GigaDevice Semiconductor Inc.

GD32G5x3 High-Resolution Timer User Guide

Application Note AN203

Revision 1.1

(Nov. 2024)

Table of Contents

List of Figures

List of Tables

1. Terminology and Abbreviations

The professional terms and abbreviations used in this document are as shown in ***Table 1-1. Common Acronym Comparison Table***.

Table 1-1. Common Acronym Comparison Table

Abbreviation	Full name
HRTIMER	High-Resolution Timer
MT	Master Timer
STx	Slave Timer x
DLL	Delay-locked loop
PER	Period
CMPx	Compare x
CBC	Cycle by cycle
LLC	LLC Resonant Converters
TRIGSEL	Trigger Selection Controller

2. Introduction

The GD32G5x3 series MCU features a High-Resolution timer (HRTIMER) module.

HRTIMER adopts a Master-Slave timer architecture, consisting of 1 Master Timer (MT) and 8 Slave Timers (STx).

The MT unit is mainly composed of a 16-bit counter, an counter auto reload register, a counter repetition counter, and four comparators.

The STx unit mainly consists of a 16-bit counter, an counter auto reload register, a counter repetition counter, four comparators, two capture registers, a set/reset crossbar switch, idle control level, and channel output level, among other components. Each STx unit can generate two independent PWM waves or a pair of complementary PWM waves with dead-time.

HRTIMER module can generate up to 16 high-resolution PWM waves, and can also be coupled into 8 pairs of complementary PWM waves with dead-time.

The HRTIMER module, due to its high-resolution and flexibility, is particularly suitable for applications such as switch power supplies and power control.

This document will introduce the basic functions and reference applications of HRTIMER based on the GD32G5x3 series MCU, takes a 216MHz system clock as an example. When the system clock is set to other values, some calculation values in the document will no longer apply, and recalculations based on the content of the document are required.

3. Basic configuration

3.1. Period value and Compare Value Settings.

The clock sources for MT and STx are provided by the HRTIMER_CK from the RCU module.

The DLL is used to generate and calibrate a high-resolution counter unit HRTIMER_HPCK = 32 * $f_{HRTIMER\_CK}$.

The prescaler divides the high-resolution timer unit HRTIMER_HPCK by the division factor to obtain the counter clock HRTIMER_PSCCK.

The frequency relationship between them can be expressed as follows: $f_{HRTIMER\_PSCCK}$ = $f_{HRTIMER\_HPCK}$ / the division ratio.

Therefore, the counter clock = HRTIMER_CK * 32 / the division ratio. The 32 / division ratio can be understood as the multiplication factor, so the counter clock = HRTIMER_CK * multiplication factor.

Taking the division ratio as 1, the counter clock is HRTIMER_CK*32, which can be regarded as a timer with a 32-fold frequency and a high-resolution clock of 6.912GHz (216MHz*32). The period value and the comparison value can be directly written into the corresponding 16-bit registers with high resolution. The counting period can be simply calculated using the formula PER = $f_{High\text{-}res}/f_{pwm}$.

For example, when you want to generate a PWM with a period of 200 kHz, the value of the period register should be set to PER = 6.912GHz / 200kHz = 34560. For specific configuration, refer to the code in ***Table 3-1. Timer Period Configuration Code***.

Table 3-1. Timer Period Configuration Code

```
/* 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. Set/Reset Crossbar

The timer unit controls two outputs through Set/Reset crossbar, allowing multiple events to be set as set or reset events, thereby increasing output flexibility. By configuring the Set/Reset events, the output can be configured as two independent PWM waves or a pair of complementary PWM waves with dead-time.

3.3. Channel Output Stage

There are three working states of the channel output level:

- Run state: normal output.
- Idle state: Output the predefined idle state.
- Fault state: A fault input triggers the fault state.

Generally, the running state is used to generate PWM waveforms, and the fault state is triggered when fault protection is activated.

3.4. Shadow Register

Some registers in MT and STx have shadow registers. The shadow registers are disabled after MCU reset. If the SHWEN bit is set to 1, shadow registers are enabled and these registers listed in ***Table 3-2. Shadow Register*** are pre-loaded. The values written into these registers are transferred into the shadow register and do not take effect immediately. Their content of the shadow is transferred into active register and take effect immediately on update event.

Table 3-2. Shadow Register

Category	Registers that contain shadow registers
DMA and interrupt enable	MTDMAINTEN / STxDMAINTEN
Counter auto-reload value	MTCAR / STxCAR
Repetition counter value	MTCREP / STxCREP
Comparison value	MTCMP0V / STxCMP0V
	STxCMP0CP
	MTCMP1V / STxCMP1V
	MTCMP2V / STxCMP2V
	MTCMP3V / STxCMP3V
Dead-time configuration	STxDTCTL
	DTFCFG[15:9] and DTRCFG[15:9] in STxACTL
Set/Reset source	STxCH0SET
	STxCH0RST
	STxCH1SET
	STxCH1RST
Counter reset source	STxCNTRST

Note: After completing the initialization configuration of HRTIMER, it is recommended to perform a manual software update to ensure that the initialized configuration values are updated to the effective registers.

3.5. Update Event

MT and STx update event sources as shown in ***Table 3-3. Update the event source configuration***.

Table 3-3. Update the event source configuration

Update source	MT Control bits	STx Control bits
Software	MTSUP	STxSUP
Repetition event	UPREP = 1	UPREP = 1
Counter reset event or roll-over event	NA	UPRST = 1
Update event from other timers	UPBMT	UPBSTX
DMA mode end event	UPSEL[1:0] = 2'b01	UPSEL[3:0] = 4'b0001
Roll-over event following a DMA mode end event	UPSEL[1:0] = 2'b10	UPSEL[3:0] = 4'b0010
Update event generated on the rising edge of STxUPINy(y=0..2)	NA	UPSEL[3:0] = 4'b0011 \ 4'b0100 \ 4'b0101;
Update event following the rising edge of STxUPINy(y=0..2)	NA	UPSEL[3:0] = 4'b0110 \ 4'b0111 4'b1000;

Note: The HRTIMER_CTL0 register provides the function of disabling update events, and the disable function for all timers is in the same register, which can take effect simultaneously. Before assigning values to multiple timers or multiple shadow registers, it is recommended to disable this function uniformly, and then restore it uniformly after the assignment is completed, to avoid the occurrence of update events in the middle of writing multiple shadow registers, causing multiple shadow registers not to be updated at the same time, which can lead to abnormal waveforms. For specific code implementation, refer to ***Table 3-4. Update Event Disabling and Recovery***.

Table 3-4. Update Event Disabling and Recovery

```
/* 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. Dead-time Mode

When DTEN = 1 in STxCHOCTL register and BLNMEN = 0 in STxCTL0 register, the STx output run in dead-time mode.

The dead-time setting can refer to ***Table 3-5. Dead-time Related Register Configuration***.

Note: It is recommended that the prescaled factor of dead-time should be consistent with the clock division of HRTIMER counter.

Table 3-5. Dead-time Related Register Configuration

Configurtion	Control bits	Value	Note
Dead-time enable	DTEN	1	CH0 and CH1 complement automatically in dead-time mode
Dead-time clock division factor	DTGCKDIV[3:0]	n	It is recommended to synchronize with the counter clock frequency division.
Falling edge dead-time value	DTFCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
Rising edge dead-time value	DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
The sign of falling edge dead-time value	DTFS	0	In power control applications, it should be set to 0
The sign of rising edge dead-time value	DTRS	0	

The control bits for the above configurations are all in the STxDTCTL and STxACTL registers.

- The dead-time values are defined with DTFCFG[15:0] and DTRCFG[15:0] bit-fields. DTFCFG[15:0] bit-fields defines the value of the dead-time following a falling edge of O0PRE, and DTRCFG[15:0] bit-fields defines the value of the dead-time following a rising edge of O0PRE.
- DTFCFG[8:0] and DTRCFG[8:0] bit-fields are in HRTIMER_STxDTCTL register. DTFCFG[15:9] and DTRCFG[15:9] bit-fields are in HRTIMER_STxACTL register
- The dead-time values are based on a specific clock division according to DTGCKDIV[3:0] bit-fields in HRTIMER_STxDTCTL register.
- DTGCKDIV[2:0] bit-fields is in the HRTIMER_STxDTCTL register, DTGCKDIV[3] bit is in the HRTIMER_STxACTL register.

For example, to generate a 200ns dead-time, the dead-time clock division factor DTGCKDIV[3:0] can be set to 4b’1000, which is 16 times the frequency, and DTFCFG[8:0]

and DTRCFG[8:0] can be set to 200ns * 16 * 216MHz = 691.

3.7. Comparison Threshold

Each timer unit has four comparators inside, which can flexibly generate multiple compare events, and through compare events, complete waveform output or sampling trigger functions.

The compare value should not be too close to 0, nor should the comparator be too close to the period value, or too close between comparators themselves, otherwise, compare events or period events may be missed, leading to abnormal waveforms…

The CAR register and compare registers have the following limitations:

- The minimum value must be greater than or equal to (3 * $t_{HRTIMER\_CK}$).
- The maximum value must be less than or equal to (0xFFFF – 1 * $t_{HRTIMER\_CK}$).

With a 216MHz system clock, DLL 32 times frequency multiplication, and 200kHz PWM as an example:

The CAR value should be set to PER = 6.912GHz / 200kHz = 34560, and should not exceed (65535-32).

The CMP value should be set not less than (3 * 32) and does not exceed (34560 - 32).

3.8. External Event Function

10 external events can simultaneously affect any one of the 8 STs. This feature can be used to quickly process external events, achieving CRM control, Cycle-by-Cycle Current Limiting, and other functions.

There are four main sources of external event inputs, which can come from comparators, digital input pins, the ADC analog watchdog, and TIMER_TRGO. External events can be configured to be triggered by a rising edge, falling edge, or level, and can be processed with digital filtering.

Refer to ***Table 3-6. External Input Register Configuration*** for relevant register configurations.

