时，

- GPIO14 输出为低

当向 CMPIN1P 提供高输入（高于 VDD/2）时，

- GPIO14 输出变为高

外部连接

- 在 CMPIN1P 上提供输入（该引脚与 ADCINB6 共用）
- 输出可在 GPIO14 上观察到

监视变量

- 无

2.6 DAC 例程

2.6.1 缓冲 DAC 使能

文件：buffdac_ex1_enable.c

这个示例在缓冲的 DAC 输出 DACOUTA/ ADCINA0 上生成电压，并使用默认的 VDAC DAC 参考设置。

外部连接

- 当 DAC 参考设置为 VDAC 时，必须在 VDAC 引脚上应用外部参考电压。可以通过连接一根跳线线从 3.3V 连接到 ADCINB3 来实现这一点。

2.6.2 缓冲 DAC 随机

文件：buffdac_ex2_random.c

这个例子在缓冲 DAC 输出 DACOUTA/ADCINA0 上生成随机电压，并使用 VDAC 的默认 DAC 参考设置。

外部连接

- 当 DAC 参考设置为 VDAC 时，必须将外部参考电压应用到 VDAC 引脚。可以通过连接一根跳线线将 3.3V 连接到 ADCINB3 来实现这一点。

2.6.3 缓冲 DAC 正弦波（buffdac_sine）

文件：buffdac_ex2_random.c

这个例子在缓冲的 DAC 输出上生成正弦波，DACOUTA/ADCINA0（HSEC 引脚 9），并使用默认的 VDAC 参考设置。当 DAC 参考设置为 VDAC 时，必须向 VDAC 引脚施加外部参考电压。这可以通过连接一根跳线线从 3.3V 到 ADCINB3 来实现。

2.7 DCCOMP 例程

2.7.1 DCCOMP 单拍时钟验证

文件：dccomp_ex1_single_shot_verification.c

该程序使用 XTAL 时钟作为参考时钟来验证 PLLRAW 时钟的频率。双时钟比较器模块 0 用于时钟验证。clocksource0 是参考时钟（Fclk0 = 20Mhz），clocksource1 是需要验证的时钟（Fclk1 = 200Mhz）。Seed 是加载到计数器中的值。请参考用户手册获取有关要设置的计数器种子值的详细信息。

外部连接

- 无

监视变量

- 状态/结果——PLLRAW 时钟验证的状态

2.7.2 DCCOMP 单拍时钟测量

文件：dccomp_ex2_single_shot_measurement.c

这个程序演示了使用 XTAL 作为参考时钟对 INTOSC2 时钟进行单次测量。

双时钟比较器模块 0 用于时钟测量。clocksource0 是参考时钟（Fclk0 = 20Mhz），clocksource1 是需要测量的时钟（Fclk1 = 10Mhz）。由于需要测量时钟 1 的频率，初始种子被设置为计数器的最大值。

请参考用户手册获取有关应设置的计数器种子值的详细信息。

外部连接

- 无

监视变量

- result——INTOSC2 时钟测量是否成功完成的状态。
- meas_freq1——测量的时钟频率，这里是针对 INTOSC2。

2.7.3 DCCOMP 持续时钟监控

文件：dccomp_ex3_continuous_monitoring_of_clock.c

这个程序演示了在系统中使用 INTOSC2 作为参考时钟连续监控 PLL 时钟。这将在任何错误时触发中断，导致减少/重新加载计数器的停止。双时钟比较器模块 0 用于时钟监控。clocksource0 是参考时钟（Fclk0 = 10Mhz），clocksource1 是需要监控的时钟（Fclk1 = 250Mhz）。clock0 和 clock1 的 seed 被设置为实现 300 微秒的窗口。Seed 是加载到计数器中的值。为了演示，在

clock1 的 seed 值上稍微变化以在连续监控中生成错误。请参考用户手册以获取要设置的计数器 seed 值的详细信息。

注意：在 flash 配置中运行时，最好在加载示例后进行重置和重新启动，以消除任何过时的标志/状态。

外部连接

- 无

监视变量

- status/result——PLLRAW 时钟监控的状态
- cnt0——生成错误时的 Counter0 值
- cnt1——生成错误时的 Counter1 值
- valid——生成错误时的 Valid0 值

2.7.4 DCCOMP 时钟故障的检测

文件：dccomp_ex4_clock_fail_detect.c

这个程序演示了在系统中使用 XTAL 作为振荡时钟源对 PLL 时钟进行连续监测的时钟失效检测。一旦振荡时钟失效，就会触发 DCCOMP 错误中断，导致计数器的递减/重新加载停止。在这个示例中，通过关闭 XTAL 振荡器来模拟时钟失效。一旦 ISR 被处理，振荡源会被更改为 INTOSC1，PLL 会被关闭。

双时钟比较器模块 0 用于时钟监测。clocksource0 是参考时钟（Fclk0 = 20Mhz），clocksource1 是需要被监测的时钟（Fclk1 = 250Mhz）。Seed 是加载到计数器中的值。

在当前示例中，XTAL 预期是以晶体模式运行的谐振器，稍后将其关闭以模拟时钟失效。如果使用 SE 晶体，则需要在板上物理断开时钟连接。

有关要设置的计数器种子值的详细信息，请参考用户手册。

注意：在闪存配置中运行时，最好在加载示例后进行重置和重新启动，以消除任何陈旧的标志/状态。

外部连接

- 无

监视变量

- 状态/结果——时钟失效检测的状态

2.8 DMA 例程

2.8.1 DMA 传输应用 (dma_ex1_sram_transfer)

文件：dma_ex1_sram_transfer.c

这个例子使用一个 DMA 通道将数据从 RAMGS0 中的一个缓冲区传输到 RAMGS1 中的一个缓冲区。该示例重复设置 DMA 通道的 PERINTFRC 位，直到完成 16 个 burst（其中每个 burst 为 8 个 16 位字）的传输。当整个传输完成时，将触发 DMA 中断。

监视变量：

- sData——要发送的数据
- rData——接收到的数据

2.8.2 DMA 传输应用 (dma_ex2_sram_transfer)

文件：dma_ex2_sram_transfer.c

这个例子使用一个 DMA 通道从 RAMGS0 中的一个缓冲区传输数据到 RAMGS1 中的一个缓冲区。该示例重复设置 DMA 通道的 PERINTFRC 位，直到完成 16 个突发（其中每个突发是 8 个 16 位字）的传输。当整个传输完成时，将触发 DMA 中断。

监视变量：

- sData——要发送的数据
- rData——接收到的数据

2.9 Flash 例程说明

2.9.1 Flash 编程使用 AutoEcc

文件：flashapi_ex1_program_autoecc.c

本示例演示了如何使用 API 的 AutoEcc 生成选项进行 Flash 编程。

外部连接：

- 无。

监控变量：

- 无。

2.9.2 Flash 编程切换 bank 模式

文件：flashapi_ex6_bank_mode_switch.c

本示例演示了如何使用 Flash API 切换 Flash bank 模式。

外部连接：

- 无。

监控变量：

- 无。

2.10 GPIO 例程说明

2.10.1 设备 GPIO 设置

文件：gpio_ex1_setup.c

只是设置了 GPIO。在设置之后实际上并没有对引脚进行任何操作。

本例程将设备的 GPIO 为配置两种不同的状态。这段代码过于冗长，是为了说明如何设置 GPIO。在实际应用中，可以将代码行组合起来以提高代码大小和效率。此例程只是设置了 GPIO，设置完成后实际上并未对引脚做任何操作。

一般来说：

- 所有上拉电阻都已启用。对于 PWM 来说，可能不希望这样。
- 通信端口（CAN，SPI，UART，I2C）的输入质量为异步。
- Trip 引脚（TZ）的输入质量为异步。
- CAP 和 QEP 信号的输入质量与 SYSCLKOUT 同步。
- 一些 I/O 和中断的输入质量可能有一个采样窗口。

2.10.2 设备 GPIO 切换

文件：gpio_ex2_toggle.c

本例程演示了 GPIO 引脚在无限循环中切换。

外部连接：

- 无

监控变量：

- 无

2.10.3 设备 GPIO 中断

文件：gpio_ex3_interrupt.c

本例程配置一个 GPIO 输出引脚和一个 GPIO 输入引脚，然后将 GPIO 输入引脚配置为外部中断的来源，该中断会切换 GPIO 输出引脚。

外部连接：

- 无

监控变量：

- error：无

2.10.4 外部中断(EXTI)

文件：gpio_ex4_aio_external_interrupt.c

本例程中，AIO 引脚被配置为数字输入。另外两个 GPIO 信号（通过外部连接到 AIO 引脚）在软件中被切换以通过 AIO224 和 AIO225 触发外部中断（AIO224 分配给 EXTI1，AIO225 分配给 EXTI2）。用户需要外部连接这些信号以使程序正常工作。每个中断按顺序触发：先 EXTI1，然后 EXTI2。

外部连接：

- 将 GPIO30 连接到 AIO224。AIO224 将被分配给 EXTI1
- 将 GPIO31 连接到 AIO225。AIO225 将被分配给 EXTI2
- 可以在示波器上监视 GPIO34

监控变量：

- xint1Count：通过 EXTI1 中断的次数
- xint2Count：通过 EXTI2 中断的次数
- loopCount：通过空闲循环的次数

2.11 HRCAP 例程说明

2.11.1 HRCAP 捕获和校准示例

文件：hrcap_ex1_capture.c

本例程配置一个 CAP 来使用 HRCAP 功能来捕获输入 GPIO2 上的边缘之间的时间。

外部连接：

- 用户必须向 GPIO2 提供信号。XCLKOUT 已配置为输出 GPIO，可以外部跳线将 GPIO16 连接到 GPIO2。

监控变量：

- onTime1, onTime2
- offTime1, offTime2
- period1, period2

2.12 HRPWM 例程说明

2.12.1 SFO 的 HRPWM 占空比控制

文件：hrpwm_ex1_duty_sfo.c

此例程修改了 MEP 控制寄存器，以显示由于相应 PWM 模块的 HRPWM 控制扩展而在向上计数模式下具有高分辨率周期的边缘位移。

例程调用 MEP 比例因子优化器（SFO）软件库 V8 函数：

SFO v8：

- 当 HRPWM 正在使用时，动态更新 MEP_ScaleFactor。
- 使用 MEP_ScaleFactor 值更新 HRMSTEP 寄存器（仅存在于 Pwm1Regs 寄存器空间）。
- 如果出现错误：MEP_ScaleFactor 大于 255 的最大值，则返回 2（在此条件下，自动转换可能无法正常工作）。
- 对于指定的通道完成时返回 1。
- 对于指定的通道未完成时返回 0。

