SR5 E1 line: getting started with the HRTIM

Introduction

This technical note introduces the high-resolution timer (HRTIM) main features and gives the background knowledge to start quickly the programming of HRTIM.

The HRTIM is a hi-resolution and complex waveform builder, able to generate up to 12 signals. It handles a large variety of input signals for control, synchronization, on protection purposes.

Thanks to its modular architecture it is possible to address several conversion topologies and multiple parallel converters, with the possibility to reconfigure them during runtime.

Even though, this peripheral may appear complex, mostly because of the large control register set, this technical note shows how the HRTIM programming is actually simple.

To complement the extensive description provided in reference manuals, this document includes some quick-start examples. The HRTIM module is available in the products listed in Table 1. Applicable products.

Table 1. Applicable products

Type	Part numbers or product lines
Microcontrollers	SR5E1E3, SR5E1E7 (1)

1. *The SR5 E1 line microcontroller is built on Arm® architecture technology, with two Arm Cortex® -M7 cores, and two instances of high-resolution timer (HRTIM).*

1 Overview

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

Section Appendix B 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 Hardware set-up

The best option to start (and go on) experimenting with the HRTIM on the SR5E1x device is the evaluation board SR5E1-EVBE7000P. It is a stand-alone unit allowing access to the CPU, access to the I/O pins as well as to any board peripherals.

An oscilloscope, with a sampling rate above 1 GS/s at least, is mandatory.

1.2 MCU and HRTIM set-up

1.2.1 System clock initialization

To provide the high-resolution, the HRTIM must be fed directly from the high frequency output of the PLL.

The clock period is divided by 32 by the DLL to provide high-resolution, as if the HRTIM clock frequency were multiplied by 32. The table below summarizes the allowed frequency ranges for the SR5 E1 Line products.

Table 2. Operating range of HRTIM input frequency

Device	PLL input	Input frequency (MHz)		HRTIM frequency (GHz)		Resolution (ps)	
		Min	Max	Min	Max	Min	Max
SR5E1x	Any	200	306.7	6.400	9.814.4	102(1)	156(2)

1. *102 ps, @306.7 MHz*
2. *156 ps, @200 MHz*

Note: The HRTIM input clock is directly derived by CPU clock branch of SR5E1x that is the system clock frequency, SYS_CLK.

The actual enabling of the peripheral clock is managed by the HRTIMERx enable bit of the RCC_ AHB1HENR register (for more details refer to the reset and clock control (RCC) chapter in the reference manual of the device, Section Appendix B Reference documents).

Note: When the peripheral clock is not active, the peripheral registers read or write accesses are not supported.

1.3 HRTIM initialization

This section details step by step how the HRTIM is initialized.

1.3.1 HRTIM clock initialization

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

1. selection of the high-speed PLL output for the HRTIM in RCC_CFGR register
2. clock enable for the registers mapped on the RCC_ AHB1HENR register.

1.4 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 the high-resolution can be used. During the HRTIM operation, if voltage or temperature conditions have changed, the software can relaunch the calibration. This procedure permits to compensate for potential voltage and temperature drifts.

It is also possible to enable periodic calibration by hardware. The programming of the HRTIM_DLLCR register gives the calibration rates as described in the following table.

Table 3. HRTIM DLL calibration rate for f_{HRTIM}=300 MHz

CALEN	CALRTE			
	00	01	10	11
0x1	1048576 * t_{HRTIM} (3.495 ms)	131072 * t_{HRTIM} (437 µs)	16384 * t_{HRTIM} (55 µs)	2048 * t_{HRTIM} (6.8 µs)

Note: The high-resolution can be used once the DLLRDY flag has been set.

Here the example code is provided:

```
/* DLL calibration: periodic calibration enabled; period set to 6.8µs */
```

```
HRTIM1->sCommonRegs.DLLCR = HRTIM_CALIBRATIONRATE_6_8| HRTIM_DLLCR_CALEN;
/* Check DLL end of calibration flag */
while(HRTIM1->sCommonRegs.ISR & HRTIM_IT_DLLRDY == RESET);
```

Note: This code causes the execution to stall if the DLL does not lock. This can happen in case that the HSE oscillator is not properly configured. The SW that performs the calibration can handle the stall with a timeout verification and redirects in an error handler if necessary.

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

1.5 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.

For the SR5 E1 Line products, the HRTIM alternate functions are available:

- on AF3 and AF8 alternate functions by configuring the AFRL (for AF3) and AFRH (for AF8) registers.

The HRTIM I/Os initialization must be done in two phases.

- The HRTIM inputs are initialized first, before the HRTIM registers.
- The HRTIM outputs must be initialized after the HRTIM control registers programming 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.

