GigaDevice Semiconductor Inc.

GD32G5x3 系列高分辨率定时器使用指南

应用笔记

AN203

1.1 版本

（2024 年 11 月）

目录

图索引

表索引

1. 术语与缩写

本文档中使用的专业术语与缩写如*表 1-1. 常用缩略语对照表*所示。

表 1-1. 常用缩略语对照表

缩略语	全称	中文术语
HRTIMER	High-Resolution Timer	高分辨率定时器
MT	Master Timer	主定时器
STx	Slave Timer x	从定时器
DLL	Delay-locked loop	延迟锁相环
PER	Period	周期值/周期事件
CMPx	Compare x	比较值/比较x事件
CBC	Cycle by cycle	逐波限流
LLC	LLC Resonant Converters	LLC谐振变换器
TRIGSEL	Trigger Selection Controller	触发选择控制器

2. 简介

GD32G5x3 系列 MCU 带有高分辨率定时器 HRTIMER 模块。

HRTIMER 采用的是主从定时器架构，由 1 个 MT 和 8 个 STx 组成。

MT 单元主要由 16 位计数器、自动重载寄存器、重复计数器和 4 个比较器等部分构成。

STx 单元则主要由 16 位计数器、自动重载寄存器、重复计数器、4 个比较器、2 个捕获寄存器、置位复位交叉开关、空闲控制级和通道输出级等部分构成。每个 STx 单元可以生成 2 个独立的 PWM 波或 1 对互补带死区的 PWM 波。

整个 HRTIMER 模块最多可以产生 16 个高分辨率的 PWM 波，也可以耦合成 8 对互补带死区的 PWM 波。

HRTIMER 模块由于其具有高分辨率、高灵活性等特点，因此比较适用于开关电源和功率控制等应用中。

本文档将在 GD32G5x3 系列 MCU 基础上，以 216MHz 系统时钟为例，对 HRTIMER 的基本功能和参考应用进行介绍。当系统时钟为其他值时，文档的一些计算值将不再适用，需要文档内容的基础上重新进行计算。

3. 基本配置

3.1. 周期值与比较值设置

MT 和 STx 的时钟源是由 RCU 模块提供的 HRTIMER_CK；

DLL 用于产生并校准高分辨率计数单元 HRTIMER_HPCK=32*$f_{HRTIMER\_CK}$；

预分频器将高分辨率计数单元 HRTIMER_HPCK 除以分频数，得到计数器时钟 HRTIMER_PSCCK；

它们之间的频率关系可以表示如下：$f_{HRTIMER\_PSCCK}$=$f_{HRTIMER\_HPCK}$/分频比；

因此，计数器时钟=HRTIMER_CK*32/分频比。可以将 32/分频比视作倍频数来理解，这样计数器时钟=HRTIMER_CK*倍频数。

以分频比为 1 为例，计数器时钟=HRTIMER_CK*32，可以看成 32 倍频的具有 6.912GHz（216MHz*32）时钟的定时器。周期值和比较值可以直接写入具有高分辨率的对应的 16 位寄存器中。计数周期可以简单地使用公式 PER = $f_{High\text{-}res}/f_{pwm}$ 进行计算。

例如，想要发出 200kHz 周期值的 PWM 时，周期值寄存器的值应当设定为 PER = 6.912GHz/200kHz = 34560。具体配置参考代码可参考*表 3-1. 定时器周期配置代码*。

表 3-1. 定时器周期配置代码

```
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL32;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
```

3.2. 置位/复位交叉开关

定时器单元通过置位/复位交叉开关来控制两个输出，可以将多个事件设置为置位或复位事件，从而增加输出灵活性。通过置位/复位事件的配置，可以将输出配置成两路独立的 PWM 波或一对互补的带死区的 PWM 波。

3.3. 通道输出级

通道输出级有三种工作状态：

- 运行状态：正常输出；
- 空闲状态：输出定义好的空闲状态；
- 故障状态：故障输入触发故障状态；

一般来说，生成 PWM 波形时使用运行状态，故障保护时触发故障状态。

3.4. 影子寄存器

MT 和 STx 中的某些寄存器具有影子寄存器。MCU 复位后，影子寄存器被禁能。如果 SHWEN 位置 1，则使能影子寄存器，*表 3-2. 影子寄存器*中列出的寄存器被预加载。写入这些寄存器的值将被传送到影子寄存器，且不会立即生效。当发生更新事件时，影子寄存器内容将转移到有效寄存器中并立即生效。

表 3-2. 影子寄存器

类别	可预加载寄存器
DMA 和中断使能	MTDMAINTEN / STxDMAINTEN
计数器自动重载值	MTCAR / STxCAR
重复计数值	MTCREP / STxCREP
比较值	MTCMP0V / STxCMP0V
	STxCMP0CP
	MTCMP1V / STxCMP1V
	MTCMP2V / STxCMP2V
	MTCMP3V / STxCMP3V
死区配置	STxDTCTL
	STxACTL 寄存器中的 DTFCFG[15:9]和 DTRCFG[15:9]
置位复位源	STxCH0SET
	STxCH0RST
	STxCH1SET
	STxCH1RST
计数器复位源	STxCNTRST

注意：在完成 HRTIMER 初始化配置后，建议使用一次手动软件更新，以便于将初始化的配置值更新到有效寄存器中。

3.5. 更新事件

MT 和 STx 更新事件源如*表 3-3. 更新事件源配置*所示：

表 3-3. 更新事件源配置

更新事件源	MT 控制位	STx 控制位
软件生成	MTSUP	STxSUP
重复事件	UPREP = 1	UPREP = 1
计数器复位或翻转事件	NA	UPRST = 1
来自其他定时器的更新事件	UPBMT	UPBSTX
DMA 模式结束事件	UPSEL[1:0] = 2’b01	UPSEL[3:0] = 4’b0001

更新事件源	MT 控制位	STx 控制位
DMA 模式结束事件之后的翻转事件	UPSEL[1:0] = 2'b10	UPSEL[3:0] = 4'b0010
STxUPINy(y=0..2)的上升沿产生更新事件	NA	UPSEL[3:0] = 4'b0011 \ 4'b0100 \ 4'b0101;
STxUPINy(y=0..2)的上升沿到来之后的更新事件，产生更新事件	NA	UPSEL[3:0] = 4'b0110 \ 4'b0111 4'b1000;

注意：HRTIMER_CTL0 寄存器中提供了更新事件禁能功能，且所有定时器的禁能功能都在同一寄存器中，可以同时生效。在对多个定时器或多个影子寄存器赋值前，建议将该功能统一禁能，等待赋值结束后再进行统一恢复，避免在写多个影子寄存器中间存在更新事件，造成多个影子寄存器未同时更新的情况，从而造成波形异常。具体代码实现可参考*表 3-4. 更新事件禁能及恢复*。

表 3-4. 更新事件禁能及恢复

```
/* ISR */
/* disable the slave timer 0 and 3 update event */
HRTIMER_CTL0(HRTIMER0) |= (HRTIMER_CTL0_ST3UPDIS | HRTIMER_CTL0_ST3UPDIS);
/* shadow register calculate and write in */
……
/* enable the slave timer 0 and 3 update event */
HRTIMER_CTL0(HRTIMER0) &= ~ (HRTIMER_CTL0_ST3UPDIS | HRTIMER_CTL0_ST3UPDIS);
```

3.6. 死区模式

当 STxCHOCTL 寄存器中的 DTEN=1 时，STxCTL0 寄存器中的 BLNMEN=0 时，STx 输出工作在死区模式下。

死区时间设置可参考*表 3-5. 死区时间相关寄存器配置*。

注意：建议死区时间分频系数与 HRTIMER 的计数器时钟分频保持一致。

表 3-5. 死区时间相关寄存器配置

配置	控制位	值	备注
死区使能	DTEN	1	死区模式下CH0与CH1自动互补
死区时间时钟分频系数	DTGCKDIV[3:0]	n	建议与计数器时钟分频一致
死区下降沿值	DTFCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
死区上升沿值	DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
死区下降沿值符号	DTFS	0	功率控制应用时应当为0

配置	控制位	值	备注
死区上升沿值符号	DTRS	0	

以上配置的控制位都在 STxDTCTL 和 STxACTL 寄存器中：

- 死区时间值是由 HRTIMER_STxDTCTL 寄存器中的 DTFCFG[15:0]位域和 DTRCFG[15:0]位域确定的。DTFCFG[15:0]位域定义在 O0PRE 下降沿之后的死区时间，而 DTRCFG[15:0]位域定义在 O0PRE 上升沿之后的死区时间。
- DTFCFG[8:0]和 DTRCFG[8:0]位域在 HRTIMER_STxDTCTL 寄存器中，DTFCFG[15:9]和 DTRCFG[15:9]位域在 HRTIMER_STxACTL 寄存器中。
- 死区时间时钟分频由 HRTIMER_STxDTCTL 寄存器中的 DTGCKDIV[3:0]位域定义的时钟确定。
- DTGCKDIV[2:0]位域在 HRTIMER_STxDTCTL 寄存器中，而 DTGCKDIV[3]位在 HRTIMER_STxACTL 寄存器中。

例如，想要生成 200ns 的死区时，可将死区时间时钟分频系数 DTGCKDIV[3:0]设置为 4b'1000，即 16 倍频，DTFCFG[8:0]及 DTRCFG[8:0]设置为 200ns*16*216MHz=691。

3.7. 比较值限值

每个定时器单元内部有 4 个比较器，可以灵活地产生多个比较事件，并通过比较事件完成波形的输出或采样的触发等功能。

比较值与 0 之间、比较器与周期值之间、比较器与比较器之间都不能相差的比较近，否则会丢失比较事件或周期事件，造成波形异常。

CAR 寄存器和比较值寄存器具有以下限值：

- 最小值必须大于或等于（3*tHRTIMER_CK）；
- 最大值必须小于或等于（0xFFFF －1*tHRTIMER_CK）。

以 216MHz 系统时钟、DLL 32 倍频、200kHz PWM 为例：

CAR 值应当设定为 PER = 6.912GHz/200kHz = 34560，不能超过（65535-32）；

CMP 值应当设定为 CMP 不能小于（3*32），不能超过（34560-32）。

3.8. 外部事件功能

10 个外部事件可以同时作用于 8 个 ST 中的任意 1 个。可以通过该功能实现对外部事件的快速处理，实现 CRM 控制、逐波限流及等功能。

外部事件输入源主要有 4 种，可以来自于比较器、数字输入引脚、ADC 的模拟看门狗和 TIMER_TRGO。外部事件可以配置为上升沿、下降沿或电平触发；并进行数字滤波处理。

相关寄存器配置可参考*表 3-6. 外部输入寄存器配置*。

表 3-6. 外部输入寄存器配置

配置	控制位	选项
外部事件源	EXEVySRC[1:0]	SRC0、SRC1、SRC2、SRC3
有效沿选择	EXEVyEG[1:0]	上升沿、下降沿、电平
极性选择	EXEVyP	高电平、低电平
快速模式	EXEVyFAST	低延迟模式、非低延迟模式
外部事件滤波配置	参考HRTIMER_STxEXEVFCFG0、HRTIMER_STxEXEVFCFG1、HRTIMER_STxEXEVFCFG2寄存器	

以内部比较器进行逐波限流为例，外部事件代码可参考*表 3-7. 外部事件参考代码*。

表 3-7. 外部事件参考代码

```
/* extern event config: EXEVENT5 SRC1 source is comparator1. High level trigger */
hrtimer_exeventcfg_struct_para_init(&exevcfg_para);
exevcfg_para.digital_filter = 0x0;
exevcfg_para.edge = HRTIMER_EXEV_EDGE_LEVEL;
exevcfg_para.polarity = HRTIMER_EXEV_POLARITY_HIGH;
exevcfg_para.source = HRTIMER_EXEV_SRC1;
exevcfg_para.fast = HRTIMER_EXEV_FAST_ENABLE;
hrtimer_exevent_config(HRTIMER0, HRTIMER_EXEVENT_5, &exevcfg_para);
/* extern event filter config */
hrtimer_exevfilter_struct_para_init(&exevfilter_para);
exevfilter_para.filter_mode = HRTIMER_EXEVFILTER_DISABLE;
exevfilter_para.memorized = HRTIMER_EXEVMEMORIZED_DISABLE;
hrtimer_slavetimer_exevent_filtering_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIMER_
EXEVENT_5, &exevfilter_para);
/* configures EXEV5 to reset PWM */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = (HRTIMER_CHANNEL_RESET_CMP0 | HRTIMER_CHANNEL_RE
SET_EXEV5);
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIME
R_ST0_CH0, &outcfg_para);
```