Table 3-6. External Input Register Configuration

Configuration	Bit-field	Options
Source	EXEVySRC[1:0]	SRC0、SRC1、SRC2、SRC3
Edge sensitivity	EXEVyEG[1:0]	Rising, falling or both
Polarity	EXEVyP	High level、low level
Fast mode	EXEVyFAST	Low-latency mode、Non-low latency mode
External event filter	Refer to the HRTIMER_STxEXEVFCFG0、	

Configuration	Bit-field	Options
	HRTIMER_STxEXEVFCFG1、HRTIMER_STxEXEVFCFG2 registers.	

Take the CBC by internal comparator as an example, the external event code can refer to ***Table 3-7. External Event Reference Code***.

Table 3-7. External Event Reference Code

```
/* 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. Fault Protection Function

Each ST has a fault protection mechanism, which can quickly handle fault inputs through this function, achieving functions such as OVP, SCP, and CBC, etc.

Fault input sources mainly come from external events, external input pins, and the output of internal comparators. The selection of fault input sources for each STy is determined by their respective FLTyINSRC bit.

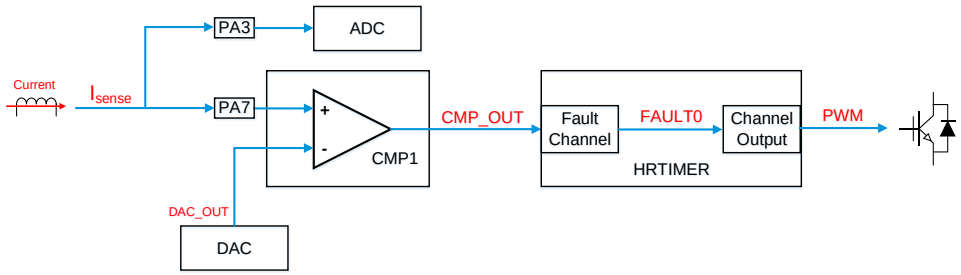
When FLTyINSRC = 00, the protection trigger signal output by the external comparator can be connected to the external fault input pin.

When FLTyINSRC = 01, the output of the MCU's internal comparator is directly connected to the fault input channel, with the positive end of the internal comparator connected to the physical quantity, and the negative end can be connected to the output of the DAC.

When FLTyINSRC = 10, the HRTIMER's external event will be directly connected to the fault input channel, where the trigger for the external event can come from external event input pins, the output of the internal comparator, signals from other internal modules, etc.

As shown in ***Figure 3-1. Fault Protection Function (Fault Source is the Internal Comparator)***, the DAC output protection threshold is compared with the actual physical quantity sampling value in the internal comparator, and the comparator output is connected to the fault channel to respond to the fault, thereby outputting the fault status (such as sealing the wave, etc.).

Figure 3-1. Fault Protection Function (Fault Source is the Internal Comparator)



Take the fault protection with an internal comparator as an example, the fault input configuration shown in ***Table 3-8. Fault Input Reference Code***.

Table 3-8. Fault Input Reference Code

```
/* 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);
```

For more information on fault protection and comparator modules, please refer to the ***AN198 GD32G5x3 Comparator User Guide***.

3.10. Trigger to ADC

The ADCs can be triggered by the TRIGSEL. Ten independent triggers (HRTIMER_ADCTRIG0 - HRTIMER_ADCTRIG9 in HRTIMER_ADCTRIGx (x=0..3), HRTIMER_ADCTRIGxA (x=0..3), HRTIMER_ADCEXTTRG, HRTIMER_ADCEXTTRGA) are

available to start the routine sequence of the ADCs.

The ADC sampling, triggered by CMP1 of ST0, configuration shown in ***Table 3-9. Trigger to ADC Reference Code***.

Table 3-9. Trigger to ADC Reference Code

```
/* 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. Double Source Trigger

Enabling double source trigger mode requires setting the TRIGEN bit in the HRTIMER_STxCTL1 register. Once the timer is running (with the STxCEN bit set), the TRIGEN bit cannot be changed. At this time, the TRIG0M and TRIG1M in the HRTIMER_STxCTL1 register configure the trigger signals TRIG0 and TRIG1 output by the Slave_TIMERx to be connected to TRIGSEL (in the TRIGSEL module, they are respectively HRTIMER_STx_TRIG0 and HRTIMER_STx_TRIG1). By configuring the TRIG0M and TRIG1M bits in the HRTIMER_STxCTL1 register, user can select the trigger signal source.

The TRIG0 signal is configured through the TRIG0M

- TRIG1M = 0 : Counter reset or roll-over event generate TRIG0 trigger

- TRIG0M = 1 : Output 0 set event generate TRIG0 trigger

The TRIG1 signal is configured through the TRIG1M

- TRIG1M = 0 : Compare 1 event generate TRIG1 trigger.
- TRIG1M = 1 : Output 0 reset event generate TRIG1 trigger.

The application of double source trigger is to solve issues such as ramp compensation and hysteresis control.

Taking the double source trigger of DAC to generate a sawtooth wave with ST0 as an example, the configuration code refers to ***Table 3-10. Double source trigger Reference Code***.

Table 3-10. Double source trigger Reference Code

```
/* 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 Synchronization

When multiple MTs or STx work together, synchronization between timers is often required, mainly including synchronized start and triggering counter reset.

3.12.1. Synchronous Start

To maintain the consistency of the PWM waveform, it is often necessary to synchronize the start of timers. The synchronization start code for MT and STx is shown in ***Table 3-11. Synchronous Startup Code***.

Table 3-11. Synchronous Startup Code

```
/* 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. Event-triggered Counter Reset

The following four types of events can reset the STx counter:

- STx itself: Update events, compare 1 event and compare 3 event;
- Other STy (for example, if x = 1, then y = 0, 2..7): Compare 0 event, compare 1 event, and compare 3 event;
- MT: Compare 0 event, compare 1 event, compare 2 event, compare 3 event, and reset event.
- External event y (y = 0..9): EXEVy is the filtered signal of the external event in STx.

This feature has a wide range of applications in interlaced topology, frequency multiplication sampling, and phase control.

Take ST1 as an example, use the MT compare 0 event to reset ST1, and the relevant code configuration is as shown in ***Table 3-12. CNTRST Function Code***:

Table 3-12. CNTRST Function Code

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

3.13. Single PWM Wave Generation

The focus of this case configuration is:

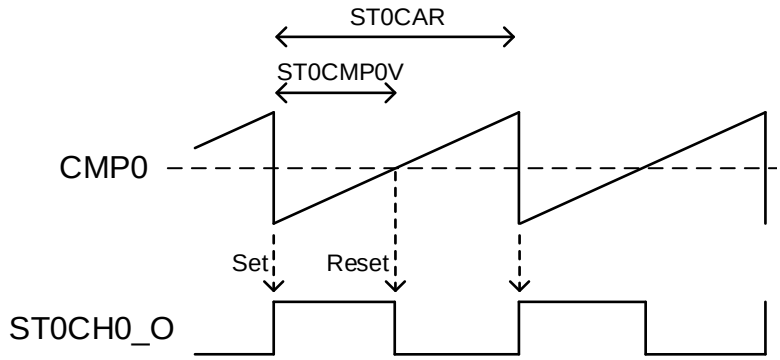
- Timer continuous mode
- Set reset event configuration

- Output stage enable

PWM signals are the fundamental components of most power converters and are also used for many other purposes, such as motors, buzzers, LEDs, etc.

As shown in ***Figure 3-2. Single PWM Generation***, a single PWM signal is generated with a 100kHz frequency and a 50% duty cycle on the ST0CH0 output.

Figure 3-2. Single PWM Generation



Take ST0 as an example, refer to ***Table 3-13. Specific Configuration of a Single PWM*** are as follows:

- The timer ST0 is configured in continuous mode.
- Counter clock division: Set to 16 times frequency multiplication.
- The counter auto reload register STxCAR writes the cycle value: (216MHz x 16) / 100kHz = 34560.
- Compare the 0 register STxCMP0V writing the duty cycle value: 0.5 x 34560 = 17280.
- Set request: Period event;
- Reset request: Compare 0 event;
- Enable the CH0_O output channel of ST0: Set the ST0CH0EN bit to 1 in the CHOUTEN register.
- Enable ST0 counting: Set the ST0CEN bit to 1 in the MTCTL0 register.

Table 3-13. Specific Configuration of a Single PWM

Configuration	Control bits	value	Note
Operating mode	CTNM	Continuous mode	
Counter clock division	CNTCKDIV[2:0]	1	Set to 16 times frequency multiplication
Counter auto reload register	STxCAR	(216MHz x 16)/100kHz = 34560	Period value
Compare 0	STxCMP0V	0.5 x 34560 = 17280	Duty cycle
Repetition counter	STxCREP	0	
Set request	CH0SPER	Period event	
Reset request	CH0RSCMP0	Compare 0 event	
Channel output enable	ST0CH0EN	Enable	

Configuration	Control bits	value	Note
Count enable	ST0CEN	Enable	

The reference code is shown in *Table 3-14. PWM wave generation reference code*:

Table 3-14. PWM wave generation reference code

```
/* 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. Complementary Dead-time PWM Wave Generation

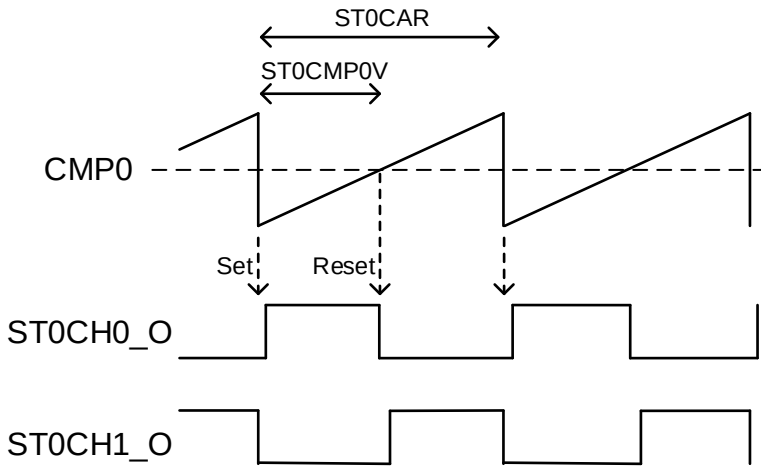
The focus of this case configuration is:

- Complementary channel enable
- Dead-time configuration

In power control, the switching tubes of the upper and lower bridge arms need to insert a dead-time to avoid short pass-through, which requires HRTIMER to issue complementary PWM signals with a dead-time.

As shown in *Figure 3-3. Complementary Dead-time PWM Generation*, a complementary PWM signal with a duty cycle of 50% at 100kHz is generated on ST0CH0 and ST0CH1, with a dead-time set to 200ns.

Figure 3-3. Complementary Dead-time PWM Generation



Taking ST as an example, refer to ***Table 3-15. Specific Configuration of Complementary Dead-time PWM***, which is as follows:

- The timer ST0 is configured in continuous mode.
- Counter clock division: Set to 16 times frequency multiplication.
- The counter auto reload register STxCAR writes the period value: (216MHz x 16) / 100kHz = 34560.
- Dead-time enable: DTEN=1;
- Dead-time falling edge value, rising edge value: 200ns * 16 * 216MHz = 691;
- Compare 0 register STxCMP0V writing duty cycle value: 0.5 x 34560 = 17280;
- Set request: Period event;
- Reset request: Compare 0 event;
- Enable the output channels CH0 and CH1 of ST0: Set the ST0CH0EN and ST0CH1EN bits of the CHOUTEN register to 1.
- Enable ST0 counter: Set the ST0CEN bit to 1 in the MTCTL0 register.

Table 3-15. Specific Configuration of Complementary Dead-time PWM

Configuration	Control bits	Value	Note
Operating mode	CTNM	Continuous mode	
Counter clock division	CNTCKDIV[2:0]	1	Set to 16 times frequency multiplication
Counter auto reload register	STxCAR	(216MHz x 16)/100kHz = 34560	Period value
Dead-time enable	DTEN	1	After dead-time is enabled, CH0 and CH1 automatically output complementary waveforms
Dead-time falling edge value, rising edge value	DTFCFG[8:0]/ DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
Compare 0	STxCMP0V	0.5 x 34560 = 17280	Duty cycle

Configuration	Control bits	Value	Note
register			
Repetition counter	STxCREP	0	
Set request	CH0SPER	Period event	
Reset request	CH0RSCMP0	Comparator 0 event	
Channel output enable	STxCH0/1EN	Enable	
Counter enable	ST0CEN	Enable	

The reference code is shown in ***Table 3-16. Complementary PWM Wave Generation Reference Code***:

Table 3-16. Complementary PWM Wave Generation Reference Code

```
/* 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 Circuit Example

4.1. Focus of Configuration

The focus of this example configuration is:

- Counter reset event interrupt
- ADC trigger (midpoint current sampling)
- Counter reset event update

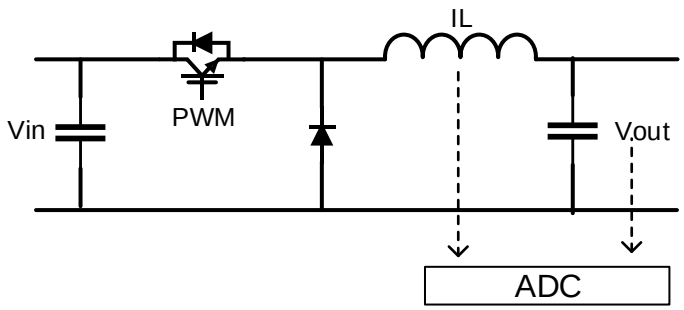
4.2. Topological Structure

The buck converter is mainly used for voltage step-down conversion, usually working at a fixed frequency, and power control is achieved by adjusting the duty cycle.

To maintain consistency between PWMs, in an ideal Buck converter, the ratio of Vin to Vout depends only on the duty cycle D of the PWM, that is, Vout = D × Vin.

Buck topology structure as shown in ***Figure 4-1. Buck Topology***:

Figure 4-1. Buck Topology



4.3. Specific Configuration

Taking ST0 as an example, the specific configuration of the Buck circuit PWM can refer to ***Table 4-1. Buck Circuit PWM Configuration***.

- The timer ST0 is configured in continuous mode.
- The counter auto reload register STxCAR is written with the period value: Corresponds to the switching cycle of the buck circuit.
- Compare 0 value register STxCMP0V is written with the duty cycle value: The duty cycle value is assigned during the period event interrupt, and updated at the period.
- Repetition counter: 0;
- Set request: Period event;

- Reset request: Compare 0 event;
- ADC trigger: Compare 1 event, during loop operation, make CMP1 equal to the latest CMP0/2.
- Interrupt event: Counter reset event, control bit is the RSTIE bit in STxDMAINTEN.
- Interrupt number: The interrupt number corresponds to ST0 is HRTIMER_IRQ1, and loop calculation and assignment are performed in the HRTIMER_IRQ1_IRQHandler.
- Shadow Register: Enabled;
- Update event: Triggered by the counter reset event, the control bit is the UPRST bit of the STxCTL0 register, meaning the new duty cycle value will be updated at the period.
- Enable the CH0_O output channel of ST0: Set the ST0CH0EN bit to 1 in the CHOUTEN register.
- Enable ST0 counting: Set the ST0CEN bit in the MTCTL0 register to 1.

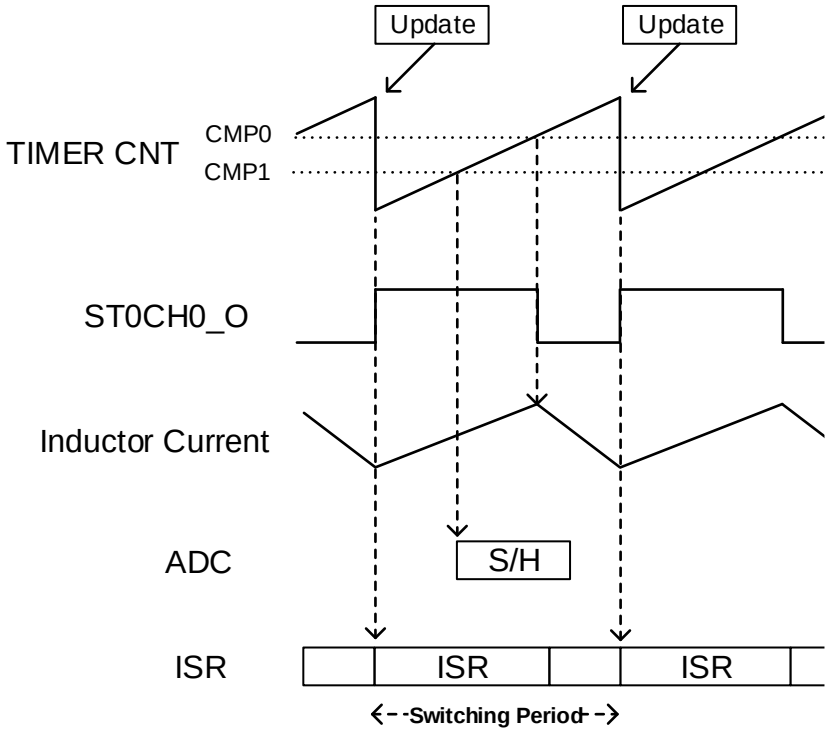
Table 4-1. Buck Circuit PWM Configuration

Options	Control bit	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	0	
Counter auto reload register	STxCAR	Corresponds to the period value	Assign a value in ISR and update at the update event (period)
Compare 0 value register	STxCMP0V	Corresponds to the duty cycle	Assign a value in ISR and update at the update event (period)
Compare 1 value register	STxCMP1V	CMP1=CMP0 / 2	Trigger ADC sampling；Assign a value in ISR and update at the update event (period)
Set request	CH0SPER	Period event	
Reset request	CH0RSCMP0	Comparator 0 event	
Interrupt request	RSTIE	Counter reset event	The interrupt number corresponding to ST0 is HRTIMER_IRQ1
Shadow registers	SHWEN	Enabled	
Update event	UPRST	Counter reset event（Period）	
Channel output enable	STxCH0\1EN = 1	Enabled	
Counter enable	STxCEN = 1	Enabled	

As shown in ***Figure 4-2. Buck Converter PWM Generation***, PER and CMP0 are used for PWM generation, while CMP1 is used to trigger ADC sampling, with the ADC configured to trigger sampling conversion at the midpoint of the output's on-time. An interrupt request is generated at the time of the period event, and the loop operation is performed in the interrupt service routine, where the loop operation will yield a new duty cycle and a new sampling point

(half of the duty cycle), which is written to the shadow register and updated in the next period.

Figure 4-2. Buck Converter PWM Generation



By updating the value of CMP1 continuously, it is possible to achieve real-time sampling of midpoint current/midpoint voltage, and to avoid the oscillations generated when the switching tube is turned on and off, ensuring the accuracy of the sampling. At the same time, in DCM mode, the average current can be compensated by dynamically modifying the value of CMP1.

If the PWM frequency is too high or the loop operation takes too long, the repetition counter can be set to n, and the interrupt request can be changed to a repetition event. This will result in an interrupt every n+1 switching cycles, performing one control loop calculation and updating the control quantity.

At the same time, the relevant ADC trigger configuration needs to be performed in the ADC and TRIGSEL modules.

PWM configuration: The trigger source is set to ADCTRIG0, and the trigger event is set to ST0CMP1.

TRIGSEL configuration: Trigger input selection HRTIMER_ADCTRIG0 (0x7D).

ADC0 Trigger Configuration: Externally trigger ADC0 regular group; ADC0 triggers Comparator 1 event, during loop operation, make CMP1 equal to half of the latest calculated value of CMP0.

