AN4539
Application note

HRTIM cookbook

Introduction

Several STM32 microcontrollers address market segments requiring digital signals with highly accurate timings, namely digital power supplies, lighting, non-interruptible power supplies, solar inverters and wireless chargers. This is possible thanks to a high resolution timer (HRTIM) peripheral able to generate up to twelve signals and to handle a large variety of input signals for control, synchronization or protection purposes. Its modular architecture makes possible to address several conversion topologies and multiple parallel converters, with the possibility to reconfigure them during run-time.

The HRTIM is available in the products listed in *Table 1*.

This peripheral may appear complex, mostly because of the large control register set. To complement the extensive description provided in reference manuals, this document includes quick-start informations and a collection of examples.

In the first section, this cookbook aims to show that HRTIM programming is simple. The environment (the kitchen) setup is first explained, followed by a number of simple examples given for understanding by practice. These basic cases introduce step by step the timer features and provide programming guidelines. This section must be read with attention by people not familiar with the HRTIM.The second part is a collection of converter recipes to use when starting a new design, either to pick up a ready-made code example, or to get ideas and programming tricks when dealing with a topology not described in this document. This cookbook does not cover the converter design itself (control techniques and components dimensioning), described in dedicated application notes.

Each example comes with a brief converter description, the control waveform(s) and a code snippet. These snippets (and the equivalent code done based the STM32 HAL library and/or LL libraries) can be downloaded from *www.st.com*.

Table 1. Applicable products

Type	Part numbers or product lines
Microcontrollers	STM32F334C4, STM32F334C6, STM32F334C8, STM32F334K4, STM32F334K6, STM32F334K8, STM32F334R4, STM32F334R6, STM32F334R8
	STM32H742, STM32H743/753, STM32H745/755 and STM32H747/757 lines, STM32H750IB, STM32H750VB, STM32H750XB, STM32H753AI, STM32H753BI, STM32H753II, STM32H753VI, STM32H753XI, STM32H753ZI, STM32H755BI, STM32H755II, STM32H755XI, STM32H755ZI, STM32H757AI, STM32H757BI, STM32H757II, STM32H757XI, STM32H757ZI
	STM32G474CB, STM32G474CC, STM32G474CE, STM32G474MB, STM32G474MC, STM32G474ME, STM32G474QB, STM32G474QC, STM32G474QE, STM32G474RB, STM32G474RC, STM32G474RE, STM32G474VB, STM32G474VC, STM32G474VE, STM32G484CE, STM32G484ME, STM32G484QE, STM32G484RE, STM32G484VE

Contents

List of tables

List of figures

1 Getting the kitchen ready

The microcontrollers concerned by this application note are based on Arm® cores[a].

This section details the ingredients needed before starting, so that the user can focus on the HRTIM programming.

Appendix C: Reference documents lists the documents related to the HRTIM and to the peripherals used in this document. A preliminary reading of the HRTIM section in the product reference manual is recommended.

1.1 Prerequisites

Before enjoying the flavors of the HRTIM, there are some prerequisites. It is expected from the reader basic C programming skills and minimal experience on MCUs and development environments, as well as a theoretical background on switched mode power supplies. Control strategies and components dimensioning details are exceeding the scope of this application note, they are available in literature.

For the sake of simplicity, this cookbook only considers logic signals or analog voltages that can be directly handled by the MCU, so as to be voltage level agnostic. However some references are made to external components and side effects from power switchings, whenever the timer or MCU has some features to handle them.

Last, it is reminded that it is required to have power applications operated by skilled technical personnel to avoid risks of electrical shocks, burns or even death, should the MCU be used in applications with hazardous voltage levels.

1.2 Hardware set-up

The STM32F334 Discovery kit is a very affordable tool and is the best option to start (and go on) experimenting with the HRTIM (order code STM32F3348-DISCO). It includes the programming interface and a USB cable is the only necessary additional material to have the chip programmed and debugged. All I/Os are made available on 2.54 mm spaced pins so that it can also be connected to a perfboard / stripboard / breadboard. The kit also features two power converters: an inverted buck for LED drive and a low-voltage buck/boost converter with independent inputs and outputs.

An oscilloscope is mandatory, eventually coupled with a logic analyzer for the configurations where more than four channels must be monitored. To visualize the subtle high-resolution steps, the oscilloscope must have a sampling rate above 1 GS/s at least with an option to have interleaved acquisition, to increase the timing accuracy.

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

One or several function generator are also of great help to emulate the feedbacks from the power converter (either logic pulses or analog signals) during the early debugging phases. The generator must have a trigger input for some specific cases. If missing, it is also possible to have the feedback signal emulated by the HRTIM itself using free timing units, with some more coding effort (or software example reuse).

1.3 Tools set-up

It is necessary to have a compiler installed (all demonstrations are fitting within 32 Kbytes), as well as an IDE supporting the ST-LINK-V2/ST-LINK-V3 debug interfaces.

The code snippets given below are compiler agnostic, so they simply need to be copied into a generic HRTIM project template for any kind of toolchain.

The software sources are delivered with workspaces for the following toolchains:

- IAR™ (EWARM 7.10.3)
- Keil® (MDK-ARM 4.7)
- System Workbench for STM32 (SW4STM32).

1.4 HRTIM versions

Since its first introduction, the HRTIM peripheral has been improved. Today it is integrated in the products listed in *Table 2*. When needed, its revision is indicated as HRTIMv1 and HRTIMv2, or simply v1 and v2.

Table 2. HRTIM features according to products where it is integrated

Part numbers	HRTIM revision	Reference	Main changes
STM32F334xx	V1.0	HRTIMv1	-
STM32H74xxx STM32H75xxx	V1.1		No DLL
STM32G47xxx STM32G48xxx	V2.0	HRTIMv2	– Addition of a sixth timer and a sixth FAULT input – New operating modes Refer to *Appendix A: HRTIM v2* for further details.

Appendix B: Software migration explains how to handle the HRTIM revision change when switching from a product to another.

1.5 MCU and HRTIM set-up

1.5.1 System clock initialization

To provide the high-resolution, the HRTIM must be fed directly by the PLL high-frequency output. The clock period is divided by 32 by the DLL (when available) to provide high-resolution, as if the HRTIM clock frequency were multiplied by 32.

Two options are available, using either the crystal-based high-speed external (HSE) oscillator, or the high-speed internal (HSI) oscillator.

Table 3 summarizes the allowed frequency ranges for each product.

Table 3. HRTIM input frequency operating range

Part numbers	PLL input	Input frequency (MHz)		HRTIM frequency (GHz)		Resolution (ps)	
		Min	Max	Min	Max	Min	Max
STM32F334xx	HSE	128	144	4.096	4.608	217	244
	HSI	128		4.096			
STM32H74xxx STM32H75xxx	Any	-	480	-	0.480	2.08 ns (no DLL)	-
STM32G47xxx STM32G48xxx		100	170	3.20	5.44	184	312

Clock initialization is done in the main routine using the *SystemClock_Config()* function right after the HAL libray initialization (*HAL_Init*).

The CPU clock is also derived from the PLL, after a division by two, so that it can be up to half of the PLL output frequency. It can also be reduced to decrease MCU consumption while keeping the high-resolution functional.

The *SystemCoreClockUpdate* function can eventually be executed in the main to verify the CPU operating frequency: the frequency is updated in the SystemCoreClock variable.

1.5.2 HRTIM initialization

This section details step by step how the HRTIM is initialized, including individual function calls. Practically, this is done within the *HAL_HRTIM_Init* and *HAL_HRTIM_MspInit* routines.

HRTIM clock initialization

Once the MCU is up and running, the HRTIM must be clocked before being programmed. This is done using the RCC (reset and clock control) peripheral, in two steps:

1. selection of the high-speed PLL output for the HRTIM in RCC_CFGR3 register

```
__HAL_RCC_HRTIM1_CONFIG(RCC_HRTIM1CLK_PLLCLK);
```

2. clock enable for the registers mapped on the APB2 bus

```
__HRTIM1_CLK_ENABLE();
```

1.5.3 HRTIM DLL initialization

The HRTIM delay-locked loop (DLL) provides fine-grained timings and divides the high-frequency clock period in 32 evenly spaced steps.

This DLL must be calibrated at least once before high-resolution can be used. The calibration can be transparently relaunched by software during HRTIM operation if voltage or temperature conditions have changed. It is also possible to enable periodic calibration by hardware.

The code snippet below (for STM32F334) shows how the calibration is done. The high-resolution can be used once the DLLRDY flag has been set.

```
/* DLL calibration: periodic calibration enabled, period set to 14µs */
  HRTIM1->sCommonRegs.DLLCR = HRTIM_CALIBRATIONRATE_14| HRTIM_DLLCR_CALEN;
  /* Check DLL end of calibration flag */
  while(HRTIM1->sCommonRegs.ISR & HRTIM_IT_DLLRDY == RESET);
```

It is recommended to have the periodic calibration enabled, with the lowest calibration period (2048 x t_{HRTIM}) as default condition.

Note: This code causes the execution to stall if the DLL does not lock (typically when the HSE oscillator is not properly configured). The HAL library includes a function to perform the calibration that has a timeout verification and redirects in an error handler if necessary. This function is the one used in the HAL-based software example.

1.5.4 HRTIM I/Os initialization

The HRTIM inputs and outputs are mapped on standard I/O ports and must be programmed as any other I/O peripheral. The HRTIM alternate functions are split:

- on AF3 and AF13 alternate functions for STM32F334 and the STM32G4 products
- on AF1, AF2 and AF3 alternate functions for the STM32H7 products.

The HRTIM I/Os initialization must be done in two phases. The HRTIM inputs are initialized first, prior to the HRTIM registers, in the HAL_HRTIM_MspInit function.

The HRTIM outputs must be initialized after the HRTIM control registers programming (done in the *GPIO_HRTIM_outputs_Config* function in the examples), and once the counters are enabled. This is to ensure that the outputs states are correctly defined inside the HRTIM before passing the control from the GPIO circuitry to the HRTIM.

1.5.5 Other peripherals initialization

The HRTIM interacts with many of the MCU peripherals, as listed below. It is not mandatory to have all of them initialized to have the HRTIM operating. Initialization codes for the peripherals below are available in some of the examples described later.

Nested vectored interrupt controller (NVIC)

The HRTIM interrupts requests are grouped in seven (HRTIMv1) or eight (HRTIMv2) interrupts vectors. All faults are grouped within a distinct vector that can be set with a very high priority.

The NVIC part related to the HRTIM is programmed in the *HAL_HRTIM_MspInit* function.

DMA controller

Most of interrupt requests can be used as DMA requests and are grouped on six (HRTIMv1) or seven (HRTIMv2) DMA channels (one per timing unit, including the master timer).

DMA-based HRTIM operation is enabled when starting the timer, with specific start/stop function such as *HAL_HRTIM_WaveformCounterStart_DMA*.

Refer to *Section 2* for more details.

Comparators

The built-in comparators can be used to condition analog signals: they must be initialized before the output is routed to the HRTIM.

The initialization includes the analog inputs programming, clock enable and polarity.

Operational amplifier

The built-in operational amplifiers can amplify low voltage signals to be routed to the ADC or to the comparators. They must be initialized similarly to the comparator.

ADCs

The HRTIM can trigger any of the ADCs. They must be initialized to receive external triggers, on their regular and/or injected sequencers.

Another possible use of the ADCs consists in using the analog watchdog to trigger external events on the HRTIM (for output set/reset or counter reset purposes).

DACs

The DACs are used to define the comparator thresholds. They can be updated synchronously with HRTIM operation by means of HRTIM DAC triggers.

General purpose timers

The HRTIM can also be linked with other on-chip timers for the following use:

- as external event
- as burst mode trigger or clock
- for HRTIM registers update triggering.

1.5.6 HRTIM functionality check

Once the whole initialization is completed, it is possible to verify that the HRTIM is ready to go with the simple code below. This example code (HRTIM_BasicPWM example) enables the HRTIM TD1 output and toggles it by software.

```
/* Use the PLLx2 clock for HRTIM */
__HAL_RCC_HRTIM1_CONFIG(RCC_HRTIM1CLK_PLLCLK);
/* Enable HRTIM clock*/
__HRTIM1_CLK_ENABLE();
/* DLL calibration: periodic calibration enabled, period set to 14µs */
HRTIM1->sCommonRegs.DLLCR = HRTIM_CALIBRATIONRATE_14| HRTIM_DLLCR_CALEN;
/* Check DLL end of calibration flag */
while(HRTIM1->sCommonRegs.ISR & HRTIM_IT_DLLRDY == RESET);

HRTIM1->sCommonRegs.OENR = HRTIM_OENR_TD1OEN; /* Enable TD1 output */
GPIO_HRTIM_outputs_Config();  /* Initialize HRTIM outputs */

while(1)
{
 /* Set and reset TD1 by software */
 HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx1R = HRTIM_SET1R_SST;
 HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx1R = HRTIM_RST1R_SRT;
}
```

The STM32F334 snippet reproduced here is available in the HRTIM_Snippets and HRTIM_BasicPWM examples. In both cases the example must be selected with the *#define HRTIM_CHECK* statement.

For the remaining part of this document, the clock and DLL initialization part are not repeated, but replaced by a call to the *HRTIM_Minimal_Config()* function.

2 HRTIM basic operating principles

Despite an apparent complexity due to the number of features and to its modular architecture, the HRTIM is basically made of six or seven 16-bit auto-reload up-counters with four compare registers each.

Period and compare programming (example for 144 MHz input clock)

The high-resolution programming is made completely transparent, so as to have the look-and-feel of a timer clocked by a 4.6 GHz clock (144 MHz x 32), when using the HSE oscillators. The timings (period and compare) can be directly written into a unique 16-bit register with high-resolution accuracy. A counting period is simply programmed using the formula PER = $T_{counting}$ / $T_{High\text{-}res}$.

For instance, a 10 µs time period is obtained by setting the period register to 10 µs / 217 ps = 46082d and is programmed as follows:

```
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = 0x0000B400;
```

If the result exceeds the 16-bit range, the high-resolution can be adjusted by multiples of 217 ps, so as to be have the period value within the 16-bit range.

Set / reset crossbar

Each timing unit holds the control of two outputs via a set / reset crossbar. Compared to usual frozen PWM modes where the output is set at the beginning of the counting period and reset on a given compare match, the crossbar offers much more flexibility in defining how an output is set or reset. It gives the possibility to have any of timer events setting or resetting an output.

Output stage

The waveform generated by the set / reset crossbar is finally passed through an output stage for “post-processing” such as

- generating complementary signal with a dead-time
- adding high-frequency modulation
- modifying the signal polarity
- shutting down the output for protection purpose.

With these few features in mind, it is now possible to elaborate the first elemental PWM signals.

2.1 Single PWM generation

This session focuses on

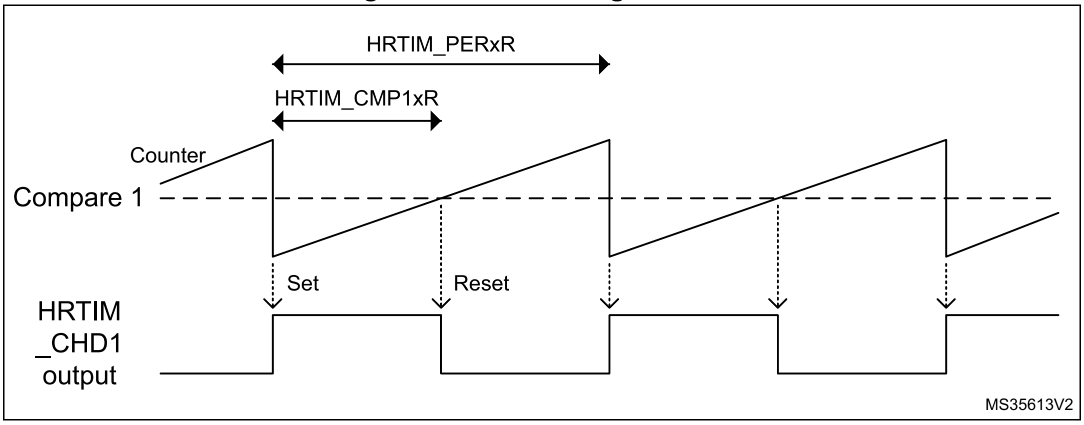
- timer continuous mode
- simplest crossbar configuration
- output stage enable.

PWM signals are the elemental bricks of most power converter and are used for many other purposes such as driving electric motors, piezo buzzers or emulating DAC converters.

This example shows that this can be achieved very simply with the HRTIM by programming a limited number of HRTIM registers.

Let us consider a 100 kHz PWM signal with 50% duty cycle to be generated on HRTIM_CHA1 output, as exemplified in *Figure 1*.

Figure 1. Basic PWM generation



Timer D must be configured in continuous (free-running) mode. The PWM period is programmed in the period register HRTIM_PERAR using the formula PER = f_{HRCK} / f_{PWM}.

Here (144 MHz x 32) / 100 kHz = 46080d (0xB400).

The 50% duty cycle is obtained by multiplying the period by the duty cycle: PER x DC.

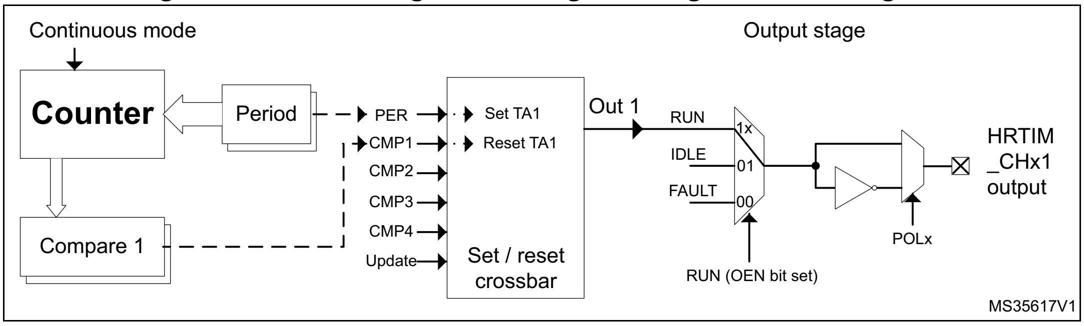
Here 0.5 x 46080d = 23040d (0x5A00).