本例程旨在解释 HRPWM 的功能，代码可以优化以提高代码效率。

外部连接：

- 在示波器上监视 PWM1/2/3/4 A/B 引脚

2.12.2 HRPWM 滑块

文件：hrpwm_ex2_slider.c

本例程修改了 MEP 控制寄存器，以显示由 HRPWM 引起的边缘位移。由于 HRPWM 逻辑，相应 PWM 模块通道 A 和 B 的控制块将具有精细的边缘移动。

外部连接：

- 在示波器上监视 PWM1/2/3/4 A/B 引脚

2.12.3 HRPWM 滑块周期控制

文件：hrpwm_ex3_prdupdown_sfo.c

本例程修改了 MEP 控制寄存器，以显示高分辨率周期中 PWM 的边缘位移，因为各自的 PWM 模块的 HRPWM 控制扩展而处于上下计数模式。

这个示例调用了 MEP 比例因子优化器（SFO）软件库 V8 函数：

SFO v8：

- 当 HRPWM 被使用时动态更新 MEP_ScaleFactor
- 使用 MEP_ScaleFactor 值更新 HRMSTEP 寄存器（仅存在于 Pwm1Regs 寄存器空间中）
- 如果出现错误，返回 2：MEP_ScaleFactor 大于 255 的最大值（在这种情况下自动转换可能无法正常运作）
- 对指定的通道完成时返回 1
- 对指定的通道未完成时返回 0

此例程旨在解释 HRPWM 的功能。代码可以进行优化以提高代码效率。

外部连接：

- 在示波器上监视 PWM1/2/3/4 A/B 引脚

2.12.4 HRPWM 占空比控制与 UPDOWN 模式

文件：hrpwm_ex4_duty_updown_sfo.c

本例程中，AIO 引脚被配置为数字输入。另外两个 GPIO 信号（通过外部连接到 AIO 引脚）在软件中被切换以通过 AIO224 和 AIO225 触发外部中断（AIO224 分配给 EXTI1，AIO225 分配给 EXTI2）。用户需要外部连接这些信号以使程序正常工作。每个中断按顺序触发：先 EXTI1，然后 EXTI2。

这个例子调用 MEP 比例因子优化器（SFO）软件库 V8 的函数：

SFO v8：

- 当 HRPWM 正在使用时动态更新 MEP_ScaleFactor
- 使用 MEP_ScaleFactor 值更新 HRMSTEP 寄存器（仅存在于 Pwm1Regs 寄存器空间中）
- 如果出错，返回 2：MEP_ScaleFactor 大于 255 的最大值（在此条件下，自动转换可能无法正常工作）
- 对于指定通道完成时返回 1
- 如果对于指定通道未完成，返回 0

此示例旨在解释 HRPWM 的功能。代码可以优化以提高代码效率。

外部连接：

- 在示波器上监视 PWM1/2/3/4 A/B 引脚

2.13 I2C 例程说明

2.13.1 I2C 数字环回与 FIFO 中断

文件：i2c_ex1_loopback.c

本例程使用 I2C 模块的内部环回测试模式。同时使用 TX 和 RX I2C FIFO 和它们的中断。

发送一系列数据，然后将其与接收到的数据进行比较。

发送的数据类似于：

0000 0001

0001 0002

0002 0003

...

00FE 00FF

00FF 0000

依此类推，这种模式将永远重复。

外部连接：

- 无

监控变量：

- sData：将发送的数据
- rData：接收到的数据
- rDataPoint：用于跟踪接收流中最后位置，以进行错误检查

2.13.2 I2C EEPROM

文件：i2c_ex2_eeprom.c

此例程将会向 EEPROM 写入 1-14 个字，并读取它们。所写入的数据和 EEPROM 地址保存在消息结构体 i2cMsgOut 中。读取回来的数据将保存在消息结构 i2cMsgIn 中。

外部连接：

对于 Eval 上的 I2C <-> EEPROM 通信：

- 在地址 0x50 处连接外部 I2C EEPROM
- 将 GPIO35（SDAA）连接到外部 EEPROM 的 SDA（串行数据）引脚
- 将 GPIO37（SCLA）连接到外部 EEPROM 的 SCL（串行时钟）引脚
- 如果 EEPROM 和 r501 设备在不同的板上，请连接 GND 引脚

监控变量：

- i2cMsgOut：将写入 EEPROM 的数据的消息
- i2cMsgIn：从 EEPROM 中读取的数据的消息

2.13.3 I2C EEPROM POLLING

文件：i2c_ex2_eeprom.c

本程序将展示如何使用 I2C 轮询方法执行不同的 EEPROM 写入和读取命令。

外部连接：

对于 Eval 上的 I2C <->EEPROM 通信：

- 在地址 0x50 处连接外部 I2C EEPROM
- 将 GPIO35（SDAA）连接到外部 EEPROM 的 SDA（串行数据）引脚
- 将 GPIO37（SCLA）连接到外部 EEPROM 的 SCL（串行时钟）引脚
- 如果 EEPROM 和 r501 设备在不同的板上，请连接 GND 引脚

监控变量：

- i2cMsgOut：将写入 EEPROM 的数据的消息
- i2cMsgIn：从 EEPROM 中读取的数据的消息

2.14 中断例程说明

2.14.1 外部中断边沿触发

文件：interrupt_ex1_external.c

此例程将 GPIO0 设置为 EXTI1（EXTI_LINE_4），并将 GPIO1 设置为 EXTI2（EXTI_LINE_5）。另外两个 GPIO 信号用于触发中断（GPIO10 触发 EXTI1，GPIO11 触发 EXTI2）。用户需要外部连接这些信号才能使程序正常工作。

EXTI1 输入与 SYSCLKOUT 同步。

EXTI2 具有长的限定时间——每个为 510*SYSCLKOUT 的 6 个样本。

GPIO16 在中断之外会变高，在中断中会变低。可以在示波器上监视此信号。

每个中断按顺序触发——先 EXTI1，然后再 EXTI2。

外部连接：

- 将 GPIO10 连接到 GPIO0。（GPIO0 将被分配给 EXTI1）
- 将 GPIO11 连接到 GPIO1。（GPIO1 将被分配给 EXTI2）

监控变量：

- xint1Count：EXTI1 中断触发的次数
- xint2Count：EXTI2 中断触发的次数
- loopCount：空闲循环执行的次数

2.14.2 多中断处理

文件：interrupt_ex2_with_i2c_uart_spi_loopback.c

此例程用于演示如何在一个例程中处理多个通信外设中断，如 I2C，UART 和 SPI 数字环回时处理多个中断。数据传输将使用 FIFO 中断完成。

它使用这些模块的内部环回测试模式。同时使用 TX 和 RX FIFO 及其中断。除了引导模式引脚配置外，不需要其他硬件配置。

发送一系列数据，然后与接收到的数据进行比较。

对于 I2C 和 UART，发送的数据如下所示：

0000 0001

0001 0002

0002 0003

...

00FE 00FF

00FF 0000

...

对于 SPI，发送的数据如下所示：

0000 0001

0001 0002

0002 0003

...

FFFE FFFF

FFFF 0000

...

这种模式将永远重复。

外部连接：

- 无

监控变量：

- sDatai2cA：通过 I2C 发送的数据
- rDatai2cA：通过 I2C 接收的数据
- rDataPoint：用于跟踪接收 I2C 数据流中的最后位置，以进行错误检查
- sDataspiA：通过 SPI 发送的数据
- rDataspiA：通过 SPI 接收的数据
- rDataPointspiA：用于跟踪接收 SPI 数据流中的最后位置，以进行错误检查
- sDatauartA：串口将发送的数据
- rDatauartA：串口接收到的数据
- rDataPointA：跟踪在串口数据流中的位置。这用于检查传输的数据

2.14.3 中断优先级设置（软件）

文件：interrupt_ex3_sw_prioritization.c

本例程演示了通过 CPU 定时器中断对中断进行软件优先级排序。通过启用中断嵌套，可以实现对中断的软件优先级排序。

在这个设备中，CPU 定时器 0、1 和 2 的硬件优先级设置为定时器 0 具有最高优先级，定时器 2 具有最低优先级。这个例子在软件中配置了 CPU 定时器 0、1 和 2 的优先级，其中定时器 2 的优先级在软件中最高，定时器 0 的优先级最低，并输出执行顺序。

对于大多数应用程序，硬件对中断的优先级排序已经足够。对于需要自定义优先级排序的应用程序，这个例子说明了如何通过软件实现这一点。用户特定的优先级可以在 sw_prioritized_isr_level.h 头文件中进行配置。

为了启用中断嵌套，在 ISRs 中需要按照以下顺序进行操作：

- 步骤 1：设置全局优先级：修改 IER 寄存器，允许 CPU 中断按照更高用户优先级进行服务。

注意：此时 IER 已经保存在堆栈中。

- 步骤 2：设置组优先级（可选）：修改具有更高用户设定优先级的组中断以进行服务
- 步骤 3：启用中断
- 步骤 4：运行 ISR 的主要部分。
- 步骤 5：禁用中断
- 步骤 6：从 ISR 返回

外部连接：

- 无

监控变量：

- traceISR：ISR 的执行顺序

2.14.4 PWM 实时中断

文件：interrupt_ex4_pwm_realtime_interrupt.c

这个例子配置了 PWM1 定时器，并在每次 ISR 执行时增加一个计数器。PWM 中断可以配置为时间关键，以演示实时模式功能和实时中断能力。该示例使用 2 个 LED0——LED1 在主循环中切换，LED2 在 PWM 定时器中断中切换。必须设置 FREE_SOFT 位和 DBGIER.INT3 位才能在 halt 命令后使 PWM1 中断启用时间关键并在实时模式下运行。