3.9. 故障保护功能

每一个 ST 都具有故障保护机制，可以通过该功能实现对故障输入的快速处理，实现过压保护、短路保护、逐波限流等功能。

故障输入源主要来自于外部事件、外部输入引脚和内部比较器的输出。每个 STy 的故障输入源的选择由各自的 FLTyINSRC 位决定。

当 FLTyINSRC = 00 时，可将外部比较器输出的保护触发信号接到外部故障输入引脚；

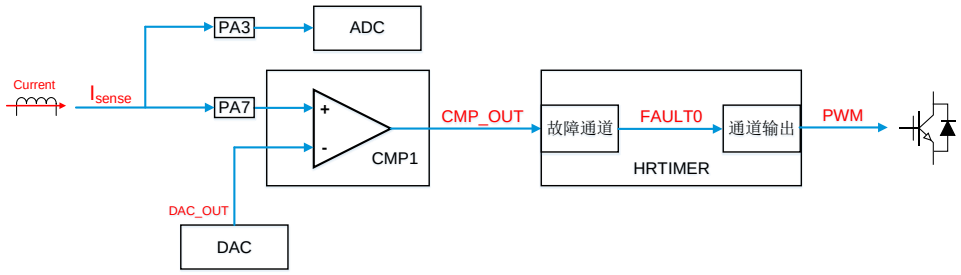
当 FLTyINSRC = 01 时，MCU 内部比较器的输出会在内部直接连到故障输入通道，此时内部

比较器的正向端接物理量，负向端可接 DAC 的输出；

当 FLTyINSRC = 10 时，HRTIMER 的外部事件将会在内部直接连到故障输入通道，其中外部事件的触发可来自于外部事件输入引脚、内部比较器输出、内部其他模块信号等。

如*图 3-1. 故障保护功能（故障源为内部比较器）*所示，DAC 输出保护阈值，与实际物理量采样值在片内比较器中进行比较，并将比较器输出连接至故障通道，响应故障，从而输出故障状态（如封波等）。

图 3-1. 故障保护功能（故障源为内部比较器）



以内部比较器进行故障保护为例，故障输入的代码可参考*表 3-8. 故障输入参考代码*。

表 3-8. 故障输入参考代码

```
/* enable fault input channel 0: fault source is from internal comparator output */
hrtimer_faultcfg_struct_para_init(&faultcfg_para);
faultcfg_para.control = HRTIMER_FAULT_CHANNEL_ENABLE;
faultcfg_para.filter = 0x0;
faultcfg_para.polarity = HRTIMER_FAULT_POLARITY_LOW;
faultcfg_para.protect = HRTIMER_FAULT_PROTECT_ENABLE;
faultcfg_para.source = HRTIMER_FAULT_SOURCE_INTERNAL;
hrtimer_fault_config(HRTIMER0, HRTIMER_FAULT_0, &faultcfg_para);
/* fault input channel 0 will effect ST2 */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.fault_enable = HRTIMER_STXFAULTENABLE_FAULT0;
timercfg_para.fault_protect = HRTIMER_STXFAULT_PROTECT_READWRITE;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER2, &timercfg_para);
```

更多有关故障保护及比较器模块的内容，请参考 ***AN198 GD32G5x3 系列比较器使用指南***。

3.10. ADC 触发

TRIGSEL 可以触发 ADC，10 个独立的触发（HRTIMER_ADCTRIGSx (x=0..3) 和 HRTIMER_ADCTRIGSxA (x=0..3)和 HRTIMER_ADCEXTTRG、HRTIMER_ADCEXTTRGA 中的 HRTIMER_ADCTRIG0 - HRTIMER_ADCTRIG9）可用于触发 ADC 的常规组。

以 ST0 的 CMP1 触发 ADC0 规则组采样的参考代码，可参考*表 3-9. ADC 触发参考代码*。

表 3-9. ADC 触发参考代码

```
/* PWM trigger config */
```

```
Hrtimer_trigger_adc0_config(void)
{……
/* ADC trigger source: HRTIMER_ADCTRIG0, trigger event is selected as ST0CMP1*/
hrtimer_adctrigcfg_struct_para_init(&triggercfg_para);
triggercfg_para.update_source = HRTIMER_ADCTRGI_UPDATE_ST0;
triggercfg_para.trigger0_3[0] = HRTIMER_ADCTRGI02_EVENT_ST0CMP1;
triggercfg_para.trigger0_3[1] = HRTIMER_ADCTRGI02_EVENT_NONE;
hrtimer_adc_trigger0_3_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg_para);
……}
/* ADC0 trigger config */
adc0_trigger_config(void)
{……
/* ADC0 routine channel triggered by external trigger */
adc_external_trigger_config(ADC0, ADC_ROUTINE_CHANNEL, EXTERNAL_TRIGGER_RISIN
G);
……}

/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ADCTRIG0 (0x7D) */
trigsel_init(TRIGSEL_OUTPUT_ADC0_ROUTRG, TRIGSEL_INPUT_HRTIMER_ADC_TRIG0);
……}
```

3.11. 双通道触发

当 HRTIMER_STxCTL1 寄存器中的 TRIGEN 位置 1 时，可启用 HRTIMER 的双通道触发模式。通过 HRTIMER_STxCTL1 寄存器中的 TRIG0M 和 TRIG1M 配置 Slave_TIMERx 输出的触发信号 TRIG0 和 TRIG1 连接到 TRIGSEL（在 TRIGSEL 模块分别为 HRTIMER_STx_TRIG0 和 HRTIMER_STx_TRIG1），可通过配置 HRTIMER_STxCTL1 寄存器中的 TRIG0M 位和 TRIG1M 位，选择触发信号源。

TRIG0 信号通过 TRIG0M 进行配置。

- TRIG0M = 0：比较 1 事件生成 TRIG0 触发信号
- TRIG0M = 1：输出 0 复位事件生成 TRIG0 触发信号

TRIG1 信号通过 TRIG1M 进行配置。

- TRIG1M = 0：计数器复位或翻转事件生成 TRIG1 触发信号
- TRIG1M = 1：输出 0 置位事件生成 TRIG1 触发信号

双通道触发模式可用于斜波补偿和迟滞控制等应用场景。

以 ST0 的双通道触发 DAC 产生锯齿波为例，配置代码参考*表 3-10. 双通道触发参考代码*。

表 3-10. 双通道触发参考代码

```
/* PWM trigger config */
Hrtimer_trigger_dac_config(void)
{……
/* DAC trigger source: TRIG0M = 0, TRIG1M = 0*/
hrtimer_double_channel_struct_para_init(&two_dac_trigger_para);
two_dac_trigger_para.trigger_enable = HRTIMER_DOUBLE_TRIG_ENABLE;
two_dac_trigger_para.trigger0 = HRTIMER_DOUBLE_TRIG0_BY_CPV1;
two_dac_trigger_para.trigger1 = HRTIMER_DOUBLE_TRIG1_BY_CNT_RESET;
hrtimer_double_trigger_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &two_dac_trigger_para);
……}
/* DAC trigger config */
dac_trigger_config(void)
{……
/* DAC triggered by external trigger */
dac_sawtooth_reset_trigger_source_config(DAC0, DAC_OUT1, DAC_SAWTOOTH_RESET_TRI
GGER_EXTERNAL);
dac_sawtooth_step_trigger_source_config(DAC0, DAC_OUT1, DAC_SAWTOOTH_STEP_TRIGG
ER_EXTERNAL);
……}

/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ST0TRIG0 (0x8A), TRIGSEL_INPUT_HRTIMER_DAC_TRIG1
(0x88) */
trigsel_init(TRIGSEL_OUTPUT_DAC0_EXTRIG1, TRIGSEL_INPUT_HRTIMER_ST0_TRIG0);
trigsel_init(TRIGSEL_OUTPUT_DAC0_ST_EXTRIG1, TRIGSEL_INPUT_HRTIMER_DAC_TRIG
1);
……}
```

3.12. HRTIMER 同步

当用到多个 MT 或 STx 协同工作时，很多情况下需要做 timer 间的同步，其中主要包括同步启动和触发计数器复位。

3.12.1. 同步启动

为保持 PWM 波形的一致性，经常需要定时器之间进行同步启动。MT 和 STx 的同步启动代码如*表 3-11. 同步启动代码*所示：

表 3-11. 同步启动代码

```
/* enable counters */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_MT_COUNTER | HRTIMER_ST1_COUN
TER | HRTIMER_ST2_COUNTER | HRTIMER_ST3_COUNTER | HRTIMER_ST4_COUNTER |
 HRTIMER_ST5_COUNTER | HRTIMER_ST6_COUNTER | HRTIMER_ST7_COUNTER);
```

3.12.2. 事件触发计数器复位

有以下四类事件可以复位 STx 的计数器：

- STx 本身：更新事件，比较 1 事件和比较 3 事件；
- 其他 STy（例如 x = 1，则 y = 0，2..7）：比较 0 事件，比较 1 事件和比较 3 事件；
- MT：比较 0 事件，比较 1 事件，比较 2 事件，比较 3 事件和复位事件；
- 外部事件 y（y = 0..9）：EXEVy 为 STx 中的外部事件的滤波信号。

该功能在做交错拓扑、倍频采样及移相控制时，会有广泛应用。

以 ST1 为例，使用 MT 的比较 0 事件对 ST1 进行复位，相关代码配置如*表 3-12. CNTRST 功能代码*所示：

表 3-12. CNTRST 功能代码

```
/* timer counter reset */
HRTIMER_STXCNTRST(HRTIMER0, HRTIMER_SLAVE_TIMER1) = HRTIMER_STXCNT_RES
ET_MASTER_CMP0;
```

3.13. 单个 PWM 波生成

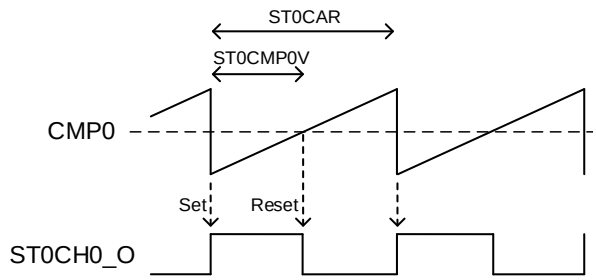
本案例配置的重点是：

- 定时器连续模式
- 置位复位事件配置
- 输出级使能

PWM 信号是大多数功率变换器的基本组成部分，也用于许多其他领域目的，如电机、蜂鸣器、LED 等。

如*图 3-2. 单个 PWM 生成*所示，是在 ST0CH0 输出上产生一个 100kHz 占空比为 50%的 PWM 信号。

图 3-2. 单个 PWM 生成



以 ST0 为例，可参考*表 3-13. 单个 PWM 具体配置*，具体配置如下：

- 定时器 ST0 配置为连续模式；
- 计数器时钟分频：设置为 16 倍频；
- 计数器自动重载寄存器 STxCAR 写入周期值：(216MHz x 16)/100kHz = 34560；
- 比较 0 寄存器 STxCMP0V 写入占空比值：0.5 x 34560 = 17280；
- 置位请求：周期事件；
- 复位请求：比较器 0 事件；
- 使能 ST0 的 CH0_O 输出通道：CHOUTEN 寄存器的 ST0CH0EN 位置 1；
- 开启 ST0 计数：MTCTL0 寄存器的 ST0CEN 位置 1。

表 3-13. 单个 PWM 具体配置

配置	控制位	值	备注
工作模式	CTNM	连续模式	
计数器时钟分频	CNTCKDIV[2:0]	1	16倍频
自动重载寄存器	STxCAR	(216MHz x 16)/100kHz = 34560	周期值
比较0寄存器	STxCMP0V	0.5 x 34560 = 17280	占空比
重复计数器	STxCREP	0	
置位请求	CH0SPER	周期事件	
复位请求	CH0RSCMP0	比较器0事件	
通道输出使能	ST0CH0EN	使能	
计数器使能	ST0CEN	使能	

参考代码如*表 3-14. PWM 波生成参考代码*：

表 3-14. PWM 波生成参考代码

```
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16;
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 17280U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);
```

```
/* enable output channel */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0);
/* enable a counter */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER);
```

3.14. 互补带死区 PWM 波生成

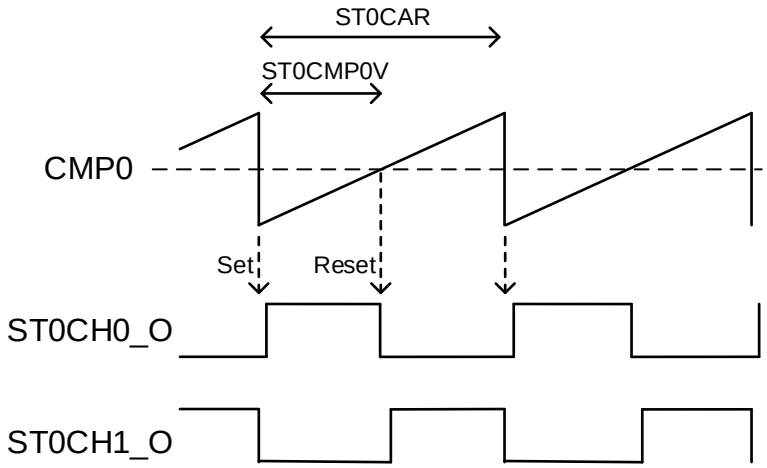
本案例配置的重点是：

- 互补通道使能
- 死区时间配置

功率控制中，上下桥臂的开关管需要插入死区，避免短时间直通，这时候就需要 HRTIMER 发出互补带死区的 PWM 波信号

如*图 3-3. 互补带死区 PWM 生成*所示，是在 ST0CH0 和 ST0CH1 上产生一个 100kHz 占空比为 50%的互补 PWM 信号，死区设定为 200ns。