1.6 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 enables the HRTIM timer D output 1 (TD1) and toggles it by software.

```
/* Enable HRTIM clock*/
_HRTIM1_CLK_ENABLE();
/* DLL calibration: periodic calibration enabled, period set to 6.8µs *
/ HRTIM1->sCommonRegs.DLLCR = HRTIM_CALIBRATIONRATE_6_8|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;
}
```

For the remaining part of this document, the clock and DLL initialization part are not repeated.

1.7 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.

1.7.1 Nested vectored interrupt controller (NVIC)

The HRTIM interrupts requests are grouped in eight interrupts vectors. All faults are grouped within a distinct vector that can be set with a very high priority. Refer to the SR5E1x vector table in the reference manual of the device, Section Appendix B Reference documents.

1.7.2 DMA controller

Most of interrupt requests can be used as DMA requests and are grouped on DMA channels (one per timing unit, including the master timer). Refer to the DMAMUX mapping section in the reference manual of the device, Section Appendix B Reference documents.

1.7.3 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.

Refer to HRTIMx (x = 1,2) external events mapping and associated features table in the reference manual of the device, Section Appendix B Reference documents.

1.7.4 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). Refer to HRTIMx (x = 1,2) external events mapping and associated features table in the reference manual of the device, Section Appendix B Reference documents.

1.7.5 DACs

The DACs are used to define the comparator thresholds. They can be updated synchronously with HRTIM operation by means of HRTIM DAC triggers. Refer to the HRTIMx (x = 1,2) DAC triggers connections table in the reference manual of the device, Section Appendix B Reference documents.

1.7.6 TIMs

The TIMx timers can trigger external events on the HRTIM (for output set/reset or counter reset purposes). Refer to HRTIMx (x = 1,2) external events mapping and associated features table in the reference manual of the device, Section Appendix B Reference documents.

1.8 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 enables the HRTIM timer D output 1 (TD1) and toggles it by software.

```
/* Enable HRTIM clock*/
_HRTIM1_CLK_ENABLE();
/* DLL calibration: periodic calibration enabled, period set to 6.8µs *
/ HRTIM1->sCommonRegs.DLLCR = HRTIM_CALIBRATIONRATE_6_8|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;
}
```

For the remaining part of this document, the clock and DLL initialization part are not repeated.

2 HRTIM basic operating principles

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

- The master timer provides synchronization signals to the 6 timer units and no output.
- The timing units (timer A to timer F) work either independently or coupled with the other timers including the master timer. Each timer unit contains the control for two outputs.

2.1 Period and compare programming (example for 300 MHz input clock)

The high-resolution programming is made completely transparent, to have the look-and-feel of a timer clocked by a 9.6 GHz clock (300 MHz x 32). 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:

$$\mathrm{PER} = \mathrm{T_{counting}}/\mathrm{T_{high-res}}$$

For instance, a 5 µs period is obtained, for f_{HRTIM} = 300 MHz, by setting the period register (HRTIM_MPER or HRTIM_PERxR, x = A to F) to 5 µs / 104 ps = 48000d (BB80h).

Note: If the result exceeds the 16-bit range, the high-resolution can be adjusted by multiples of 104 ps, so as to have the period value within the 16-bit range.

2.2 Set / reset crossbar

Each timing unit (timer A to timer F) holds the control of two outputs via a set / reset crossbar. Up to 32 events that can be selected among the event sources listed in the following table.

Table 4. Set/reset event sources

	Event source	Max n° of events
Timer_x unit (x=A..F)	• Period: PER • Compare: CMP1, CMP2, CMP3, CMP4 • Update: UPDATE	6
Master timer	• Period: MSTPER • Compare: MSTCMP1, MSTCMP2, MSTCMP3, MSTCMP4 • HRTIM synchronization: RESYN	6
Timer_y unit (y=A..F)	TIMEVNTx, x=1,..,9	9
External events	EXTEVNT x, x=1,..,10	10
Software trigger	SST	1

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.

2.3 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.