4.4. Reference Code

Taking 216MHz main frequency, 100kHz switching frequency, and 100kHz control frequency as examples, the reference code is as shown in ***Table 4-2. Buck Circuit Configuration Code***:

Table 4-2. Buck Circuit Configuration Code

```
/* 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. Synchronous Buck Circuit Example

5.1. Focus of Configuration

The focus of this example configuration is:

- Counter reset event interrupt
- Dead-time configuration
- Counter reset event update.

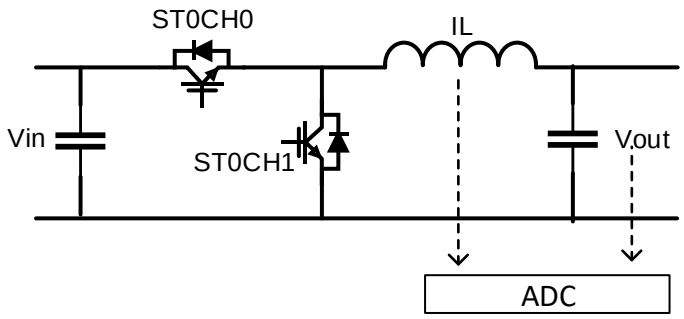
5.2. Topological Structure

Compared to the buck circuit, the synchronous rectification buck circuit replaces the freewheeling diode with a switch tube, thereby effectively reducing the conduction loss brought by the freewheeling diode in the buck circuit. It is mainly used for voltage step-down conversion and usually works at a fixed frequency, controlling power by adjusting the duty cycle.

In an ideal Buck converter, the ratio of Vin to Vout depends only on the duty cycle D of the PWM, that is, Vout = D × Vin.

Buck topology structure as shown in ***Figure 5-1. Synchronous Buck Topology Structure***:

Figure 5-1. Synchronous Buck Topology Structure



5.3. Specific Configuration

Take ST0 as an example, refer to ***Table 5-1. Synchronous Buck Circuit PWM Configuration***, the specific configuration is as follows:

- The timer ST0 is configured in continuous mode.
- The counter auto reload register STxCAR is written with the peroid value: Corresponds to the switching cycle of the buck circuit.
- Compare 0 value register STxCMP0V is written with the duty cycle values: The duty cycle

value is assigned in the period event interrupt, and updated at the period.

- Repetition counter: 0;
- Set request: Period event.
- Reset request: Comparator 0 event;
- Dead-time enable: DTEN=1;
- Dead-time value of falling edge or rising edge: Actual dead-time * HRTIMER_DTGCK.
- ADC trigger: Compare 1 event, during loop operation, make CMP1 equal to the latest CMP0/2.
- Interrupt event: Counter reset event, control bit is the RSTIE bit in STxDMAINTEN.
- Interrupt number: ST0 corresponds to the interrupt number HRTIMER_IRQ1, and loop computation and assignment are performed in the HRTIMER_IRQ1_IRQHandler.
- Shadow registers: Enabled;
- Update event: Generated by the counter reset event, the control bit is the UPRST bit in the STxCTL0 register, which means the new duty cycle value will be updated at the peroid.
- Enable the CH0 and CH1 output channels of ST0: Set the ST0CH0EN and ST0CH1EN bits to 1 in the CHOUTEN register.
- Enable ST0 counting: Set the ST0CEN bit to 1 in the MTCTL0 register.

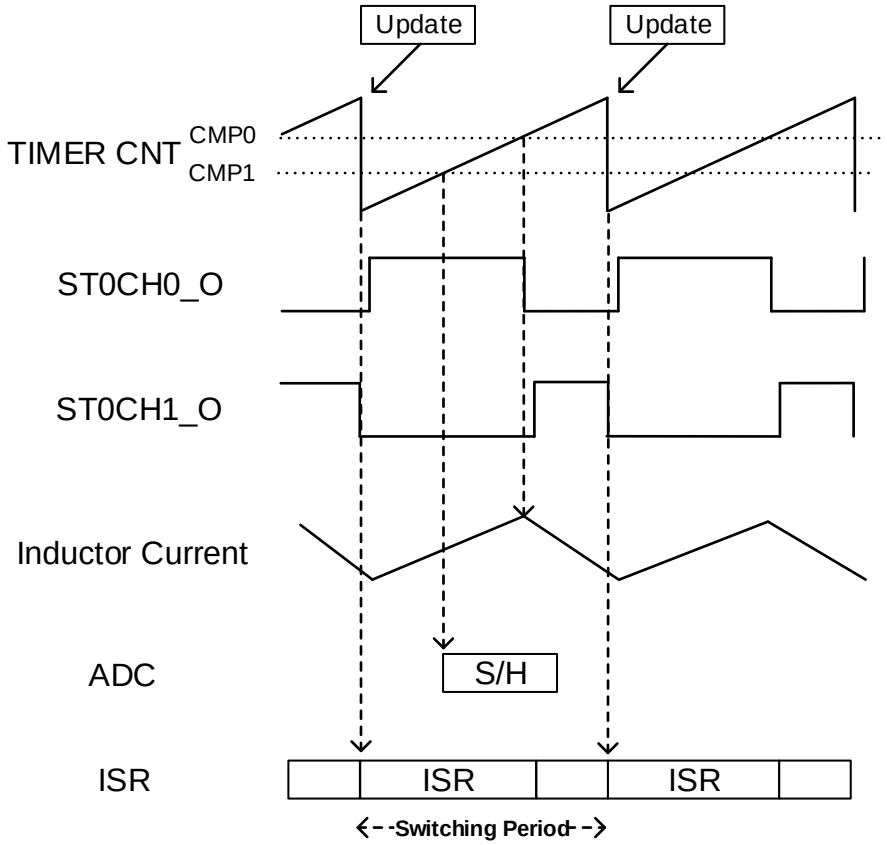
Table 5-1. Synchronous Buck Circuit PWM Configuration

Options	Control bits	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	0	
Counter auto reload register	STxCAR	Corresponds to the period value	Assign a value in ISR and update at the update event (period)
Compare 0 value register	STxCMP0V	Corresponds to the duty cycle	Assign a value in ISR and update at the update event (period)
Compare 1 value register	STxCMP1V	CMP1=CMP0 / 2	Trigger ADC sampling；Assign a value in ISR and update at the update event (period)
Set request	CH0SPER	Period event	
Reset request	CH0RSCMP0	Compare 0 event	
Dead-time enable	DTEN	1	After dead-time is enabled, CH0 and CH1 automatically output complementary waveforms
Falling edge dead-time value , rising edge dead-time value	DTFCFG[8:0]/ DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	

Options	Control bits	Specific configuration	Note
Interrupt request	RSTIE	Counter reset event	ST0 corresponds to the interrupt number HRTIMER_IRQ1
Shadow registers	SHWEN	Enabled	
Update event	UPRST	Counter reset event （period）	
Channel output enable	STxCH0EN	Enabled	
Counter enable	STxCEN	Enabled	

As shown in ***Figure 5-2. Synchronous Buck Circuit PWM Generation***, PER and CMP0 are used for PWM generation, while CMP1 is used to trigger ADC sampling. The ADC is configured to trigger sampling conversion at the midpoint of the output's on-time. At the period event, an interrupt request is generated, and the loop operation is performed in the interrupt service routine. The loop operation will derive a new duty cycle and a new sampling point (half of the duty cycle), which are written to the shadow register and updated in the next period.

Figure 5-2. Synchronous Buck Circuit PWM Generation



By updating the value of CMP1 continuously, it is possible to achieve the sampling of midpoint current/midpoint voltage, and to avoid the oscillations generated during the on and off of the switching tube, ensuring the accuracy of the sampling.

If the PWM frequency is too high or the loop operation takes too long, the repetition counter can be set to n, and the interrupt request can be changed to a repetition event. This will cause an interrupt to be entered only once every n+1 switching cycles, and a control loop calculation and control quantity update will be performed once.

At the same time, the relevant ADC trigger configuration needs to be carried out in the ADC and TRIGSEL modules.

PWM configuration: The trigger source is set to ADCTRIG0, and the trigger event is set to ST0CMP1.

TRIGSEL configuration: Trigger input selection HRTIMER_ADCTRIG0 (0x7D).

ADC0 trigger configuration: Externally trigger ADC0 regular group; ADC0 triggers Comparator 1 event, during loop operation, making CMP1 equal to half of the latest calculated value of CMP0.

5.4. Reference Code

Taking the synchronous Buck circuit configuration code as an example, with a main frequency of 216MHz, a switching frequency of 100kHz, a control frequency of 100kHz, and a dead-time of 200ns, the reference code is as shown in ***Table 5-2. Synchronous Buck Circuit Configuration Code***.

Table 5-2. Synchronous Buck Circuit Configuration Code

```
/* 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. Multi-phase Interleaved Buck Circuit Example

6.1. Focus of Configuration

The focus of this case configuration is on multiphase interleaving.

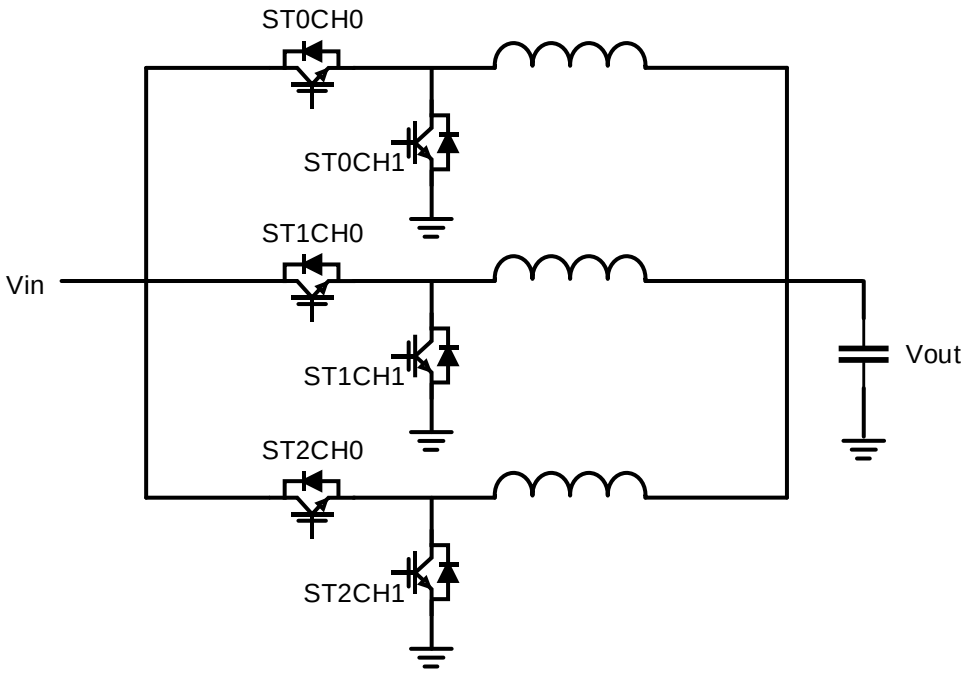
- Global counter reset event interrupt
- Phase interleaved configuration
- Event-triggered counter reset
- Reset events for individual counters update.