运行此示例的关键点：

- 添加如下所述的观察变量，并启用连续刷新。
- 启用实时模式（Run->Advanced->启用实时模式）。
- 初始时，DBGIER 寄存器设置为 0，PWM 仿真模式设置为 PWM_EMULATION_STOP_AFTER_NEXT_TB（FREE_SOFT = 0）。
- 当应用程序运行时，您会发现两个 LED 在切换，观察变量 Pwm1TimerIntCount 和 Pwm1Regs.TBCTR 得到更新。
- 当应用程序暂停时，两个 LED 停止切换，观察变量保持不变。在调试器暂停时，PWM 计数器停止。
- 若要在调试器暂停期间启用 PWM 计数器运行，请将仿真模式设置为 PWM_EMULATION_FREE_RUN（FREE_SOFT = 2）。您会发现 Pwm1Regs.TBCTR 在运行，但 Pwm1TimerIntCount 保持不变。这意味着 PWM 计数器在运行，但中断服务程序未得到执行。

- 若要启用实时中断，请设置 DBGIER.INT3 = 1（PWM1 中断是 NVIC Group 3 的一部分）。您会发现 Pwm1TimerIntCount 在递增，LED 开始切换。即使在调试器暂停时，PWM 中断服务程序也会得到执行。

外部连接：

- 无

监控变量：

- Pwm1TimerIntCount：PWM1 中断服务程序计数器
- Pwm1Regs.TBCTR.TBCTR：PWM1 时间基准计数器
- Pwm1Regs.TBCTL.FREE_SOFT：将其设置为 2 以启用自由运行
- DBGIER.INT3：设置为 1 以启用实时中断

2.14.5 中断优先级设置（硬件）

文件：interrupt_ex5_hw_prioritization.c

本例程演示了通过 CPU 定时器中断实现硬件中断优先级。

程序配置了 CPU 定时器 0、1 和 2 的优先级，其中定时器 2 的优先级最高，定时器 0 的优先级最低，并打印执行顺序的跟踪。

对于大多数应用程序，中断的默认优先级设置足够了。对于需要自定义优先级设置的应用程序，此程序演示了如何实现。

外部连接：

- 无

监控变量：

- traceISR：ISR 的执行顺序

2.15 IPC 例程说明

2.15.1 IPC 邮箱使用中断模式

文件：ipc_ex1_mailbox_interrupt_cpu0.c

本示例展示了如何使用处理器间通信（IPC）模块的邮箱，以中断模式在两个 CPU 间互相收发数据。

在这个示例中：

1. CPU0 通过 IPC 模块以中断模式发送消息给 CPU1。

2. CPU1 以中断模式发送消息回 CPU0。

3. CPU0 以中断模式接收来自 CPU1 发送的消息。

注意：IPC 例程包含 CPU0 和 CPU1 工程，编译 CPU0 工程之前，请先编译 CPU1 工程。

运行该应用程序，需要使用串口终端打开 COM 端口，设置如下：

- 查找正确的 COM 端口
- 每秒比特数 = 115200
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

外部连接：

- GPIO28 是 UART——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

监控变量：

- 无。

2.15.2 IPC 邮箱使用轮询模式

文件：ipc_ex2_mailbox_polling_cpu0.c

本示例展示了如何使用处理器间通信（IPC）模块的邮箱，以轮询模式在两个 CPU 间互相收发数据。

在这个示例中：

1. CPU0 通过 IPC 模块以轮询模式发送消息给 CPU1。

2. CPU1 以轮询模式发送消息回 CPU0。

3. CPU0 以轮询模式接收来自 CPU1 发送的消息，以此类推。

注意：IPC 例程包含 CPU0 和 CPU1 工程，编译 CPU0 工程之前，请先编译 CPU1 工程。

运行该应用程序，需要使用串口终端打开 COM 端口，设置如下：

- 查找正确的 COM 端口
- 每秒比特数 = 115200
- 数据位 = 8

- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

外部连接：

- GPIO28 是 UART——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

监控变量：

- 无。

2.15.3 IPC 通用中断

文件：ipc_ex3_general_interrupt_cpu0.c

本示例展示了如何使用处理器间通信（IPC）模块的通用中断功能。CPU0 和 CPU1 间互相触发各自的通用中断。

注意：IPC 例程包含 CPU0 和 CPU1 工程，编译 CPU0 工程之前，请先编译 CPU1 工程。

运行该应用程序，需要使用串口终端打开 COM 端口，设置如下：

- 查找正确的 COM 端口
- 每秒比特数 = 115200
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

外部连接：

- GPIO28 是 UART——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

监控变量：

- 无。

2.15.4 IPC 资源共享

文件：ipc_ex4_resource_share_cpu0.c

本示例展示了如何使用处理器间通信（IPC）模块的通用中断和邮箱，CPU0 和 CPU1 互斥可以

访问同一块内存空间和同一个外设（UART_A）。

注意：IPC 例程包含 CPU0 和 CPU1 工程，编译 CPU0 工程之前，请先编译 CPU1 工程。

运行该应用程序，需要使用串口终端打开 COM 端口，设置如下：

- 查找正确的 COM 端口
- 每秒比特数 = 115200
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

外部连接：

- GPIO28 是 UART——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

监控变量：

- 无。

2.15.5 IPC 事件控制

文件：ipc_ex5_event_control_cpu0.c

本示例展示了如何使用处理器间通信（IPC）模块的事件控制功能。CPU0 通过 TASK 寄存器发送任务给 CPU1，CPU1 根据不同的任务 ID 执行不同的任务。

在这个示例中：

1. CPU0 通过 IPC_issueTask 函数发送任务 ID 为 0x55AA0001 的任务给 CPU1，然后等待 CPU1 执行完该任务。
2. CPU1 通过 IPC_fetchTask 函数获取到任务 ID 为 0x55AA0001 任务，然后执行该任务，在该例程中是 LED2 以 1 秒闪灯 10 次。
3. CPU0 继续发送任务 ID 为 0x55AA0002 的任务，然后等待 CPU1 执行完该任务。
4. CPU1 获取 0x55AA0002 任务，并执行。在该任务中，CPU1 通过__WFE 指令进入低功耗模式。
5. CPU0 判断 CPU1 是否已经进入低功耗模式，并且 CPU0 通过 IPC_issueTask 函数发布任务，把 CPU1 从低功耗模式中唤醒。
6. CPU1 被 CPU0 从低功耗模式中唤醒，并继续执行完成任务。

注意：IPC 例程包含 CPU0 和 CPU1 工程，编译 CPU0 工程之前，请先编译 CPU1 工程。

运行该应用程序，需要使用串口终端打开 COM 端口，设置如下：

- 查找正确的 COM 端口
- 每秒比特数 = 115200
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

外部连接：

- GPIO28 是 UART——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

监控变量：

- 无。

2.16 LED 例程说明

2.16.1 LED 闪烁

文件：led_ex1_blinky.c

此例程演示了如何让一个 LED 闪烁。

外部连接：

- 无

监控变量：

- 无

2.17 LIN 例程说明

2.17.1 带中断的 LIN 内部环回

文件：lin_ex1_loopback_interrupts.c

本例程配置 LIN 模块为主模式，进行带有中断的内部环回。该模块被设置为执行 8 次数据传输，使用不同的传输 ID 和变化的传输数据。在接收到 ID 头后，将在线路 0 上触发一个中断，并调用中断服务例程（ISR），检查接收到的数据是否准确。

此例程可以通过取消注释"LIN_setInterruptLevel1()"来调整为使用中断线 1。

外部连接：

- 无

监控变量：

- txData：被发送数据的数组
- rxData：收到的数据数组
- result：例程完成状态（PASS = 0xABCD, FAIL = 0xFFFF）
- level0Count：中断线 0 的数量
- level1Count：中断线 1 的数量。

2.17.2 LIN UART 模式下的内部环回与中断

文件：lin_ex2_uart_loopback.c

此例程配置 LIN 模块为 UART 模式，使用中断进行内部环回。LIN 模块以非多缓冲模式下特定字符和帧长度的 UART 功能。该模块被设置为连续发送一个字符，等待接收该字符，并重复。

外部连接：

- 无

监控变量：

- 无

2.17.3 LIN UART 模式内部环回与 DMA

文件：lin_ex3_uart_dma.c

这个示例配置了 LIN 模块使用 DMA 以 UART 模式进行内部环回。LIN 模块在多缓冲区模式下以设置的字符和帧长度作为 UART 运行。当 LINTD0 和 LINTD1 寄存器中的传输缓冲区有足够的空间时，DMA 将从全局变量 sData 传输数据到这些传输寄存器中。一旦 LINRD0 和 LINRD1 寄存器中的接收缓冲区包含数据，DMA 将数据传输到全局变量 rdata 中。

当所有数据都被放入 rData 中时，将在一个 DMA 通道的 ISR 中执行对数据有效性的检查。

外部连接：

- 无

监控变量：

- sData：待发送的数据
- rData：接收到的数据

2.17.4 无中断 LIN 内部环回（轮询模式）

文件：lin_ex4_loopback_polling.c

本例程配置了 LIN 模块为主模式，进行内部环回测试，没有使用中断。该模块被设置为执行 8 次数据传输，传输 ID 和变化的传输数据。等待接收 ID 头部。然后检查接收到的数据是否准确。

外部连接：

- 无

监控变量：

- 无

2.18 低功耗模式例程说明

2.18.1 空闲模式的进入和退出

文件：lpm_ex1_idlewake_gpio.c

本例程将设备置于空闲模式，然后通过在 GPIO0 的下降沿触发 EXTI1 来唤醒设备。

GPIO0 引脚必须由外部代理从高电平拉低以唤醒。 GPIO0 被配置为 EXTI1 引脚，以在检测到下降沿时触发 EXTI1 中断。

最初，通过外部将 GPIO0 拉高。要通过触发 EXTI1 中断从空闲模式唤醒设备，请将 GPIO0 拉低（下降沿）。 当 GPIO0 保持低电平的时间达到设备数据表中指示的时间时，唤醒过程开始。