The waveform is elaborated in the set/reset crossbar with the registers HRTIM_SETx1R (PER bit set) and HRTIM_RSTx1R (CMP1 bit set).

Finally, the output is enabled with the HRTIM_OENR register.

The sequence just described gives an overview of the timer features involved in a simple PWM generation. The configuration is schematized in *Figure 2*.

Figure 2. HRTIM configuration for generating basic PWM signals



The example code is provided in the HRTIM_BasicPWM example, and the snippet reproduced below is available in the HRTIM_Snippets example. In both cases the example must be selected with the #define SINGLE_PWM statement.

```
/* TIMD counter operating in continuous mode */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].TIMxCR = HRTIM_TIMCR_CONT;

/* Set period to 100kHz and duty cycle (CMP1) to 50% */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = 0x0000B400;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = 0x00005A00;

/* TD1 output set on TIMD period and reset on TIMD CMP1 event*/
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx1R = HRTIM_SET1R_PER;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx1R = HRTIM_RST1R_CMP1;

HRTIM1->sMasterRegs.MCR = HRTIM_MCR_TDCEN;    /* Start Timer D */
HRTIM1->sCommonRegs.OENR = HRTIM_OENR_TD1OEN; /* Enable TD1 output */
GPIO_HRTIM_outputs_Config();  /* Initialize HRTIM GPIO outputs */
```

Refer to *Section 10* for the target board and firmware access path.

2.2 Generating multiple PWMs

This section focuses on:

- multiple timing unit usage
- register preload.

The HRTIM is able to generate up to ten PWM signals with five independent frequencies (or six frequencies when the master timer is used, see *Section 2.3*).

In the example below, four PWM signals with two different time-bases are generated.

Timer D generates two phase-shifted 100 kHz with 25% duty cycle on HRTIM_CHD1 and HRTIM_CHD2, with the following conditions:

- HRTIM_CHD1: set on TD Period, reset on TD CMP1
- HRTIM_CHD2: set on TD CMP2, reset on TD Period.

Timer A generates two phase-shifted 33.333 kHz PWMs with 25% duty cycle on HRTIM_CHA1 and HRTIM_CHA2, with the following conditions:

- HRTIM_CHA1: set on TA period, reset on TA CMP1
- HRTIM_CHA2: set on TA CMP2, reset on TA CMP3

Timer A period is below the minimum frequency available at the highest resolution, as shown in *Table 4* and *Table 5*.

Table 4. Timer resolution / minimum PWM frequency for f_{HRTIM}=144 MHz

CKPSC[2:0]	Prescaling ratio	f_{HRCK} equivalent frequency	Resolution	Minimum PWM frequency
000	1	144 x 32 MHz = 4.608 GHz	217 ps	70.3 kHz
001	2	144 x 16 MHz = 2.304 GHz	434 ps	35.1 kHz
010	4	144 x 8 MHz = 1.152 GHz	868 ps	17.6 kHz
011	8	144 x 4 MHz = 576 MHz	1.73 ns	8.8 kHz

Table 4. Timer resolution / minimum PWM frequency for f_{HRTIM}=144 MHz (continued)

CKPSC[2:0]	Prescaling ratio	f_{HRCK} equivalent frequency	Resolution	Minimum PWM frequency
100	16	144 x 2 MHz = 288 MHz	3.47 ns	4.4 kHz
101	32	144 MHz	6.95 ns	2.2 kHz
110	64	144 / 2 MHz = 72 MHz	13.88 ns	1.1 kHz
111	128	144 / 4 MHz = 36 MHz	27.7 ns	550 Hz

Table 5. Timer resolution / minimum PWM frequency for f_{HRTIM}=170 MHz

CKPSC[2:0]	Prescaling ratio	f_{HRCK} equivalent frequency	Resolution	Minimum PWM frequency
000	1	170 x 32 MHz = 5.44 GHz	184 ps	83 kHz
001	2	170 x 16 MHz = 2.72 GHz	368 ps	41.5 kHz
010	4	170 x 8 MHz = 1.36 GHz	735 ps	20.8.6 kHz
011	8	170 x 4 MHz = 680 MHz	1.47 ns	10.4 kHz
100	16	170 x 2 MHz = 340MHz	2.94 ns	5.194 kHz
101	32	170 MHz	5.88 ns	2.59 kHz
110	64	170 / 2 MHz = 85 MHz	11.76 ns	1.3 kHz
111	128	170 / 4 MHz = 42.5 MHz	23.53 ns	650 Hz

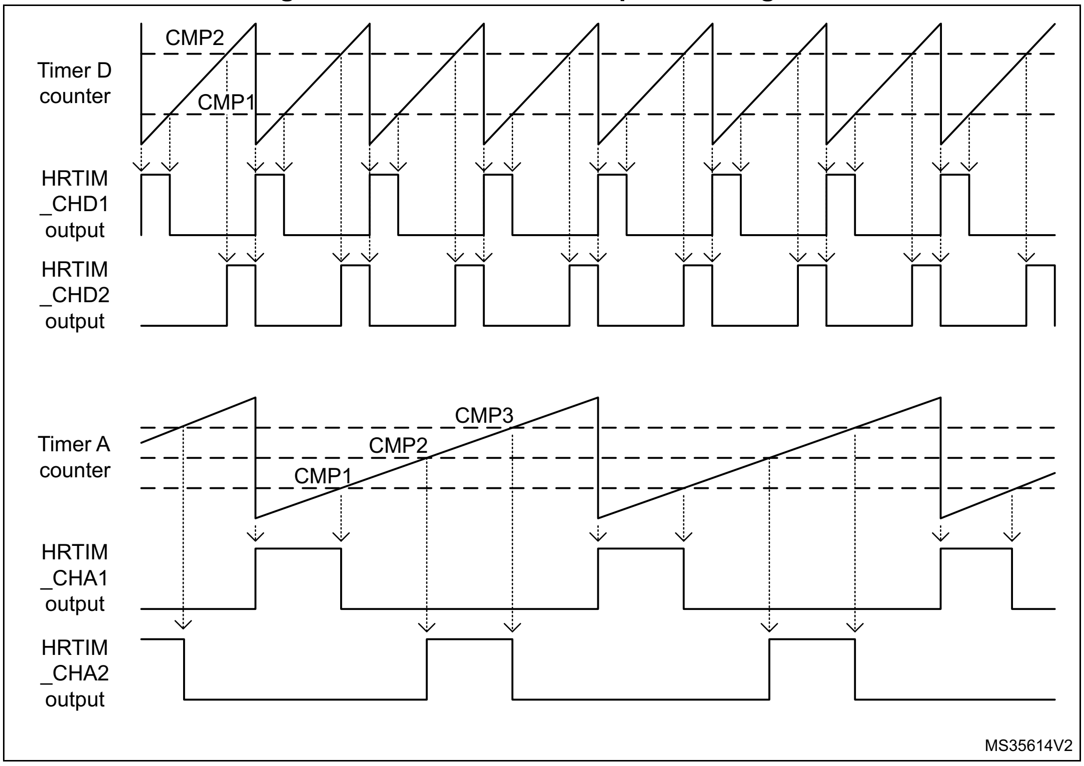
To get a 33.33 kHz switching frequency with a 144 MHz input clock, the frequency prescaler must be set to 4 and the period calculated as PER = f_{HRCK} / f_{PWM}.

Here, a frequency of 33.333 kHz is obtained by setting PER register to (144 MHz * 8) / 33.333 kHz = 34560d (0x8700).

Even though the duty cycle is not updated in this example, and to be close to a practical use case, the register preload mechanism is enabled with the PREEN bit. The REP (repetition) event triggers the transfer of the preload registers at the beginning of each and every period.

Note: A delay can be noticed at the PWM start-up between HRTIM_CHA1/HRTIM_CHA2 and HRTIM_CHD1/HRTIM_CHD2 waveforms. This delay is normal and due to the fact that the first update event (causing compare registers to take their programmed value) occurs only after the first counting period is elapsed. Should the waveforms be started without this delay, it is possible to force a register update by software (using TASWU and TDSWU bits) to have all active compare registers contents updated at once.

Figure 3. Generation of multiple PWM signals



The example code is provided in the HRTIM_BasicPWM example, and the snippet reproduced here-below is available in the HRTIM_Snippets example. In both cases the example must be selected with the #define MULTIPLE_PWM statement.

```
/* --------------------- Timer D initialization ------------------------ */
  /* TIMD counter operating in continuous mode, preload enabled on REP event
*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].TIMxCR =
HRTIM_TIMCR_CONT + HRTIM_TIMCR_PREEN + HRTIM_TIMCR_TREPU;

   /* Set period to 100kHz, CMP1 to 25% and CMP2 to 75% of period */
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _100KHz_PERIOD;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _100KHz_PERIOD/4;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP2xR =
(3*_100KHz_PERIOD)/4;

  /* TD1 output set on TIMD period and reset on TIMD CMP1 event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx1R = HRTIM_SET1R_PER;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx1R = HRTIM_RST1R_CMP1;
  /* TD2 output set on TIMD CMP2 and reset on TIMD period event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx2R = HRTIM_SET2R_CMP2;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx2R = HRTIM_RST2R_PER;
```

```
  /* --------------------- Timer A initialization ---------------------- */
  /* TIMA counter operating in continuous mode with prescaler = 010b (div.
by 4) */
  /* Preload enabled on REP event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].TIMxCR = HRTIM_TIMCR_CONT
+ HRTIM_TIMCR_PREEN + HRTIM_TIMCR_TREPU + HRTIM_TIMCR_CK_PSC_1;

  /* Set period to 33kHz and duty cycles to 25% */
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].PERxR = _33KHz_PERIOD;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP1xR = _33KHz_PERIOD/4;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP2xR = _33KHz_PERIOD/2;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP3xR =
(3*_33KHz_PERIOD)/4;

  /* TA1 output set on TIMA period and reset on TIMA CMP1 event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].SETx1R = HRTIM_SET1R_PER;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].RSTx1R = HRTIM_RST1R_CMP1;
  /* TA2 output set on TIMA CMP2 and reset on TIMA period event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].SETx2R = HRTIM_SET2R_CMP2;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].RSTx2R = HRTIM_RST2R_CMP3;

   /* Enable TA1, TA2, TD1 and TD2 outputs */
   HRTIM1->sCommonRegs.OENR = HRTIM_OENR_TA1OEN + HRTIM_OENR_TA2OEN +
   HRTIM_OENR_TD1OEN + HRTIM_OENR_TD2OEN;
   GPIO_HRTIM_outputs_Config();  /* Initialize HRTIM GPIO outputs */

   /* Start Timer A and Timer D */
   HRTIM1->sMasterRegs.MCR = HRTIM_MCR_TACEN + HRTIM_MCR_TDCEN;
```

Refer to *Section 10* for the target board and firmware access path.

2.3 Generating PWM with other timing units and the master timer

This section focuses on the generation of signals on outputs not related to a given timer.

This example shows that thanks to the set/reset crossbar, it is possible to have PWM signals (or other waveforms) generated on a given output with any other available timer. This is interesting in the following cases:

- to generate a sixth PWM independent frequency with the master timer, as in the example below
- to work-around pin-out constraints, for instance using Timer E to generate waveforms even if HRTIM_CHE1 and HRTIM_CHE2 outputs are not available (typically on small pin-count package).

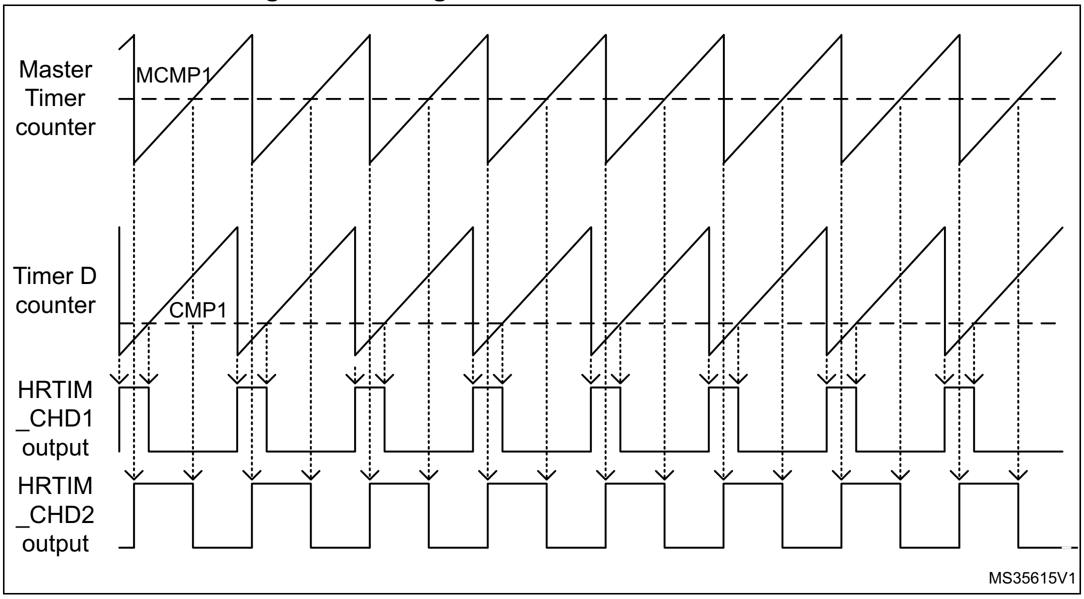
Note: It is mandatory to have the same prescaling factors for all timers sharing resources (for instance master timer and Timer A must have identical CKPSC[2:0] values if master timer is controlling HRTIM_CHA1 or HRTIM_CHA2 outputs).

In the example below, two PWM signals with slightly different switching frequencies are generated on HRTIM_CHD1 and HRTIM_CHD2 outputs, with the following conditions:

- HRTIM_CHD1: set on TD period, reset on TD CMP1
- HRTIM_CHD2: set on master period, reset on master CMP1.

The frequencies are set to slightly different values to ease visualization on oscilloscope (the signals have a relatively slow phase-shift variation), and to demonstrate that the signals are completely asynchronous.

Figure 4. PWM generation with the master timer



The provided code is the HRTIM_BasicPWM example, and the snippet reproduced below is available in the HRTIM_Snippets example. In both cases the example must be selected with the #define PWM_MASTER statement.

```
/* --------------------- Master Timer initialization ------------------- */
  /* Master counter operating in continuous mode, Preload enabled on REP
event*/
  HRTIM1->sMasterRegs.MCR = HRTIM_MCR_CONT + HRTIM_MCR_PREEN +
HRTIM_MCR_MREPU;
  /* Set period to 101kHz and duty cycle to 50% */
  HRTIM1->sMasterRegs.MPER = _100KHz_Plus_PERIOD;
  HRTIM1->sMasterRegs.MCMP1R = _100KHz_Plus_PERIOD/2;

/* --------------------- Timer D initialization ----------------------- */
  /* TIMD counter operating in continuous mode, preload enabled on REP event
*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].TIMxCR =
HRTIM_TIMCR_CONT + HRTIM_TIMCR_PREEN + HRTIM_TIMCR_TREPU;

  /* Set period to 100kHz and duty cycle (CMP1) to 50% */
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _100KHz_PERIOD;
```

```
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _100KHz_PERIOD/4;

  /* TD1 output set on TIMD period and reset on TIMD CMP1 event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx1R = HRTIM_SET1R_PER;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx1R = HRTIM_RST1R_CMP1;

  /* TD2 output set on TIMD CMP2 and reset on TIMD period event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx2R =
HRTIM_SET2R_MSTPER;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx2R =
HRTIM_RST2R_MSTCMP1;

  /* Enable TD1 and TD2 outputs */
  HRTIM1->sCommonRegs.OENR = HRTIM_OENR_TD1OEN + HRTIM_OENR_TD2OEN;
  GPIO_HRTIM_outputs_Config();  /* Initialize HRTIM GPIO outputs */

  /* Start Master Timer and Timer D */
  HRTIM1->sMasterRegs.MCR |= HRTIM_MCR_MCEN + HRTIM_MCR_TDCEN;
```

Refer to *Section 10* for the target board and firmware access path.

2.4 Arbitrary waveform generation

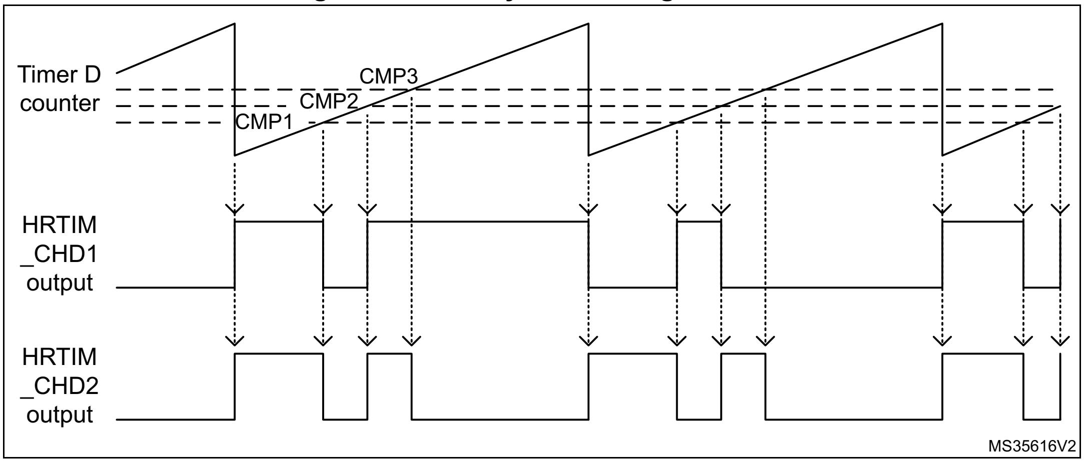
This section focuses on waveforms with multiple set/reset/toggle requests per period.