6.2. Topological Structure

The multi-phase interleaved parallel technology can be applied to various power conversion topologies (buck, boost, LLC). It has the advantages of reducing ripple, lowering EMI, and improving efficiency at light load. For multi-phase interleaved buck, generally, the relative phase shift of each phase is 360° divided by the number of phases. Taking the three-phase as an example, each phase is staggered by 120°.

Three-phase interleaved Buck topology structure as shown in ***Figure 6-1. Three-phase Interleaved Buck Topology***.

Figure 6-1. Three-phase Interleaved Buck Topology



6.3. Specific Configuration

Take ST0, ST1, and ST2 as an example with a phase difference of 120°. The phase-related configuration is shown in ***Table 6-1. STx Phase Configuration***:

Table 6-1. STx Phase Configuration

TIMER	ST0	ST1	ST2
Phase/degree	0	120	240
Counter reset	Automatic reload at cycle	Triggered by ST0's CMP1	Triggered by ST0's CMP3
Set request	Period event	ST1 CMP1	ST2 CMP1
Reset request	ST0 CMP0	ST1 CMP0	ST2 CMP0
Update event	ST0 Period event	ST1 Period event	ST2 Period event
CAR value	Period value	0xFFFF	0xFFFF
Compare 0 value	Duty cycle	Duty cycle	Duty cycle
Compare 1 value	Period value *1/3	DLL*3	DLL*3
Compare 2 value	Trigger sampling	Trigger sampling	Trigger sampling
Compare 3 value	Period value *2/3		

ST0 operates in continuous mode, with its wave generation and period being the same as the single-phase buck wave generation. ST1 counter is reset by the CMP1 event of ST0. ST2 counter is reset by the CMP2 event of ST0. The duty cycle is programmed into each timer, independent of phase shift, and can vary during operation. The implementation of phase difference is mainly as follows:

- The CAR value of ST0 is configured as a period value, and the CMP0 value is configured as a duty cycle value.
- The CAR value of ST1 is configured as 0xFF, and the CMP0 value is set to the duty cycle value.
- The CAR value of ST2 is set to 0xFF, and the CMP0 value is set to the duty cycle value.
- The CMP1 event of ST0 triggers the reset of ST1 counter, thereby determining the phase difference and period value of ST1, ST0CMP1 = Period value * 1/3.
- The CMP3 event of ST0 triggers the reset of ST1 counter, thereby determining the phase difference and cycle value of ST1, ST0CMP3 = Period value * 2/3.
- ST0, ST1, and ST2 are updated at their respective periodic event points.
- ST0, ST1, ST2 are uniformly processed in the loop operation within the interrupt service routine of ST0.

The specific configuration of each phase's PWM is as follows, for details, please refer to ***Table 6-2. Multiphase Interleaved Buck Circuit PWM Configuration***:

- Timer STx is configured in continuous mode.
- Counter auto reload register STxCAR is written with period value: Corresponds to the switching cycle of the buck circuit.
- Compare 0 value register STxCMP0V is written with duty cycle value: The duty cycle

value is assigned during the repetition event interrupt, and updated at the period.

- Repetition counter: 0;
- Set request:
  ST0: Period event;
  ST1: ST1 CMP1;
  ST2: ST2 CMP1;
- Reset request: Compare 0 event.
- Dead-time enable: DTEN=1;
- Dead-time falling edge value, rising edge value: Actual dead-time * HRTIMER_DTGCK.
- ADC trigger: Compare 2 event, during loop operation, make CMP2 equal to the latest CMP0/2.
- Interrupt event: Counter reset event, control bit is the RSTIE bit in STxDMAINTEN.
- Interrupt number: ST0 corresponds to the interrupt number HRTIMER_IRQ1, and loop computation and assignment are performed in the HRTIMER_IRQ1_IRQHandler.
- Shadow register: Enabled;
- Update event: Triggered by the counter reset event, the control bit is the UPRST bit of the STxCTL0 register, that is, the new duty cycle value will be updated at the period.
- Enable the CH0 and CH1 output channels of STx: Set the STxCH0EN and STxCH1EN bits in the CHOUTEN register to 1.
- Enable STx counting: Set the ST0CEN, ST1CEN, and ST2CEN bits to 1 in the MTCTL0 register.

Table 6-2. Multiphase Interleaved Buck Circuit PWM Configuration

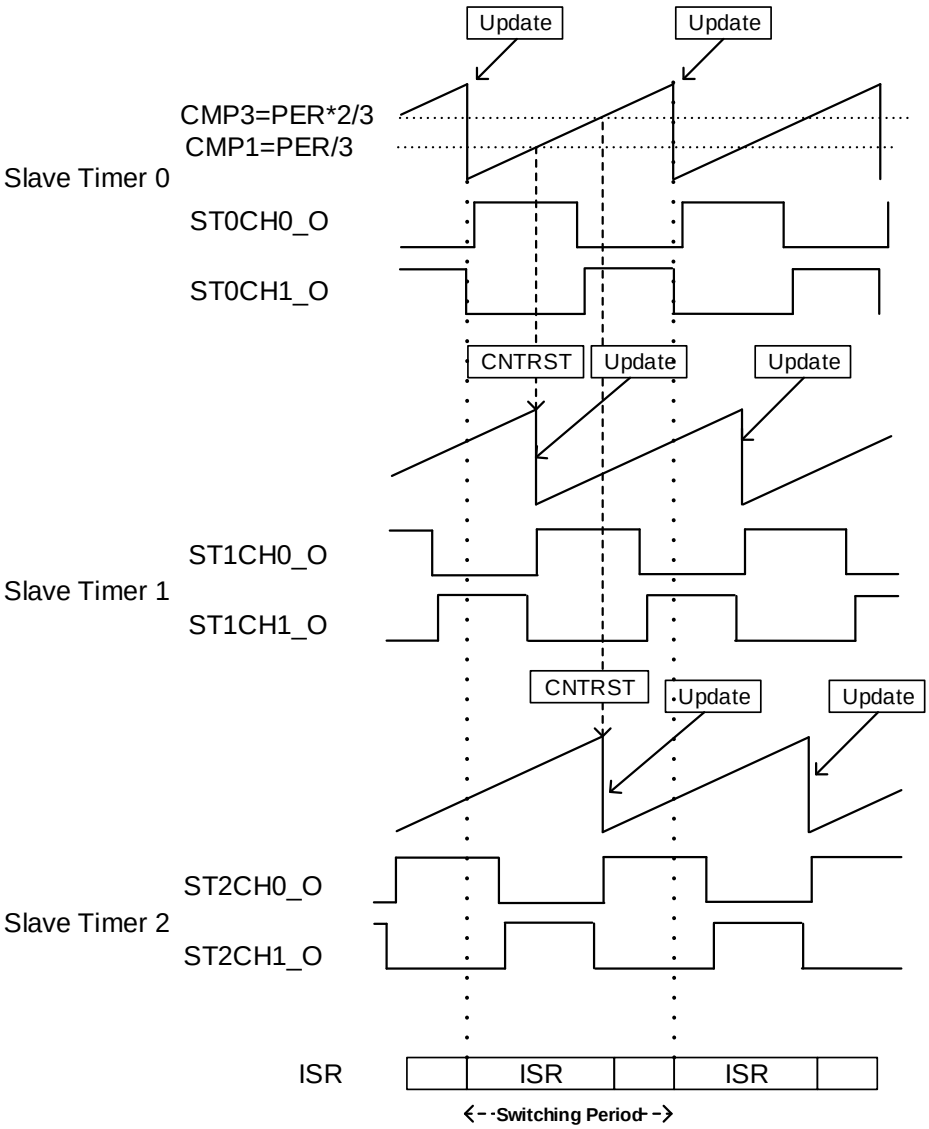
Options	Control bits	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	0	
Counter auto reload register	STxCAR	Corresponds to the period value	Assign a value in ISR and update at the update event (period)
Compare 0 value register	STxCMP0V	Corresponds to the duty cycle	Assign a value in ISR and update at the update event (period)
Compare 2 value register	STxCMP2V	CMP2=CMP0 / 2	Trigger ADC sampling. Assign a value in ISR and update at the update event (period)
Set request	CH0SPER	Period event/ ST1 CMP1/ ST2 CMP1	
Reset request	CH0RSCMP0	Compare 0 event	
Dead-time enable	DTEN	1	After dead-time is enabled, CH0 and CH1 automatically output complementary waveforms

Options	Control bits	Specific configuration	Note
Falling edge dead-time value , rising edge dead-time value	DTFCFG[8:0]/ DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
Interrupt request	RSTIE	Counter reset event	ST0 corresponds to the interrupt number HRTIMER_IRQ1
Shadow registers	SHWEN	Enabled	
Update event	UPRST	Counter reset event （period）	
Channel output enable	STxCH0\1EN	Enabled	
Counter enable	STxCEN	Enabled	

As shown in ***Figure 6-2. Synchronous Buck Circuit PWM Generation***, PER and CMP0 are used for PWM generation, while CMP2 is used to trigger ADC sampling, and the ADC is configured to trigger sampling conversion at the midpoint of the output's conduction time.