在进入空闲模式之前，将 GPIO1 拉高，并在外部中断 ISR 中将其拉低。

外部连接：

- 将 GPIO0 拉低以唤醒设备
- 设备唤醒时，GPIO1 将为低电平，LED1 将开始闪烁

2.18.2 空闲模式的进入和退出

文件：lpm_ex2_idlewake_watchdog.c

此例程将设备置于空闲模式，然后使用看门狗定时器唤醒设备。

当看门狗定时器溢出时，设备将从空闲模式唤醒，触发一个中断。

为了改变计数器溢出时间，为看门狗定时器设置了一个预分频器。

在进入空闲模式之前，GPIO1 被拉高，当在唤醒中断服务程序中时，GPIO1 被拉低。

外部连接：

- 设备唤醒时，GPIO1 将为低电平，LED1 将开始闪烁

2.18.3 停止模式的进入和退出

文件：lpm_ex5_haltwake_gpio.c

这个示例将设备置于 HALT 模式。如果需要在 HALT 模式下实现最低可能的电流消耗，则在设备处于 HALT 模式时必须将 JTAG 连接器从设备板上拔下。

对于需要在 HALT 模式下实现最小功耗的应用程序，应用软件在进入 HALT 模式之前应该关掉晶振（XTAL），通过设置 XTALCR.OSCOFF 位或使用 driverlib 函数 SysCtl_turnOffOsc(SYSCTL_OSCSRC_XTAL);。如果 OSCCLK 源被配置为 XTAL，则应用程序在设置 XTALCR.OSCOFF 之前应先将 OSSCLK 源切换到 INTOSC1 或 INTOSC2。

这个示例将设备置于 HALT 模式，然后通过一个 LPM 唤醒引脚将设备从 HALT 模式唤醒。

引脚 GPIO0 被配置为 LPM 唤醒引脚，以在检测到低脉冲时触发 WAKEINT 中断。GPIO0 引脚必须由外部代理从高电平拉低以唤醒。

在进入 STANDBY 模式之前，GPIO1 被拉高，并且在唤醒 ISR 时被拉低。

外部连接：

- 设备唤醒时，GPIO1 将为低电平，LED1 将开始闪烁

2.18.4 停止模式的进入和退出

文件：lpm_ex6_haltwake_gpio_watchdog.c

本例程将设备置于 HALT 模式。如果在 HALT 模式下需要最低可能的电流消耗，则必须在设备处于 HALT 模式时从设备板上移除 JTAG 连接器。

对于需要在 HALT 模式下最小功耗的应用程序，应用软件在进入 HALT 模式之前应该通过设置 XTALCR.OSCOFF 位或使用 driverlib 函数 SysCtl_turnOffOsc(SYSCTL_OSCSRC_XTAL);来关闭 XTAL。如果 OSCCLK 源配置为 XTAL，则应用程序在设置 XTALCR.OSCOFF 之前应该先将 OSSCLK 源切换到 INTOSC1 或 INTOSC2。

此例程将设备置于 HALT 模式，然后使用 LPM 唤醒引脚来唤醒设备。

引脚 GPIO0 被配置为 LPM 唤醒引脚，以在检测到低脉冲时触发 WAKEINT 中断。GPIO0 引脚必须由外部代理从高电平拉低以唤醒。

在此例程中，看门狗定时器被时钟化，并配置为作为超时机制产生看门狗复位。

在进入 STANDBY 模式之前，GPIO1 被拉高，并在唤醒 ISR 时拉低。

外部连接：

- 设备唤醒后，GPIO1 将变为低电平，LED1 将开始闪烁

2.19 PWM 例程

2.19.1 PWM 触发应用

文件：pwm_ex1_trip_zone.c

这个例子配置 PWM1 和 PWM2 如下：

- PWM1 的一次触发源为 TZ1
- PWM2 的逐周期触发源为 TZ1

最初将 TZ1 连接到高电平。在测试过程中，通过示波器监视 PWM1 或 PWM2 的输出。将 TZ1 拉低以查看效果。

外部连接

- PWM1A 在 GPIO0 上
- PWM2A 在 GPIO2 上
- TZ1 在 GPIO12 上

这个例子还利用了输入 XBAR。GPIO12（外部触发器）被路由到输入 XBAR，然后被路由到 TZ1。

TZ 事件被定义为 PWM1A 将进行一次性触发，PWM2A 将进行逐周期触发。

2.19.2 PWM 上下计数

文件：pwm_ex2_updown.c

这个例子配置了 PWM1、PWM2 和 PWM3，以产生具有独立调制的 PWMxA 和 PWMxB 的波形。在 PWM 的 ISR 中修改了比较值 CMPA 和 CMPB。在此示例中，TB 计数器处于向上/向下计数模式。在示波器上查看 PWM1A/B（GPIO0 和 GPIO1）、PWM2A/B（GPIO2 和 GPIO3）和 PWM3A/B（GPIO4 和 GPIO5）的波形。

2.19.3 PWM 同步应用

文件：pwm_ex3_synchronization.c

这个例子配置了 PWM1、PWM2、PWM3 和 PWM4 如下：

- PWM1 作为同步源没有相位移动
- PWM2 相位移动 300 TBCLKs
- PWM3 相位移动 600 TBCLKs
- PWM4 相位移动 900 TBCLKs

外部连接：

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO2 PWM2A
- GPIO3 PWM2B
- GPIO4 PWM3A
- GPIO5 PWM3B
- GPIO6 PWM4A
- GPIO7 PWM4B

监视变量：

- 无。

2.19.4 PWM 数字比较

文件：pwm_ex4_digital_compare.c

这个例子配置 PWM1 如下：

- 使用 DCAEVT1 使 PWM 输出低电平
- GPIO25 用作输入到输入交换输入 1
- 输入 1（来自输入交换）用作 DCAEVT1 的源
- 为了测试触发，启用 GPIO25 的上拉电阻，将此引脚拉到地

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO25 TZ1，拉低此引脚以触发 PWM

监视变量

- 无。

2.19.5 PWM 数字比较事件滤波屏蔽窗口

文件：pwm_ex5_digital_compare_event_filter.c

这个例子配置 PWM1 如下：

- PWM1 的 DCAEVT1 强制 PWM 输出为低电平
- GPIO25 被用作输入到 INPUT XBAR INPUT1
- INPUT1（来自 INPUT XBAR）被用作 DCAEVT1 的源
- 为了测试 trip，启用 GPIO25 的上拉电阻，将此引脚拉到 GND
- PWM1 的 DCBEVT1 强制 PWM 输出为低电平
- GPIO25 被用作输入到 INPUT XBAR INPUT1
- INPUT1（来自 INPUT XBAR）被用作 DCAEVT1 的源
- 为了测试 trip，启用 GPIO25 的上拉电阻，将此引脚拉到 GND
- DCBEVT1 使用 DCBEVT1 的滤波版本
- DCFILT 信号使用空白窗口来忽略 DCBEVT1 的持续时间的 DC 空白窗口

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO25 TRIPIN1，拉低此引脚以触发 PWM

监视变量

- 无。

2.19.6 PWM 谷值开关

文件：pwm_ex6_valley_switching.c

这个示例配置了 PWM1 如下：

- PWM1 具有 DCAEVT1 强制 PWM 输出为低电平
- GPIO25 被用作 INPUT XBAR INPUT1 的输入
- INPUT1（来自 INPUT XBAR）被用作 DCAEVT1 的源
- GPIO25 被设置为输出，并在主循环中切换以触发 PWM
- PWM1 具有 DCBEVT1 强制 PWM 输出为低电平
- GPIO25 被用作 INPUT XBAR INPUT1 的输入
- INPUT1（来自 INPUT XBAR）被用作 DCAEVT1 的源
- GPIO25 被设置为输出，并在主循环中切换以触发 PWM
- DCBEVT1 使用 DCBEVT1 的滤波版本

- DCFILT 信号使用谷底开关模块将 DCFILT 信号延迟一个软件定义的 DELAY 值。

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO25 TRIPIN1（输出引脚，通过软件切换）

监视变量

- 无。

2.19.7 PWM 数字比较边缘滤波器

文件：pwm_ex7_edge_filter.c

这个例子配置 PWM1 如下：

- PWM1 通过 DCBEVT2 将 PWM 输出强制置为低电平作为 CBC 源
- GPIO25 被用作输入到 INPUT XBAR INPUT1
- INPUT1（来自 INPUT XBAR）被用作 DCBEVT2 的源
- GPIO25 被设置为输出，并在主循环中切换以触发 PWM
- DCBEVT2 是 DCFILT 的源
- DCFILT 将计算 DCBEVT2 的边缘并在 DCBEVT2 的第四个边缘生成信号来触发 PWM

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO25 TRIPIN1（输出引脚，通过软件切换）

监视变量

- 无。

2.19.8 PWM 死区

文件：pwm_ex8_deadband.c

这个例子配置 PWM1 到 PWM6 如下：

- PWM1 禁用死区 (参考)
- PWM2 死区激活高电平

- PWM3 死区激活低电平
- PWM4 死区激活高电平互补
- PWM5 死区激活低电平互补
- PWM6 死区输出交换 (交换 A 和 B 输出)

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO2 PWM2A
- GPIO3 PWM2B
- GPIO4 PWM3A
- GPIO5 PWM3B
- GPIO6 PWM4A
- GPIO7 PWM4B
- GPIO8 PWM5A
- GPIO9 PWM5B
- GPIO10 PWM6A
- GPIO11 PWM6B

监视变量

- 无。

2.19.9 PWM DMA

文件：pwm_ex9_dma.c

这个示例配置了 PWM1 和 DMA 如下：

- PWM1 被设置为生成 PWM 波形
- DMA5 被设置为在每个周期更新 CMPAHR、CMPA、CMPBHR 和 CMPB，使用配置数组中的下一个值。这允许用户为所有的 CMPx 和 CMPxHR 寄存器创建一个 DMA 启用的 fifo，以生成非传统的 PWM 波形。
- DMA6 被设置为在每个周期使用配置数组中的下一个值更新 TBPHSHR、TBPHS、TBPRDHR 和 TBPRD。
- 其他寄存器如 AQCTL 也可以通过 DMA 进行控制，通过遵循相同的过程。（本示例中未使