图 3-3. 互补带死区 PWM 生成



以 ST 为例，可参考*表 3-15. 互补带死区 PWM 具体配置*，具体配置如下：

- 定时器 ST0 配置为连续模式；
- 计数器时钟分频：设置为 16 倍频；
- 计数器自动重载寄存器 STxCAR 写入周期值：(216MHz x 16)/100kHz = 34560；
- 死区使能：DTEN=1；
- 死区下降沿值、上升沿值：200ns*16*216MHz = 691；
- 比较 0 寄存器 STxCMP0V 写入占空比值：0.5 x 34560 = 17280；
- 置位请求：周期事件；
- 复位请求：比较器 0 事件；
- 使能 ST0 的 CH0 和 CH1 输出通道：CHOUTEN 寄存器的 ST0CH0EN 位和 ST0CH1EN 位置 1；
- 开启 ST0 计数：MTCTL0 寄存器的 ST0CEN 位置 1。

表 3-15. 互补带死区 PWM 具体配置

配置	控制位	值	备注
工作模式	CTNM	连续模式	
计数器时钟分频	CNTCKDIV[2:0]	1	16倍频
自动重载寄存器	STxCAR	(216MHz x 16)/100kHz = 34560	周期值
死区使能	DTEN	1	死区使能后CH0和CH1自动输出互补波形
死区下降沿值、上升沿值	DTFCFG[8:0]/ DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
比较0寄存器	STxCMP0V	0.5 x 34560 = 17280	占空比
重复计数器	STxCREP	0	
置位请求	CH0SPER	周期事件	
复位请求	CH0RSCMP0	比较器0事件	
通道输出使能	STxCH0/1EN	使能	
计数器使能	ST0CEN	使能	

参考代码如*表 3-16. 互补 PWM 波生成参考代码*:

表 3-16. 互补 PWM 波生成参考代码

```
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16;
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 17280U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
/* enable deadtime */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_ENABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
/* configures the deadtime mode :deadtime=200ns */
hrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL16;
deadtimecfg_para.rising_value = 691;
deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;
deadtimecfg_para.falling_value = 691;
deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &deadtimecfg_par
```

```
a);
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);
/* enable output channel */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0|HRTIMER_ST0_CH1);
/* enable a counter */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER);
```

4. Buck 电路示例

4.1. 配置重点

本示例配置的重点是：

- 计数器复位事件中断
- ADC 触发（中点电流采样）
- 计数器复位事件更新

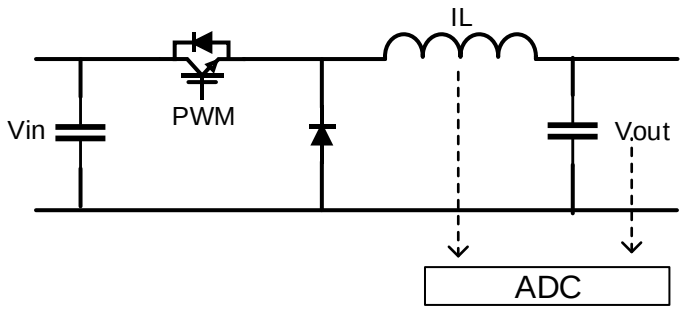
4.2. 拓扑结构

降压转换器主要用于降压电压转换，通常工作在固定频率情况下，通过调节占空比进行功率控制。

为保持 PWM 间的一致性，在理想的 Buck 变换器中，V_{in}/V_{out} 比值仅取决于 PWM 的占空比 D，即 Vout = D × Vin。

Buck 拓扑结构如*图 4-1. Buck 拓扑结构*：

图 4-1. Buck 拓扑结构



4.3. 具体配置

以 ST0 为例，可参考*表 4-1. Buck 电路 PWM 配置*，具体配置如下：

- 定时器 ST0 配置为连续模式；
- 计数器自动重载寄存器 STxCAR 写入周期值：与 buck 电路开关周期对应；
- 比较 0 寄存器 STxCMP0V 写入占空比值：占空比值在周期事件中断赋值，周期处更新；
- 重复计数器：0；
- 置位事件：周期事件；
- 复位事件：比较器 0 事件；
- ADC 触发：比较器 1 事件，环路运算时使 CMP1 等于最新的 CMP0/2；
- 中断事件：计数器复位事件，控制位为 STxDMAINTEN 中的 RSTIE 位；

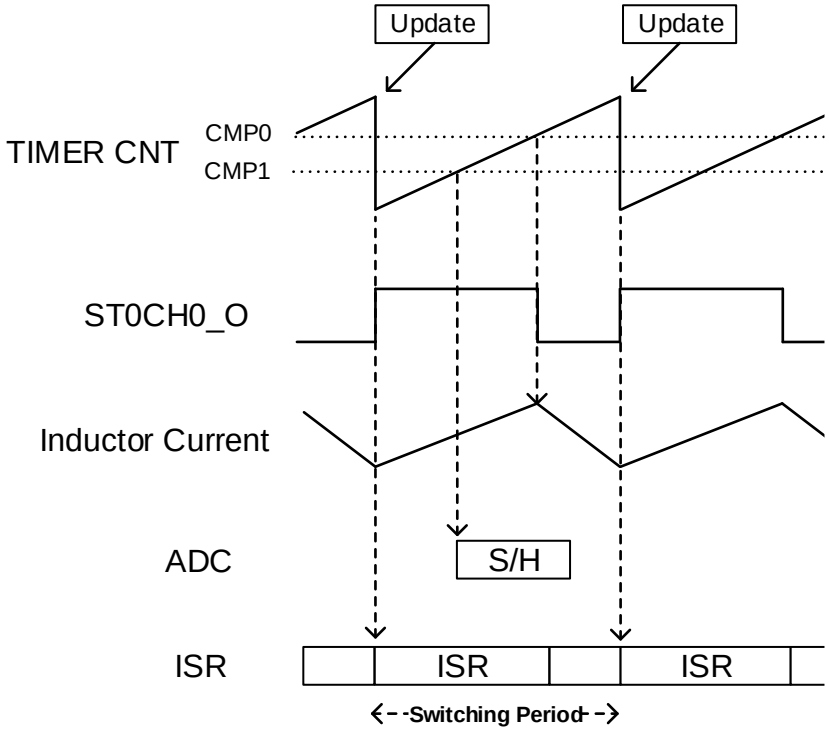
- 中断号：ST0 对应的中断号为 HRTIMER_IRQ1，在 HRTIMER_IRQ1_IRQHandler 中进行环路运算与赋值；
- 影子寄存器：使能；
- 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPRST 位，即新占空比值将在周期处更新；
- 使能 ST0 的 CH0_O 输出通道：CHOUTEN 寄存器的 ST0CH0EN 位置 1；
- 开启 ST0 计数：MTCTL0 寄存器的 ST0CEN 位置 1。

表 4-1. Buck 电路 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	0	
自动重载寄存器	STxCAR	对应周期值	在ISR中赋值，更新事件（周期）处更新
比较0寄存器	STxCMP0V	对应占空比值	在ISR中赋值，更新事件（周期）处更新
比较1寄存器	STxCMP1V	CMP1=CMP0 / 2	触发ADC采样； 在ISR中赋值，更新事件（周期）处更新
置位请求	CH0SPER	周期事件	
复位请求	CH0RSCMP0	比较器0事件	
中断请求	RSTIE	计数器复位事件	ST0对应的中断号为HRTIMER_IRQ1
影子寄存器	SHWEN	使能	
更新事件	UPRST	计数器复位事件（周期）	
通道输出使能	STxCH0\1EN = 1	使能	
计数器使能	STxCEN = 1	使能	

如*图 4-2. Buck 电路 PWM 生成*所示，PER 和 CMP0 用于 PWM 的生成，而 CMP1 用于触发 ADC 采样，ADC 配置为在输出的导通时间的中点处触发采样转换。在周期事件时，产生中断请求，并在中断服务程序中进行环路运算，环路运算会得出新的占空比和新的采样点（占空比的一半），写入到影子寄存器，并在下一周期处进行更新。

图 4-2. Buck 电路 PWM 生成



通过实时更新 CMP1 的值，这样就能实现中点电流/中点电压的采样，并使得采样点避开开关管导通关断时产生的震荡，保证采样的准确性。同时，在 DCM 模式下，可以通过动态修改 CMP1 的值，补偿平均电流。

如果 PWM 频率过高或环路运算耗时过长时，可以将重复计数器设置为 n，中断请求改为重复事件，这样将会在每 n+1 个开关周期时才进 1 次中断，进行 1 次控制环路的计算和控制量的更新。

同时，在 ADC 和 TRIGSEL 模块需要进行相关 ADC 的触发配置：

PWM 配置：触发源选定为 ADCTRIG0，触发事件选定为 ST0CMP1

TRIGSEL 配置：触发输入选择 HRTIMER_ADCTRIG0（0x7D）。

ADC0 触发配置：外部触发 ADC0 规则组；ADC0 触发比较器 1 事件，环路运算时使 CMP1 等于最新算出的的 CMP0 值的一半；

4.4. 参考代码

以 216MHz 主频、开关频率 100kHz、控制频率 100kHz 为例，参考代码如*表 4-2. Buck 电路配置代码*:

表 4-2. Buck 电路配置代码

```
/* PWM wave config */
hrtimer_config(void)
{
```

```
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16;
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/* configure shadow enable */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);
/* config counter reset event as upadate event */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_DISABLED;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_ENABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
/* set counter reset event as main loop interrupt:*/
hrtimer_timers_interrupt_flag_clear(HRTIMER0,HRTIMER_SLAVE_TIMER0,HRTIMER_ST_INT_F
LAG_CNTRST);
hrtimer_timers_interrupt_enable(HRTIMER0,HRTIMER_SLAVE_TIMER0,  HRTIMER_ST_INT_CN
TRST);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 20736U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
/* init sample point value */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = 10368U;
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);
/* software update to ensure that the initial configuration takes effect before the timer start */
HRTIMER_CTL1(HRTIMER0) |= (HRTIMER_CTL1_ST0SUP);
/* enable output channel */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0);
/* enable a counter */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER);
}
```

```
/* ISR: counter reset event or repetition event   */
void HRTIMER_IRQ1_IRQHandler(void)
{
……
/* Sampling value processing and loop calculation */
……
/* Update the value of CMP0 per control cycle：CMP0 = duty*baseinit_para.period */
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = cmp0;
/* Update the value of CMP1 per control cycle：CMP1 = CMP0/2 */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = cmp0/2U;
……
}

/* PWM trigger config */
Hrtimer_trigger_adc0_config(void)
{……
/* ADC trigger source: HRTIMER_ADCTRIG0, trigger event is selected as ST0CMP1*/
hrtimer_adctrigcfg_struct_para_init(&triggercfg_para);
triggercfg_para.update_source = HRTIMER_ADCTRGI_UPDATE_ST0;
triggercfg_para.trigger0_3[0] = HRTIMER_ADCTRGI02_EVENT_ST0CMP1;
triggercfg_para.trigger0_3[1] = HRTIMER_ADCTRGI02_EVENT_NONE;
hrtimer_adc_trigger0_3_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg_para);
……}

/* ADC0 trigger config */
adc0_trigger_config(void)
{……
/* ADC0 routine channel triggered by external trigger */
adc_external_trigger_config(ADC0, ADC_ROUTINE_CHANNEL, EXTERNAL_TRIGGER_RISING);
……}

/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ADCTRIG0 (0x7D) */
trigsel_init(TRIGSEL_OUTPUT_ADC0_ROUTRG, TRIGSEL_INPUT_HRTIMER_ADC_TRIG0);
……}
```

5. 同步 Buck 电路示例

5.1. 配置重点

本示例配置的重点是：

- 计数器复位事件中断
- 死区时间配置
- 计数器复位事件更新

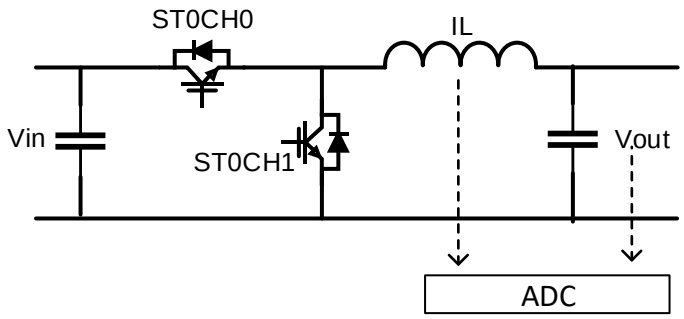
5.2. 拓扑结构

与 buck 电路相比，同步整流 buck 电路使用开关管替代了续流二极管，从而有效减少在 buck 电路中续流二极管带来的导通损耗。其主要用于降压电压转换，通常工作在固定频率情况下，通过调节占空比进行功率控制。