This example shows how waveforms other than PWMs can be easily generated thanks to the 32 concurrent set/reset sources of the crossbar.

In the example below, two arbitrary waveforms are generated on HRTIM_CHD1 and HRTIM_CHD2 outputs, with the following conditions:

- HRTIM_CHD1: toggle on TD period, toggle on TD CMP1, toggle on TD CMP2
- HRTIM_CHD2: set on TD period and TD CMP3, reset on TD CMP2 and TD CMP3.

Figure 5. Arbitrary waveform generation



The provided code is the HRTIM_BasicPWM example, and the snippet reproduced below is available in the HRTIM_Snippets example. In both cases the example must be selected with the #define ARBITRARY_WAVEFORM statement.

```
/* -------------------- Timer D initialization ---------------------- */
  /* TIMD counter operating in continuous mode, preload enabled on REP event
*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].TIMxCR = HRTIM_TIMCR_CONT
+ HRTIM_TIMCR_PREEN + HRTIM_TIMCR_TREPU;

  /* Set period to 100kHz and edge timings */
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _100KHz_PERIOD;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _100KHz_PERIOD/4;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP2xR =
(3*_100KHz_PERIOD)/8;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP3xR = _100KHz_PERIOD/2;

  /* TD1 toggles on TIMD period, CMP1 and CMP2 event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx1R = HRTIM_SET1R_PER +
HRTIM_SET1R_CMP1 + HRTIM_SET1R_CMP2;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx1R = HRTIM_RST1R_PER +
HRTIM_RST1R_CMP1 + HRTIM_RST1R_CMP2;

  /* TD2 output set on TIMD PER and CMP2 and reset on TIMD CMP1 and CMP3
event*/
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].SETx2R = HRTIM_SET2R_PER +
HRTIM_SET2R_CMP2;
  HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].RSTx2R = HRTIM_RST2R_CMP1 +
HRTIM_RST2R_CMP3;

  /* Enable TD1 and TD2 outputs */
  HRTIM1->sCommonRegs.OENR = HRTIM_OENR_TD1OEN + HRTIM_OENR_TD2OEN;
  GPIO_HRTIM_outputs_Config();  /* Initialize HRTIM GPIO outputs */

  /* Start Timer D */
  HRTIM1->sMasterRegs.MCR = HRTIM_MCR_TDCEN;
```

Refer to *Section 10* for the target board and firmware access path.

3 Voltage mode dual buck converter

This section focuses on:

- repetition counter interrupts
- ADC trigger
- register update on repetition event.

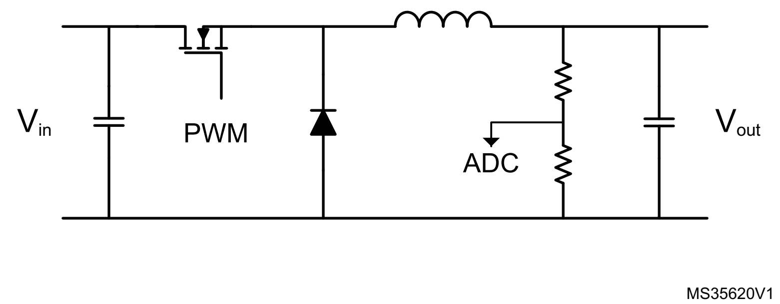
Buck converters are mostly used to perform step-down voltage conversions. They are usually operating at fixed frequency and variable duty cycle.

In the ideal converter, the V_{in} / V_{out} ratio depends only upon the duty cycle D applied to the power switch, as V_{out} = D x V_{in}.

Practically, the duty cycle is adjusted by the control algorithm to maintain the desired output voltage. In this example, the voltage mode buck converter is considered: the duty cycle is controlled based on the converter output voltage reading. An input voltage reading can be added to implement a feed-forward compensation.

The power stage topology is sketched in *Figure 6*, where the output voltage reading carried out by the ADC (to perform the regulation) is also shown.

Figure 6. Voltage-mode buck converter



In the example provided with the software package (HRTIM_DualBuck), the HRTIM is programmed to perform the control of two buck converters operating in parallel (with the same frequency and non-overlapping ON-time).

The HRTIM is operating in continuous mode, and the PWM signals are defined as following:

- HRTIM_CHD1: set on TD Period, reset on TD CMP1
- HRTIM_CHD2: set on TD CMP2, reset on TD period.

HRTIM_CHD2 is generated as in *Section 2.2*, it is not represented in the following figures.

Two other compare events are used in this example to trigger the ADC.

CMP3 and CMP4 timings are calculated to be in the middle of the switch ON time, on HRTIM_CHD1 and HRTIM_CHD2. This makes possible to have an average voltage measurement and guarantees that the conversion is done away from the ringing and switching noise due to power switches.

Figure 7 shows the counter, the generated PWM output and a magnified view of the resulting output voltage ripple, together with the ADC sampling point for HRTIM_CHD1 channel. Because of the high PWM frequency, the repetition counter is used to decrease the number of interrupts service routines. In *Figure 7* the repetition autoreload rate is set to 1, so that the ISR rate is half the PWM frequency. In the example, the repetition rate is set to 127 to have a very low change rate.

Figure 7. VM buck waveforms, including ADC sampling and interrupts

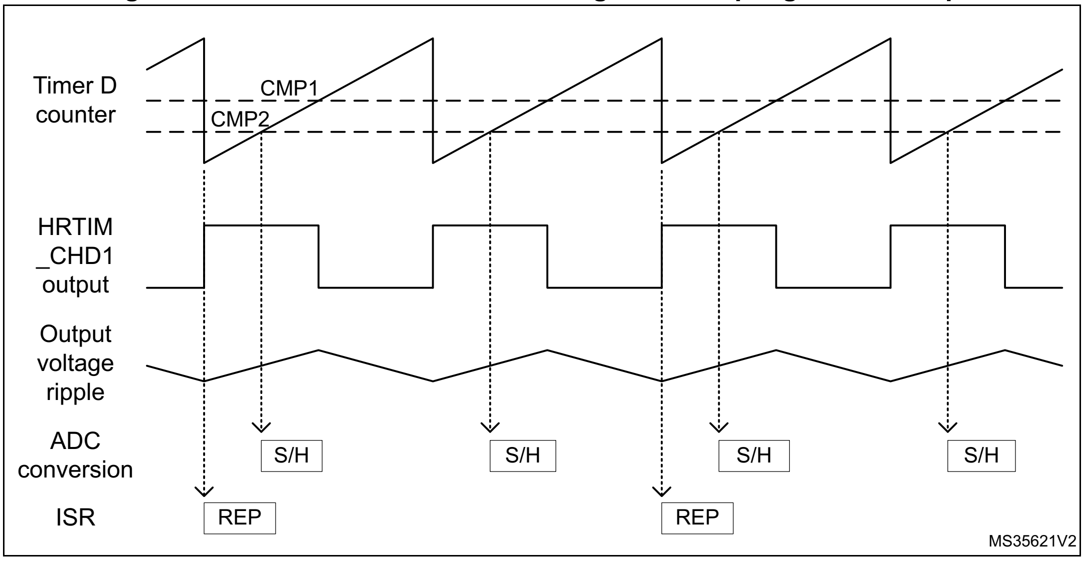
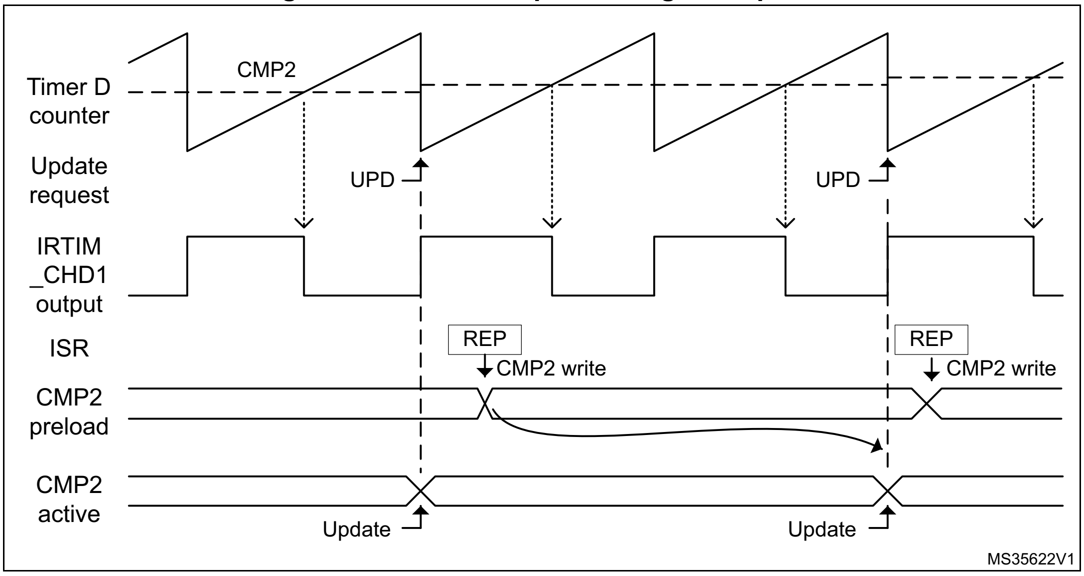


Figure 8 shows how the duty cycle is updated, once the duty cycle on HRTIM_CHD1 has been completed in the interrupt routine. The timer is programmed to have a register update occurring on each repetition event.

Figure 8. PWM interrupts and register update



Dual buck demonstration overview

The duty cycle on HRTIM_CHD1 is continuously varied in an interrupt service routine generated on repetition event, to mimic a real converter management. A low-pass filtered signal reflecting the PWM duty cycle can be monitored on STM32F334 discovery kit TP3 test point.

The duty cycle on HRTIM_CHD2 is constant.

The ADC is configured to have conversions triggered in the middle of the ON time of each output. The two conversions are done with the same ADC injected trigger (this is possible because the two signals are not overlapping). The ADC is programmed in discontinuous injected mode (a conversion triggered on CMP3, another on CMP4).

The two conversions are done on the same input (PA4): HRTIM_CHD1 low-pass filtered PWM can be connected to PA4 for monitoring ADC conversions with the debugger. For a real use case, it is easy to reprogram the ADC to have the conversions done on each buck output voltage.

The HRTIM_FLT1 input is enabled on PA12 (active low) to demonstrate PWM shut down (see *Section 4* for details). When the fault is triggered (PA12 input connected to GND), HRTIM_CHD1 signal only is stopped. The system can be re-armed by pressing the user button.

LEDs are indicating the following:

- blue: blinks during normal operation
- red: blinks when FAULT is triggered
- orange: indicates the occurrence and duration of the PWM refresh ISR.

Demonstration code

The example code is provided in the HRTIM_DualBuck example.

Refer to *Section 10* for the target board and firmware access path.

4 Voltage mode buck converter with synchronous rectification and fault protection

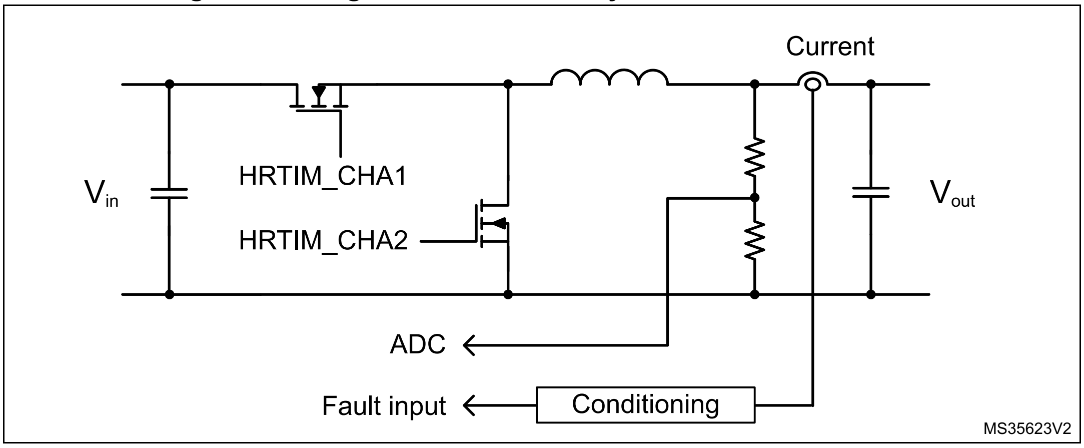
This section focuses on:

- dead time
- digital FAULT protection
- burst mode controller.

This example presents a voltage mode buck converter where the freewheeling diode is replaced by a MOSFET for synchronous rectification purpose. It is thus possible to increase the converter efficiency by reducing the losses when the inductance is demagnetized into the output capacitor.

Figure 9 shows the power stage topology. It includes the output voltage reading and an over-current protection using the FAULT input, to have the converter shut down when the current exceeds a programmable threshold. For simplicity, the current sensor and the conditioning circuitry are not discussed here; the expected FAULT feedback on HRTIM_FLT1 input is a digital signal (on PA12 input).

Figure 9. Voltage mode buck with synchronous rectification



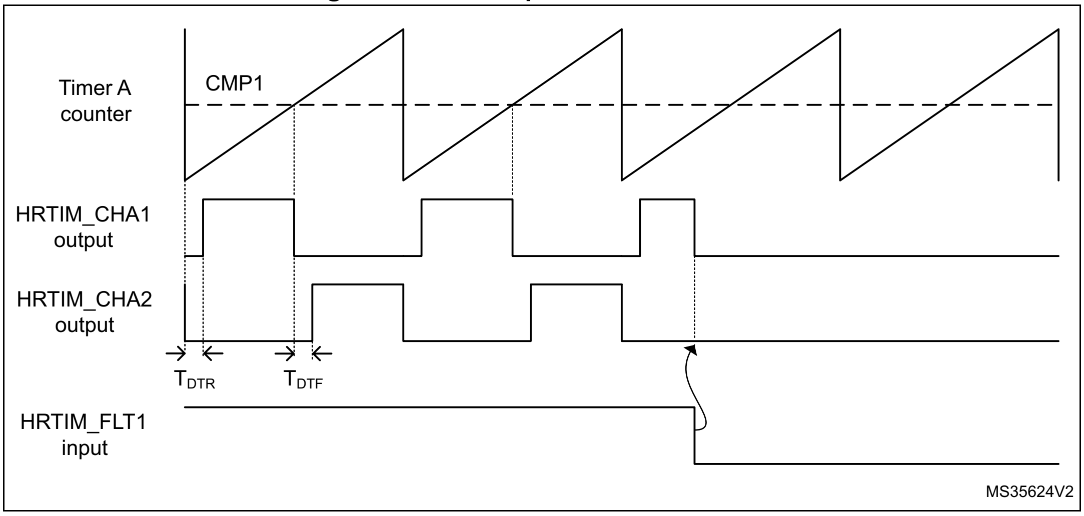
The HRTIM is operating in continuous mode, and the PWMs signals are defined as follows:

- HRTIM_CHA1: set on TA period, reset on TA CMP2
- HRTIM_CHA2: complementary of HRTIM_CHA1 with dead time generator (identical rising and falling edge dead times).

The converter is over-current protected with the HRTIM_FLT1 digital input, low level sensitive. The HRTIM_CHA1 and HRTIM_CHA2 outputs are forced to the low level in case of HRTIM_FLT1 event, by programming a positive output polarity and an inactive fault state.

Figure 10 shows the buck control waveforms on HRTIM_CHA1 and HRTIM_CHA2 outputs and includes the case where a fault occurs.

Figure 10. Buck operation with FAULT



Buck with synchronous rectification demonstration overview

The duty cycle on HRTIM_CHA1 is continuously varied in an interrupt service routine generated on repetition event, to mimic a real converter management. The HRTIM_FLT1 input is enabled on PA12 (active low) to demonstrate PWM shut down (low level sensitive), for both HRTIM_CHA1 and HRTIM_CHA2.

When the fault is triggered (PA12 input connected to GND) HRTIM_CHA1 and HRTIM_CHA2 signals are shut-down. The system can be re-armed by pressing the user button.

LEDs are indicating the following:

- blue: blinks during normal operation
- red: blinks when FAULT is triggered
- orange: indicates the occurrence and duration of the PWM update ISR.

The ADC is configured to have conversions triggered in the middle of the converter ON time, on PA1 (V_{in}) and PA3 (V_{out}) inputs. To run the demo, the V_{out} input of the BUCK-BOOST converter must be connected to the 5V_O supply. The resulting voltage is available on V_{out} pin.

Note: To have the Discovery kit BOOST stage bypassed, the PA11 pin must be forced to 1 (to have the T6 PMOS switched ON).

Demonstration code

The example code is provided in the HRTIM_BuckSyncRect example.

Refer to *Section 10* for the target board and firmware access path.

5 Non-inverting buck-boost converter

This section focuses on:

- 4-switch converter drive
- 0% and 100% PWM generation.

This example shows how to configure the HRTIM to drive a non-inverter buck-boost converter and switch on the fly from a step-down to a step-up mode. Although this configuration requires more switches than the conventional inverting buck-boost, it has the advantage of providing a ground-referred positive output voltage.

Figure 11 shows the power stage topology, as implemented on the STM32F334 Discovery kit.