用）

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B

监视变量

- 无。

2.19.10 PWM 斩波器

文件：pwm_ex10_chopper.c

这个示例配置了 PWM1、PWM2、PWM3 和 PWM4 如下：

- PWM1 禁用斩波器（基准）
- PWM2 斩波器启用，占空比为 1/8
- PWM3 斩波器启用，占空比为 6/8
- PWM4 斩波器启用，占空比为 1/2，并启用一次脉冲

外部连接

- GPIO0 PWM1A
- GPIO1 PWM1B
- GPIO2 PWM2A
- GPIO3 PWM2B
- GPIO4 PWM3A
- GPIO5 PWM3B
- GPIO6 PWM4A
- GPIO7 PWM4B

监视变量

- 无。

2.19.11 PWM 信号配置

文件：pwm_ex11_configure_signal.c

这个例子配置 PWM1、PWM2、PWM3 以产生所需频率和占空比的信号。它还配置了这些模块

之间的相位。在 PWMxA 和 PWMxB 上配置了频率为 10kHz、占空比为 0.5 的信号，并且 PWMxB 被反转。此外，配置了 PWM1 到 PWM3 信号之间的 120 度相位。

在测试期间，使用示波器监视 PWM1、PWM2 和/或 PWM3 的输出。

- PWM1A 在 GPIO0
- PWM1B 在 GPIO1
- PWM2A 在 GPIO2
- PWM2B 在 GPIO3
- PWM3A 在 GPIO4
- PWM3B 在 GPIO5

2.19.12 单脉冲模式实现

文件：pwm_ex12_monoshot_mode.c

该示例演示了如何基于外部触发器生成单脉冲 PWM 输出，即在接收外部触发器时仅生成一个脉冲输出。下一个脉冲只有在下一个触发器到达时才会生成。该示例利用外部同步和 T1 动作限定器事件功能来实现所需的输出。

PWM1 用于生成单脉冲输出，而 PWM2 用作外部触发器。无需外部连接，因为 PWM2A 会自动作为触发器通过输入 XBAR 进行馈送。

当接收外部触发器时，PWM1 被配置为生成 0.4 微秒的单脉冲。通过启用相位同步功能，并将 PWMxSYNCI 配置为 EXTSYNCIN1 来实现这一目标。而且，将 PWMxSYNCI 配置为动作限定器的 T1 事件，以在“CTR = PRD”动作时将输出设置为高电平。

PWM2 被配置为生成频率为 125 千赫兹，占空比为 1%的信号（用于模拟上升沿触发器），并通过输入 XBAR 路由到 EXTSYNCIN1。

在示波器上观察 GPIO0（PWM1A：单脉冲输出）和 GPIO2（PWM2：外部触发器）。

2.19.13 PWM 动作限定器 (pwm_up_aq)

文件：pwm_ex13_up_aq.c

这个例子配置 PWM1、PWM2 和 PWM3 以产生一个具有独立调制的波形，PWMxA 和 PWMxB 分别进行调制。比较值 CMPA 和 CMPB 在 PWM 的 ISR 中被修改。在这个例子中，TB 计数器处于向上计数模式。通过示波器查看 PWM1A/B（GPIO0 和 GPIO1）、PWM2A/B（GPIO2 和 GPIO3）和 PWM3A/B（GPIO4 和 GPIO5）的波形。

2.19.14 PWM 独立调制波形

文件：pwm_ex14_global_load_use_case.c

该应用报告中的用例描述如下：

- PWM1/2/3 的输出频率为 500 kHz
- PWM2 相对于 PWM1 具有 120 度的相位移
- PWM3 相对于 PWM1 具有 240 度的相位移
- PWM1/2/3 的占空比为 45%
- PWM1/2/3 的激活高电平互补信号配对，上升/下降沿延迟为 200 纳秒
- 通过 PWM2 上的比较器信号进行周期性触发保护
- 通过 PWM3 上的 GPIO 进行单次触发保护
- 每当时间基准计数器等于零时，会产生中断信号
- 全局加载以支持动作限定器设置的异步更新
- 将 PWM1 的 CMPA/CMPB 连接到 PWM2 和 PWM3

2.20 QEP 例程

2.20.1 QEP 频率测量

文件：qep_ex1_freq_cal.c

这个示例将使用 QEP 模块来计算输入信号的频率。PWM1A 配置为生成频率为 5kHz 的输入信号。每个周期它会中断一次并调用频率计算函数。这个示例使用 IQMath 库来简化高精度计算。

除了主示例文件外，这个项目中还必须包含以下文件：

- qep_ex1_calculation.c——包含频率计算函数
- qep_ex1_calculation.h——包含频率结构的初始化值

这个示例的配置如下：

- 最大频率配置为 10KHz（baseFreq）
- 最小频率假定为 50Hz 用于捕获预分频选择

SPEED_FR：高频率测量通过计算外部输入脉冲，计数时间为 10ms（定时器设置为 100Hz）。

SPEED_PR：低频率测量通过测量输入边缘的时间周期获得。时间测量平均值取自 64 个边缘以获得更好的结果，并且捕获单元使用经过预分频的 APBCLK 进行时间测量。

请注意，捕获单元时钟的预分频器被选择为使捕获定时器在所需的最小频率下不会溢出。这个示例会持续运行直到用户停止它。

有关频率计算的更多信息，请参阅 qep_ex1_calculation.c 开头的注释

外部连接

- 连接 GPIO10/QEP1A 到 GPIO0/PWM1A

监视变量

- freq.freqHzFR——使用位置计数器/单元超时进行频率测量
- freq.freqHzPR——使用捕获单元进行频率测量

2.20.2 QEP 监测位置和速度

文件：qep_ex2_pos_speed.c

这个示例使用 QEP 模块的捕获单元提供位置和速度测量，使用时间单位的速度测量。PWM1 和一个 GPIO 被配置为生成模拟的 QEP 信号。PWM 模块将在每个周期中断一次，并调用位置/速度计算函数。这个示例使用 IQMath 库简化高精度计算。

除了主示例文件外，此项目还必须包括以下文件：

- qep_ex2_calculation.c——包含位置/速度计算函数
- qep_ex2_calculation.h——包含位置/速度结构的初始化值

此示例的配置如下：

- 最大速度配置为 6000rpm（baseRPM）
- 最小速度假设为 10rpm 以进行捕获预分频选择
- 极对被配置为 2（polePairs）
- 编码器分辨率配置为每转 4000 计数（mechScaler）
- 这意味着：4000 / 4 = 每转 1000 线的正交编码器（由 PWM1 模拟）
- PWM1（模拟 QEP 编码器信号）配置为 5kHz 频率或 300rpm（= 4 * 5000 cnts/sec * 60 sec/min）/ 4000 cnts/rev）

SPEEDRPM_FR：通过计数 QEP 输入脉冲 10ms（单位定时器设置为 100Hz）获得高速测量。

SPEEDRPM_FR = （位置增量/10ms）* 60 rpm

SPEEDRPM_PR：通过测量 QEP 边缘的时间段获得低速测量。时间测量在 64 个边缘上平均以获得更好的结果，并且捕获单元使用预分频 APBCLK 执行时间测量。

请注意，捕获单元时钟的预分频器被选择为使捕获定时器不会在所需的最小频率下溢出。这个示例会无限期地运行，直到用户停止它。

有关位置/速度计算的更多信息，请参阅 qep_ex2_calculation.c 开头的注释。

外部连接

- 将 GPIO6/QEP1A 连接到 GPIO0/PWM1A（模拟 QEP A 相信号）
- 将 GPIO7/QEP1B 连接到 GPIO1/PWM1B（模拟 QEP B 相信号）
- 将 GPIO9/QEP1I 连接到 GPIO4（模拟 QEP 索引信号）

观察变量

- posSpeed.speedRPMFR——使用 QEP 位置计数器测量的速度（rpm）
- posSpeed.speedRPMPR——使用捕获单元测量的速度（rpm）
- posSpeed.thetaMech——电机机械角度（Q15）
- posSpeed.thetaElec——电机电气角度（Q15）

2.20.3 QEP 使用中断计算信号频率

文件：qep_ex4_freq_cal_interrupt.c

这个示例将使用 QEP 模块计算输入信号的频率。PWM1A 被配置为以 5 kHz 的频率生成这个输入信号。设置了 QEP 单元超时，它会每 UNIT_PERIOD 微秒生成一个中断，并且频率计算会持续进行。

这个示例的配置如下：

- PWM 频率指定为 5000Hz
- UNIT_PERIOD 指定为 10000 微秒
- 最小频率为(1/(2*10 毫秒))即 50Hz
- 最高频率可以是(2^32)/((2*10 毫秒))
- 频率测量的分辨率为 50hz

频率：通过计算单位周期（单位定时器设置为 10 毫秒）内外部输入脉冲的数量来获得频率测量。

外部连接：

- 将 GPIO6/QEP1A 连接到 GPIO10/PWMA

观察变量：

- freq——使用位置计数器/单元超时进行频率测量
- pass——如果测量得到的频率与 PWM 频率匹配，则 pass = 1，否则为 0

2.20.4 四分位编码器模式下感知电机的速度和方向

文件：qep_ex5_speed_dir_motor.c

这个示例可用于使用 QEP 在四分位编码器模式下感知电机的速度和方向。 PWM1A 被配置为在 A 和 B 引脚上模拟电机编码器信号，频率为 5kHz，相位差为 90 度（以便在没有电机的情况下运行此示例）。设置 QEP 单元超时，将每 UNIT_PERIOD 微秒生成一个中断，并且根据电机的方向连续进行速度计算。

该示例的配置如下

- PWM 频率指定为 5000Hz
- UNIT_PERIOD 指定为 10000 微秒
- 模拟的四分编码信号频率为 20000Hz（4 * 5000）
- 假设编码器孔数为 1000
- 因此，模拟的电机速度为 300rpm（5000 *（60 / 1000））

freq：通过计算 UNIT_PERIOD（单元定时器设置为 10 毫秒）内的外部输入脉冲来测量的模拟四分编码信号频率。

speed：以 rpm 为单位测量电机速度

dir：表示顺时针（1）或逆时针（-1）

外部连接（如果通过 PWM 模拟电机编码器信号）

- 将 GPIO6/QEP1A 连接到 GPIO10/PWMA（模拟 QEP A 相信号）
- 将 GPIO7/QEP1B 连接到 GPIO11/PWMB（模拟 QEP B 相信号）