在理想的 Buck 变换器中，V_{in}/V_{out} 比值仅取决于 PWM 的占空比 D，即 Vout = D × Vin。

Buck 拓扑结构如*图 5-1. 同步 buck 拓扑结构*：

图 5-1. 同步 buck 拓扑结构



5.3. 具体配置

以 ST0 为例，可参考*表 5-1. 同步 buck 电路 PWM 配置*，具体配置如下：

- 定时器 ST0 配置为连续模式；
- 计数器自动重载寄存器 STxCAR 写入周期值：与 buck 电路开关周期对应；
- 比较 0 寄存器 STxCMP0V 写入占空比值：占空比值在周期事件中断赋值，周期处更新；
- 重复计数器：0；
- 置位事件：周期事件；
- 复位事件：比较器 0 事件；
- 死区使能：DTEN=1；
- 死区下降沿值、上升沿值：实际死区时间*HRTIMER_DTGCK；

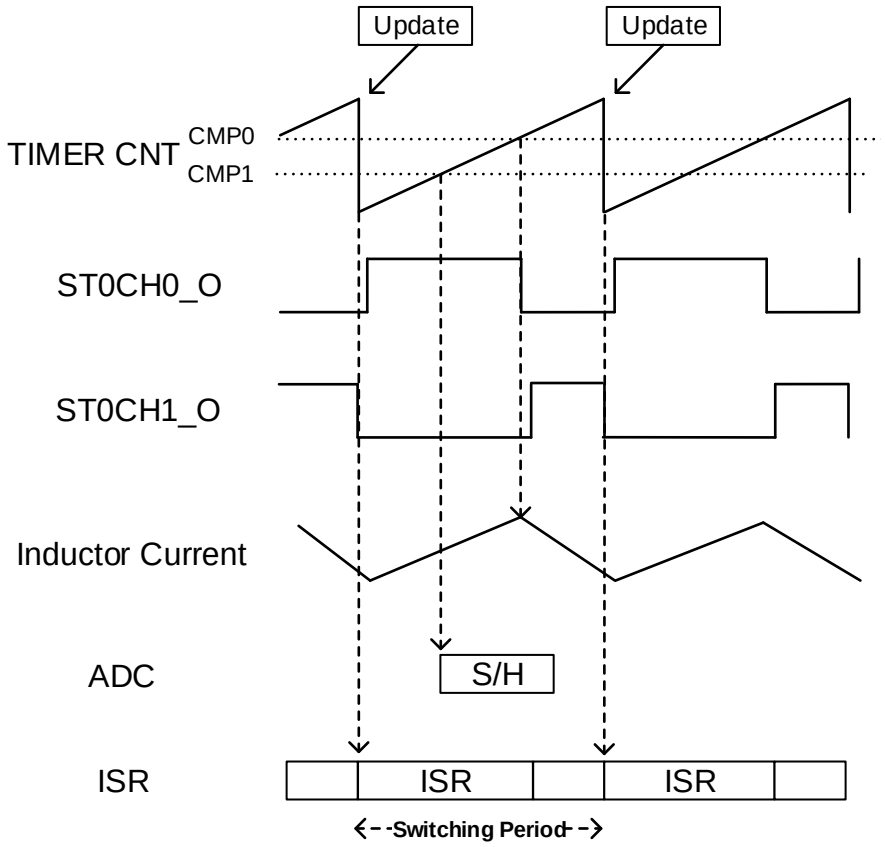
- ADC 触发：比较器 1 事件，环路运算时使 CMP1 等于最新的 CMP0/2；
- 中断事件：计数器复位事件，控制位为 STxDMAINTEN 中的 RSTIE 位；
- 中断号：ST0 对应的中断号为 HRTIMER_IRQ1，在 HRTIMER_IRQ1_IRQHandler 中进行环路运算与赋值；
- 影子寄存器：使能；
- 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPRST 位，即新占空比值将在周期处更新；
- 使能 ST0 的 CH0 和 CH1 输出通道：CHOUTEN 寄存器的 ST0CH0EN 和 ST0CH1EN 位置 1；
- 开启 ST0 计数：MTCTL0 寄存器的 ST0CEN 位置 1。

表 5-1. 同步 buck 电路 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	0	
自动重载寄存器	STxCAR	对应周期值	在ISR中赋值，更新事件（周期）处更新
比较0寄存器	STxCMP0V	对应占空比值	在ISR中赋值，更新事件（周期）处更新
比较1寄存器	STxCMP1V	CMP1=CMP0 / 2	触发ADC采样； 在ISR中赋值，更新事件（周期）处更新
置位请求	CH0SPER	周期事件	
复位请求	CH0RSCMP0	比较器0事件	
死区使能	DTEN	1	死区使能后CH0和CH1自动输出互补波形
死区下降沿值、上升沿值	DTFCFG[8:0]/ DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
中断请求	RSTIE	计数器复位事件	ST0对应的中断号为HRTIMER_IRQ1
影子寄存器	SHWEN	使能	
更新事件	UPRST	计数器复位事件（周期）	
通道输出使能	STxCH0EN	使能	
计数器使能	STxCEN	使能	

如*图 5-2. 同步 buck 电路 PWM 生成*所示，PER 和 CMP0 用于 PWM 的生成，而 CMP1 用于触发 ADC 采样，ADC 配置为在输出的导通时间的中点处触发采样转换。在周期事件时，产生中断请求，并在中断服务程序中进行环路运算，环路运算会得出新的占空比和新的采样点（占空比的一半），写入到影子寄存器，并在下一周期处进行更新。

图 5-2. 同步 buck 电路 PWM 生成



通过实时更新 CMP1 的值，这样就能实现中点电流/中点电压的采样，并使得采样点避开开关管导通关断时产生的震荡，保证采样的准确性。

如果 PWM 频率过高或环路运算耗时过长时，可以将重复计数器设置为 n，中断请求改为重复事件，这样将会在每 n+1 个开关周期时才进 1 次中断，进行 1 次控制环路的计算和控制量的更新。

同时，在 ADC 和 TRIGSEL 模块需要进行相关 ADC 的触发配置：

PWM 配置：触发源选定为 ADCTRIG0，触发事件选定为 ST0CMP1

TRIGSEL 配置：触发输入选择 HRTIMER_ADCTRIG0（0x7D）。

ADC0 触发配置：外部触发 ADC0 规则组；ADC0 触发比较器 1 事件，环路运算时使 CMP1 等于最新算出的的 CMP0 值的一半；

5.4. 参考代码

以 216MHz 主频、开关频率 100kHz、控制频率 100kHz、200ns 死区时间为例，参考代码如 *表 5-2. 同步 Buck 电路配置代码*:

表 5-2. 同步 Buck 电路配置代码

```
/* PWM wave config */
```

```
hrtimer_config(void)
{
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/* configure shadow enable */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);
/* enable deadtime and config counter reset event as upadate event */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_ENABLED;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_ENABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
/* configures the deadtime mode :deadtime=200ns */
hrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL16;
deadtimecfg_para.rising_value = 691;
deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;
deadtimecfg_para.falling_value = 691;
deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &deadtimecfg_par
a);
/* set counter reset event as main loop interrupt:*/
hrtimer_timers_interrupt_flag_clear(HRTIMER0,HRTIMER_SLAVE_TIMER0,HRTIMER_ST_INT_F
LAG_CNTRST);
hrtimer_timers_interrupt_enable(HRTIMER0,HRTIMER_SLAVE_TIMER0, HRTIMER_ST_INT_CN
TRST);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 20736U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
/* init sample point value */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = 10368U;
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
```

```
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);
/* software update to ensure that the initial configuration takes effect before the timer start */
HRTIMER_CTL1(HRTIMER0) |= (HRTIMER_CTL1_ST0SUP);
/* enable output channel */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0| HRTIMER_ST0_CH1);
/* enable a counter */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER);
}

/* ISR: counter reset event or repetition event */
void HRTIMER_IRQ1_IRQHandler(void)
{
……
/* Sampling value processing and loop calculation */
……
/* Update the value of CMP0 per control cycle：CMP0 = duty*baseinit_para.period */
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = cmp0;
/* Update the value of CMP1 per control cycle：CMP1 = CMP0/2 */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = cmp0/2U;
……
}

/* PWM trigger config */
hrtimer_trigger_adc0_config(void)
{……
/* ADC trigger source: HRTIMER_ADCTRIG0, trigger event is selected as ST0CMP1*/
hrtimer_adctrigcfg_struct_para_init(&triggercfg_para);
triggercfg_para.update_source = HRTIMER_ADCTRGI_UPDATE_ST0;
triggercfg_para.trigger0_3[0] = HRTIMER_ADCTRGI02_EVENT_ST0CMP1;
triggercfg_para.trigger0_3[1] = HRTIMER_ADCTRGI02_EVENT_NONE;
hrtimer_adc_trigger0_3_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg_para);
……}

/* ADC0 trigger config */
adc0_trigger_config(void)
{……
/* ADC0 routine channel triggered by external trigger */
adc_external_trigger_config(ADC0, ADC_ROUTINE_CHANNEL, EXTERNAL_TRIGGER_RISING);
……}
```

```
/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ADCTRIG0 (0x7D) */
trigsel_init(TRIGSEL_OUTPUT_ADC0_ROUTRG, TRIGSEL_INPUT_HRTIMER_ADC_TRIG0);
……}
```

6. 多相交错 Buck 电路示例

6.1. 配置重点

多相交错并本案例配置的重点是：

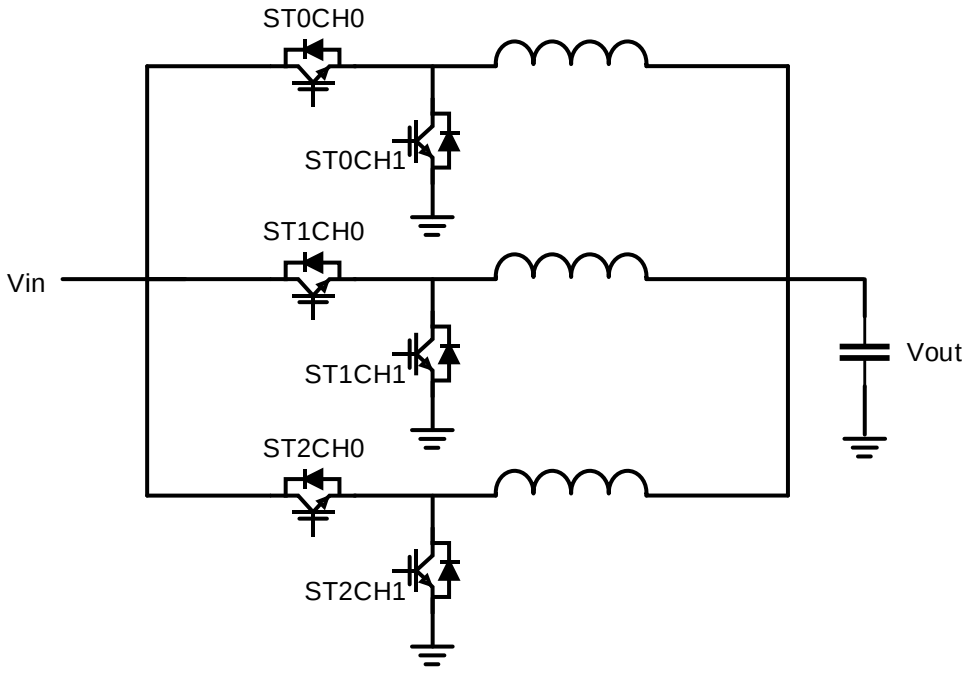
- 全局计数器复位事件中断
- 相位交错配置
- 事件触发计数器复位
- 各自计数器复位事件更新

6.2. 拓扑结构

多相交错并联技术可应用于多种功率转换拓扑(buck、boost、LLC)。具有减少纹波、降低 EMI、提升轻载效率等优点。对于多相交错 buck，一般来说，每相的相对相移等于 360° 除以相数，本节以三相为例，每相交错 120° 。

三相交错 buck 拓扑结构如*图 6-1. 三相交错 buck 拓扑结构*:

图 6-1. 三相交错 buck 拓扑结构



6.3. 具体配置

以 ST0、ST1、ST2 相差 120° 为例，相位相关配置的如*表 6-1. STx 相位配置*:

表 6-1. STx 相位配置

定时器	相位/度	计数器复位	置位请求	复位请求	更新事件	CAR值	CMP0值	CMP1值	CMP2值	CMP3值
ST0	0	周期处自动重载	周期事件	ST0 CMP0	ST0周期事件	周期值	占空比值	周期值*1/3	触发采样	周期值*2/3
ST1	120	ST0的CMP1触发	ST1 CMP1	ST1 CMP0	ST1周期事件	0xFFFF	占空比值	DLL*3	触发采样	
ST2	240	ST0的CMP3触发	ST2 CMP1	ST2 CMP0	ST2周期事件	0xFFFF	占空比值	DLL*3	触发采样	