Figure 11. Non-inverting buck-boost converter

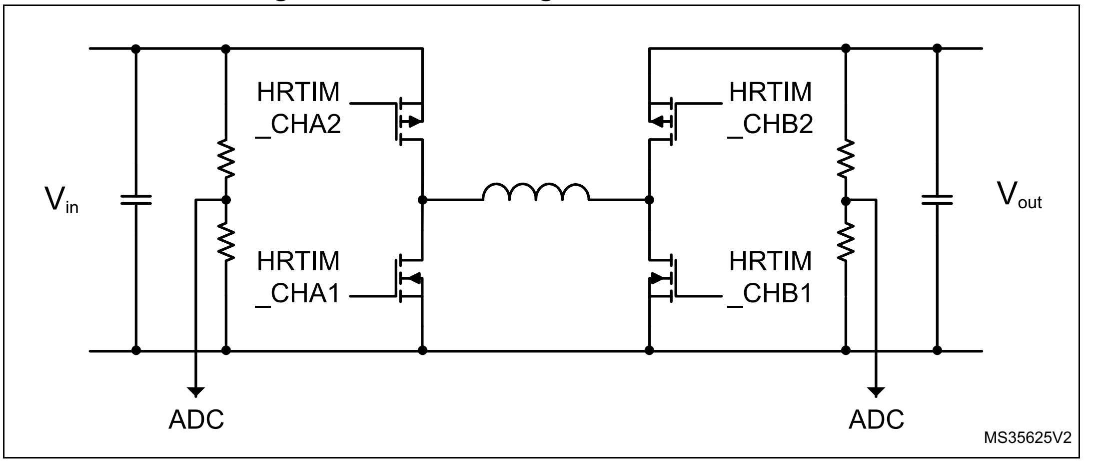
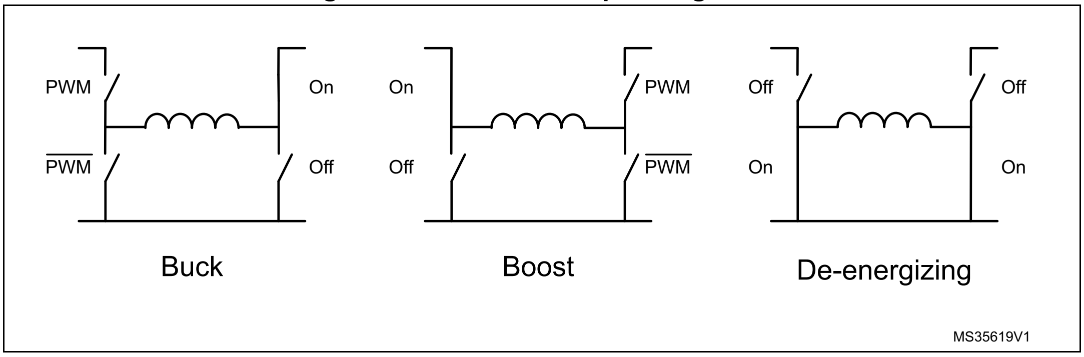


Figure 12 summarizes the three operating modes of the demonstration with simplified schematics (MOSFETs anti-parallel diodes are not represented). The synchronous rectification schemes are applied for both buck and boost modes (the complementary MOSFETs are driven to reduce the conduction losses in their intrinsic diodes). A third mode “de-energizing” is necessary to avoid having current flowing back into the input source when the converter is switched from boost to buck mode: the MCU maintains this mode until the output capacitor is discharged below the input voltage.

The constant ON/OFF states are imposed by forcing either 0% or 100% PWM on the complementary outputs driving the half-bridges.

Figure 12. Buck-boost operating mode



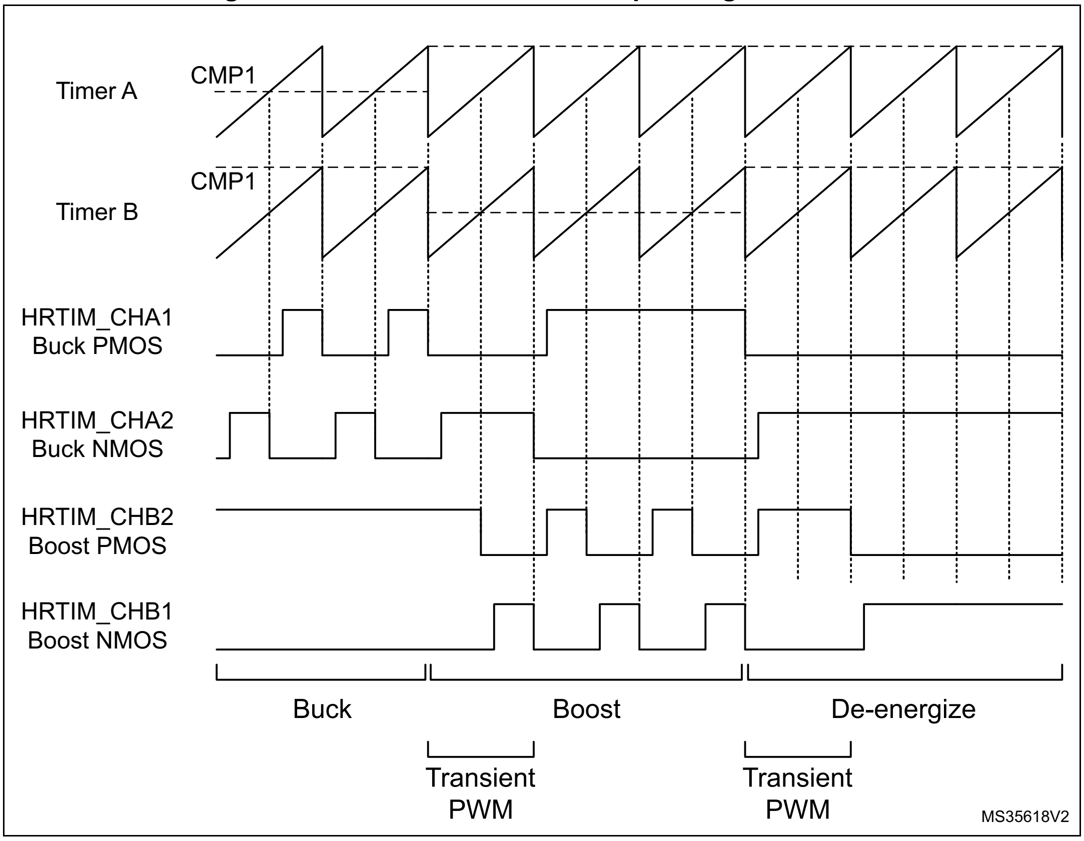
The HRTIM is operating in continuous mode, and the PWMs signals are defined as follows:

- HRTIM_CHA1: set on TA CMP1, reset on TA period
- HRTIM_CHA2: complementary of HRTIM_CHA1 with dead time generator (identical rising and falling edge dead times)
- HRTIM_CHB1: set on TB CMP1, reset on TB period
- HRTIM_CHB2: complementary of HRTIM_CHB1 with dead time generator (identical rising and falling edge dead times).

The converter is protected with the HRTIM_FLT1 digital input, low level sensitive. All outputs are forced to the low level in case of FAULT1 event, by programming a positive output polarity and an inactive fault state.

Figure 13 shows the buck-boost control waveforms on the four outputs for the three operating modes.

Figure 13. Buck-boost converter operating waveforms



The 0% and 100% duty cycle are obtained by programming compare register values to 0 or 100%, causing the two complementary outputs to be constantly ON/OFF:

- when CMP1 value is equal to the PER value, the duty cycle is 100% (CMP1 set event wins over PER reset event: refer to the hardware priority scheme detailed in the reference manual: when two simultaneous events occur, priority is the following: CMP4 → CMP3 → CMP2 → MP1 → PER)
- when CMP1 value is above to the PER value, the duty cycle is 0% (no more set event generated).

Buck-boost demonstration overview

To run the demonstration , the V_{in} input pin of the buck-boost converter must be connected to the 5V_O output pin of the STM32F334 Discovery kit supply. The resulting voltage is available on V_{out} pin.

The demonstration starts in buck mode, and the duty cycle is slowly adjusted in the TIMA IRQ handler to have V_{out} continuously varying below V_{in} value. If the push-button is pressed and the voltage is below 5 V, the boost mode is enabled (the voltage check prevents exceeding the kit maximum output voltage). The voltage is increased above V_{in} value with a fixed duty cycle. If the push-button is pressed again, the output capacitor is de-energized down to 4.5 V before re-enabling the buck mode.

The ADC is configured to have conversions triggered in the middle of the converter ON time, on PA1 (Vin) and PA3 (V_{out}) inputs. The values are converted in mV in the main routine to have a direct voltage reading with the debugger (using a run-time refreshed watch) during the demonstration.

The HRTIM_FLT1 input is enabled on PA12 (active low) to demonstrate PWM shut down (low level sensitive), for all outputs. When the fault is triggered (PA12 input connected to GND), HRTIM_CHA1, HRTIM_CHA2, HRTIM_CHB1 and HRTIM_CHB2 signals are shut-down. The system can be re-armed by pressing the user button.

LEDs are indicating the following:

- green: blinks during BUCK operation
- blue: blinks during BOOST operation
- red: blinks when FAULT is triggered
- orange: indicates the occurrence and duration of the PWM update ISR.

Demonstration code

The example code is provided in the HRTIM_BuckBoost example.

Refer to *Section 10* for the target board and firmware access path.

6 Transition mode power factor controller

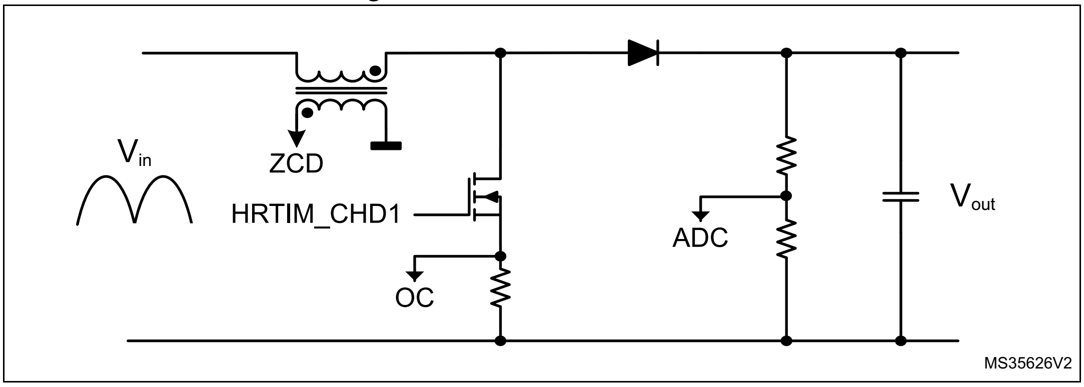
This section focuses on:

- external events conditioning and filtering
- timer counter reset
- events blanking windows
- capture unit
- constant T_{on} time converter.

This example presents a transition mode (also known as boundary conduction mode) power factor controller (PFC).

Figure 14 shows the topology, similar to a boost converter. It features an output voltage reading and a demagnetization detection winding coupled with the main inductor for ZCD (zero-current detection).

Figure 14. Transition mode PFC



The basic operating principle is to build-up current into an inductor during a fixed T_{on} time.

This current decays during the T_{off} time, and the period is re-started when it becomes null. This is detected using a ZCD circuitry, made with an auxiliary winding on the main inductor. With a constant T_{on}, the peak current value in the inductor is directly proportional to the rectified AC input voltage, which provides power factor correction.

This converter is operating with a constant T_{on} and a variable frequency due the T_{off} variation (depending on the input voltage). It must also include some features to operate when no zero-crossing is detected, or to limit T_{on} in case of over-current (OC). The OC feedback is usually conditioned with the built-in comparator and routed onto an external event channel.

Note: The operating principle below can also be applied to a constant off time converter.

Figure 15 and *Figure 16* show the waveforms during the various operating modes, with the following parameters defined:

- T_{on} min: during this period, spurious over-currents are discarded (typically these are freewheeling diode recovery currents). It is represented as OC blanking and programmed with CMP3.
- T_{on} max: practically, the converter set-point, defined with CMP1.
- T_{off} min: limits the frequency when current limit is close to zero (demagnetization is very fast). It is defined with CMP2 in auto-delayed mode.
- T_{off} max: prevents the system to be stuck if no ZCD occurs. It is defined with CMP4 in auto-delayed mode.

Figure 15. Transition mode PFC operation at T_{on} max and during over-current

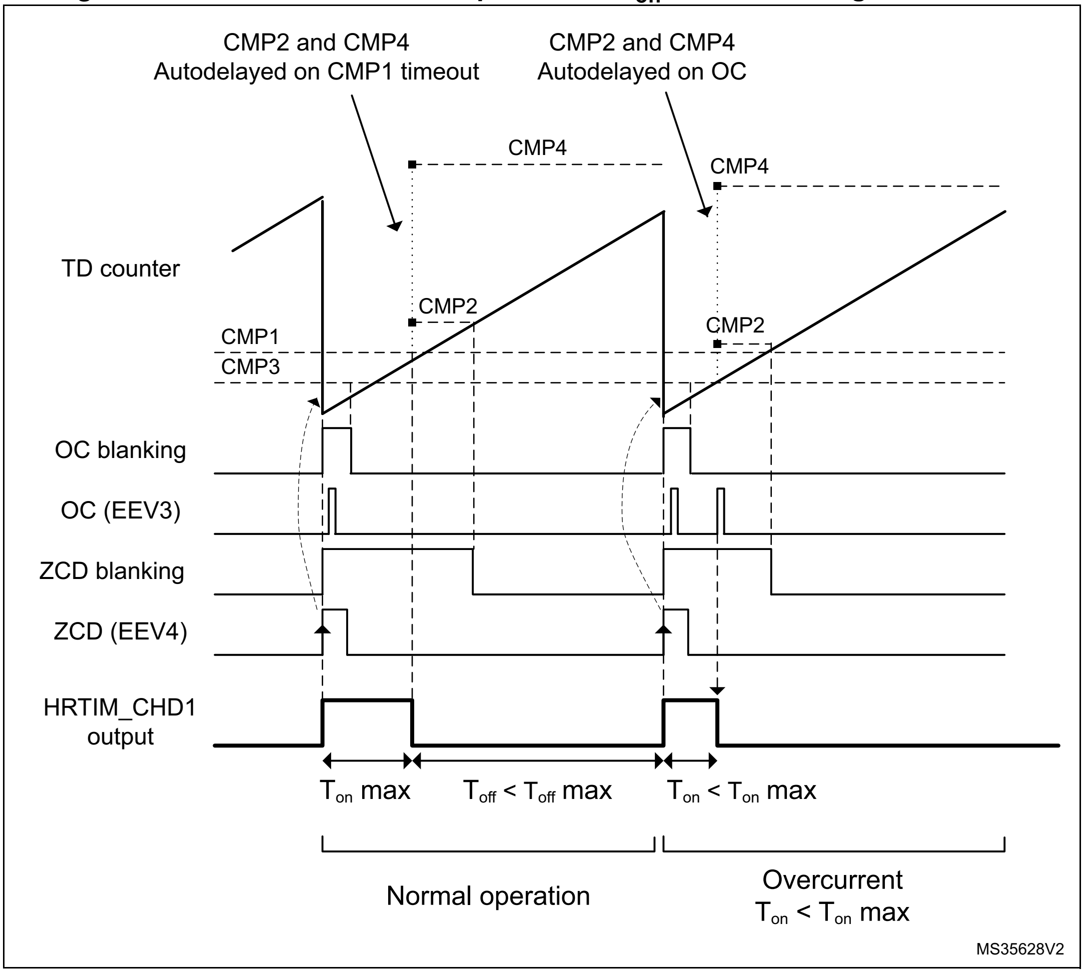
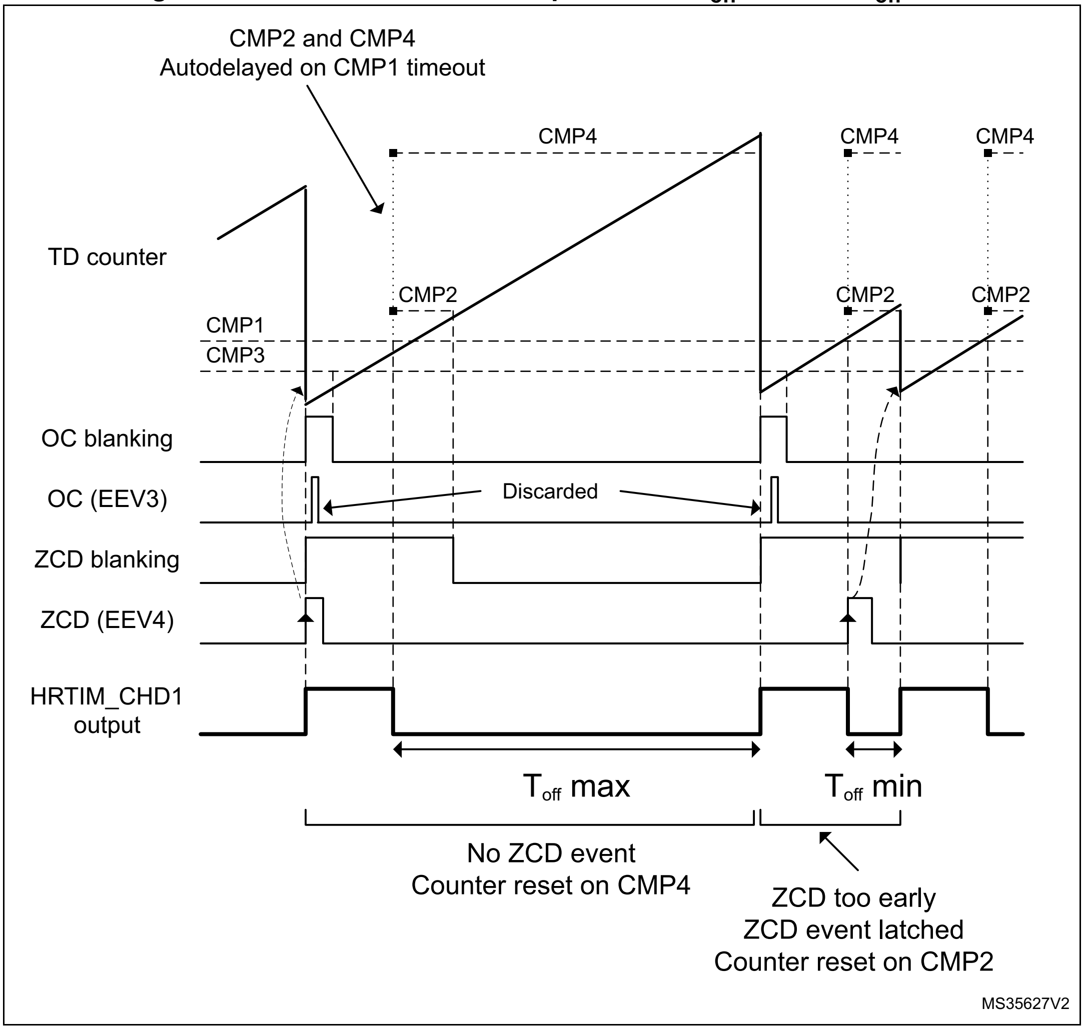


Figure 16. Transition mode PFC operation at T_{off} max and T_{off} min



The HRTIM is operating in continuous mode, and the HRTIM_CHD1 signal is defined as follows:

- set on (TD CMP4 or ($ZCD_{latched}$. TDCMP2))
- reset on (TD CMP1 or (OC. TDCMP3)).