监控变量

- freq：通过计算 UNIT_PERIOD（单元定时器设置为 10 毫秒）内的外部输入脉冲，得出的模拟电机频率测量。
- speed：以 rpm 为单位测量电机速度
- dir：表示顺时针（1）或逆时针（-1）
- pass——如果测得的四分频率与输入四分频率（4 * PWM 频率）相匹配，则 pass = 1，否则 fail = 1（**仅当“MOTOR”被注释时）

2.21 SDF 例程说明

2.21.1 SDF 滤波器 CPU 同步

文件：sdf_ex1_filter_sync_cpuread.c

在此示例中，SDF 滤波器数据由 CPU 在 SDF ISR 例程中读取。SDF 配置如下：

- 本示例使用的 SDF：SDF1

- 选择的输入控制模式：MODE0
- 比较器设置：
  - 选择 Sinc3 滤波器
  - OSR = 32
  - HLT = 0x7FFF（高阈值设置）
  - LLT = 0x0000（低阈值设置）
- 数据滤波器设置：
  - 启用所有 4 个滤波模块
  - 选择 Sinc3 滤波器
  - OSR = 128
  - 所有 4 个滤波器通过使用 MFE（主滤波器使能位）进行同步
  - 滤波器输出以 16 位格式表示
  - 为了将 25 位数据滤波器转换为 16 位格式，用户需要对 OSR = 128 的 Sinc3 滤波器右移 7 位
- SDF 滤波器的中断模块设置：
  - 禁用所有 4 个高阈值比较器中断
  - 禁用所有 4 个低阈值比较器中断
  - 禁用所有 4 个调制器故障中断
  - 当有新滤波器数据可用时，所有 4 个滤波器将生成中断

外部连接：

- 将 Sigma——Delta 流连接到 GPIO24——GPIO31 上的（SD——D1、SD——C1 到 SD——D4、SD——C4）

监控变量：

- filter1Result——滤波器 1 的输出
- filter2Result——滤波器 2 的输出
- filter3Result——滤波器 3 的输出
- filter4Result——滤波器 4 的输出

2.21.2 SDF 滤波器同步 DMA

文件：sdf_ex3_filter_sync_dmaread.c

在此示例中，SDF 滤波器数据由 DMA 读取。SDF 配置如下：

- 本示例使用的 SDF：SDF1
- 选择的输入控制模式：MODE0
- 比较器设置：
  - 选择 Sinc3 滤波器
  - OSR = 32
  - HLT = 0x7FFF（高阈值设置）
  - LLT = 0x0000（低阈值设置）
- 数据滤波器设置：
  - 启用所有 4 个滤波模块
  - 选择 Sinc3 滤波器
  - OSR = 256
  - 所有 4 个滤波器通过使用 MFE（主滤波器使能位）进行同步
  - 滤波器输出以 16 位格式表示
  - 为了将 25 位数据滤波器转换为 16 位格式，用户需要对 OSR = 256 的 Sinc3 滤波器右移 10 位
- SDF 滤波器的中断模块设置：
  - 禁用所有 4 个高阈值比较器中断
  - 禁用所有 4 个低阈值比较器中断
  - 禁用所有 4 个调制器故障中断
  - 当有新滤波器数据可用时，所有 4 个滤波器将生成中断

外部连接：

- 将 Sigma——Delta 流连接到 GPIO24——GPIO31 上的（SD——D1、SD——C1 到 SD——D4、SD——C4）

监控变量：

- filter1Result——滤波器 1 的输出

- filter2Result——滤波器 2 的输出
- filter3Result——滤波器 3 的输出
- filter4Result——滤波器 4 的输出

2.21.3 SDF PWM 同步

文件：sdf_ex4_pwm_sync_cpuread.c

在此示例中，SDF 滤波器数据由 CPU 在 SDF ISR 例程中读取。SDF 配置如下：

- 本示例使用的 SDF：SDF1
- 选择的输入控制模式：MODE0
- 比较器设置：
  - 选择 Sinc3 滤波器
  - OSR = 32
  - HLT = 0x7FFF（高阈值设置）
  - LLT = 0x0000（低阈值设置）
- 数据滤波器设置：
  - 启用所有 4 个滤波模块
  - 选择 Sinc3 滤波器
  - OSR = 256
  - 所有 4 个滤波器通过使用 PWM（主滤波器使能位）进行同步
  - 滤波器输出以 16 位格式表示
  - 为了将 25 位数据滤波器转换为 16 位格式，用户需要对 OSR = 256 的 Sinc3 滤波器右移 10 位
- SDF 滤波器的中断模块设置：
  - 禁用所有 4 个高阈值比较器中断
  - 禁用所有 4 个低阈值比较器中断
  - 禁用所有 4 个调制器故障中断
  - 当有新滤波器数据可用时，所有 4 个滤波器将生成中断

外部连接：

- 将 Sigma——Delta 流连接到 GPIO24——GPIO31 上的（SD——D1、SD——C1 到 SD—

—D4、SD——C4）

监控变量：

- filter1Result——滤波器 1 的输出
- filter2Result——滤波器 2 的输出
- filter3Result——滤波器 3 的输出
- filter4Result——滤波器 4 的输出

2.21.4 SDF1 滤波器 FIFO 的使用

文件：sdf_ex5_filter_sync_fifo_cpuread.c

该示例配置 SDF1 滤波器以演示通过 CPU 在 FIFO 和非 FIFO 模式下的数据读取。数据滤波器配置为 mode 0，选择 OSR 为 256 的 SINC3 滤波器。滤波器输出配置为 16 位格式，并使用 10 位的数据移位。

此示例演示了 FIFO 的使用（如果启用）。FIFO 长度设置为 16，当 FIFO 满时，将触发数据就绪中断。在此示例中，SDF 滤波器数据由 CPU 在 SDF 数据就绪 ISR 例程中读取。

外部连接：

- 将 Sigma——Delta 流连接到 GPIO24——GPIO31 上的（SD——D1、SD——C1 到 SD——D4、SD——C4）

监控变量：

- filter1Result——滤波器 1 的输出

2.22 SPI 例程说明

2.22.1 SPI 回环测试

文件：spi_ex1_loopback.c

该程序使用了 SPI 模块的内部回环测试模式。这是一个非常基础的回环，不使用 FIFO 或中断。发送一串数据，然后与接收到的数据进行比较。引脚复用和 SPI 模块可以根据实际需要重新配置。

发送的数据格式如下：

0000 0001 0002 0003 0004 0005 0006 0007 ... FFFE FFFF 0000

此模式会无限重复。

外部连接：

- 无

监视变量：

- sData——要发送的数据
- rData——接收到的数据

2.22.2 带有 FIFO 中断的 SPI 回环测试

文件：spi_ex2_loopback_fifo_interrupts.c

该程序使用了 SPI 模块的内部回环测试模式，使用了 SPI 的 FIFO 和中断。

发送一串数据，然后与接收到的数据进行比较。发送的数据格式如下：

0000 0001

0001 0002

0002 0003

...

FFFE FFFF

FFFF 0000

等等...

此模式会无限重复。

外部连接：

- 无

监视变量：

- sData——要发送的数据
- rData——接收到的数据
- rDataPoint——用于跟踪接收流中的最后位置以进行错误检查

2.22.3 无 FIFO 中断的 SPI 外部回环测试

文件：spi_ex3_external_loopback.c

该程序使用了两个 SPI 模块之间的外部回环。在此示例中没有使用 SPI 的 FIFO 和中断。SPIA 配置为从设备，SPIB 配置为主控制器。这个示例展示了全双工通信，使控制器和从设备可以同时发送和接收数据。

外部连接：

- GPIO7 和 GPIO16——SPISIMO

- GPIO25 和 GPIO17——SPISOMI
- GPIO26 和 GPIO56——SPICLK
- GPIO27 和 GPIO57——SPISTE

监视变量：

- TxData_SPIA——从 SPIA（从设备）发送的数据
- TxData_SPIB——从 SPIB（主控制器）发送的数据
- RxData_SPIA——SPIA（从设备）接收的数据
- RxData_SPIB——SPIB（主控制器）接收的数据

2.22.4 带有 FIFO 中断的 SPI 外部回环测试

文件：spi_ex4_external_loopback_fifo_interrupts.c

该程序使用了两个 SPI 模块之间的外部回环，使用了 SPI 的 FIFO 和中断。SPIA 配置为从设备，从 SPIB（配置为主控制器）接收数据。

发送一串数据，然后与接收到的数据进行比较。发送的数据格式如下：

0000 0001

0001 0002

0002 0003

...

FFFE FFFF

FFFF 0000

等等...