ST0 在连续模式下工作，其发波及周期与单相 buck 发波相同。ST1 计数器由 ST0 的 CMP1 事件复位。ST2 计数器由 ST0 的 CMP2 事件复位。占空比被编程到每个定时器中，独立于相移，在操作期间可以变化。其相位差的实现主要如下：

- ST0 的 CAR 值配置为周期值，CMP0 值配置为占空比值；
- ST1 的 CAR 值配置为 0xFF，CMP0 值配置为占空比值，；
- ST2 的 CAR 值配置为 0xFF，CMP0 值配置为占空比值；
- ST0 的 CMP1 事件触发 ST1 计数器复位，从而确定 ST1 的相位差与周期值，ST0CMP1=周期值*1/3；
- ST0 的 CMP3 事件触发 ST1 计数器复位，从而确定 ST1 的相位差与周期值，ST0CMP3=周期值*2/3；
- ST0、ST1、ST2 在各自的周期事件处进行更新；
- ST0、ST1、ST2 统一在 ST0 的中断服务程序中进行环路运算；

每一相的 PWM 的具体配置如下，具体可参考*表 6-2. 多相交错 buck 电路 PWM 配置*：

- 定时器 STx 配置为连续模式；
- 计数器自动重载寄存器 STxCAR 写入周期值：与 buck 电路开关周期对应；
- 比较 0 寄存器 STxCMP0V 写入占空比值：占空比值在重复事件中断赋值，周期处更新；
- 重复计数器：0；
- 置位事件：
  ST0：周期事件；
  ST1：ST1 CMP1；
  ST2：ST2 CMP1；
- 复位事件：比较器 0 事件；
- 死区使能：DTEN=1；
- 死区下降沿值、上升沿值：实际死区时间*HRTIMER_DTGCK；
- ADC 触发：比较器 2 事件，环路运算时使 CMP2 等于最新的 CMP0/2；
- 中断事件：计数器复位事件，控制位为 STxDMAINTEN 中的 RSTIE 位；
- 中断号：ST0 对应的中断号为 HRTIMER_IRQ1，在 HRTIMER_IRQ1_IRQHandler 中进行环路运算与赋值；
- 影子寄存器：使能；
- 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPRST 位，即新占空比值将在周期处更新；
- 使能 STx 的 CH0 和 CH1 输出通道：CHOUTEN 寄存器的 STxCH0EN 和 STxCH1EN 位

置 1；

- 开启 STx 计数：MTCTL0 寄存器的 ST0CEN、ST1CEN、ST2CEN 位置 1。

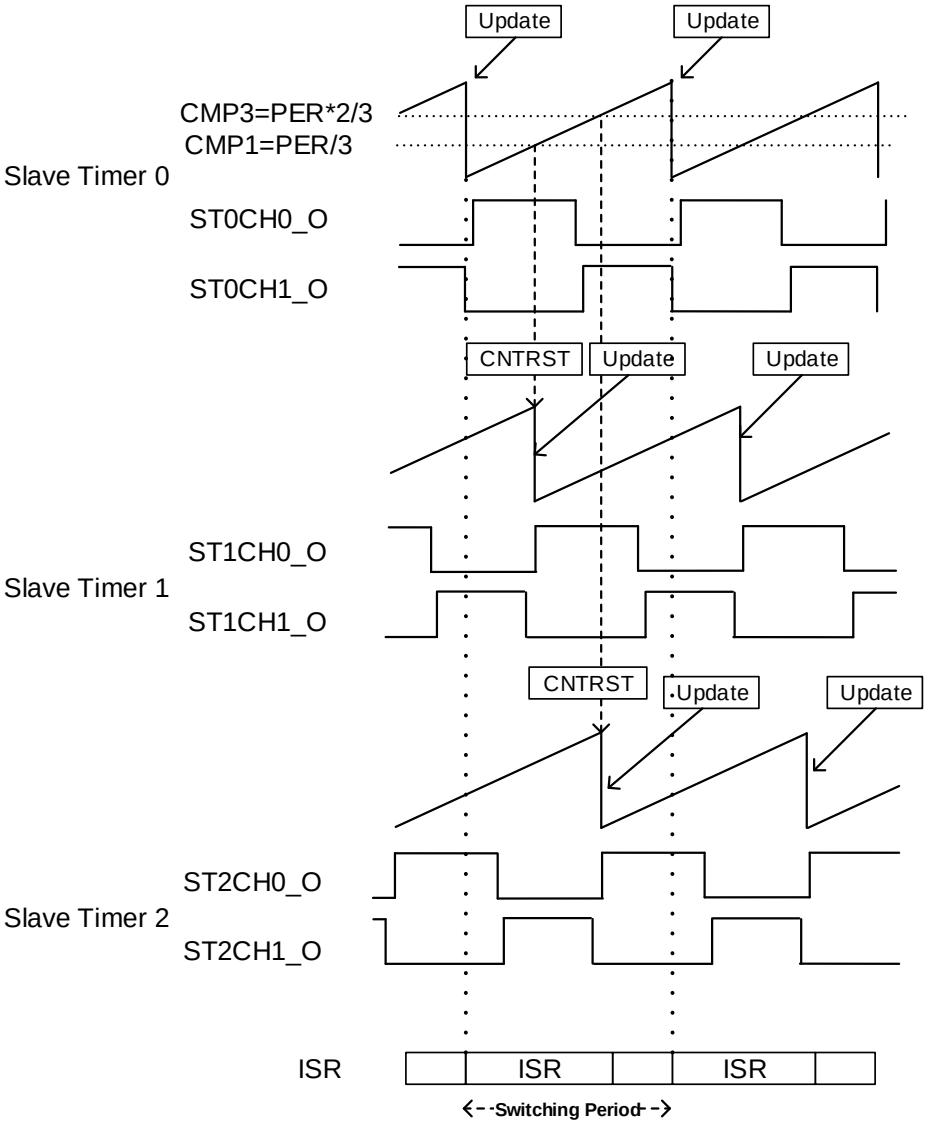
表 6-2. 多相交错 buck 电路 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	0	
自动重载寄存器	STxCAR	对应周期值	在ISR中赋值，更新事件（周期）处更新
比较0寄存器	STxCMP0V	对应占空比值	在ISR中赋值，更新事件（周期）处更新
比较1寄存器	STxCMP2V	CMP2=CMP0 / 2	触发ADC采样； 在ISR中赋值，更新事件（周期）处更新
置位请求	CH0SPER	周期事件/ ST1 CMP1/ ST2 CMP1	
复位请求	CH0RSCMP0	比较器0事件	
死区使能	DTEN	1	死区使能后CH0和CH1自动输出互补波形
死区下降沿值、上升沿值	DTFCFG[8:0]/ DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
中断请求	RSTIE	计数器复位事件	ST0对应的中断号为HRTIMER_IRQ1
影子寄存器	SHWEN	使能	
更新事件	UPRST	计数器复位事件（周期）	
通道输出使能	STxCH0\1EN	使能	
计数器使能	STxCEN	使能	

如*图 6-2. 同步 buck 电路 PWM 生成*所示，PER 和 CMP0 用于 PWM 的生成，而 CMP2 用于触发 ADC 采样，ADC 配置为在输出的导通时间的中点处触发采样转换。

在 ST0 的周期事件时，产生中断请求，并在中断服务程序中进行环路运算，环路运算会得出 ST0、ST1、ST2 的新的占空比和新的采样点（占空比的一半），写入到 ST0、ST1、ST2 的影子寄存器中，并在其各自的周期处进行更新。

图 6-2. 同步 buck 电路 PWM 生成



通过实时更新 CMP2 的值，这样就能实现中点电流/中点电压的采样，并使得采样点避开开关管导通关断时产生的震荡，保证采样的准确性。

如果 PWM 频率过高或环路运算耗时过长时，可以将重复计数器设置为 n，中断请求改为重复事件，这样将会在每 n+1 个开关周期时才进 1 次中断，进行 1 次控制环路的计算和控制量的更新。

同时，在 ADC 和 TRIGSEL 模块需要进行相关 ADC 的触发配置：

PWM 配置：触发源选定为 ADCTRIG0，触发事件选定为 ST0CMP2

TRIGSEL 配置：触发输入选择 HRTIMER_ADCTRIG0（0x7D）。

ADC0 触发配置：外部触发 ADC0 规则组；ADC0 触发比较器 1 事件，环路运算时使 CMP2 等于最新算出的的 CMP0 值的一半；

6.4. 参考代码

以 216MHz 主频、开关频率 100kHz、控制频率 100kHz、200ns 死区时间为例，参考代码如 *表 6-3. 三相交错 buck 电路配置代码*:

表 6-3. 三相交错 buck 电路配置代码

```
/* PWM wave config */
hrtimer_config(void)
{
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
baseinit_para.period = 0xFFFF;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER1, &baseinit_para);
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER2, &baseinit_para);
/* configure shadow enable */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER1, &timerinit_para);
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER2, &timerinit_para);
/* enable deadtime and config counter reset event as upadate event */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_ENABLED;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_ENABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER1, &timercfg_para);
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER2, &timercfg_para);
/* configures the deadtime mode :deadtime=200ns */
hrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL16;
deadtimecfg_para.rising_value = 691;
deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;
deadtimecfg_para.falling_value = 691;
deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &deadtimecfg_par
a);
```

```
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER1, &deadtimecfg_para);
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER2, &deadtimecfg_para);
/* set counter reset event as main loop interrupt:*/
hrtimer_timers_interrupt_flag_clear(HRTIMER0,HRTIMER_SLAVE_TIMER0,HRTIMER_ST_INT_FLAG_CNTRST);
hrtimer_timers_interrupt_enable(HRTIMER0,HRTIMER_SLAVE_TIMER0, HRTIMER_ST_INT_CNTRST);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 20736U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER1,
HRTIMER_COMPARE0, &comparecfg_para);
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER2,
HRTIMER_COMPARE0, &comparecfg_para);
/* init sample point value */
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = 10368U;
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER1) = 10368U;
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = 10368U;
/* ST0CMP1 to counter reset ST1, ST0CMP3 to counter reset ST2 */
HRTIMER_STXCNTRST(HRTIMER0, HRTIMER_SLAVE_TIMER1) = HRTIMER_STXCNT_RESET_OTHER0_CMP1);
HRTIMER_STXCNTRST(HRTIMER0, HRTIMER_SLAVE_TIMER2) = HRTIMER_STXCNT_RESET_OTHER0_CMP3);
/* generate phase difference: 0 \ 120 \ 240 */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = 34560*1/3;
HRTIMER_STXCMP3V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = 34560*2/3;
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0, HRTIMER_ST0_CH0, &outcfg_para);
outcfg_para.set_request = HRTIMER_CHANNEL_SET_ CMP1;
/* Set the value of CMP1 min = DLL*3 */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER1) = 48U;
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = 48U;

hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER1, HRTIMER_ST1_CH0, &outcfg_para);
```

```
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER2, HRTIMER_ST2_CH0, &outcfg_para);
/* software update to ensure that the initial configuration takes effect before the timer start */
HRTIMER_CTL1(HRTIMER0) |= (HRTIMER_CTL1_ST0SUP| HRTIMER_CTL1_ST0SUP| HRTIMER_CTL1_ST2SUP);
/* enable output channels synchronization */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0 | HRTIMER_ST0_CH1| HRTIMER_ST1_CH0 | HRTIMER_ST1_CH1 | HRTIMER_ST2_CH0| HRTIMER_ST2_CH1);
/* enable counters synchronization */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER | HRTIMER_ST1_COUNTER | HRTIMER_ST2_COUNTER);
}

/* ISR: counter reset event or repetition event */
void HRTIMER_IRQ1_IRQHandler(void)
{
……
/* Sampling value processing and loop calculation */
……
/* disable all the slave timer update event */
HRTIMER_CTL0(HRTIMER0) |= (HRTIMER_CTL0_ST0UPDIS|HRTIMER_CTL0_ST1UPDIS|HRTIMER_CTL0_ST2UPDIS);
/* Update the value of CMP0 per control cycle：CMP0 = duty*baseinit_para.period */
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = st0cmp0;
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER1) = st1cmp0;
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = st2cmp0;
/* Update the value of CMP2 per control cycle：CMP2 = CMP0/2 */
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = st0cmp0/2U;
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER1) = st1cmp0/2U;
HRTIMER_STXCMP2V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = st2cmp0/2U;
/* enable all the slave timer update event */
HRTIMER_CTL0(HRTIMER0) &= ~ (HRTIMER_CTL0_ST0UPDIS|HRTIMER_CTL0_ST1UPDIS|HRTIMER_CTL0_ST2UPDIS);
……
}

/* PWM trigger config */
hrtimer_trigger_adc0_config(void)
{……
/* ADC trigger source: HRTIMER_ADCTRIG0, trigger event is selected as ST0CMP2*/
hrtimer_adctrigcfg_struct_para_init(&triggercfg_para);
triggercfg_para.update_source = HRTIMER_ADCTRGI_UPDATE_ST0;
triggercfg_para.trigger0_3[0] = HRTIMER_ADCTRGI02_EVENT_ST0CMP2;
```

```
triggercfg_para.trigger0_3[1] = HRTIMER_ADCTRGI02_EVENT_NONE;
hrtimer_adc_trigger0_3_config(HRTIMER0, HRTIMER_ADCTRIG_0, &triggercfg_para);
……}

/* ADC0 trigger config */
adc0_trigger_config(void)
{……
/* ADC0 routine channel triggered by external trigger */
adc_external_trigger_config(ADC0, ADC_ROUTINE_CHANNEL, EXTERNAL_TRIGGER_RISING);
……}

/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ADCTRIG0 (0x7D) */
trigsel_init(TRIGSEL_OUTPUT_ADC0_ROUTRG, TRIGSEL_INPUT_HRTIMER_ADC_TRIG0);
……}
```