During the period event of ST0, an interrupt request is generated, and the loop operation is performed in the interrupt service routine. The loop operation will yield the new duty cycles and new sampling points (half of the duty cycle) for ST0, ST1, and ST2, which are written into the shadow registers of ST0, ST1, and ST2, respectively, and updated at their respective periods.

Figure 6-2. Synchronous Buck Circuit PWM Generation



By updating the value of CMP2 continuously, it is possible to achieve the sampling of midpoint current/midpoint voltage, and to avoid the oscillations generated when the switching tube is turned on and off, ensuring the accuracy of the sampling.

If the PWM frequency is too high or the loop operation takes too long, the repetition counter can be set to n, and the interrupt request can be changed to a repetition event. This will cause an interrupt to be entered only once every n+1 switching cycles, and a control loop calculation and control quantity update will be performed once.

At the same time, the ADC and TRIGSEL modules need to perform the relevant ADC trigger configuration.

PWM configuration: The trigger source is set to ADCTRIG0, and the trigger event is set to ST0CMP2.

TRIGSEL Configuration: Trigger input selection HRTIMER_ADCTRIG0 (0x7D).

ADC0 trigger configuration: Externally trigger ADC0 regular group; ADC0 triggers Comparator

1 event, during loop operation, makes CMP2 equal to half of the latest calculated value of CMP0.

6.4. Reference Code

Taking the three-phase interleaved buck circuit configuration as an example, with a main frequency of 216MHz, a switching frequency of 100kHz, a control frequency of 100kHz, and a 200ns dead-time, the reference code is as shown in ***Table 6-3. Three-phase Interleaved Buck Circuit Configuration Code***.

Table 6-3. Three-phase Interleaved Buck Circuit Configuration Code

```
/* 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_para);
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 Circuit Example

7.1. Focus of Configuration

The focus of this example configuration is:

- Repeated event interruption
- Frequency conversion control
- Counter reset event update.

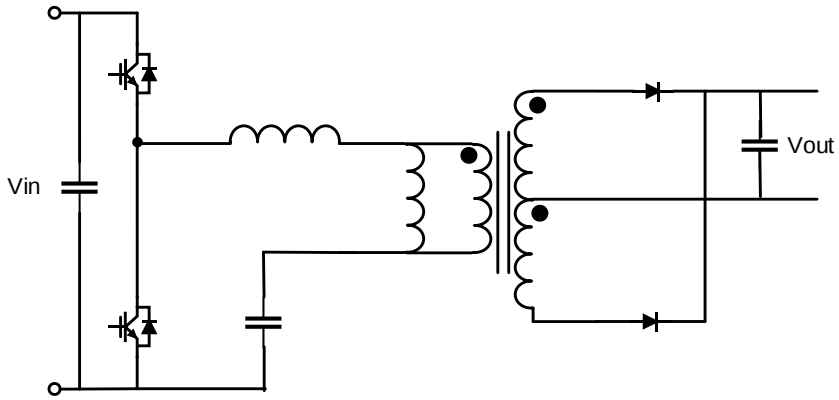
7.2. Topological Structure

The LLC circuit is a very popular circuit topology in industry in recent years, due to its soft switching characteristics, it has features such as high frequency and high efficiency. It is widely used in server power, portable energy storage, mobile phone fast charging, EV charging and other scenarios.

There are various control methods for LLC, but it usually works at a fixed 50% duty cycle and a fixed phase, with power control achieved by adjusting the frequency.

Take the half-bridge LLC as an example, its topological structure is shown in ***Figure 7-1. LLC Topology Structure***:

Figure 7-1. LLC Topology Structure



7.3. Specific Configuration

Take ST0 as an example, the specific configuration of PWM is as follows, for details, please refer to ***Table 7-1. LLC Circuit PWM Configuration***:

- Timer ST0 is configured in continuous mode.
- The counter auto reload register STxCAR is written with the cycle value: Corresponds to the LLC circuit switching cycle.

- Compare 0 value register STxCMP0V is written with duty cycle value: The duty cycle value is assigned in the repetitive event interrupt, and updated at the period.
- Repetition counter: 2;
- Set request: Period event.
- Reset request: Compare 0 event, 50% cycle value.
- Dead-time enable: DTEN=1;
- Dead-time falling edge value, rising edge value: Actual dead-time * HRTIMER_DTGCK.
- Interrupt event: Repetition event, control bit is the REPIE bit in STxDMAINTEN.
- Interrupt number: ST0 corresponds to the interrupt number HRTIMER_IRQ1, and loop computation and assignment are performed in the HRTIMER_IRQ1_IRQHandler.
- Shadow register: Enabled;
- Update event: Triggered by the counter reset event, the control bit is the UPRST bit of the STxCTL0 register, that is, the new duty cycle value will be updated at the period.
- Enable the CH0 and CH1 output channels of ST0: Set the ST0CH0EN and ST0CH1EN bits in the CHOUTEN register to 1.
- Enable ST0 counter: Set ST0CEN bit to 1 in MTCTL0 register.

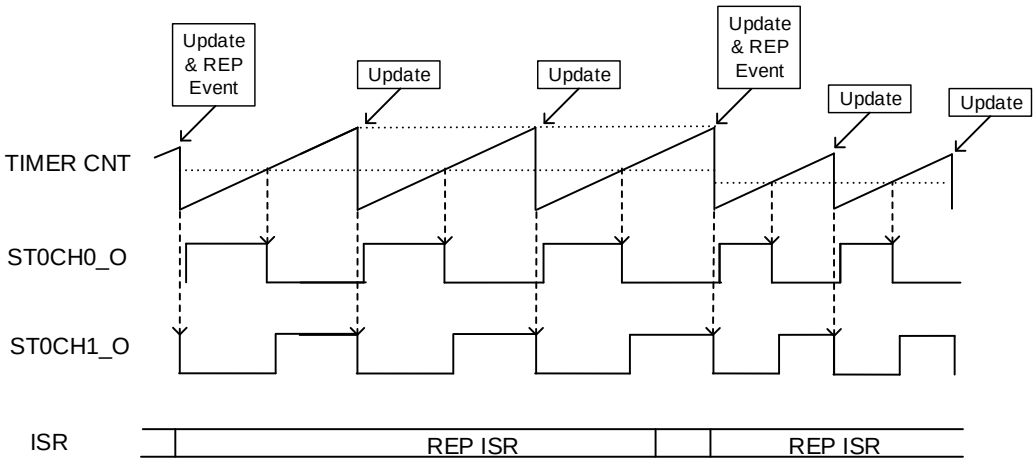
Table 7-1. LLC Circuit PWM Configuration

Options	Control bits	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	2	
Auto reload register	STxCAR	Period value	Assign a value in ISR and update at the update event (period)
Compare 0 value register	STxCMP0V	CMP0=PER / 2	Assign a value in ISR and update at the update event (period)
Set request	CH0SPER	Period event	
Reset request	CH0RSCMP0	Compare 0 event	
Dead-time enable	DTEN	1	After dead-time is enabled, CH0 and CH1 automatically output complementary waveforms
Falling edge dead-time value 、Rising edge dead-time value	DTFCFG[8:0]/ DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
Interrupt request	REPIE	Repetition event	ST0 corresponds to the interrupt number HRTIMER_IRQ1
Shadow registers	SHWEN	Enabled	
Update event	UPRST	Counter reset event （period）	

Options	Control bits	Specific configuration	Note
Channel output enable	STxCH0EN	Enabled	
Counter enable	STxCEN	Enabled	

As shown in ***Figure 7-2. LLC Circuit PWM Generation***, PER and CMP0 are used for PWM generation. During the repetition event, an interrupt request is generated, and in the interrupt service routine, the loop operation is performed, which results in a new period value and CMP0 value (half a cycle), written into the shadow register, and updated in the next period.

Figure 7-2. LLC Circuit PWM Generation



By calculating the loop once every 3 cycles and updating the values of PER and CMP0 simultaneously, the frequency modulation control of LLC can be achieved.

If the switching frequency is too high or the loop computation takes too long, the repetition counter can be set larger, for example, to n, so that the interrupt will be entered and the control loop calculation and update of the control amount will be performed only once every n+1 switching periods.

7.4. Reference Code

Translate the sentence into English, only return the translation result: Taking 216MHz main frequency, switching frequency of 100kHz, and 200ns dead-time as an example, the reference code is as shown in ***Table 7-2. LLC Circuit Configuration Code***:

Table 7-2. LLC Circuit Configuration Code

```
/* 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-bridge Inverter Circuit Example

8.1. Focus of Configuration

The focus of this example configuration is:

- Counter reset event interrupt
- SPWM modulation
- synchronous start
- Counter reset event update

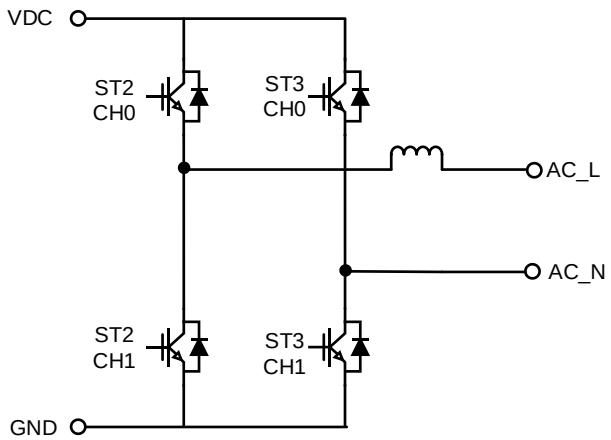
8.2. Topological Structure

The H-bridge inverter is a common topology for single-phase inverters and is widely used in bidirectional energy storage, UPS, and other scenarios.

The H-bridge inverter usually adopts the SPWM modulation method. Common SPWM modulation methods include unipolar modulation and bipolar modulation. This example uses unipolar SPWM modulation to generate waves.