此模式会无限重复。

外部连接：

- GPIO7 和 GPIO16——SPISIMO
- GPIO25 和 GPIO17——SPISOMI
- GPIO26 和 GPIO3——SPICLK
- GPIO27 和 GPIO11——SPISTE

监视变量：

- sData——要发送的数据

- rData——接收到的数据
- rDataPoint——用于跟踪接收流中的最后位置以进行错误检查

2.22.5 带 DMA 的 SPI 回环测试

文件：spi_ex5_loopback_dma.c

该程序使用了 SPI 模块的内部回环测试模式，使用了 DMA 中断和 SPI 的 FIFO。当 SPI 发送 FIFO 有足够的空间（由 FIFO 级别中断信号指示）时，DMA 会将数据从全局变量 sData 传输到 FIFO，这些数据通过内部回环被传输到接收 FIFO。

当接收 FIFO 中有足够的数据（由 FIFO 级别中断信号指示）时，DMA 会将数据从 FIFO 传输到全局变量 rData。

当所有数据都被放入 rData 后，将在 DMA 通道的一个 ISR 中进行数据有效性检查。

外部连接：

- 无

监视变量：

- sData——要发送的数据
- rData——接收到的数据

2.22.6 SPI EEPROM

文件：spi_ex6_eeprom.c

该程序将 8 字节数据写入 EEPROM 并读回。设备通过 SPI 和特定操作码与 EEPROM 通信。此示例适用于 SPI 串行 EEPROM AT25128/256。

外部连接：

连接外部 SPI EEPROM

- 连接 GPIO16（PICO）到外部 EEPROM SI 引脚
- 连接 GPIO17（POCI）到外部 EEPROM SO 引脚
- 连接 GPIO56（CLK）到外部 EEPROM SCK 引脚
- 连接 GPIO11（CS）到外部 EEPROM CS 引脚
- 连接外部 EEPROM 的 VCC 和 GND 引脚

监视变量：

- writeBuffer——写入外部 EEPROM 的数据

- readBuffer——从 EEPROM 读回的数据
- error——错误计数

2.22.7 SPI DMA EEPROM

文件：spi_ex7_eeprom_dma.c

该程序将 8 字节数据写入 EEPROM 并读回。设备通过 SPI 使用 DMA 和特定操作码与 EEPROM 通信。此示例适用于 SPI 串行 EEPROM AT25128/256。

外部连接：

连接外部 SPI EEPROM

- 连接 GPIO16（PICO）到外部 EEPROM SI 引脚
- 连接 GPIO17（POCI）到外部 EEPROM SO 引脚
- 连接 GPIO56（CLK）到外部 EEPROM SCK 引脚
- 连接 GPIO11（CS）到外部 EEPROM CS 引脚
- 连接外部 EEPROM 的 VCC 和 GND 引脚

监视变量：

- writeBuffer——写入外部 EEPROM 的数据
- SPI_DMA_Handle.RXdata——当接收字节数小于 4 时，从 EEPROM 读回的数据
- SPI_DMA_Handle.pSPIRXDMA->pbuffer——从 EEPROM 接收数据的起始地址
- error——错误计数

2.23 SYSCTL 例程说明

2.23.1 缺失时钟检测

文件：sysctl_ex1_missing_clock_detection.c

该程序演示了缺失时钟检测（MCD）功能及其处理方法。当通过断开 OSCCLK 与 MCD 模块的连接来模拟 MCD 时，将产生一个 NMI 中断。这个 NMI 中断表明由于时钟故障生成了 MCD，并在 ISR 中处理。

在发生 MCD 之前，时钟频率按照设备初始化设置为 250MHz。MCD 发生后，频率将更改为 10MHz 或 INTOSC1。

该示例还展示了在缺失时钟检测后如何重新锁定 PLL，首先明确地将时钟源切换到 INTOSC1，重置缺失时钟检测电路，然后重新锁定 PLL。重新锁定后，时钟频率将为 250MHz，但使用 INTOSC1 作为时钟源。

外部连接：

- 无

监视变量：

- fail——表示缺失时钟未被检测到或未正确处理。
- mcd_clkfail_isr——表示由于缺失时钟故障触发了 NMI 中断并调用 ISR 进行处理。
- mcd_detect——表示检测到缺失时钟。
- result——表示缺失时钟检测成功处理的状态。

2.23.2 XCLKOUT 配置

文件：sysctl_ex2_xclkout_config.c

该程序演示了如何配置 XCLKOUT 引脚，通过外部引脚观察内部时钟，用于调试和测试目的。

在这个示例中，INTOSC1 被用作 XCLKOUT 时钟源，分频器配置为 8。

XCLKOUT 的预期频率计算如下：

XCLKOUT frequency = INTOSC1 frequency / 8 =10MHz / 8 = 1.25MHz

外部连接：

- 使用示波器观察 GPIO16 上的 XCLKOUT。

监视变量：

- 无

2.24 Timer 例程说明

2.24.1 CPU 定时器示例

文件：timer_ex1_tmrs.c（timer_ex1_cputimers.c）

该程序例子配置了 CPU 定时器 0、1 和 2，并在每次定时器触发中断时递增一个计数器。

外部连接：

- 无

监视变量：

- cpuTimer0IntCount——CPU 定时器 0 中断计数
- cpuTimer1IntCount——CPU 定时器 1 中断计数
- cpuTimer2IntCount——CPU 定时器 2 中断计数

2.25 UART 例程说明

2.25.1 通过 UART 调整波特率示例

文件：baud_tune_via_uart.c

本示例演示了如何根据来自另一设备的 UART 输入调整 G32R501 设备的 UART 波特率。由于 UART 没有时钟信号，可靠的通信需要波特率大致匹配。本示例解决了设备之间时钟不匹配超过可接受范围的情况，要求设备之间波特率补偿。由于可靠通信只需要匹配有效的波特率，无论是哪两个设备进行调整（时钟源不太准确的设备不需要进行调整，只要其中一个设备调整到另一个设备，那么就可以建立正确的通信）。

要调整本设备的波特率，必须将 UART 数据（所需的波特率）发送到本设备。输入的 UART 波特率必须在下面选择的 TARGETBAUD 的+/—— MARGINPERCENT 范围内。这两个变量在下面定义，应根据应用要求选择。在嘈杂环境中，较高的 MARGINPERCENT 将允许更多的数据被视为“正确”，但可能会降低精度。TARGETBAUD 是预期的波特率，但由于时钟差异，需要调整以增强与另一设备的通信鲁棒性。

对于 Eval 和自定义设备，如果 GPIO9 和 GPIO8 不能使用，可能需要配置不同的 GPIO 用于 UART_RX 和 UART_TX 引脚。打开 UART 外设，选择给定板上的可用 GPIO。更新下面的 GPIO_UARTRX_NUMBER 以匹配 RX 选择。请参阅 Eval 用户指南以获取可用 GPIO 列表。

注意：较低的波特率在寄存器选项中具有更多的粒度，因此在这些速度下调整更有效。

外部连接：

- UART_RX/eCAP1 位于 GPIO9，连接到传入的 UART 通信
- UART_TX 位于 GPIO8，用于外部观察

监控变量：

- avgBaud——调整后的检测和设置的波特率

2.25.2 UART FIFO 回环测试

文件：uart_ex1_loopback.c

该程序使用外设的内部回环测试模式。除了引导模式引脚配置外，不需要其他硬件配置。本测试使用 UART 模块的回环测试模式发送从 0x00 到 0xFF 的字符。测试将发送一个字符，然后检查接收缓冲区是否正确匹配。

监控变量：

- loopCount——发送的字符数
- errorCount——检测到的错误数
- sendChar——发送的字符

- receivedChar——接收的字符

2.25.3 UART 中断回环

文件：uart_ex2_interrupts.c

本测试通过中断接收和回环数据通过 UART——A 端口。可以使用如'putty'之类的终端查看来自 UART 的数据并将信息发送到 UART。UART 端口接收到的字符将被发送回主机。

运行应用程序，使用终端打开带有以下设置的 COM 端口：

- 找到正确的 COM 端口
- 比特率 = 9600
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

程序将打印出问候语，然后要求您输入一个字符，它将回环到终端。

监控变量：

- counter——发送的字符数

外部连接：

通过收发器和电缆将 UART——A 端口连接到 PC。

- GPIO28 是 UART_A——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART_A——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

2.25.4 带 FIFO 的 UART 中断回环

文件：uart_ex3_interrupts_fifo.c

本测试通过中断一次接收和回环两个字符数据通过 UART——A 端口。当 FIFO 状态级别为两个或更多时，触发 Rx 中断。一旦两个字符在 RXFIFO 中，UART Rx ISR 将被触发，并从 FIFO 中读取两个字符并写入发送缓冲区。然后 UART Tx ISR 将再次触发以请求更多来自终端的数据。

运行应用程序，使用终端打开带有以下设置的 COM 端口：

- 找到正确的 COM 端口
- 比特率 = 9600
- 数据位 = 8

- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无

程序将打印出问候语，然后要求您输入两个字符，它将回环到终端。

监控变量:

- counter——发送的字符对数

外部连接:

通过收发器和电缆将 UART——A 端口连接到 PC。

- GPIO28 是 UART_A——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART_A——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

2.25.5 UART 回环测试

文件：uart_ex4_echoback.c

本测试通过 UART——A 端口接收和回环数据。可以使用如'putty'之类的终端查看来自 UART 的数据并将信息发送到 UART。UART 端口接收到的字符将被发送回主机。

运行应用程序，使用终端打开带有以下设置的 COM 端口：

- 找到正确的 COM 端口
- 比特率 = 9600
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无
- 程序将打印出问候语，然后要求您输入一个字符，它将回环到终端。

监控变量:

- loopCounter——发送的字符数

外部连接:

通过收发器和电缆将 UART——A 端口连接到 PC。

- GPIO28 是 UART_A——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART_A——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

2.26 Watchdog 例程说明

2.26.1 Watchdog 例程

文件：watchdog_ex1_service.c

该程序示例展示了如何服务看门狗或使用看门狗生成唤醒中断。默认情况下，该示例将生成一个唤醒中断。要服务看门狗并且不生成中断，请取消注释主循环中的 SysCtl_serviceWatchdog()行。

外部连接：

- 无

监视变量：

- wakeCount——进入看门狗 ISR 的次数
- loopCount——未在 ISR 中执行的循环次数

2.27 Zidian 例程说明

2.27.1 Zidian 数学示例

文件：zidian_ex1_math.c

该程序示例展示了如何使用 zidian_math.h。只有 math.h 中的 sqrtf()、sinf()、cosf()、atanf()和 atan2f()可以通过 Zidian 加速。用户可以通过 traceTime 观察加速效果。

注意：请避免直接使用兼容函数，例如__sin()、__cos()和__atan()。

外部连接：

- 无

监视变量：

- traceResult——计算结果
- traceTime——计算所花费的时间

3 Device Support 例程

注意：这些例程位于 G32R5xx_SDK 的以下位置：

G32R5xx_SDK_VERSION：
/device_support/DEVICE_GPN/examples/CORE_IF_MULTICORE/

3.1 ADC 例程说明

3.1.1 ADC PWM 触发

文件：adc_ex1_soc_pwm.c

该程序示例设置了 PWM1 以定期触发 ADCA 上的转换。

外部连接：

- A1 应连接到需要转换的信号

监视变量：

- adcAResults——来自引脚 A1 的一系列模数转换样本。样本之间的时间间隔基于 PWM 定时器的周期确定。

3.2 CAP 例程说明

3.2.1 CAP APWM 例程

文件：cap_ex1_apwm.c

该程序示例在 APWM 模式下设置 CAP 模块。PWM 波形将会出现在 GPIO5 上。通过使用影子寄存器加载下一个周期/比较值，PWM 的频率被配置为在 5Hz 和 10Hz 之间变化。