7. LLC 电路示例

7.1. 配置重点

本示例配置的重点是：

- 重复事件中断
- 变频控制
- 计数器复位事件更新

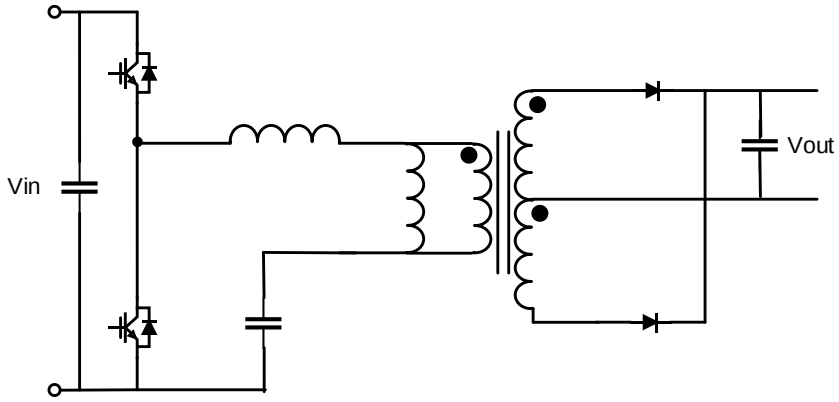
7.2. 拓扑结构

LLC 电路是近年来工业上非常流行的电路拓扑，由于其软开关特性，使其具有高频、高效率等特点。广泛应用于服务器电源、便携式储能、手机快充、车充等场景中。

LLC 控制方式有多种，但通常工作在固定 50%占空比、固定相位情况下，通过调节频率进行功率控制。

以半桥 LLC 为例，其拓扑结构如*图 7-1. LLC 拓扑结构*：

图 7-1. LLC 拓扑结构



7.3. 具体配置

以 ST0 为例，PWM 具体配置如下，具体可参考*表 7-1. LLC 电路 PWM 配置*：

- 定时器 ST0 配置为连续模式；
- 计数器自动重载寄存器 STxCAR 写入周期值：与 LLC 电路开关周期对应；
- 比较 0 寄存器 STxCMP0V 写入占空比值：占空比值在重复事件中断赋值，周期处更新；
- 重复计数器：2；
- 置位事件：周期事件；
- 复位事件：比较器 0 事件，50%周期值；
- 死区使能：DTEN=1；

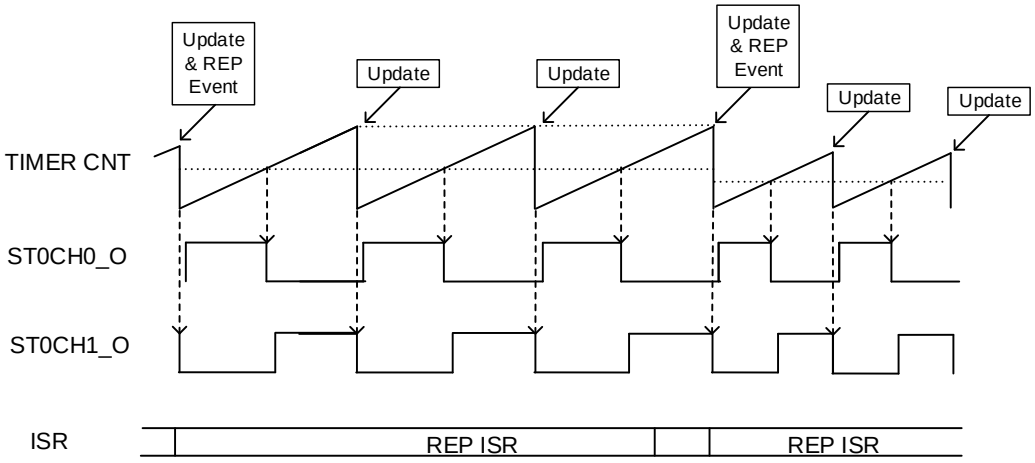
- 死区下降沿值、上升沿值：实际死区时间*HRTIMER_DTGCK；
- 中断事件：重复事件，控制位为 STxDMAINTEN 中的 REPIE 位；
- 中断号：ST0 对应的中断号为 HRTIMER_IRQ1，在 HRTIMER_IRQ1_IRQHandler 中进行环路运算与赋值；
- 影子寄存器：使能；
- 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPRST 位，即新占空比值将在周期处更新；
- 使能 ST0 的 CH0 和 CH1 输出通道：CHOUTEN 寄存器的 ST0CH0EN 和 ST0CH1EN 位置 1；
- 开启 ST0 计数：MTCTL0 寄存器的 ST0CEN 位置 1。

表 7-1. LLC 电路 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	2	
自动重载寄存器	STxCAR	周期值	在ISR中赋值，更新事件（周期）处更新
比较0寄存器	STxCMP0V	CMP0=PER / 2	在ISR中赋值，更新事件（周期）处更新
置位请求	CH0SPER	周期事件	
复位请求	CH0RSCMP0	比较器0事件	
死区使能	DTEN	1	死区使能后CH0和CH1自动输出互补波形
死区下降沿值、上升沿值	DTFCFG[8:0]/ DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
中断请求	REPIE	重复事件	ST0对应的中断号为HRTIMER_IRQ1
影子寄存器	SHWEN	使能	
更新事件	UPRST	计数器复位事件（周期）	
通道输出使能	STxCH0EN	使能	
计数器使能	STxCEN	使能	

如*图 7-2. LLC 电路 PWM 生成*所示，PER 和 CMP0 用于 PWM 的生成。在重复事件时，产生中断请求，并在中断服务程序中进行环路运算，环路运算会得出新的周期值和 CMP0 值（半个周期），写入到影子寄存器，并在下一周期处进行更新。

图 7-2. LLC 电路 PWM 生成



通过每 3 个周期计算一次环路，并同时更新一次 PER 和 CMP0 的值，可以保证 LLC 的调频控制的实现。

如果开关频率过高或环路运算耗时过长时，可以将重复计数器设置得更大，例如为 n，这样将会在每 n+1 个开关周期时才进 1 次中断，进行 1 次控制环路的计算和控制量的更新。

7.4. 参考代码

以 216MHz 主频、开关频率 100kHz、200ns 死区时间为例，参考代码如*表 7-2. LLC 电路配置代码*:

表 7-2. LLC 电路配置代码

```
/* PWM wave config */
hrtimer_config(void)
{
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16
baseinit_para.repetitioncounter = 2U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/* configure shadow enable , config Rep event as upadate event */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_ENABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);
/* enable deadtime */
hrtimer_timercfg_struct_para_init(&timercfg_para);
```

```
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_ENABLED;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_DISABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
/* configures the deadtime mode :deadtime=200ns */
hrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL16;
deadtimecfg_para.rising_value = 691;
deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;
deadtimecfg_para.falling_value = 691;
deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;
hrtimer_slavetimer_deadtime_config(HRTIMER0,  HRTIMER_SLAVE_TIMER0,  &deadtimecfg_par
a);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 17280U;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_COMPARE0, &comparecfg_para);
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP0;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER0,
HRTIMER_ST0_CH0, &outcfg_para);
/* software update to ensure that the initial configuration takes effect before the timer start */
HRTIMER_CTL1(HRTIMER0) |= (HRTIMER_CTL1_ST0SUP);
/* enable output channel */
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0| HRTIMER_ST0_CH1);
/* enable a counter */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER);
}

/* ISR: repetition event   */
void HRTIMER_IRQ1_IRQHandler(void)
{
……
/* Sampling value processing and loop calculation */
……
/* Update the value of PER per control cycle：PER = period */
HRTIMER_STXCAR(HRTIMER0, HRTIMER_SLAVE_TIMER0) = new_period;
/* Update the value of CMP0 per control cycle：CMP0 = period / 2 */
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER0) = new_period / 2;
……
}
```

8. H 桥逆变电路示例

8.1. 配置重点

本示例配置的重点是：

- 计数器复位事件中断
- SPWM 调制
- 同步启动
- 计数器复位事件更新

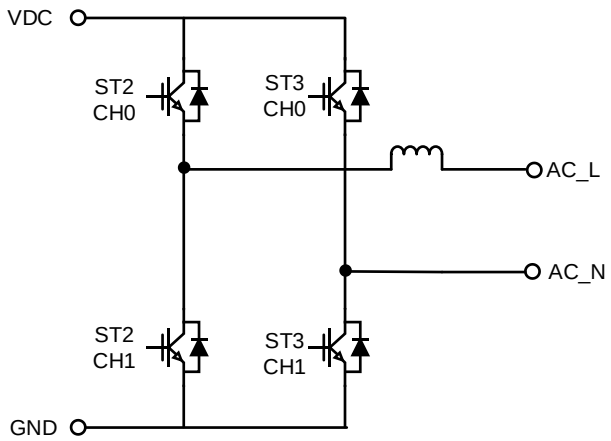
8.2. 拓扑结构

H 桥逆变是单相逆变器的常用拓扑，广泛应用于双向储能、UPS 等场景中。

H 桥逆变通常采用 SPWM 的调制方式。常用的 SPWM 调制方式有单极性调制和双极性调制方式。本示例采用单极性 SPWM 调制的方式发波。

以 H 桥为例，其拓扑结构如*图 8-1. H 桥逆变拓扑结构*：

图 8-1. H 桥逆变拓扑结构



发波方式如下：正半工频周期时，ST2 的 CH0 和 CH1 在高频（开关频率）下互补，ST3 的 CH0 和 CH1 在低频(50Hz)下互补；负半工频周期时，ST3 的 CH0 和 CH1 在高频下开关，ST2 的 CH0 和 CH1 在低频(50Hz)下开关。

8.3. 具体配置

以上述发波方式为例，ST2 和 ST3 的 PWM 配置如下，具体可参考*表 8-1. H 桥逆变电路 PWM 配置*：

- 定时器 ST2、ST3 配置为连续模式；
- 计数器自动重载寄存器 STxCAR 写入周期值：与开关周期对应；
- 比较 0 寄存器 STxCMP0V 写入(PER – duty)/2；
- 比较 1 寄存器 STxCMP2V 写入(PER + duty)/2；
- 重复计数器：0；
- 置位事件：比较器 0 事件；
- 复位事件：比较器 1 事件；
- 死区使能：DTEN=1；
- 死区下降沿值、上升沿值：实际死区时间*HRTIMER_DTGCK；
- 中断事件：计数器复位事件，控制位为 STxDMAINTEN 中的 RSTIE 位；
- 中断号：ST2 对应的中断号为 HRTIMER_IRQ3，在 HRTIMER_IRQ3_IRQHandler 中进行环路运算与赋值；
- 影子寄存器：使能；
- 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPRST 位，即新占空比值将在周期处更新；
- 使能 ST2、ST3 的 CH0、CH1 输出通道：CHOUTEN 寄存器的 ST2CH0EN、ST2CH1EN、ST3CH0EN、ST3CH1EN 位置 1；
- 开启 ST2 和 ST3 计数：MTCTL0 寄存器的 ST2CEN、ST3CEN 位同时置 1。