$ZCD_{latched}$. TDCMP2 indicates that ZCD event is filtered by a blanking window starting from Timer D counter reset and finishing at TD CMP2 match. The ZCD event is latched: if it occurs during the blanking window, it is not discarded and is effective at the end of the blanking period. The ZCD signal is applied on the external event 4.

OC. TDCMP3 indicates that OC event is filtered by a blanking window starting from timer D counter reset and finishing at TD CMP3 match. The OC event is not latched: if it occurs during the blanking window is discarded. The OC signal is applied on the external event 3.

Both T_{off} values (based on CMP2 and CMP4) are auto-delayed: these timings must be relative to the output signal falling edge (occurring either on CMP1 match or OC event). CMP2 and CMP4 events are generated in auto-delayed mode with CMP1 timeout, based on capture 1 and capture 2 events, respectively. The two capture units are triggered by the OC signal (EEV3).

Timer D counter is reset either by the ZCD event or by CMP4 match (in timeout conditions).

The converter is protected with the HRTIM_FLT1 digital input, low level sensitive. The HRTIM_CHD1 output is forced to a low level in case of FAULT1 event, by programming a positive output polarity and an inactive fault state.

Transition mode PFC demonstration overview

To run the demonstration and test all operating modes, it is necessary to simulate the feedback two input signals with a function generator:

- over-current (OC, on EEV3/PB7)
- zero-crossing detection (ZCD, on EEV4/PB6).

The various operating modes can be tested as following:

- if the OC signal is generated during the T_{on} time, the pulse is shortened
- the ZCD signal is resetting the timer counter and causes the switching frequency to change accordingly.

The FAULT1 input is enabled on PA12 (active low) to demonstrate PWM shut down (low level sensitive). When the fault is triggered (PA12 input connected to GND) HRTIM_CHD1 signal is stopped. The system can be re-armed by pressing the user button.

LEDs are indicating the following:

- green: blinks during normal operation;
- orange: blinks when FAULT is triggered.

Demonstration code

The example code is provided in the HRTIM_TM_PFC example.

Refer to *Section 10* for the target board and firmware access path.

7 Multiphase buck converter

This section focuses on:

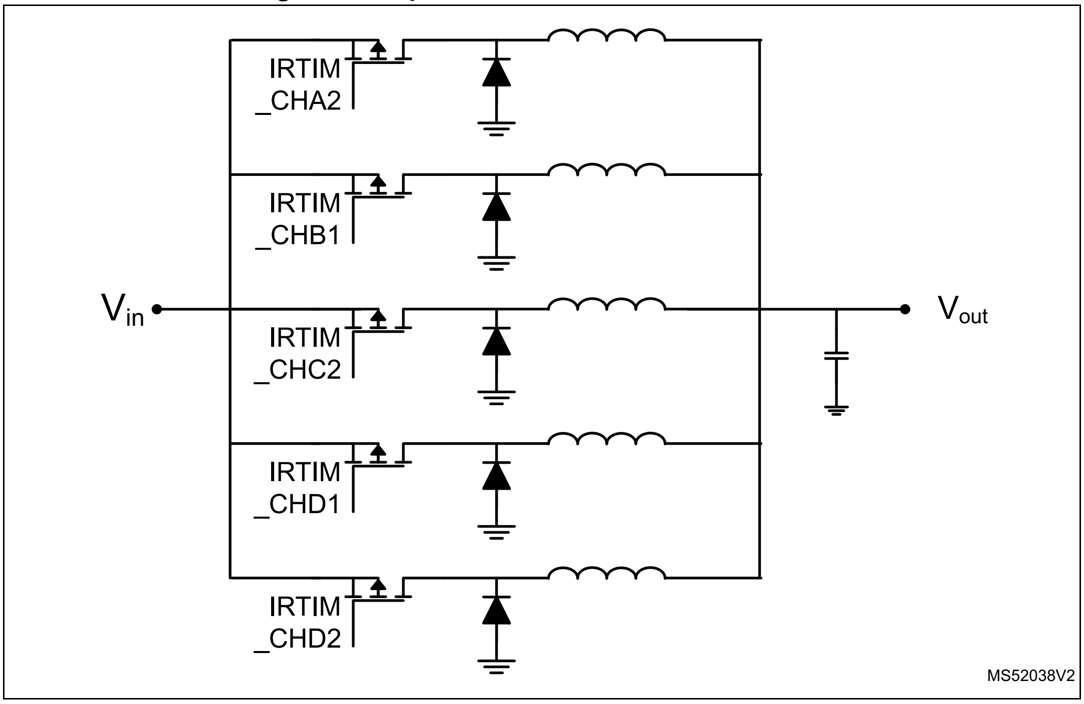
- master timer
- cross timer reset
- multiple timer synchronization
- burst mode controller.

Multiphase techniques can be applied to multiple power conversion topologies (buck, boost). Their main benefits are:

- reduction of the current ripple on the input and output capacitors
- reduced EMI
- higher efficiency at light load by dynamically changing the number of phases (phase shedding).

A 5-phase interleaved buck driven by the HRTIM is shown in *Figure 17*.

Figure 17. 5-phase interleaved buck converter



The master timer handles the phase management: it defines the phase relationship between the individual buck converters by resetting the slave timers periodically. The relative phase-shift is 360° divided by the number of phases, 72° in this given example.

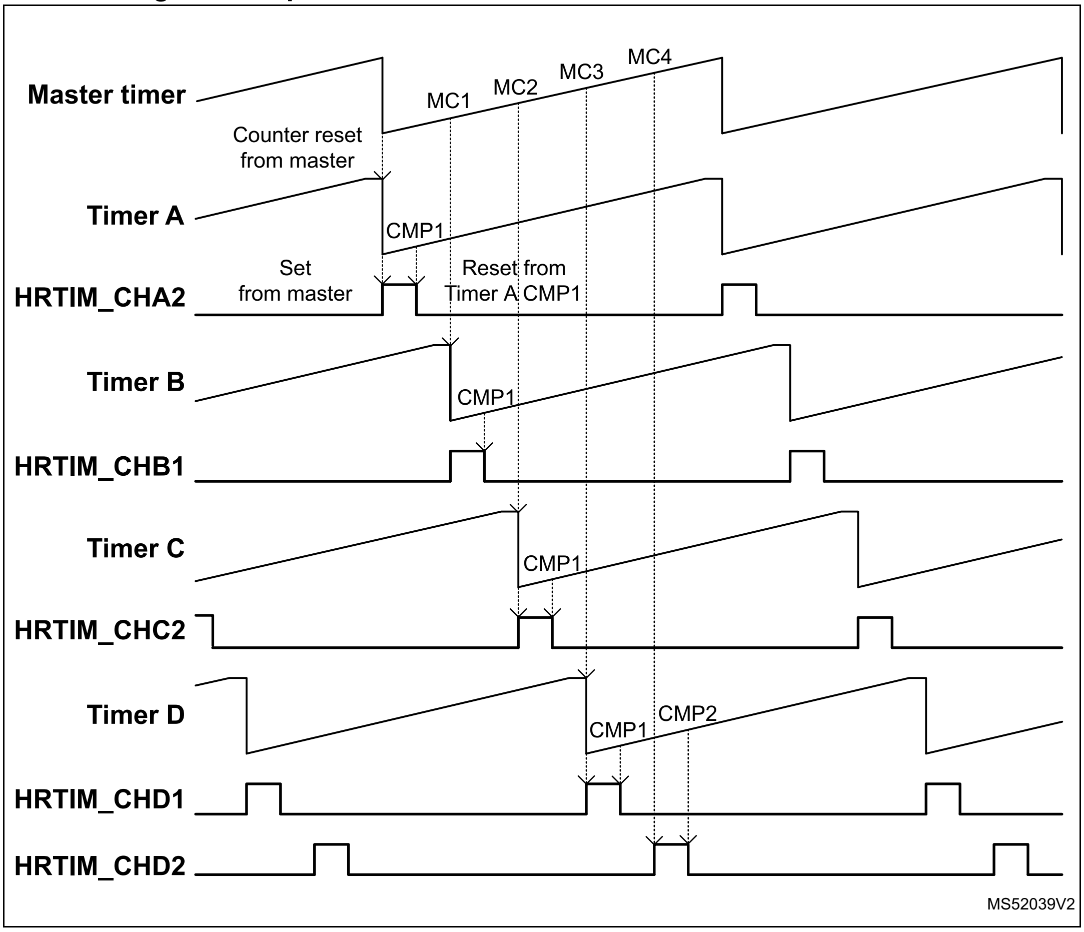
The master operates in continuous mode while the slave timers are in single-shot retriggerable mode. Slave timer counters are reset by master timer events (Timer A reset on master period, Timer B, C and D reset, respectively, by Master compare 1, 2, 3). The duty cycle is programmed into each of the timers, independently from the phase shift, which can vary during operation. The output waveforms are defined as following:

- HRTIM_CHA2: set on master timer period, reset on TACMP1
- HRTIM_CHB1 set on master timer MCMP1, reset on TBCMP1
- HRTIM_CHC2 set on master timer MCMP2, reset on TCCMP1
- HRTIM_CHD1 set on master timer MCMP3, reset on TDCMP1
- HRTIM_CHD2 set on master timer MCMP4, reset on TDCMP2

The waveforms and main events (output set/reset and slave timer counter reset) are represented in *Figure 18*, where MCx stands for master timer CMPx and Cx stands for slave timer CMPx. The master timer manages both counter reset and output set with the same event. The duty cycle is programmed using compare events from the slave timers. This makes possible to decouple the phase management for efficiency and EMI (reduction of the number of active phase, phase-shift variation, spread spectrum frequency dithering), from the duty cycle management for output voltage regulation.

Note: The example given here is intended to be tested with the STM32F334 Discovery hardware, which imposes the output mapping and a few tricks. A regular implementation would use the following outputs: HRTIM_CHA1, HRTIM_CHB1, HRTIM_CHC1, HRTIM_CHD1, HRTIM_CHE1, with the option for synchronous rectification on complementary outputs HRTIM_CHA2, HRTIM_CHB2, HRTIM_CHC2, HRTIM_CHD2, HRTIM_CHE2.

Figure 18. 5-phase interleaved buck converter control waveforms



The master timer is updated on its repetition event (once every N PWM period, N from 1 to 256, to have the CPU load decoupled from the switching frequency). This update event triggers the transfer from preload to active registers, so that all modified values are taken into account simultaneously, without any risk of glitches.

The event is also propagated to all slave registers, so that the new operating set-points (phase-shift and duty-cycles) are synchronously updated for the full converter. This is achieved by setting the MPREPU bit in MCR register, and MSTU bit in TIMxCR registers (TIMx update is done on master update event).

The converter is controlled during run-time by a single interrupt service routine (master repetition event), so as to let the maximum time for the software before the next update takes place.

The multiphase converter gives the possibility to adjust the number of phases dynamically, depending on the output load, to reduce the losses and improve the efficiency. Once a phase is disabled, it is needed to adjust the phase-shift of the remaining phases. This is done adjusting the four master timer compare registers. When a master compare value is set above the master timer period value, the slave timer is not re-triggered (nor its output(s) set) and the corresponding phase is switched off, as shown in *Figure 19*.

Figure 19. Disabling a phase with the master timers

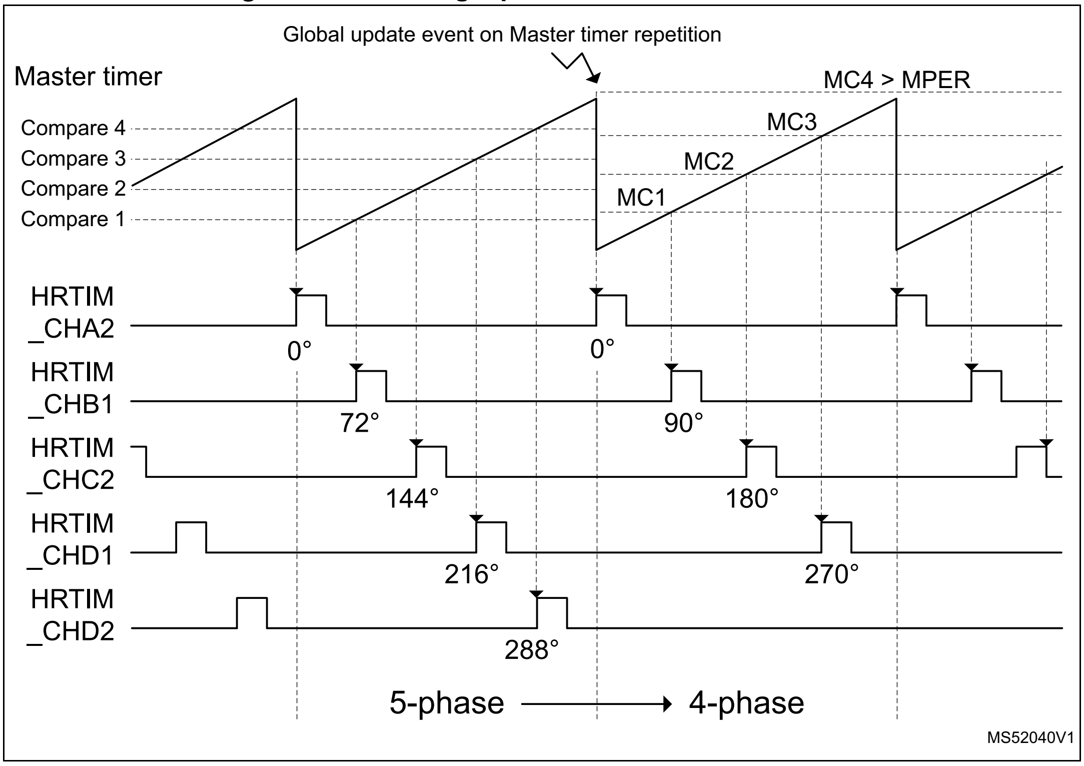
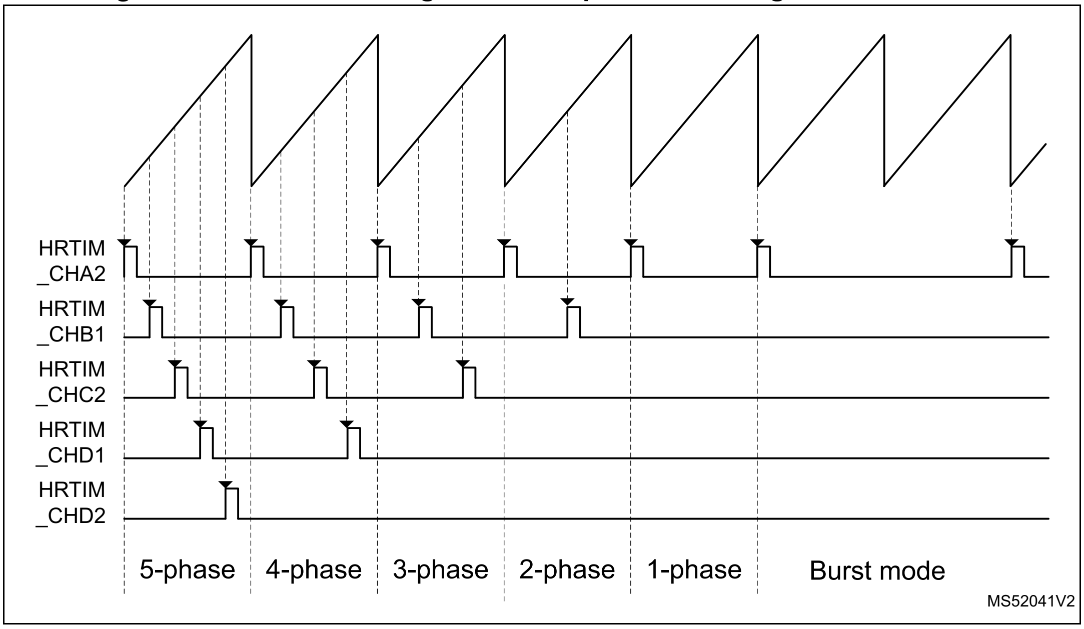


Figure 20 represents the six possible phase arrangements, from 5- to 1-phase, followed by a burst mode for phase skipping.