3 Single PWM generation

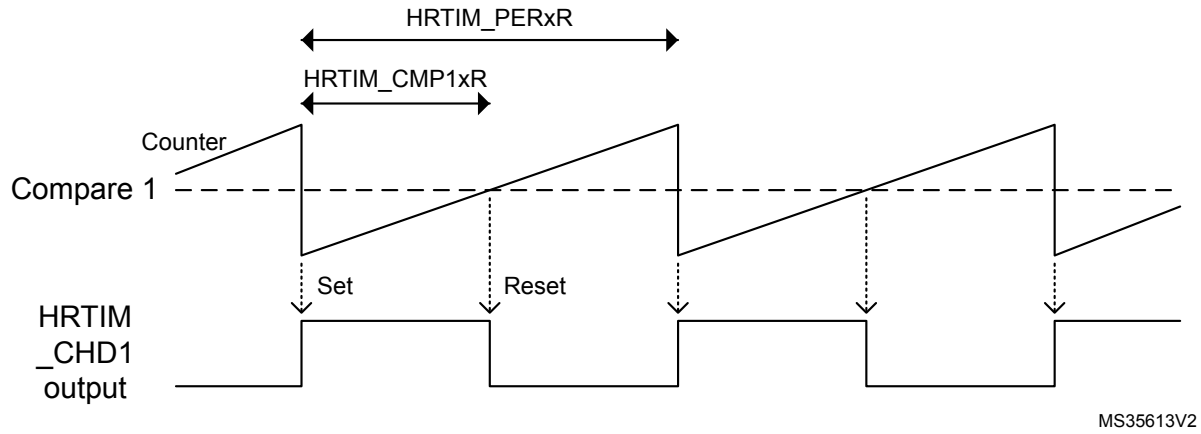
This session focuses on:

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

This example shows how very simply it is possible to achieve the PWM signals with the HRTIM by programming a limited number of HRTIM registers.

Let us consider a **200** kHz PWM signal with 50% duty cycle to be generated on output 1 of timer D, HRTIM_CHD1 output, as exemplified in the following figure.

Figure 1. Basic PWM generation



The timer D must be configured in continuous (free-running) mode. The PWM period is programmed in the period register HRTIM_PERDR using the formula:

PER = f_{HRCK} / f_{PWM} = (300 MHz x 32) / 200 kHz = 48000d (0xBB80).

The 50% duty cycle is obtained by multiplying the period by the duty cycle:

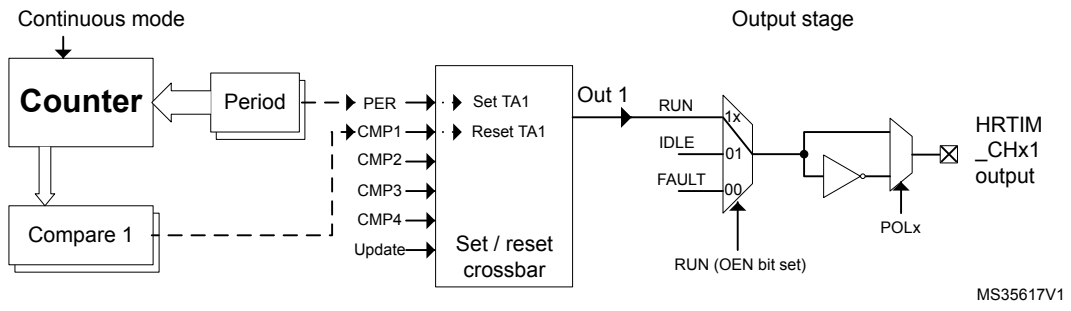
PER x DC = 0.5 x 48000d = 24000d (0x5DC0).

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 the following figure.

Figure 2. HRTIM configuration for generating basic PWM signals



Here the example code is provided:

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

```
/* Set period to 200kHz and duty cycle (CMP1) to 50% */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR =  0x0000BB80;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = 0x00005DC0;
```

```
/* 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 */
```

4 Generating multiple PWMs

This section focuses on:

- multiple timing unit usage
- register preload.

The HRTIM is able to generate up to 12 PWM signals with six independent frequencies (or seven frequencies when the master timer is used, see Section 5 Generating PWM with other timing units and the master timer).

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

The Timer D generates two phase-shifted **200 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.

The Timer A generates two phase-shifted 40kHz 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

The Timer A period is below the minimum frequency available at the highest resolution, clocked by a 9.6 GHz clock (300 MHz x 32), as shown in the table below.

Therefore, to get a 40 kHz switching frequency with a 300 MHz input clock, the frequency prescaler must be set to 4, clocking the HRTIM at 2.4 GHz clock (300 MHz x 8).

The frequency of 40 kHz is obtained by setting PER register to:

- PER = f_{HRCK} / f_{PWM} = (300 MHz * 8) / 40 kHz = 60000d (0xEA60).