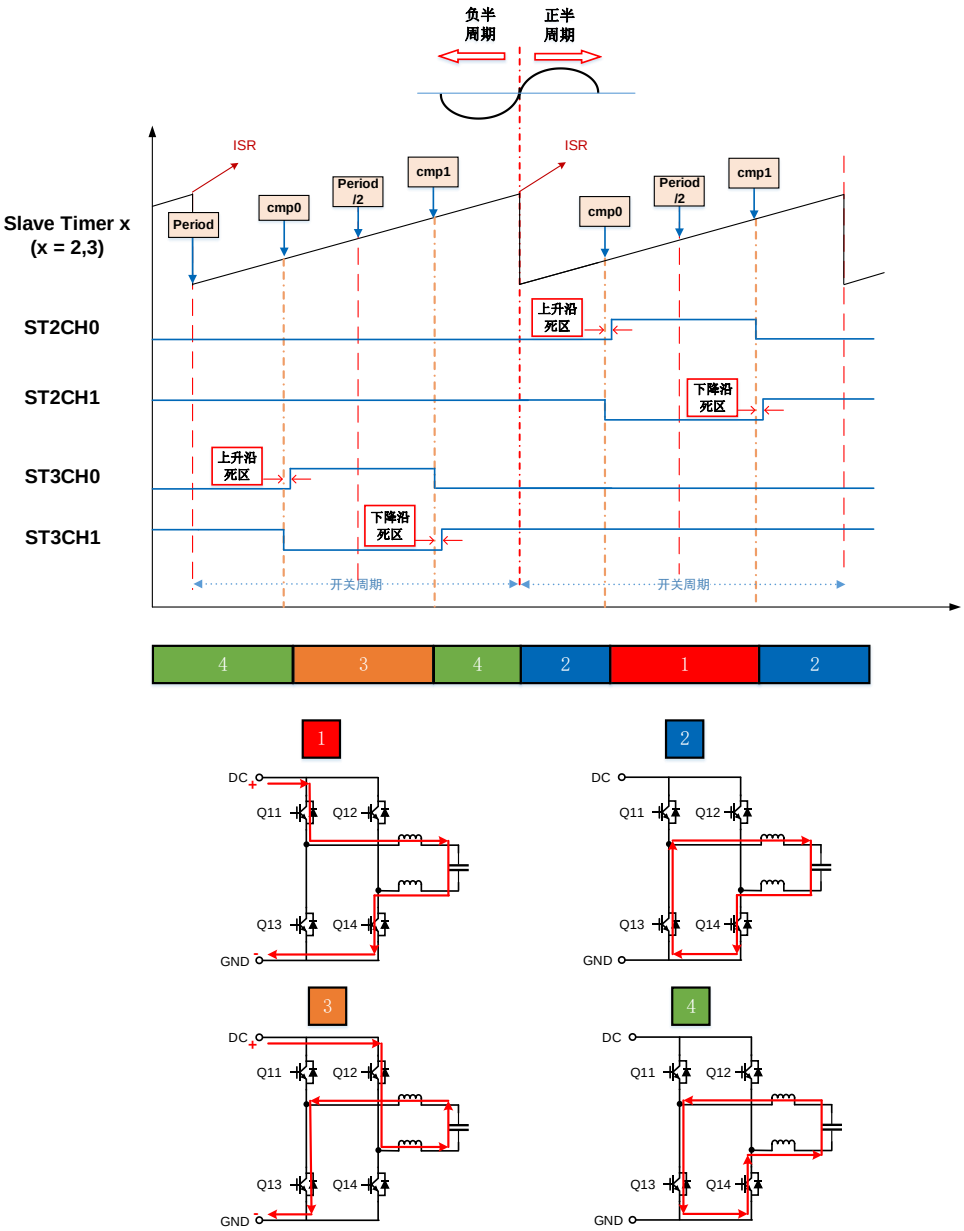
表 8-1. H 桥逆变电路 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	0	
自动重载寄存器	STxCAR	周期值	在ISR中赋值，更新事件（周期）处更新
比较0寄存器	STxCMP0V	CMP0=(PER – duty)/2	在ISR中赋值，更新事件（周期）处更新
比较1寄存器	STxCMP1V	CMP1=(PER + duty)/2	在ISR中赋值，更新事件（周期）处更新；注意运算溢出；
置位请求	CH0SCMP0	比较器0事件	
复位请求	CH0RSCMP1	比较器1事件	
死区使能	DTEN	1	死区使能后CH0和CH1自动输出互补波形
死区下降沿值、上升沿值	DTFCFG[8:0]/ DTRCFG[8:0]	实际死区时间 *HRTIMER_DTGCK	
中断请求	REPIE	重复事件	ST0对应的中断号为HRTIMER_IRQ1
影子寄存器	SHWEN	使能	
更新事件	UPRST	计数器复位事件（周期）	
通道输出使能	STxCH0EN	使能	
计数器使能	STxCEN	使能	需同步打开

如*图 8-2. H 桥逆变电路 PWM 生成*所示，CMP0 和 CMP1 用于 PWM 的生成。在周期事件时，

产生中断请求，并在中断服务程序中进行环路运算，环路运算会得出新的 CMP0 和 CMP1 值，写入到影子寄存器，并在下一周期处进行更新。

图 8-2. H 桥逆变电路 PWM 生成



8.4. 参考代码

以 216MHz 主频、开关频率 100kHz、200ns 死区时间为例，参考代码如*表 8-2. H 桥逆变电路配置代码*:

表 8-2. H 桥逆变电路配置代码

```
/* PWM wave config */
hrtimer_config(void)
{
```

```
/* configure period, continuous mode and DLL value */
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 34560U;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL16;
baseinit_para.repetitioncounter = 0U;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER2, &baseinit_para);
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER3, &baseinit_para);
/* configure shadow enable */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER2, &timerinit_para);
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER3, &timerinit_para);
/* enable deadtime, config period event as upadate event */
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_ENABLED;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_ENABLED;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER2, &timercfg_para);
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER3, &timercfg_para);
/* configures the deadtime mode :deadtime=200ns */
hrtimer_deadtimercfg_struct_para_init(&deadtimecfg_para);
deadtimecfg_para.prescaler = HRTIMER_DEADTIME_PRESCALER_MUL16;
deadtimecfg_para.rising_value = 691;
deadtimecfg_para.rising_sign = HRTIMER_DEADTIME_RISINGSIGN_POSITIVE;
deadtimecfg_para.falling_value = 691;
deadtimecfg_para.falling_sign = HRTIMER_DEADTIME_FALLINGSIGN_POSITIVE;
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER2, &deadtimecfg_par
a);
hrtimer_slavetimer_deadtime_config(HRTIMER0, HRTIMER_SLAVE_TIMER3, &deadtimecfg_par
a);
/* configure duty value */
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = (34560 – duty)/2;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER2,
HRTIMER_COMPARE0, &comparecfg_para);
comparecfg_para.compare_value = (34560 + duty)/2;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER2,
HRTIMER_COMPARE1, &comparecfg_para);
comparecfg_para.compare_value = (34560 – duty)/2;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER3,
HRTIMER_COMPARE0, &comparecfg_para);
```

```
comparecfg_para.compare_value = (34560 + duty)/2;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0,HRTIMER_SLAVE_TIMER3,
HRTIMER_COMPARE1, &comparecfg_para);
/* configure set and reset event */
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.reset_request = HRTIMER_CHANNEL_RESET_CMP1;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_CMP0;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER2,
HRTIMER_ST2_CH0, &outcfg_para);
hrtimer_slavetimer_waveform_channel_config(HRTIMER0,HRTIMER_SLAVE_TIMER3,
HRTIMER_ST3_CH0, &outcfg_para);
/* software update to ensure that the initial configuration takes effect before the timer start */
HRTIMER_CTL1(HRTIMER0) |= (HRTIMER_CTL1_ST2SUP| HRTIMER_CTL1_ST3SUP);
/* enable output channels synchronization*/
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST2_CH0| HRTIMER_ST2_CH1| HRTIM
ER_ST3_CH0| HRTIMER_ST3_CH1);
/* enable counters synchronization */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST2_COUNTER | HRTIMER_ST3_COU
NTER);
}

/* ISR: ST2 counter reset event  */
void HRTIMER_IRQ3_IRQHandler(void)
{
……
/* Sampling value processing and loop calculation */
……
/* Update the value of CMP0 per control cycle：CMP0 = period */
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = (period-new_duty)/2;
HRTIMER_STXCMP0V(HRTIMER0, HRTIMER_SLAVE_TIMER3) = (period-new_duty)/2;
/* Update the value of CMP1 per control cycle：CMP1 = period / 2 */
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER2) = (period+new_duty)/2;
HRTIMER_STXCMP1V(HRTIMER0, HRTIMER_SLAVE_TIMER3) = (period+new_duty)/2;
……
}
```

9. 斜坡补偿示例

9.1. 配置重点

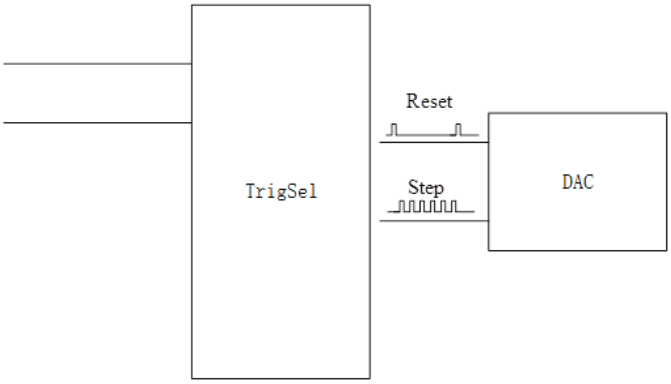
本示例配置的重点是：

- 计数器复位
- DAC 复位触发源
- DAC 步进触发源

9.2. 补偿方法

斜坡补偿需要一个与 PWM 同步的锯齿波信号，该锯齿波信号可由 MCU 触发 DAC 产生，触发信号通过 TRIGSEL 模块产生 DAC 的复位与步进信号，触发示意图如*图 9-1. TrigSel 触发 DAC* 所示。锯齿波的复位信号需与开关 PWM 同步，可将 DAC 的复位信号配置为由 HRTIMER 或 TIMER 输出通道触发。

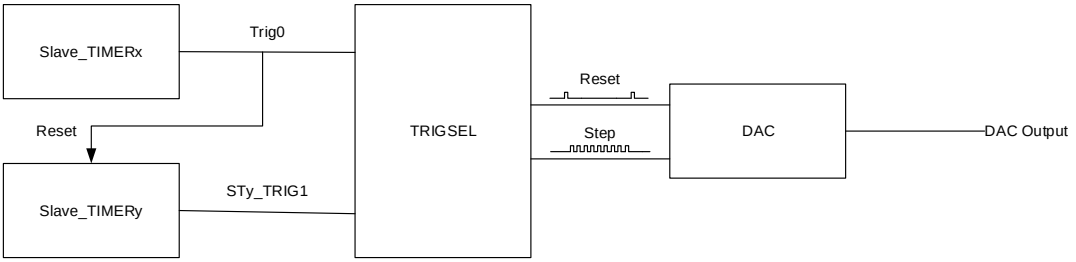
图 9-1. TrigSel 触发 DAC



触发 DAC 产生锯齿波信号有如下几种方法：

- 使用两个 ST，其中 STx 产生锯齿波复位信号，STy 产生锯齿波步进信号，如*图 9-2. 双 ST 触发 DAC* 所示：

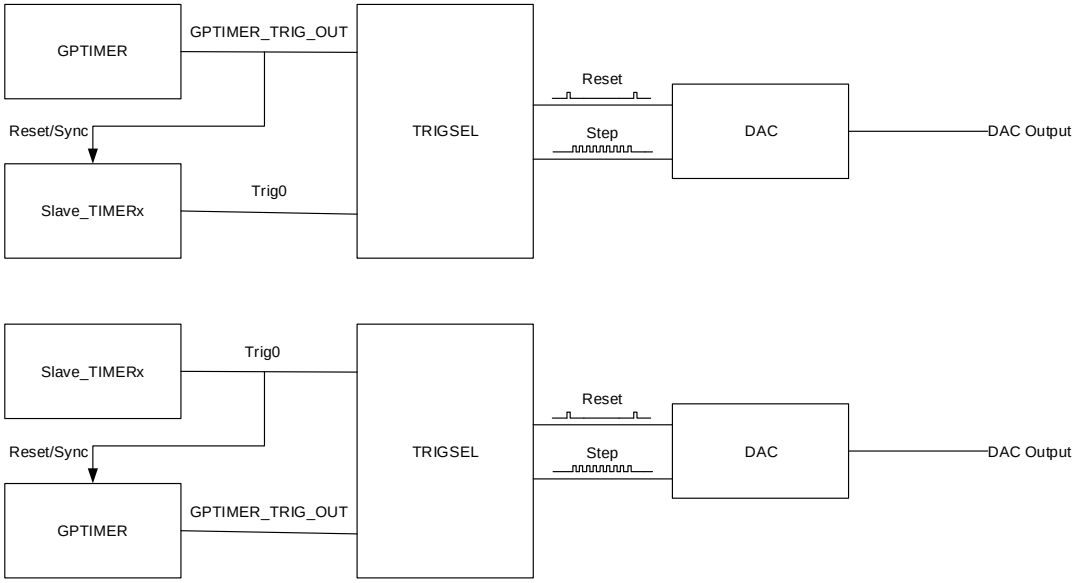
图 9-2. 双 ST 触发 DAC



- 使用一个 ST 和一个 ADTimer，STx 产生锯齿波复位/步进信号，与 STx 保持同步的

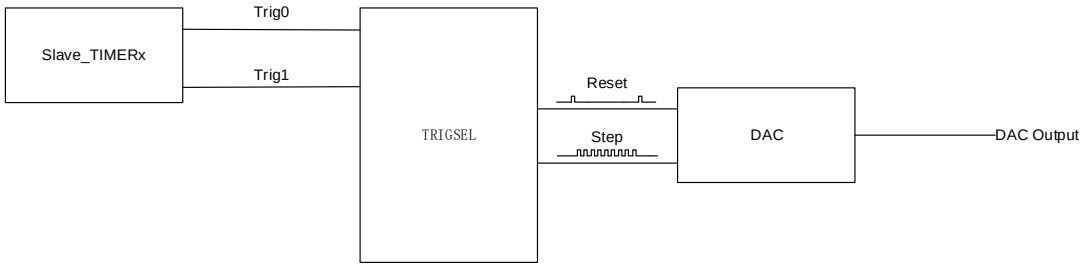
ADTimer 产生锯齿波步进/复位信号，如*图 9-3. ST+GPTimer 触发 DAC* 所示：