Take the H-bridge as an example, its topological structure is shown in ***Figure 8-1. H-bridge Inverter Topology Structure***:

Figure 8-1. H-bridge Inverter Topology Structure



The wave generation method is as follows: During the positive half of the working frequency period, ST2's CH0 and CH1 are complementary at high frequency (switching frequency), while ST3's CH0 and CH1 are complementary at low frequency (50Hz); During the negative half of the working frequency period, ST3's CH0 and CH1 switch at high frequency, and ST2's CH0 and CH1 switch at low frequency (50Hz).

8.3. Specific Configuration

Taking the above wave-emitting method as an example, the PWM configurations for ST2 and ST3 are as follows, for details, please refer to ***Table 8-1. H-bridge Inverter Circuit PWM Configuration***:

- Timer ST2 and ST3 are configured in continuous mode.
- The counter auto reload register STxCAR is written with the period value: Corresponds to the switch cycle.
- Compare 0 value register STxCMP0V is written with (PER - duty) / 2.
- Compare 1 value register STxCMP1V is written with (PER + duty) / 2;
- Repetition counter: 0;
- Set request: Compare 0 event.
- Reset request: Compare 1 event;
- Dead-time enable: DTEN=1;
- Dead-time falling edge value, rising edge value: Actual dead-time * HRTIMER_DTGCK.
- Interrupt event: Counter reset event, control bit is the RSTIE bit in STxDMAINTEN.
- Interrupt number: ST2 corresponds to the interrupt number HRTIMER_IRQ3, and loop computation and assignment are performed in the HRTIMER_IRQ3_IRQHandler.
- Shadow register: Enabled;
- Update event: Triggered by a counter reset event, the control bit is the UPRST bit of the STxCTL0 register, that is, the new duty cycle value will be updated at the period.
- Enable the CH0 and CH1 output channels of ST2 and ST3: Set the ST2CH0EN, ST2CH1EN, ST3CH0EN, and ST3CH1EN bits in the CHOUTEN register to 1.
- Enable ST2 and ST3 counting: Set the ST2CEN and ST3CEN bits in the MTCTL0 register simultaneously.

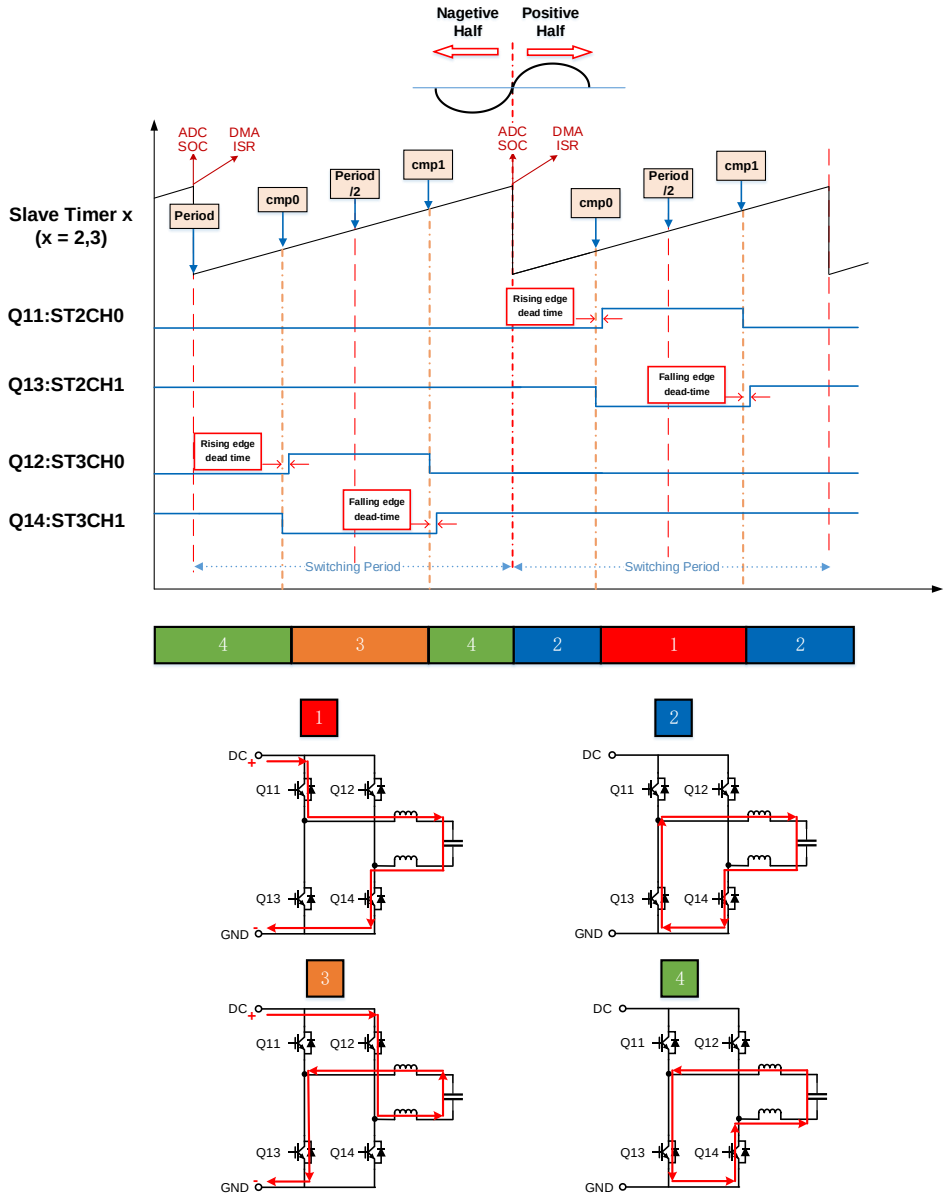
Table 8-1. H-bridge Inverter Circuit PWM Configuration

Options	Control bits	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	0	
Counter auto reload register	STxCAR	Period value	Assign a value in ISR and update at the update event (period)
Compare 0 value register	STxCMP0V	CMP0=(PER – duty)/2	Assign a value in ISR and update at the update event (period)
Compare 1 value register	STxCMP1V	CMP1=(PER + duty)/2	Assign a value in ISR and update at the update event (period). Pay attention to operation overflow;
Set request	CH0SCMP0	Compare 0 event	

Options	Control bits	Specific configuration	Note
Reset request	CH0RSCMP1	Compare 1 event	
Dead-time enable	DTEN	1	After dead-time is enabled, CH0 and CH1 automatically output complementary waveforms
Falling edge dead-time value 、Rising edge dead-time value	DTFCFG[8:0]/ DTRCFG[8:0]	Actual dead-time *HRTIMER_DTGCK	
Interrupt request	REPIE	Repetition event	ST0 corresponds to the interrupt number HRTIMER_IRQ1
Shadow registers	SHWEN	Enabled	
Update event	UPRST	Counter reset event （period）	
Channel output enable	STxCH0EN	Enabled	
Counter enable	STxCEN	Enabled	Need to enable synchronously

As shown in ***Figure 8-2. PWM Generation for H-Bridge Inverter Circuit***, CMP0 and CMP1 are used for PWM generation. At the timing of the period event, an interrupt request is generated, and the loop operation is performed in the interrupt service routine. The loop operation will yield new values for CMP0 and CMP1, which are written into the shadow registers and updated at the next period.

Figure 8-2. PWM Generation for H-Bridge Inverter Circuit



8.4. Reference Code

Take the example of a 216MHz main frequency, a switching frequency of 100kHz, and a 200ns dead-time, the reference code is as shown in ***Table 8-2. H-bridge Inverter Circuit Configuration Code***:

Table 8-2. H-bridge Inverter Circuit Configuration Code

```
/* 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. Slope Compensation Example

9.1. Focus of Configuration

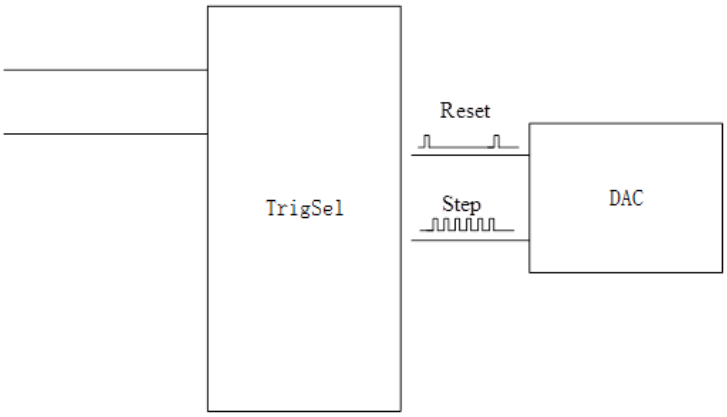
The focus of this example configuration is:

- Counter reset
- The DAC resets the trigger source
- DAC step trigger source

9.2. Compensation Methods

Slope compensation requires a saw-tooth wave signal that is synchronized with PWM, which can be generated by the MCU triggering the DAC. The trigger signal is produced by the TRIGSEL module to generate the reset and step signals for the DAC, as shown in ***Figure 9-1. TRIGSEL triggers DAC***. The reset signal of the saw-tooth wave needs to be synchronized with the switching PWM, and the reset signal of the DAC can be configured to be triggered by the HRTIMER or TIMER output channel.

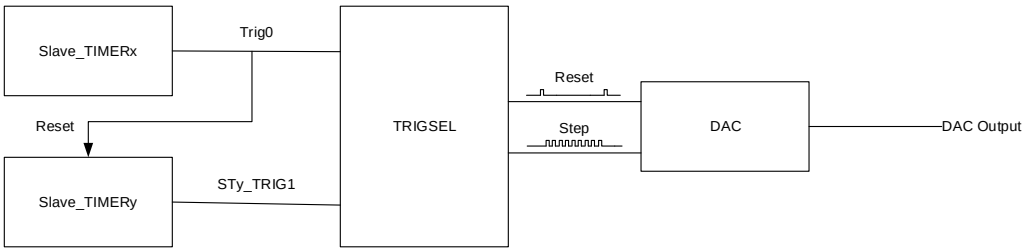
Figure 9-1. TRIGSEL triggers DAC



Slope compensation requires the MCU to provide synchronized PWM decrementing saw-tooth wave signals. Methods to generate saw-tooth waves include:

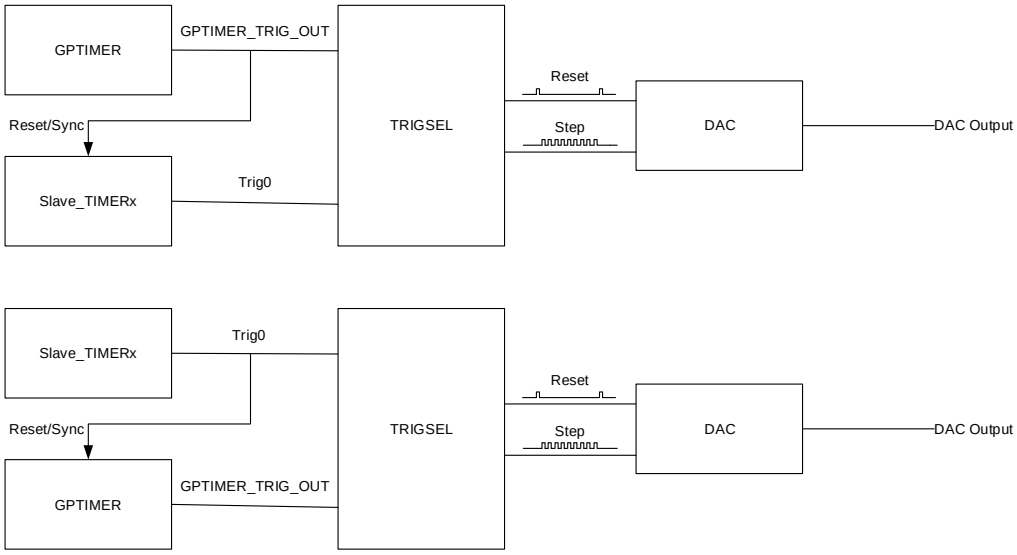
- Using two STs, where STx generates a saw-tooth wave reset signal, and STy generates a saw-tooth wave step signal, as shown in ***Figure 9-2. Dual ST triggers DAC***:

Figure 9-2. Dual ST triggers DAC



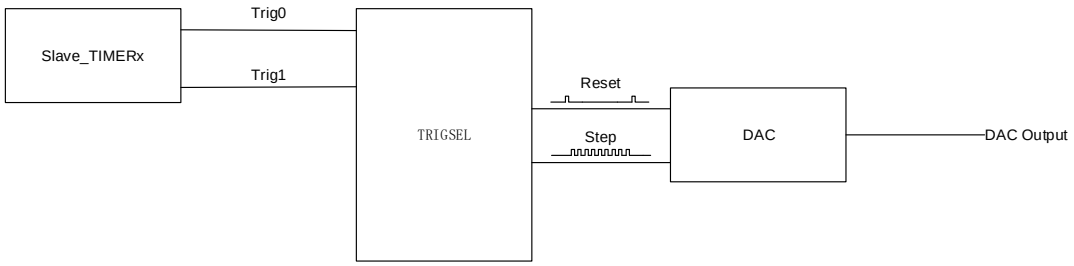
- Using one ST and one GPTimer, STx generates a saw-tooth wave reset/step signal, and the GPTimer synchronized with STx generates a saw-tooth wave step/reset signal, as shown in ***Figure 9-3. ST+GPTimer triggers DAC***:

Figure 9-3. ST+GPTimer triggers DAC



- Using the HRTIMER dual-channel trigger mode, configure the TRIG0M and TRIG1M bits in the HRTIMER_STxCTL1 register to 0, as shown in ***Figure 9-4. ST Double Source Trigger DAC***:

Figure 9-4. ST Double Source Trigger DAC



This example uses dual ST triggers DAC to generate a saw-tooth wave.

9.3. Specific Configuration

This example uses ST0 to generate the reset signal for the ramp compensation DAC, and ST4 to generate the step signal for the ramp compensation DAC. The configurations for ST0 and ST4 are as follows, for details, please refer to ***Table 9-1. Slope Compensation PWM Configuration***:

- HRTIMER clock is quadrupled;
- The timer ST0 is configured in continuous mode.
- The counter auto reload register ST0CAR is written with a period value of 24000, corresponds to the switching period; ST4CAR is written with a period value of 4320, corresponds to a 50us period.
- Repetition counter: 0;
- Set request of ST0: Period event.
- Reset request of ST0: Compare 2 event;
- Compare 0 value register STxCMP0V is written with (PER – 15).
- Compare 2 value register STxCMP2V is written with (PER / 2);
- Shadow register of ST0: Disabled;
- Shadow register of ST4: Enabled;
- Update event of ST4: Triggered by the counter reset event, the control bit is the UPREP bit of the STxCTL0 register.
- Enable the CH0 output channels of ST0: Set the ST0CH0EN bits in the CHOUTEN register to 1.
- Enable ST0 and ST4 counting: Set the ST0CEN and ST4CEN bits in the MTCTL0 register simultaneously.

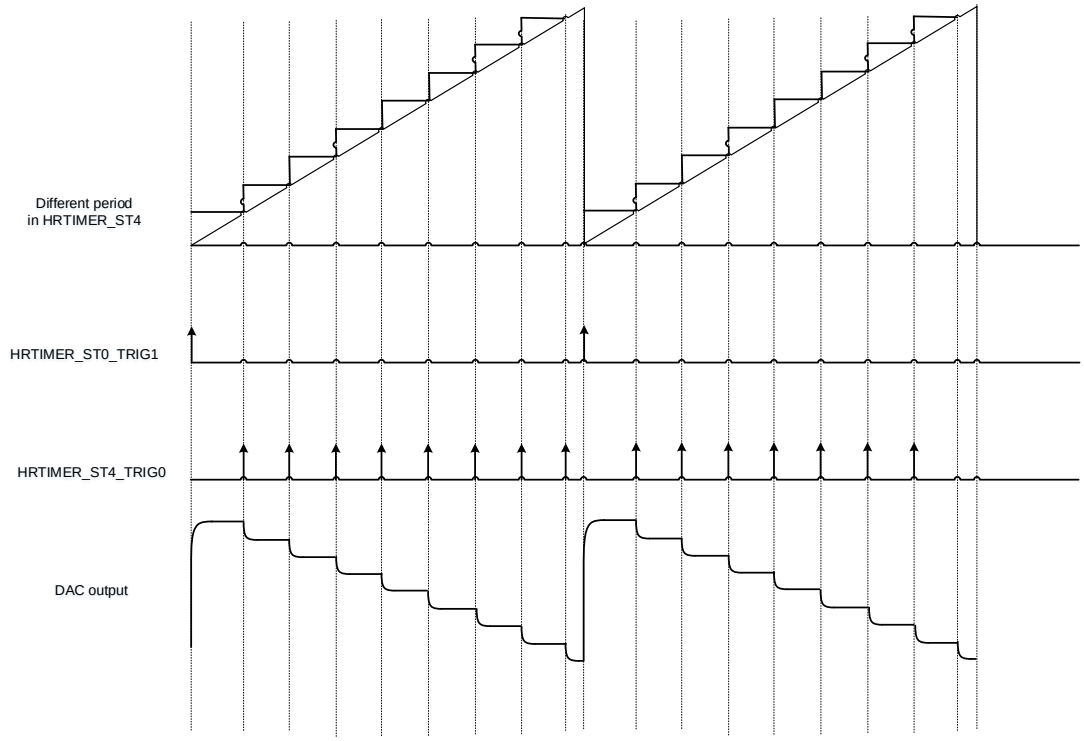
Table 9-1. Slope Compensation PWM Configuration

Options	Control bits	Specific configuration	Note
Operating mode	CTNM	Continuous mode	
Repetition counter	STxCREP	0	
Counter auto reload register	STxCAR	ST0:24000 ST4: 4320	The period value of ST0 corresponds to the PWM switching period. The period value of ST4 corresponds to the 50us step trigger signal.
Compare 0 value register	ST0CMP0V	CMP0=PER-15	
Compare 2 value register	ST0CMP2V	CMP2=PER/2	
Set request	CH0SPER	ST0:Period event	
Reset request	CH0RSCMP2	ST0: Compare 2 event	
Dead-time enable	DTEN	0	The Dead-time is disabled

Options	Control bits	Specific configuration	Note
Shadow registers	SHWEN	ST0:Disabled ST4:Enabled	
Update event	UPRST	ST0:NONE ST4: Counter reset event	
Channel output enable	STxCH0EN	Enabled	
Counter enable	STxCEN	Enabled	Need to enable synchronously

As shown in ***Figure 9-5. Signal Generation for Slope Compensation***, ST0_TRIG1 is used to generate the ramp reset signal, and ST4_TRIG0 is used to generate the ramp step signal. The reset signal for the ST4 counter is generated by the CMP0 event of ST0.

Figure 9-5. Signal Generation for Slope Compensation



9.4. Reference Code

Taking 216MHz master frequency and 36kHz switching frequency as an example, the reference code is as shown in ***Table 9-2. Slope Compensation Configuration Code***.

Table 9-2. Slope Compensation Configuration Code

```
/* 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, HRTIMER_COMPARE2, &comparecfg_para);
comparecfg_para.compare_value = 23985;
hrtimer_slavetimer_waveform_compare_config(HRTIMER0, HRTIMER_SLAVE_TIMER0, HRTIMER_COMPARE0, &comparecfg_para);
/* configures double 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_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_TRIG1);
……}
```

10. Other Examples

For more examples, please refer to www.gd32mcu.com or contact your local dealer.

11. Revision history

Table 11-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Aug 13, 2024
1.1	Added example of Slope Compensation	Nov 18, 2024

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 parties referred to therein (if any) are the property of their respective owners and are 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 have 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 its 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 its 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.

The 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 the products and services described herein at any time, without notice.