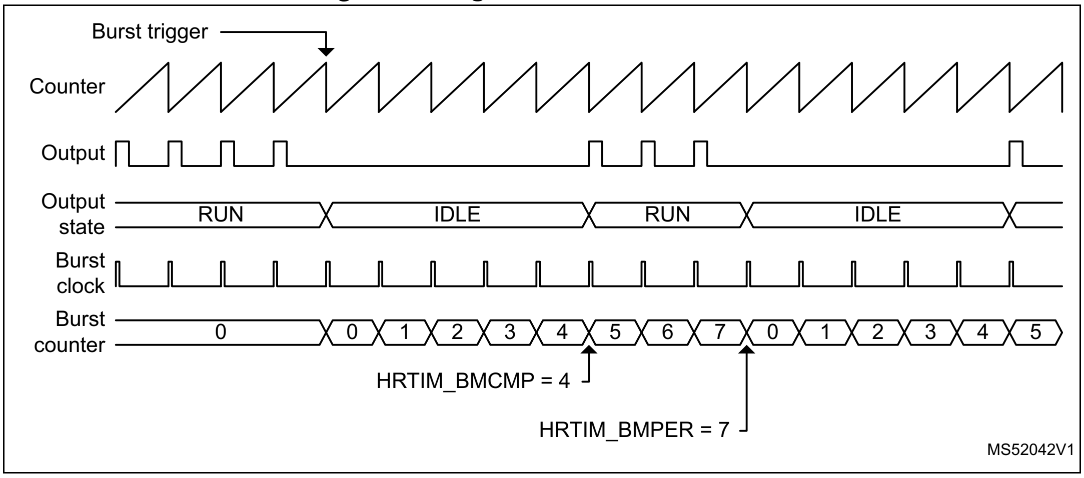
Figure 20. Low-load management with phase shedding and burst mode



The burst mode controller makes it possible to increase the buck controller efficiency during light load operation by periodically disabling the PWM outputs. This applies to either single or multiphase converter.

Figure 21 shows the PWM waveforms for an RUN / IDLE ratio of 3 / 7. In this case the burst mode is triggered by the master timer counter roll-over event and is operating continuously. It is terminated by software when full load operation must be resumed.

Figure 21. Light load buck converter



The ADC triggers are generated by the spare TxCMPy compare events. In the example provided, the ADC is configured to have a conversion done in the middle of each phase ON time. Since all ADC trigger sources are phase-shifted, it is possible to have all of them combined into a single ADC trigger to save ADC resources (here a single ADC regular channel is sufficient for the full multi-phase converter monitoring).

The ADC operates in discontinuous mode: the sequencer is scanned one conversion at a time, each trigger launching a unique conversion. The conversion result is stored in 5-location RAM table by the DMA controller, programmed in circular mode. It is also possible to have the RAM table increased (modulo 5) to store conversion history with any depth.

Note: It is also possible to define a subsequence of several conversions per trigger (typically one voltage and one current), by programming the DISCNUM bitfield in the ADCx_CFGR register.

The sampling strategy depends on the number of active phases: the ADC triggering points must be adapted, since some slave timers may be stopped. The proposed implementation allows to have dummy conversions generated so as to have the phase data reading in the very same RAM locations. *Table 6* indicates the active ADC triggers depending on the number of active phases.

Table 6. Sampling triggers and results in Data[0..4] RAM table

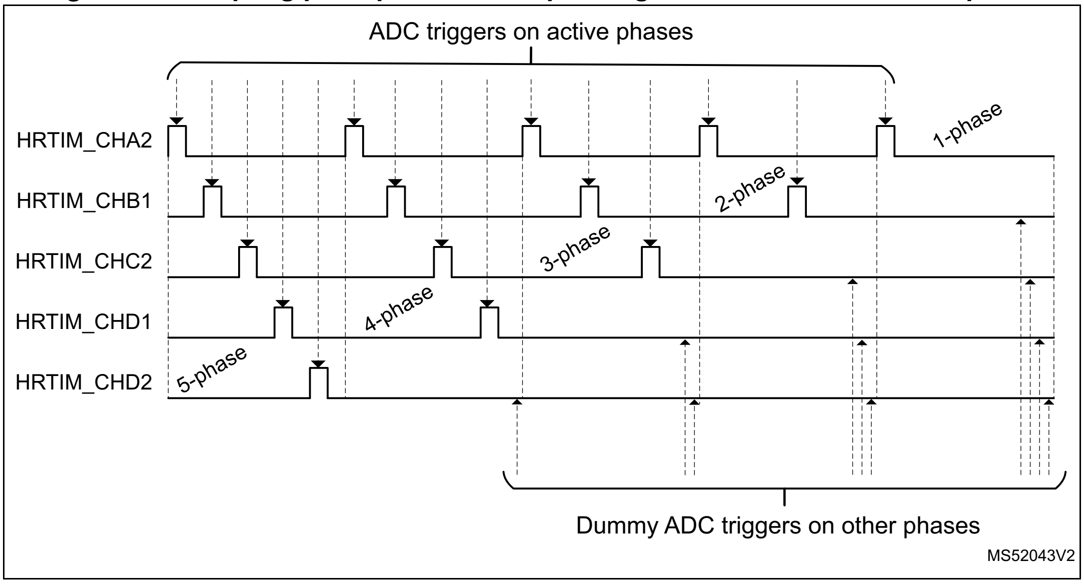
<table>
<tr><th colspan="2">Trigger</th><th>5-phase</th><th>4-phase</th><th>3-phase</th><th>2-phase</th><th>1-phase</th></tr>
<tr><td rowspan="2">1</td><td>Source</td><td colspan="5">TACMP2</td></tr>
<tr><td>Result: Data[0]</td><td colspan="5">Phase1_value</td></tr>
<tr><td rowspan="2">2</td><td>Source</td><td colspan="4">TBCMP2</td><td>TACMP3</td></tr>
<tr><td>Result: Data[1]</td><td colspan="4">Phase1_value</td><td>Dummy</td></tr>
<tr><td rowspan="2">3</td><td>Source</td><td colspan="3">TCCMP2</td><td>TBCMP3</td><td>TACMP4</td></tr>
<tr><td>Result: Data[2]</td><td colspan="3">Phase3_value</td><td colspan="2">Dummy</td></tr>
<tr><td rowspan="2">4</td><td>Source</td><td colspan="2">TDCMP3</td><td>TCCMP3</td><td>TBCMP4</td><td>TAPER</td></tr>
<tr><td>Result: Data[3]</td><td colspan="2">Phase4_value</td><td colspan="3">Dummy</td></tr>
<tr><td rowspan="2">5</td><td>Source</td><td colspan="3">TDCMP4</td><td>TDCMP3</td><td>MCMP4</td></tr>
<tr><td>Result: Data[4]</td><td>Phase5_value</td><td colspan="4">Dummy</td></tr>
</table>

A dummy trigger is still generated with the remaining timer resources when a phase is disabled. This makes it possible to have a given phase reading always located in the same place and avoids the need for re-programming the ADC or the DMA controller during converter operation to change the sequencer or DMA table length.

Let us consider the DMA is operating in circular mode and continuously fills the Data[0..4] RAM table with conversion results: *Table 6* shows that Phase 3_value is always stored in Data[2] location. For 2- and 1-phase configurations, Data[2] location is filled with a dummy conversion result, using respectively TBCMP3 and TACMP4 triggers (Timer B is stopped in 1-phase configuration and cannot be used for triggers anymore).

The dummy sampling points (see *Figure 22*) are placed close to the end the of multiphase converter period, back-to-back, so as to be let the maximum freedom for placing the relevant sampling point.

Figure 22. Sampling point placement depending on the number of active phase



Although this example is limited to five sampling points, it is possible to have a higher number of sampling events, to convert other inputs. A second ADC instance can also be used to monitor the operation of one (or several) additional independent converter(s).

7.1 Debugging ADC operation

This section gives a trick to debug ADC operation and verify easily the sampling points placement. It basically consists of building a simple DAC with the converter PWM outputs, as shown in *Figure 23*. Since the ADC samples are placed in the middle of the converter T_{on} time, it is easy to verify simultaneously that:

- the sampling point happens during ON time (or it results in a null value)
- the RAM table is filled in the correct order: first phase sampling must return V_{DD} / 2, second phase V_{DD} / 3, and so on
- no sampling points are missed during phase-shedding: in case of missing sample, the phase data are rotating within the table.

Figure 23. Making sure ADC sampling points are correctly placed

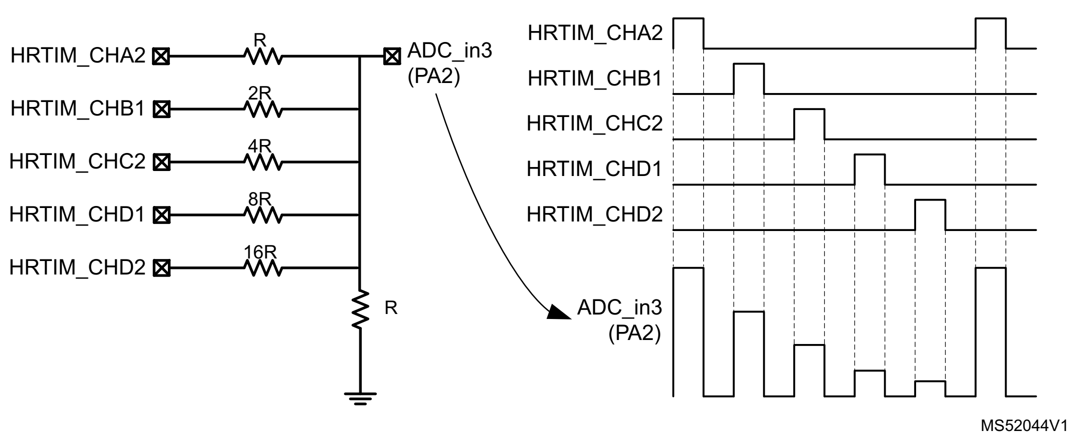


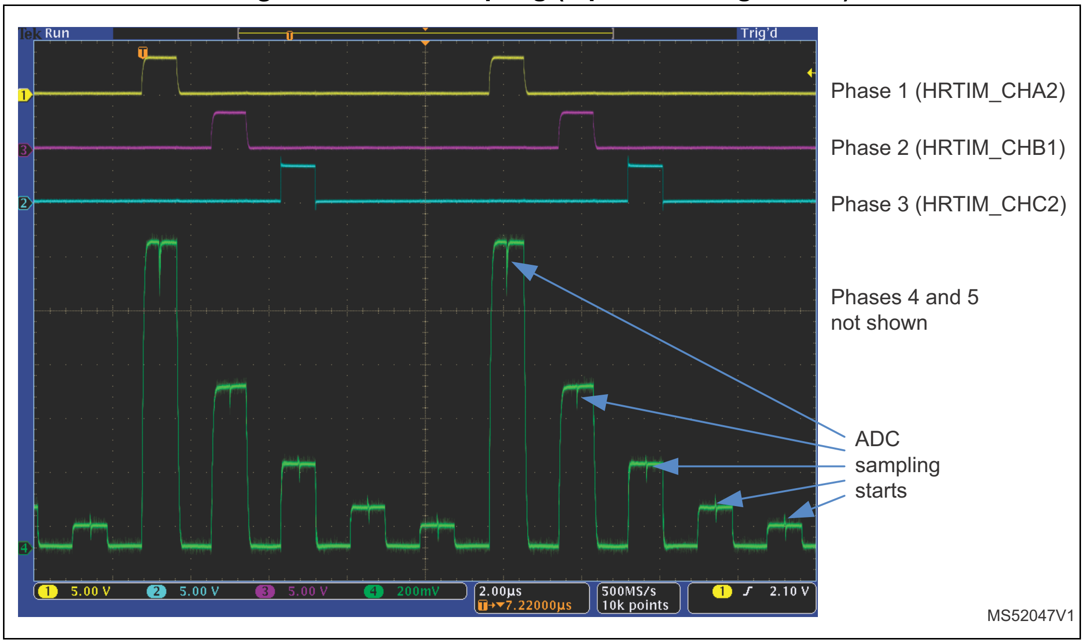
Table 7 gives the expected values for the different configurations. For burst mode, the configuration is similar to that for 1-phase.

Table 7. Expected conversion results

Trigger		5-phase	4-phase	3-phase	2-phase	1-phase
1	Result: Data[0]	V_{REF+} / 2				
2	Result: Data[1]	V_{REF+} / 3				0
3	Result: Data[2]	V_{REF+} / 5			0	
4	Result: Data[3]	V_{REF+} / 9		0		
5	Result: Data[4]	V_{REF+} / 17	0			

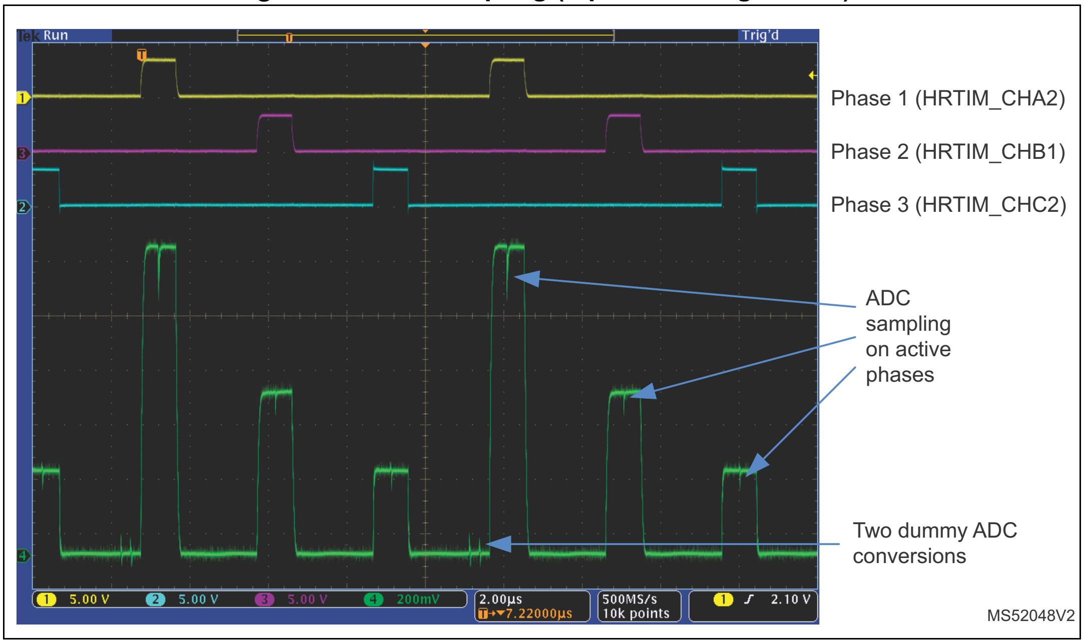
Another way of monitoring the ADC operation, with higher precision, is to use the oscilloscope. As described in *How to get the best ADC accuracy in STM32 microcontrollers* (AN2834), the SAR ADCs are providing a signature when the sampling phase starts, due to the sample and hold switch. This can be seen (provided the impedance of the analog signal sources is not very low) as small negative spikes, as in *Figure 24*.

Figure 24. ADC sampling (5-phase configuration)



The dummy conversions are also visible, see the bottom trace in *Figure 25*.

Figure 25. ADC sampling (3-phase configuration)



Caution: When using timer reset functions, the TIMxCR register holding the configuration must be updated before being active. Since the update source is the reset, the very first update must be forced by software before enabling the counters and starting converter operation.This is done using the TxSWU bits in TIMx registers.

Multiphase demonstration overview

The demonstration only allows the user to monitor the five HRTIM PWM channels, without any HW support (the active PWM outputs are either free or have no effect on the LED nor on the on-board buck-boost converter). The needed connections are:

- a test connection for FAULT input to tie PA12 to ground
- a test connection to monitor ADC operation.

The PWM signals are available on the following outputs:

- Phase 1: HRTIM_CHA2 (PA9)
- Phase 2: HRTIM_CHB1 (PA10)
- Phase 3: HRTIM_CHC2 (PB13)
- Phase 4: HRTIM_CHD1 (PB14)
- Phase 5: HRTIM_CHD2 (PB15)

The demonstration starts in 5-phase mode. If the push-button is pressed, the demonstration mode changes so that all phase shedding options are scanned: from 5- to 1-phase, and finally burst mode, as in *Figure 20*.

The ADC is configured to have conversions triggered in the middle of the converter ON time of each of five phases, on PA2 input, for this example (usually a sequence of five conversions on five inputs).

The FAULT1 input is enabled on PA12 (active low) to demonstrate PWM shut down (low level sensitive), for all outputs. When the fault is triggered (PA12 input connected to GND) HRTIM_CHA2, HRTIM_CHB1, HRTIM_CHC2, HRTIM_CHD1 and HRTIM_CHD2 signals are shut-down. The system can be re-armed by pressing the user button.

LEDs are indicating the following:

- green LED5 blinking: 5-phase operation
- blue LED6 blinking: 4-phase operation
- green LED5 continuous: 3-phase operation
- blue LED6 continuous: 2-phase operation
- both blue and green LEDs continuous: 1-phase operation
- both blue and green LEDs blinking: Burst mode operation
- red LED3: blinks when FAULT is triggered
- orange LED4: indicates the occurrence and duration of the PWM update ISR.

Demonstration code

The example code is provided is the HRTIM_Multiphase example. The #define SNIPPET statement provides an alternative to the HAL-based demonstration, for size-constrained applications.

Refer to *Section 10* for the target board and firmware access path.