图 9-3. ST+GPTimer 触发 DAC



- 使用 HRTIMER 双通道触发模式，配置 HRTIMER_STxCTL1 寄存器中的 TRIG0M 位和 TRIG1M 位分别为 0，如*图 9-4. ST 双通道触发 DAC* 所示：

图 9-4. ST 双通道触发 DAC



本示例采用双 ST 触发 DAC 生成递降锯齿波。

9.3. 具体配置

本示例使用 ST0 产生斜坡补偿 DAC 的复位信号，ST4 产生斜坡补偿 DAC 的 step 信号。ST0 和 ST4 的配置如下，具体可参考*表 9-1. 斜坡补偿 PWM 配置*:

- HRTIMER 时钟 4 倍频；
- 定时器 ST0、ST4 配置为连续模式;
- 计数器自动重载寄存器 ST0CAR 写入周期值:24000；对应开关周期；ST4CAR 写入周期值：4320；对应 50us 周期
- 重复计数器：0；
- ST0 置位事件：周期事件；
- ST0 复位事件：比较器 2 事件；
- ST4 计数器复位事件：ST0 的比较器 0 事件；

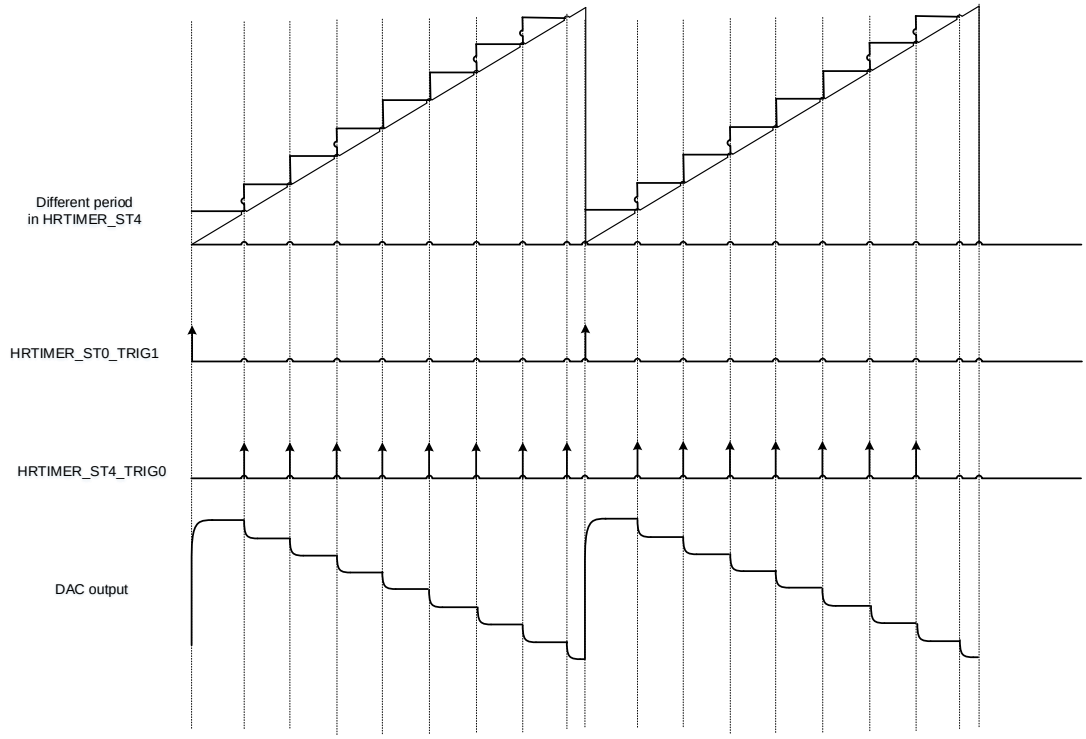
- ST0 比较 0 寄存器 ST0CMP0V 写入(PER - 15)；
- ST0 比较 2 寄存器 ST0CMP2V 写入（PER/2）；
- ST0 影子寄存器：失能；
- ST4 影子寄存器：使能；
- ST4 更新事件：由计数器复位事件产生，控制位为 STxCTL0 寄存器的 UPREP 位；
- 使能 ST0 的 CH0 输出通道：CHOUTEN 寄存器的 ST0CH0EN 位置 1；
- 开启 ST0 和 ST4 计数：MTCTL0 寄存器的 ST0CEN、ST4CEN 位同时置 1。

表 9-1. 斜坡补偿 PWM 配置

选项	控制位	具体配置	备注
工作模式	CTNM	连续模式	
重复计数器	STxCREP	0	
自动重载寄存器	STxCAR	ST0:24000 ST4: 4320	ST0的周期值对应PWM开关周期，ST4的周期值对应50us的step触发信号
比较0寄存器	ST0CMP0V	CMP0=PER-15	
比较2寄存器	ST0CMP2V	CMP2=PER/2	
置位请求	CH0SPER	ST0:周期事件	
复位请求	CH0RSCMP2	ST0:比较器2事件	
死区使能	DTEN	0	不使能死区时间
影子寄存器	SHWEN	ST0:失能 ST4:使能	
更新事件	UPRST	ST0:NONE ST4:计数器复位事件	
通道输出使能	STxCH0EN	使能	
计数器使能	STxCEN	使能	需同步打开

如*图 9-5. 斜坡补偿信号生成*所示，ST0_TRIG1 用于触发 DAC 产生锯齿波复位信号，ST4_TRIG0 用于触发 DAC 产生锯齿波 step 信号。ST4 计数器复位事件由 ST0 的 CMP0 事件产生。

图 9-5. 斜坡补偿信号生成



9.4. 参考代码

以 216MHz 主频、开关频率 36kHz 为例，参考代码如*表 9-2. 斜坡补偿配置代码*:

表 9-2. 斜坡补偿配置代码

```
/* PWM wave config */
hrtimer_config(void)
{
/*Slave_TIMER0 time base clock config*/
hrtimer_baseinit_struct_para_init(&baseinit_para);
baseinit_para.period = 24000;
baseinit_para.prescaler = HRTIMER_PRESCALER_MUL4;
baseinit_para.repetitioncounter = 0;
baseinit_para.counter_mode = HRTIMER_COUNTER_MODE_CONTINOUS;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &baseinit_para);
/*Slave_TIMER4 time base clock config*/
baseinit_para.period =4320;
hrtimer_timers_base_init(HRTIMER0, HRTIMER_SLAVE_TIMER4, &baseinit_para);
/* initialize Slave_TIMER0 to work in waveform mode */
hrtimer_timerinit_struct_para_init(&timerinit_para);
timerinit_para.cnt_bunch = HRTIMER_TIMERBUNCHNMODE_MAINTAINCLOCK;
timerinit_para.dac_trigger = HRTIMER_DAC_TRIGGER_NONE;
```

```
timerinit_para.half_mode = HRTIMER_HALFMODE_DISABLED;
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_DISABLED;
timerinit_para.reset_sync = HRTIMER_SYNCRESET_DISABLED;
timerinit_para.shadow = HRTIMER_SHADOW_DISABLED;
timerinit_para.start_sync = HRTIMER_SYNISTART_DISABLED;
timerinit_para.update_selection = HRTIMER_MT_ST_UPDATE_SELECTION_INDEPENDENT;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timerinit_para);
/*for DAC step trigger.*/
timerinit_para.dac_trigger = HRTIMER_DAC_TRIGGER_DACTRIG1;
timerinit_para.repetition_update = HRTIMER_UPDATEONREPETITION_ENABLED;
timerinit_para.shadow = HRTIMER_SHADOW_ENABLED;
hrtimer_timers_waveform_init(HRTIMER0, HRTIMER_SLAVE_TIMER4, &timerinit_para);
/* configure the general behavior of a Slave_TIMER0 which work in waveform mode */
/*OUTA*/
hrtimer_timercfg_struct_para_init(&timercfg_para);
timercfg_para.balanced_mode = HRTIMER_STXBALANCEDMODE_DISABLED;
timercfg_para.cnt_reset = HRTIMER_STXCNT_RESET_NONE;
timercfg_para.deadtime_enable = HRTIMER_STXDEADTIME_DISABLED;
timercfg_para.delayed_idle = HRTIMER_STXDELAYED_IDLE_DISABLED;
timercfg_para.fault_enable = HRTIMER_STXFAULTENABLE_NONE;
timercfg_para.fault_protect = HRTIMER_STXFAULT_PROTECT_READWRITE;
timercfg_para.reset_update = HRTIMER_STXUPDATEONRESET_DISABLED;
timercfg_para.update_source = HRTIMER_STXUPDATETRIGGER_NONE;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &timercfg_para);
/*for slavetimer4 bit19,ST0CMP0RST*/
timercfg_para.cnt_reset = HRTIMER_STXCNT_RESET_OTHER0_CMP0;
hrtimer_slavetimer_waveform_config(HRTIMER0, HRTIMER_SLAVE_TIMER4, &timercfg_para);
/* configures the compare unit of a Slave_TIMER0 which work in waveform mode */
/*OUTA*/
hrtimer_comparecfg_struct_para_init(&comparecfg_para);
comparecfg_para.compare_value = 12000;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIM
ER_COMPARE2, &comparecfg_para);
comparecfg_para.compare_value = 23985;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIM
ER_COMPARE0, &comparecfg_para);
/* configures double source trigger */
hrtimer_double_channel_struct_para_init(&two_dac_trigger_para);
two_dac_trigger_para.trigger_enable = HRTIMER_DOUBLE_TRIG_ENABLE;
two_dac_trigger_para.trigger0 = HRTIMER_DOUBLE_TRIG0_BY_CPV1;
two_dac_trigger_para.trigger1 = HRTIMER_DOUBLE_TRIG1_BY_CNT_RESET;
hrtimer_double_trigger_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, &two_dac_trigger_para);
/* configures the ST0_CH0_O output of a Slave_TIMER0 work in waveform mode */
```

```
hrtimer_channel_outputcfg_struct_para_init(&outcfg_para);
outcfg_para.carrier_mode = HRTIMER_CHANNEL_CARRIER_DISABLED;
outcfg_para.deadtime_bunch = HRTIMER_CHANNEL_BUNCH_ENTRY_REGULAR;
outcfg_para.fault_state = HRTIMER_CHANNEL_FAULTSTATE_INACTIVE;
outcfg_para.idle_bunch = HRTIMER_CHANNEL_BUNCH_IDLE_DISABLE;
outcfg_para.idle_state = HRTIMER_CHANNEL_IDLESTATE_INACTIVE;
outcfg_para.polarity = HRTIMER_CHANNEL_POLARITY_HIGH;
/*OUTA*/
outcfg_para.reset_request =  HRTIMER_CHANNEL_RESET_CMP2 ;
outcfg_para.set_request = HRTIMER_CHANNEL_SET_PER;
hrtimer_slavetimer_waveform_channel_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIMER_ST0_CH0, &outcfg_para);
/* enable output channel */
hrtimer_timers_counter_enable(HRTIMER0, HRTIMER_ST0_COUNTER | HRTIMER_ST4_COUNTER );
hrtimer_output_channel_enable(HRTIMER0, HRTIMER_ST0_CH0 );
}

/* DAC trigger config */
dac_trigger_config(void)
{……
/* DAC triggered by external trigger */
dac_sawtooth_reset_trigger_source_config(DAC0, DAC_OUT1, DAC_SAWTOOTH_RESET_TRIGGER_EXTERNAL);
dac_sawtooth_step_trigger_source_config(DAC0, DAC_OUT1, DAC_SAWTOOTH_STEP_TRIGGER_EXTERNAL);
……}

/* Trigsel config */
trigsel_config(void)
{……
/* Trigger input select HRTIMER_ST0TRIG1 (0x8A), TRIGSEL_INPUT_HRTIMER_DAC_TRIG1 (0x88) */
trigsel_init(TRIGSEL_OUTPUT_DAC0_EXTRIG1, TRIGSEL_INPUT_HRTIMER_ST0_TRIG1);
trigsel_init(TRIGSEL_OUTPUT_DAC0_ST_EXTRIG1, TRIGSEL_INPUT_HRTIMER_DAC_TRIG1);
……}
```

10. 其他示例

其他更多示例可参考 www.gd32mcu.com.或联系当地经销商。

11. 版本历史

表 11-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2024 年 08 月 13 日
1.1	新增斜坡补偿示例	2024 年 11 月 18 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.