外部连接：

- 无

监视变量：

- 无

3.3 DAC 例程说明

3.3.1 缓冲 DAC 使能例程

文件：dac_ex1_enable.c

本示例在缓冲 DAC 输出 DACOUTA/ADCINA0 上生成电压，并使用默认的 DAC 参考设置

VDAC。

外部连接

- 当 DAC 参考设置为 VDAC 时，必须在 VDAC 引脚上施加外部参考电压。这可以通过将跳线从 3.3V 连接到 ADCINB3 来实现。

监视变量

- 无

3.4 DMA 例程说明

3.4.1 DMA SRAM 传输

文件：dma_ex1_sram_transfer.c

本示例使用一个 DMA 通道将数据从 SRAM1 中的缓冲区传输到 SRAM1 中的缓冲区。示例重复设置 DMA 通道 PERINTFRC 位，直到完成 16 个突发传输（每个突发为 8 个 16 位字）。整个传输完成时，将触发 DMA 中断。

监视变量

- sdata——发送的数据
- rdata——接收的数据

3.5 模板例程说明

3.5.1 位域库标准模板

文件：empty_bitfield_main.c

本示例是为位字段开发设置的空项目。

3.5.2 位域库与驱动库空标准模板

文件：empty_bitfield_driverlib_main.c

本示例是为位字段和驱动库开发设置的空项目。

3.6 GPIO 例程说明

3.6.1 GPIO 设置

文件：gpio_ex1_setup.c

配置 G32R5xx 的 GPIO 为两种不同的配置。此代码较为详细，以说明如何设置 GPIO。在实际应用中，可以合并代码行以提高代码大小和效率。本示例仅设置 GPIO。设置完成后未对引脚进行任何操作。

外部连接

- 无

监视变量

- 无

3.7 HRPWM 例程说明

3.7.1 HRPWM SFO V1 高分辨率周期（上计数）

文件：hrpwm_ex1_duty_sfo_v1.c

本示例修改 MEP 控制寄存器，以显示在 PWM 上计数模式下高分辨率周期的边缘位移，这是由于相应 PWM 模块的 HRPWM 控制扩展。

本示例调用以下 Geehy 的 MEP 缩放因子优化器（SFO）软件库 V1 函数：

int SFO();

- 当使用 HRPWM 时，动态更新 MEP_ScaleFactor
- 使用 MEP_ScaleFactor 值更新 HRMSTEP 寄存器（仅在 Pwm1Regs 寄存器空间中存在）
- 返回 2 表示错误：MEP_ScaleFactor 大于最大值 255（在此情况下，自动转换可能无法正常工作）
- 返回 1 表示指定通道完成
- 返回 0 表示指定通道未完成

本示例旨在解释 HRPWM 能力。代码可以优化以提高代码效率。

运行本示例：

- 在最大 SYSCLKOUT 下运行本示例
- 激活实时模式
- 运行代码

外部连接

- 在示波器上监视 PWM1 A/B 引脚。

监视变量

- status——示例运行状态
- UpdateFine——设置为 1 以使用 HRPWM 功能并观察细 MEP 步长（默认）。设置为 0 以禁用 HRPWM 功能并观察粗 SYSCLKOUT 周期步长。

3.7.2 HRPWM SFO V1 高分辨率周期（上下计数）

文件：hrpwm_ex2_prdupdown_sfo_v1.c

本示例修改 MEP 控制寄存器，以显示在 PWM 上下计数模式下高分辨率周期的边缘位移，这是由于相应 PWM 模块的 HRPWM 控制扩展。

本示例调用以下 Geehy 的 MEP 缩放因子优化器（SFO）软件库 V1 函数：

int SFO();

- 当使用 HRPWM 时，动态更新 MEP_ScaleFactor
- 使用 MEP_ScaleFactor 值更新 HRMSTEP 寄存器（仅在 Pwm1Regs 寄存器空间中存在）
- 返回 2 表示错误：MEP_ScaleFactor 大于最大值 255（在此情况下，自动转换可能无法正常工作）
- 返回 1 表示指定通道完成
- 返回 0 表示指定通道未完成

本示例旨在解释 HRPWM 能力。代码可以优化以提高代码效率。

运行本示例：

- 在最大 SYSCLKOUT 下运行本示例
- 激活实时模式
- 运行代码

外部连接

- 在示波器上监视 PWM1 A/B 引脚。

监视变量

- UpdateFine——设置为 1 以使用 HRPWM 功能并观察细 MEP 步长（默认）。设置为 0 以禁用 HRPWM 功能并观察出 SYSCLKOUT 周期步长。

3.8 I2C 例程说明

3.8.1 I2C 主设备

文件：i2c_ex1_master.c

本程序展示了如何在主配置中使用 I2CA。本示例使用轮询方式，不使用中断或 FIFO。

需要两个控制卡：

- 一个配置为主设备，另一个配置为从设备。
- 主设备将运行由“i2c_ex1_master.uvprojx”生成的二进制文件。
- 从设备将运行由“i2c_ex1_slave.uvprojx”生成的二进制文件。

外部连接：

控制卡上的外部连接应如下所示：

表格 3 控制卡上的外部连接示意

Signal	I2CA on Board1	I2CA on Board 2
SCL	GPIO_PIN_SCLA	GPIO_PIN_SCLA
SDA	GPIO_PIN_SDAA	GPIO_PIN_SDAA
GND	GND	GND

- 示例 1：主发送器和从接收器
- 示例 2：主接收器和从发送器
- 示例 3：主发送器和从接收器，然后是主接收器和从发送器
- 示例 4：主接收器和从发送器，然后是主发送器和从接收器

监视变量：

- I2CA_TXdata
- I2CA_RXdata

3.8.2 I2C 从设备

文件：i2c_ex1_slave.c

本程序展示了如何在从配置中使用 I2CA。本示例使用 I2C 中断，不使用 FIFO。

需要两个控制卡：

- 一个配置为主设备，另一个配置为从设备。
- 主设备将运行由“i2c_ex1_master.uvprojx”生成的二进制文件。
- 从设备将运行由“i2c_ex1_slave.uvprojx”生成的二进制文件。

外部连接：

控制卡上的外部连接应如下所示：

表格 4 控制卡上的外部连接示意

Signal	I2CA on Board1	I2CA on Board 2
SCL	GPIO_PIN_SCLA	GPIO_PIN_SCLA
SDA	GPIO_PIN_SDAA	GPIO_PIN_SDAA
GND	GND	GND

- 示例 1：主发送器和从接收器
- 示例 2：主接收器和从发送器
- 示例 3：主发送器和从接收器，然后是主接收器和从发送器
- 示例 4：主接收器和从发送器，然后是主发送器和从接收器

监视变量：

- I2CA_TXdata
- I2CA_RXdata

3.9 LED 例程说明

3.9.1 LED 闪烁

文件：led_ex1_blinky.c

本示例演示如何使 LED 闪烁。

3.10 SPI 例程说明

3.10.1 SPI 回环测试

文件：spi_ex1_loopback.c

该程序使用了 SPI 模块的内部回环测试模式。这是一个非常基础的回环，不使用 FIFO 或中断。发送一串数据，然后与接收到的数据进行比较。引脚复用和 SPI 模块可以根据实际需要重新配置。

发送的数据格式如下：

0000 0001 0002 0003 0004 0005 0006 0007 ... FFFE FFFF 0000

此模式会无限重复。

外部连接：

- 无

监视变量：

- sData——要发送的数据
- rData——接收到的数据

3.10.2 带 DMA 的 SPI 回环测试

文件：spi_ex2_dma_loopback.c

该程序使用了 SPI 模块的内部回环测试模式，使用了 DMA 中断和 SPI 的 FIFO。当 SPI 发送 FIFO 有足够的空间（由 FIFO 级别中断信号指示）时，DMA 会将数据从全局变量 sData 传输到 FIFO，这些数据通过内部回环被传输到接收 FIFO。

当接收 FIFO 中有足够的数据（由 FIFO 级别中断信号指示）时，DMA 会将数据从 FIFO 传输到全局变量 rData。

当所有数据都被放入 rData 后，将在 DMA 通道的一个 ISR 中进行数据有效性检查。

外部连接：

- 无

监视变量：

- sData——要发送的数据
- rData——接收到的数据

3.11 Timer 例程说明

3.11.1 CPU 定时器示例

文件：timer_ex1_tmrs.c（timer_ex1_cputimers.c）

该程序例子配置了 CPU 定时器 0、1 和 2，并在每次定时器触发中断时递增一个计数器。

外部连接：

- 无

监视变量：

- cpuTimer0IntCount——CPU 定时器 0 中断计数
- cpuTimer1IntCount——CPU 定时器 1 中断计数
- cpuTimer2IntCount——CPU 定时器 2 中断计数

3.12 UART 例程说明

3.12.1 UART 回环测试

文件：uart_ex4_echoback.c

本测试通过 UART——A 端口接收和回环数据。可以使用如'putty'之类的终端查看来自 UART 的数据并将信息发送到 UART。UART 端口接收到的字符将被发送回主机。

运行应用程序，使用终端打开带有以下设置的 COM 端口：

- 找到正确的 COM 端口
- 比特率 = 9600
- 数据位 = 8
- 校验位 = 无
- 停止位 = 1
- 硬件控制 = 无
- 程序将打印出问候语，然后要求您输入一个字符，它将回环到终端。

监控变量:

- loopCounter——发送的字符数

外部连接:

通过收发器和电缆将 UART——A 端口连接到 PC。

- GPIO28 是 UART_A——RXD（连接到串行 DB9 电缆的 Pin3，PC——TX）
- GPIO29 是 UART_A——TXD（连接到串行 DB9 电缆的 Pin2，PC——RX）

4 版本历史

表格 5 文件版本历史

日期	版本	变更历史
2025.01	1.0	新建

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。