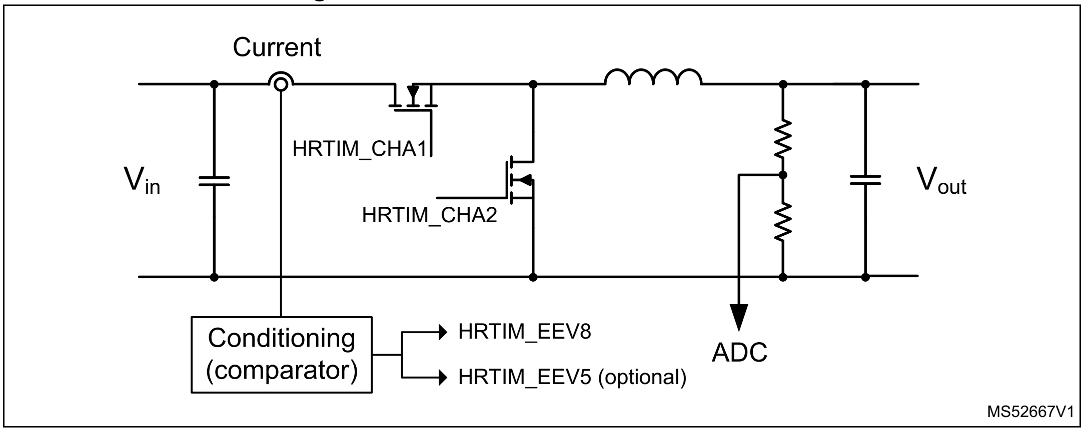
8 Cycle-by-cycle protection without deadtime insertion

This section focuses on:

- deadtime management during over-current
- autodelayed mode for deadtime insertion.

An over-current protected buck converter driven by the HRTIM is shown in *Figure 26*.

Figure 26. Current-mode buck converter

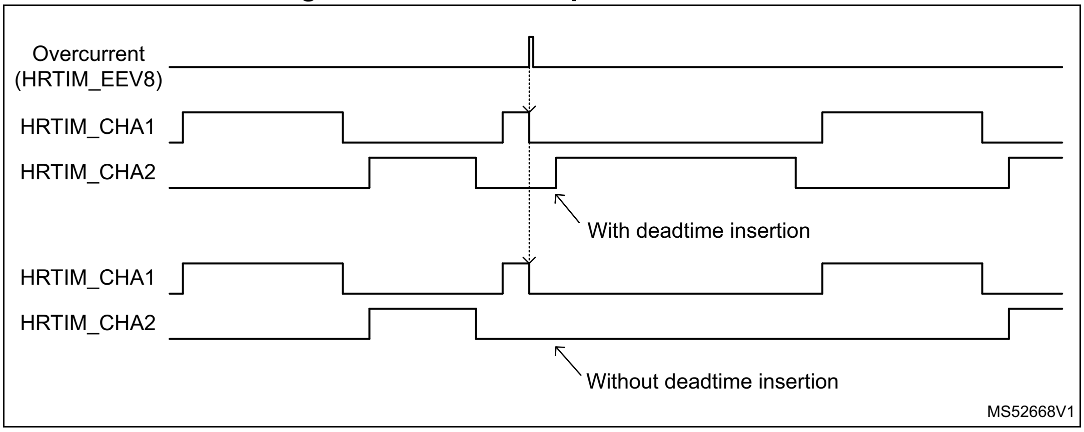


The high-side MOSFET driven by HRTIM_CHA1 is the main power switch, controlling the output voltage with fixed PWM frequency and varying duty cycle. The low-side MOSFET controlled by HRTIM_CHA2 performs the synchronous rectification.The two transistors are controlled with complementary PWM signals and deadtime insertion to avoid cross-conduction.

Deadtime can be generated with the deadtime generation unit, as detailed in *Section 4*. The deadtime insertion unit implies that the signals are always complementary, even in case of over-current protection resetting the high-side MOSFET command.

It can be necessary to have an alternative protection scheme where both MOSFETs are turned-off during the switching period, following an over-current protection. *Figure 27* shows the two different behaviors.

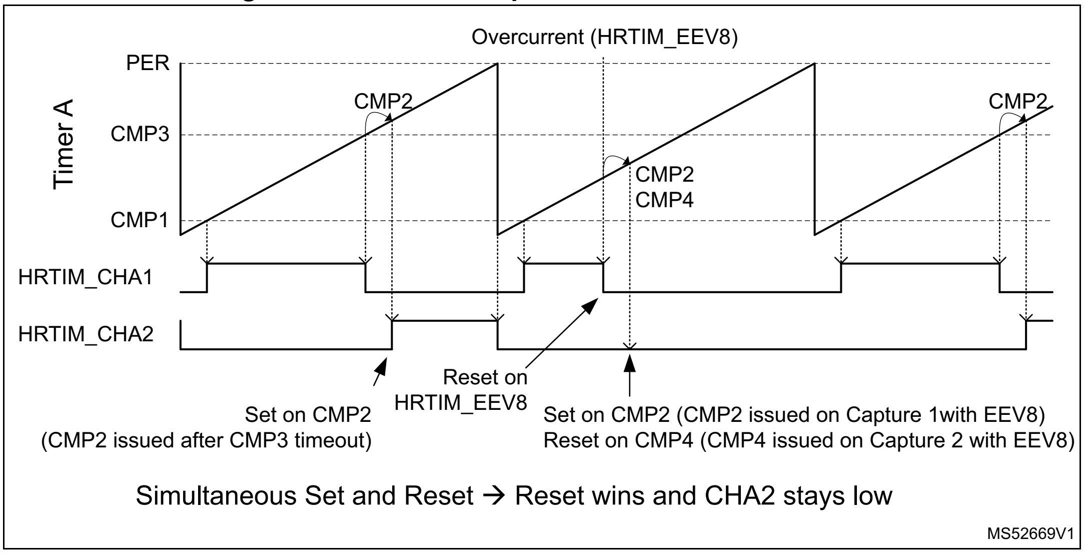
Figure 27. Overcurrent protection scheme



To implement a protection scheme without using the deadtime insertion unit the leading-edge deadtime is generated with the CMP1 (HRTIM_CHA2 is reset on Timer A period and HRTIM_CHA1 is set on Timer A CMP1). The trailing edge deadtime is generated using CMP2 autodelayed mode triggered on HRTIM_EEV8 input, with timeout on CMP3, as follows:

- the CMP3 register defines the duty cycle on HRTIM_CHA1 during regular operating conditions (no over-current)
- when there is no overcurrent, the autodelayed mode is triggered on CMP3 timeout, so that the HRTIM_CHA2 output is set after a deadtime defined by the CMP2 register value
- in case of overcurrent, the autodelayed mode is triggered by the HRTIM_EEV8 input and it issues a set request for the HRTIM_CHA2 output after the deadtime defined by CMP2. To cancel out this set request, a simultaneous reset request is also issued, using the CMP4 register in autodelayed mode. The reset request has a higher priority compared to the set request, consequently the HRTIM_CHA2 output stays low, as shown in *Figure 28*.

Figure 28. Overcurrent protection without deadtime



The autodelayed CMP4 must be triggered by the same source as the autodelayed CMP2 (here HRTIM_EEV8), with the same compare value. The timeout mode must be disabled: the CMP4 event is not generated when there is no over-current condition.

The autodelayed CMP2 is triggered by the capture 1 event, on HRTIM_EEV8 rising and falling-edges, and the autodelayed CMP4 is triggered by the capture 2 event, also on HRTIM_EEV8 rising and falling-edges.

Where the over-current pulse occurs before the turn-on of CMP1, the HRTIM_EEV8 is latched and delayed up to CMP1 event, to make sure the over-current information is acknowledged and maintains both outputs at zero, even in case of early arrival.

This scheme is working for over-current pulses with short duration, within the switching period. This is usually the case in case of short-term overload.

For applications requiring PWM shut-down for a longer duration, across several PWM periods (multi-cycle overload), the proposed scheme must be completed by a second HRTIM_EEVx input, here the HRTIM_EEV5, connected in parallel with HRTIM_EEV8.

The HRTIM_EEV5 event is programmed to be active on level, so that the two outputs are maintained low as long as the over-current condition is present (see *Figure 29*). *Figure 30* shows the waveforms when there is only the HRTIM_EEV8 event active.

Figure 29. Multi-cycle overload protection with dual external events

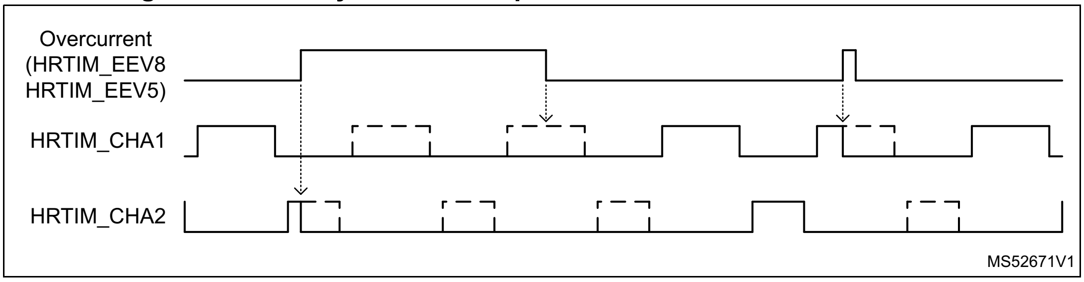
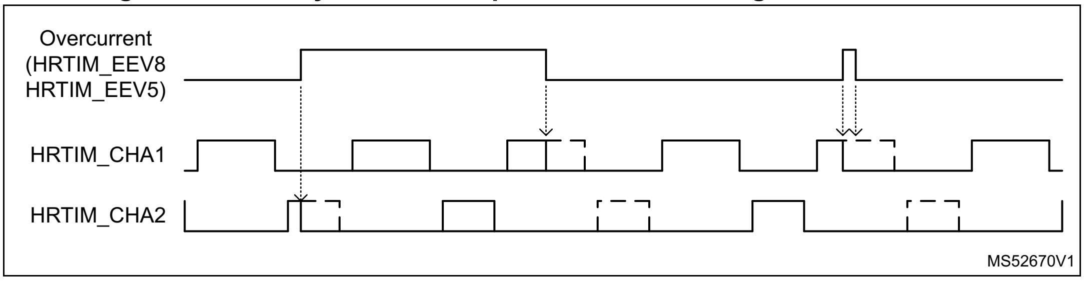


Figure 30. Multi-cycle overload protection with a single external events



In summary, the output configuration (muti-cycle overload option between brackets) is:

- HRTIM_CHA1 set: CMP1
- HRTIM_CHA1 reset: CMP3 or HRTIM_EEV8 edge (or HRTIM_EEV5 active)
- HRTIM_CHA2 set: CMP2 (autodelayed)
- HRTIM_CHA2 reset: Timer A period or CMP4 autodelayed (or HRTIM_EEV5 active)

Demonstration code

The example code is provided in the HRTIM_CBC_Deadtime example.

Refer to *Section 10* for the target board and firmware access path.

9 Other examples

The cookbook will be extended gradually, for new examples visit the STM32 product pages on *www.st.com*.

10 Where to find software examples?

This section summarizes on which boards the examples are running, and where to find them in the latest CubeFW Expansion Packages.

An icon in *Table 8* indicates if the project can be regenerated using STM32CubeMX. If so, the corresponding ioc file is available at the same firmware location.

Table 8. Summary of examples

Example	Board (compatible board)	Firmware location	STM32 CubeMX
HRTIM basic operating principles	32F3348DISCOVERY (NUCLEO-F334R8)	STM32Cube_FW_F3_V1.10.0\Projects\ STM32F3348-Discovery\Examples\ HRTIM\HRTIM_BasicPWM	-
		STM32Cube_FW_F3_V1.10.0\Projects\ STM32F3348-Discovery\Examples\ HRTIM\HRTIM_Snippets	-
	NUCLEO-G474RE	STM32Cube_FW_G4_Vx.y.z\Projects\ NUCLEO-G474RE\Examples_LL\HRTIM\ HRTIM_Basic_Single_PWM HRTIM_Basic_PWM_Master HRTIM_Basic_Multiple_PWM HRTIM_Basic_Arbitrary_Waveform	MX
		STM32Cube_FW_G4_Vx.y.z\Projects\ NUCLEO-G474RE\Examples\HRTIM\ HRTIM_Basic_Single_PWM HRTIM_Basic_PWM_Master HRTIM_Basic_Multiple_PWM HRTIM_Basic_Arbitrary_Waveform	MX
Voltage mode dual buck converter	32F3348DISCOVERY (NUCLEO-F334R8)	STM32Cube_FW_F3_Vx.y.z\Projects\ STM32F3348-Discovery\Examples\ HRTIM\HRTIM_DualBuck	-
	B-G474E-DPOW1 (NUCLEO-G474RE)	STM32Cube_FW_G4_Vx.y.z\Projects\ B-G474E-DPOW1\Examples_MIX\ HRTIM\HRTIM_Dual_Buck	MX
Voltage mode buck converter with synchronous rectification and fault protection	32F3348DISCOVERY	STM32Cube_FW_F3_Vx.y.z\Projects\ STM32F3348-Discovery\Examples\ HRTIM\HRTIM_BuckSyncRect	-
	B-G474E-DPOW1	STM32Cube_FW_G4_Vx.y.z\Projects\ B-G474E-DPOW1\Examples_MIX\ HRTIM\HRTIM_Buck_Sync_Rect	MX

Table 8. Summary of examples (continued)

Example	Board (compatible board)	Firmware location	STM32 CubeMX
Non-inverting buck-boost converter	32F3348DISCOVERY	STM32Cube_FW_F3_Vx.y.z\Projects\STM32F3348-Discovery\Examples\HRTIM\HRTIM_BuckBoost	-
Non-inverting buck-boost converter	32F3348DISCOVERY	STM32Cube_FW_F3_Vx.y.z\Projects\STM32F3348-Discovery\Examples_LL\HRTIM\HRTIM_BuckBoost	-
Non-inverting buck-boost converter	B-G474E-DPOW1	STM32Cube_FW_G4_Vx.y.z\Projects\B-G474E-DPOW1\Examples_MIX\HRTIM\HRTIM_Buck_Boost	MX
Transition mode power factor controller	32F3348DISCOVERY (NUCLEO-F334R8)	STM32Cube_FW_F3_Vx.y.z\Projects\STM32F3348-Discovery\Examples\HRTIM\HRTIM_TM_PFC	-
Multiphase buck converter	32F3348DISCOVERY (NUCLEO-F334R8)	STM32Cube_FW_F3_Vx.y.z\Projects\STM32F3348-Discovery\Examples\HRTIM\HRTIM_Multiphase	-
Cycle-by-cycle protection without deadtime insertion	NUCLEO-G474RE	STM32Cube_FW_G4_Vx.y.z\Projects\NUCLEO-G474RE\Examples_LL\HRTIM\HRTIM_CBC_Deadtime	MX

Appendix A HRTIM v2

A.1 Features

This sections summarizes the main improvements of the HRTIM v2.0. For further details refer to the reference manual.

A.1.1 Sixth timing unit (Timer F)

The HRTIM now has twelve outputs, to cover larger interleaved topologies, such as triple-interleaved half-bridge LLC (3 x 4 outputs required, two at primary for half-bridge control and two at secondary for synchronous rectification), dual interleaved full-bridge LLC (2 x 6 outputs) and dual phase-shifted full bridge converters (2 x 6 outputs).

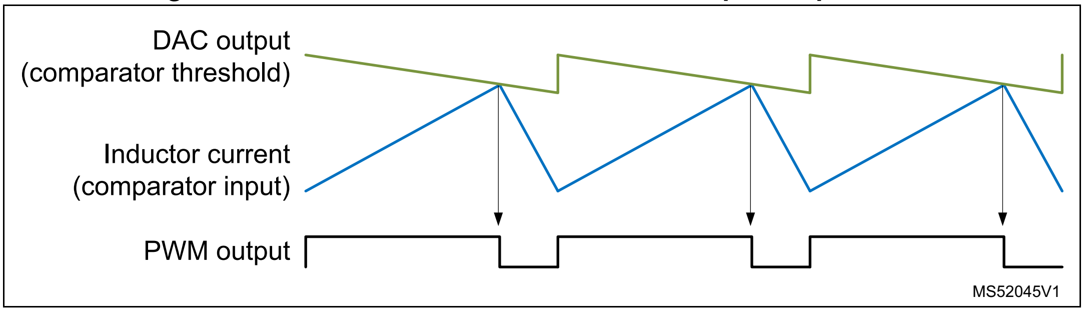
This comes with a sixth FAULT input circuitry.

A.1.2 Dual channel DAC triggers

This feature, coupled with the high-speed DAC, can be used to generate a sawtooth wave synchronized with the PWM signals. As indicated by the arrows in *Figure 31*, the PWM output is reset every time the comparator input (in blue) reaches the threshold (in green).

This technique is known as slope compensation, and is necessary to ensure the stability of peak current-mode converters.

Figure 31. Peak current-mode control with slope compensation



A.1.3 External event counter

The HRTIM v2 features event-driven filtering: an external event must occur a given number of times before being considered valid. This is typically for implementing valley skipping mode for flyback converters.

A.1.4 Extended fault features

The FAULT events can be filtered out with a programmable blanking window. Each channel has also a fault counter, so that a fault is active only if it has occurred during a given number of PWM periods.

A.1.5 Push-pull improvements

It is now possible to use the push-pull mode with dead time insertion and in single-shot operating mode (this mode is not allowed on HRTIMv1).

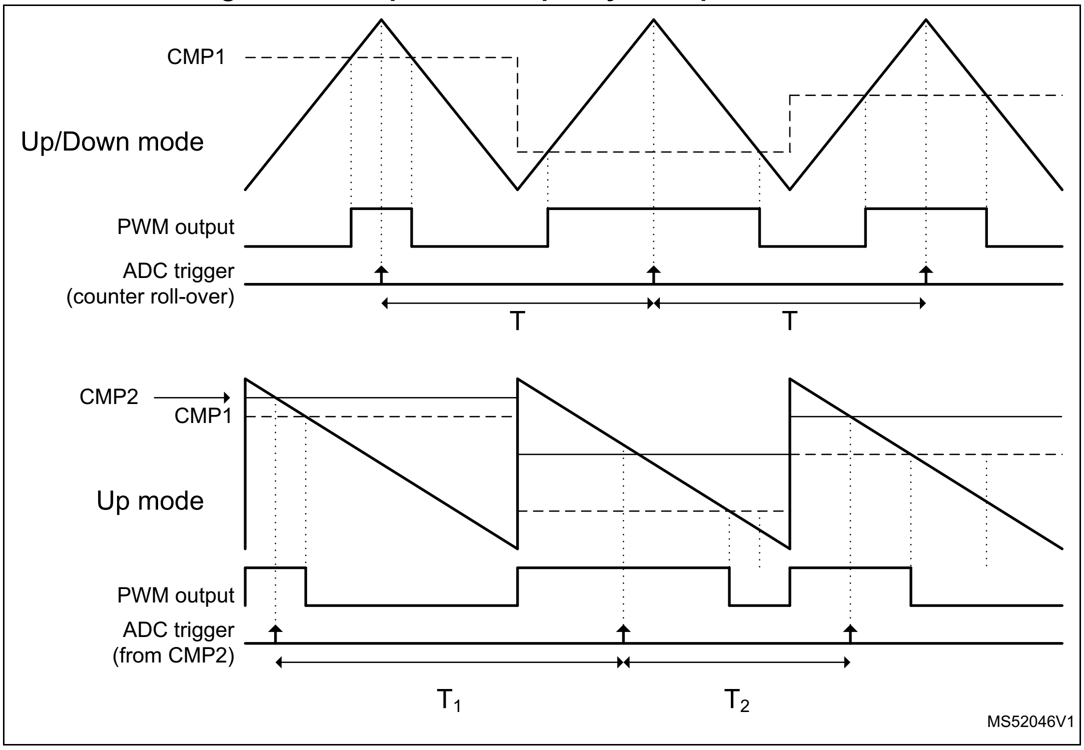
A.1.6 Classical PWM mode

This mode gives the possibility to work without any shadow register and preloading mechanism, to force the state of an output as soon as the compare register is written by the software. This makes it possible to have early turn-off or late turn-on, and can slightly improve the phase margin of converters.

A.1.7 Up-down mode

This counter operating mode is commonly used for motor control applications. It offers benefits for power converters as well. It simplifies the ADC sampling, when, as shown in *Figure 32*, a constant sampling frequency and sampling at the middle of the pulse is needed.

Figure 32. Comparison of up-only and up-down modes



A.1.8 ADC post-scaler

For high switching-frequency application, it is possible to reduce the ADC triggering rate with the ADC post-scaler. Each ADC trigger can be individually adjusted down to 1 out of 32 PWM periods.

A.1.9 Null duty cycle mode

On HRTIMv2, it is possible to force a null duty cycle by writing a null value in the Compare1 and/or Compare3 register.

A.1.10 New interleaving modes

Two new interleaved modes have been added for triple and quad interleaved or multiphase converters.

A.1.11 Swap mode

It is possible to swap two outputs in a single register access, without having to reprogram the output crossbars. All control bits are located in the same register to swap multiple PWM pairs simultaneously.

Appendix B Software migration

B.1 HRTIM v1.0 to HRTIMv1.1

The only difference between v1 and v1.1 versions is the removal of DLL in the HRTIMv1.1.

The HRTIM_DLLCR register is reserved. All accesses to HRTIM_DLLCR the must be removed and the DLL calibration procedure must be removed when porting the code.

As a consequence:

- all clock prescaler ratios using the high-resolution (CKPSC[2:0] = 000, 001, 010, 011, 100 in HRTIM_MCR register) are reserved
- the timings must be adjusted considering that the highest possible resolution is 2.5 ns.

Apart from these points, all features and operating modes are the same. The register and bit mapping are strictly compatible from one version to the other.

B.2 HRTIMv1 to HRTIMv2

The HRTIMv2 includes a sixth timing unit and several additional features.

Although the new control and status bits have been added in HRTIMv1 reserved areas to maintain compatibility whenever possible, the two timers do not have binary compatibility.

As some registers are full on HRTIMv1, it has been chosen to maintain all bits in single 32-bit registers and break compatibility. When bit positions are changed, this is managed transparently when using the HAL and LL libraries, otherwise it is required to modify the source code to adapt to the HRTIMv2.

The following tables summarize all non-binary compatible differences between v1 and v2, in gray shaded cells (NA indicates not available).

Table 9. Events mapping

Destination and version		Source																							
		Timer A				Timer B				Timer C				Timer D				Timer E				Timer F			
		CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4
Timer A	v1	-	-	-	-	1	2	-	3	-	4	5	-	6	7	-	-	-	-	8	9	NA			
	v2	-	-	-	-	1	2	-	-	-	3	4	-	5	6	-	-	-	-	7	8	-	-	-	9
Timer B	v1	1	2	-	3	-	-	-	-	-	-	4	5	-	-	6	7	8	9	-	-	NA.			
	v2	1	2	-	-	-	-	-	-	-	-	3	4	-	-	5	6	7	8	-	-	-	-	9	-
Timer C	v1	-	1	2	-	-	3	4	-	-	-	-	-	-	5	-	6	-	7	8	9	NA.			
	v2	-	1	2	-	-	3	4	-	-	-	-	-	-	5	-	6	-	-	7	8	-	9	-	-
Timer D	v1	1	-	-	2	-	3	-	4	5	-	6	7	-	-	-	-	8	-	-	9	NA.			
	v2	1	-	-	2	-	3	-	4	-	-	-	5	-	-	-	-	6	-	-	7	8	-	9	-

Table 9. Events mapping (continued)

Destination and version		Source																							
		Timer A				Timer B				Timer C				Timer D				Timer E				Timer F			
		CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4	CMP1	CMP2	CMP3	CMP4
Timer E	v1	-	-	1	2	-	-	3	4	5	6	-	-	7	8	-	9	-	-	-	-	-	-	-	
	v2	-	-	-	1	-	-	2	3	4	5	-	-	6	7	-	-	-	-	-	-	-	-	8	9
Timer F	v1	NA																							
	v2	-	-	1	-	2	-	-	3	4	-	-	5	-	-	6	7	-	8	9	-	-	-	-	-

Table 10. Windowing signals mapping per timer (EEFLTR[3:0] = 1111)

HRTIM version	Destination	Timer A	Timer B	Timer C	Timer D	Timer E	Timer F
v1	TIMWIN (source)	Timer B CMP2	Timer A CMP2	Timer D CMP2	Timer C CMP2	Timer D CMP2	NA
v2		Timer B CMP2	Timer A CMP2	Timer D CMP2	Timer C CMP2	Timer F CMP2	Timer E CMP2

Table 11. Filtering signals mapping per timer

Destination and version		Source																							
		Timer A				Timer B				Timer C				Timer D				Timer E				Timer F			
		CMP1	CMP2	CMP4	HRTIM_CHA2	CMP1	CMP2	CMP4	HRTIM_CHB2	CMP1	CMP2	CMP4	HRTIM_CHC2	CMP1	CMP2	CMP4	HRTIM_CHD2	CMP1	CMP2	CMP4	HRTIM_CHE2	CMP1	CMP2	CMP4	HRTIM_CHF2
Timer A	v1	-	-	-	-	1	-	2	3	4	-	5	6	7	-	-	-	-	8	-	-	NA			
	v2	-	-	-	-	1	-	2	3	4	-	5	-	7	-	-	-	-	8	-	-	6	-	-	-
Timer B	v1	1	-	2	3	-	-	-	-	4	5	-	6	-	7	-	-	8	-	-	-	NA			
	v2	1	-	2	3	-	-	-	-	4	5	-	-	-	7	-	-	8	-	-	-	-	6	-	-
Timer C	v1	-	1	-	-	2	-	3	4	-	-	-	-	5	-	6	7	-	-	8	-	NA			
	v2	-	1	-	-	2	-	3	-	-	-	-	-	5	-	6	7	-	-	8	-	4	-	-	-
Timer D	v1	1	-	-	-	-	2	-	-	3	4	-	5	-	-	-	-	6	-	7	8	NA			
	v2	1	-	-	-	-	2	-	-	3	4	-	5	-	-	-	-	6	-	7	-	-	-	8	-
Timer E	v1	-	1	-	-	2	-	-	-	3	-	4	5	6	-	7	8	-	-	-	-	NA			
	v2	-	1	-	-	2	-	-	-	3	-	-	-	6	-	7	8	-	-	-	-	-	-	4	5
Timer F	v1	NA																							
	v2	-	-	1	-	-	2	-	-	-	-	3	-	-	4	5	-	6	-	7	8	-	-	-	-

Table 12. HRTIM ADC trigger 1 and 3 registers (HRTIM_ADC1R, HRTIM_ADC3R)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
v1	ADC1 TEPER	ADC1 TEC4	ADC1 TEC3	ADC1 TEC2	ADC1 TDPER	ADC1 TDC4	ADC1 TDC3	ADC1 TDC2	ADC1 TCPER	ADC1 TCC4	ADC1 TCC3	ADC1 TCC2	ADC1 TBRST	ADC1 TBPER	ADC1 TBC4	ADC1 TBC3
v2	ADC1 TEPER	ADC1 TEC4	ADC1 TEC3	ADC1 TFRST	ADC1 TDPER	ADC1 TDC4	ADC1 TDC3	ADC1 TFPER	ADC1 TCPER	ADC1 TCC4	ADC1 TCC3	ADC1 TFC4	ADC1 TBRST	ADC1 TBPER	ADC1 TBC4	ADC1 TBC3
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
v1	ADC1 TBC2	ADC1 TARST	ADC1 TAPER	ADC1 TAC4	ADC1 TAC3	ADC1 TAC2	ADC1 EEV5	ADC1 EEV4	ADC1 EEV3	ADC1 EEV2	ADC1 EEV1	ADC1 MPER	ADC1 MC4	ADC1 MC3	ADC1 MC2	ADC1 MC1
v2	ADC1 TFC3	ADC1 TARST	ADC1 TAPER	ADC1 TAC4	ADC1 TAC3	ADC1 TFC2	ADC1 EEV5	ADC1 EEV4	ADC1 EEV3	ADC1 EEV2	ADC1 EEV1	ADC1 MPER	ADC1 MC4	ADC1 MC3	ADC1 MC2	ADC1 MC1

Table 13. HRTIM ADC trigger 2 and 4 registers (HRTIM_ADC2R, HRTIM_ADC4R)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
v1	ADC2 TERST	ADC2 TEC4	ADC2 TEC3	ADC2 TEC2	ADC2 TDRST	ADC2 TDPER	ADC2 TDC4	ADC2 TDPER	ADC2 TDC2	ADC2 TCRST	ADC2 TCPER	ADC2 TCC4	ADC2 TCC3	ADC2 TCC2	ADC2 TBPER	ADC2 TBC4
v2	ADC2 TERST	ADC2 TEC4	ADC2 TEC3	ADC2 TEC2	ADC2 TDRST	ADC2 TDPER	ADC2 TDC4	ADC2 TFPER	ADC2 TDC2	ADC2 TCRST	ADC2 TCPER	ADC2 TCC4	ADC2 TFC4	ADC2 TCC2	ADC2 TBPER	ADC2 TBC4
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
v1	ADC2 TBC3	ADC2 TBC2	ADC2 TAPER	ADC2 TAC4	ADC2 TAC3	ADC2 TAC2	ADC2 EEV10	ADC2 EEV9	ADC2 EEV8	ADC2 EEV7	ADC2 EEV6	ADC2 MPER	ADC2 MC4	ADC2 MC3	ADC2 MC2	ADC2 MC1
v2	ADC2 TFC3	ADC2 TBC2	ADC2 TAPER	ADC2 TAC4	ADC2 TFC2	ADC2 TAC2	ADC2 EEV10	ADC2 EEV9	ADC2 EEV8	ADC2 EEV7	ADC2 EEV6	ADC2 MPER	ADC2 MC4	ADC2 MC3	ADC2 MC2	ADC2 MC1

Appendix C Reference documents

This section lists the documentation (available on www.st.com) related to the HRTIM, grouped by product.

Some documents, such as the application notes, contain informations and concepts that can be applied to any product.

C.1 STM32F334x

STM32F334x4/x6/x8 datasheet

RM0364 *STM32F334xx advanced Arm®-based 32-bit MCUs*

UM1733 *Getting started with STM32F334 Discovery kit*

UM1735 *Discovery kit for STM32F3 series with STM32F334C8 MCU*

UM1736 *Getting started with STM32F334 discovery kit software development tools*

AN4885 *High brightness LED dimming using the STM32F3348 Discovery kit*

AN4449 *Buck-boost converter using the STM32F334 Discovery kit*

AN4296 *Use STM32F3/STM32G4 CCM SRAM with IAR™ EWARM, Keil® MDK-ARM and GNU-based toolchains*

C.2 STM32H74x / STM32H75x

STM32H743/753xI datasheet

STM32H750xB datasheet

RM0364 *STM32H743/753 and STM32H750 advanced Arm®-based 32-bit MCUs*

C.3 STM32G47x

STM32G471/3/4 datasheet

STM32G484 datasheet

RM0440 *STM32G4xx advanced Arm®-based 32-bit MCUs*

AN5094 *Migrating between STM32F334/303 lines and STM32G474xx/G431xx microcontrollers*

AN4767 *On-the-fly firmware update for dual bank STM32 microcontrollers / Software expansion for STM32Cube*

STM32G4 Online training

C.4 Others

AN4232 *Getting started with analog comparators for STM32F3 Series and STM32G4 Series devices*

AN4013 *STM32 cross-series timer overview*

AN4856 *STEVAL-ISA172V2: 2 kW fully digital AC - DC power supply (D-SMPS) evaluation board*

AN4930 *500 W fully digital AC-DC power supply (D-SMPS) evaluation board*

UM2348 *Getting started with the STEVAL-DPSLLCK1 evaluation kit for the 3 kW full bridge LLC digital power supply*

STEVAL-DPSLLCK1: 3 kW Full Bridge LLC resonant digital power supply evaluation kit

Revision history

Table 14. Document revision history

Date	Revision	Changes
19-Jun-2014	1	Initial release.
26-Mar-2019	2	Updated *Introduction*, *Section 1: Getting the kitchen ready*, *Section 1.2: Hardware set-up*, *Section 1.3: Tools set-up*, *Section 1.5.1: System clock initialization*, *Section 1.5.4: HRTIM I/Os initialization*, *Section 1.5.5: Other peripherals initialization*, *Section 2: HRTIM basic operating principles*, *Section 2.2: Generating multiple PWMs* and *Section 9: Other examples*. Added *Section 1.4: HRTIM versions*, *Section 7: Multiphase buck converter*, *Appendix A: HRTIM v2*, *Appendix B: Software migration*, *Appendix C: Reference documents* and their subsections. Updated *Table 1: Applicable products*. Added *Table 2: HRTIM features according to products where it is integrated*, *Table 3: HRTIM input frequency operating range* and *Table 5: Timer resolution / minimum PWM frequency for fHRTIM=170 MHz*. Minor text edits across the whole document.
16-Jul-2019	3	Updated *Introduction*, *Section 2.2: Generating multiple PWMs*, *Section 2.3: Generating PWM with other timing units and the master timer*, *Section 2.4: Arbitrary waveform generation*, *Section 3: Voltage mode dual buck converter*, *Section 4: Voltage mode buck converter with synchronous rectification and fault protection*, *Section 5: Non-inverting buck-boost converter*, *Section 6: Transition mode power factor controller*, *Section 7: Multiphase buck converter* and *Multiphase demonstration overview*. Added *Section 8: Cycle-by-cycle protection without deadtime insertion* and *Section 10: Where to find software examples?*. Updated *Table 3: HRTIM input frequency operating range*. Updated *Figure 1: Basic PWM generation*, *Figure 2: HRTIM configuration for generating basic PWM signals*, *Figure 3: Generation of multiple PWM signals*, *Figure 5: Arbitrary waveform generation*, *Figure 7: VM buck waveforms, including ADC sampling and interrupts*, *Figure 8: PWM interrupts and register update*, *Figure 9: Voltage mode buck with synchronous rectification*, *Figure 10: Buck operation with FAULT*, *Figure 11: Non-inverting buck-boost converter*, *Figure 13: Buck-boost converter operating waveforms*, *Figure 14: Transition mode PFC*, *Figure 15: Transition mode PFC operation at Ton max and during over-current*, *Figure 16: Transition mode PFC operation at Toff max and Toff min*, *Figure 17: 5-phase interleaved buck converter*, *Figure 18: 5-phase interleaved buck converter control waveforms*, *Figure 19: Disabling a phase with the master timers*, *Figure 20: Low-load management with phase shedding and burst mode*, *Figure 22: Sampling point placement depending on the number of active phase*, *Figure 23: Making sure ADC sampling points are correctly placed*, *Figure 24: ADC sampling (5-phase configuration)* and *Figure 25: ADC sampling (3-phase configuration)*.

Table 14. Document revision history (continued)

Date	Revision	Changes
09-Jan-2020	4	Updated *Table 1: Applicable products* and *Table 8: Summary of examples*. Minor text edits across the whole document.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to *www.st.com/trademarks*. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2020 STMicroelectronics – All rights reserved