Table 5. Timer resolution / minimum PWM frequency for f_{HRTIM} = 300 MHz

CKPSC[2:0]	Prescaling ratio	fHRCK equivalent frequency	Resolution	Minimum PWM frequency
000	1	300 x 32 MHz = 9.6 GHz	104 ps	146.6 kHz
001	2	300 x 16 MHz = 4.8 GHz	208 ps	73.26 kHz
010	4	300 x 8 MHz = 2.4 GHz	416 ps	36.63 kHz
011	8	300 x 4 MHz = 1.2 GHz	833 ps	18.31 kHz
100	16	300 x 2 MHz = 600 MHz	1.66 ns	9.16 kHz
101	32	300 MHz	3.33 ns	4.58 kHz
110	64	300 / 2 MHz = 150 MHz	6.66 ns	2.29 kHz
111	128	300 / 4 MHz = 75 MHz	13.33 ns	1.14 kHz

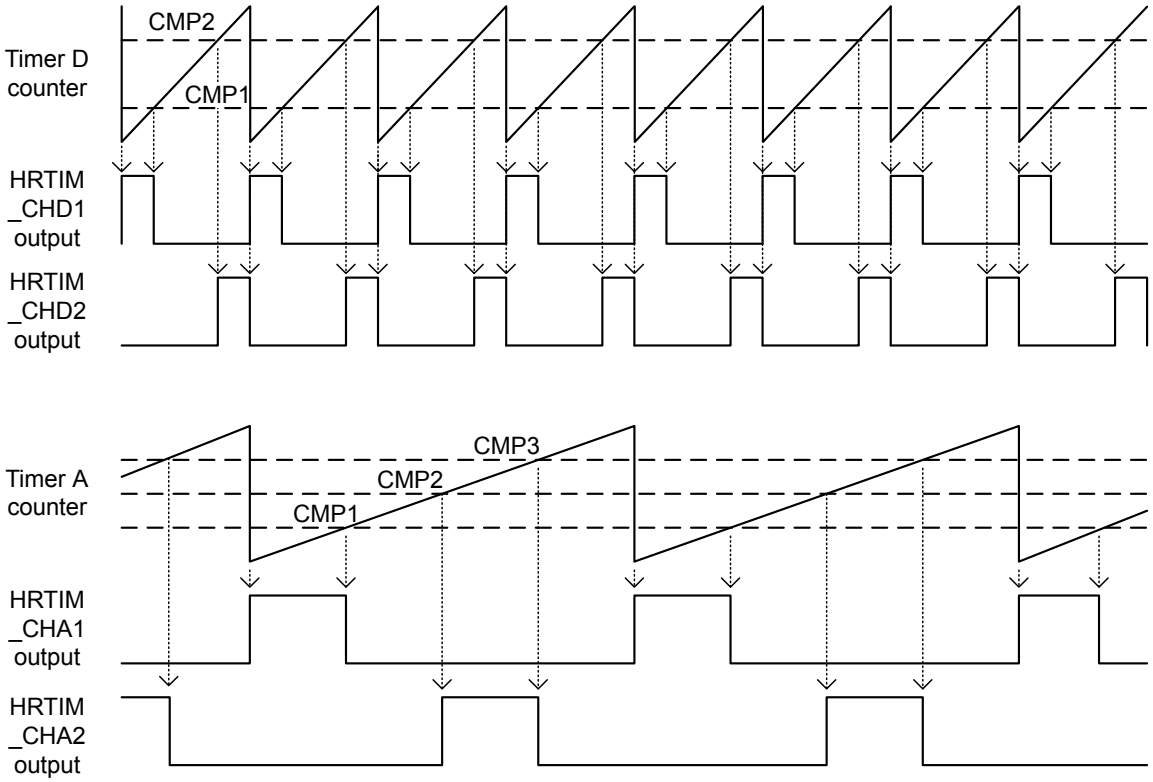
In this example the duty cycle is not updated.

However, a practical use case it is to access shadow registers by enabling the register preload mechanism (refer to device Reference Manual, for the whole list of HRTIM preloadable registers).

By enabling the feature by the PREEN bit in HRTIM_MCR and selecting the repetition event, this event will trigger the transfer of the preload registers at the beginning of each 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 since the first update event (causing compare registers to take their programmed value) occurs only after the first counting period is elapsed. To remove this delay, it is possible to force a register update by software (using TASWU and TDSWU bits in HRTIM control register 2) to have all active compare registers contents updated at once.

Figure 3. **Generation of multiple PWM signals**



MS35614V2

Here the example code is provided:

```
#define _200KHz_PERIOD 480000U
#define _40KHz_PERIOD 600000U
```

```
/*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 200kHz, CMP1 to 25% and CMP2 to 75% of period */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _200KHz_PERIOD;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _200KHz_PERIOD/4;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP2xR = (3*_200KHz_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_4;
```

```
/* Set period to 40kHz and duty cycles to 25% */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].PERxR =_40KHz_PERIOD;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP1xR = _40KHz_PERIOD/4;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP2xR = _40KHz_PERIOD/2;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_A].CMP3xR= (3*_40KHz_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;
```

5 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 seventh PWM independent frequency with the master timer, as in the example below.
- to work-around pin-out constraints (typically on small pin-count package), for instance using Timer F to generate waveforms even if HRTIM_CHF1 and HRTIM_CHF2 outputs of HRTIM2 are not available.

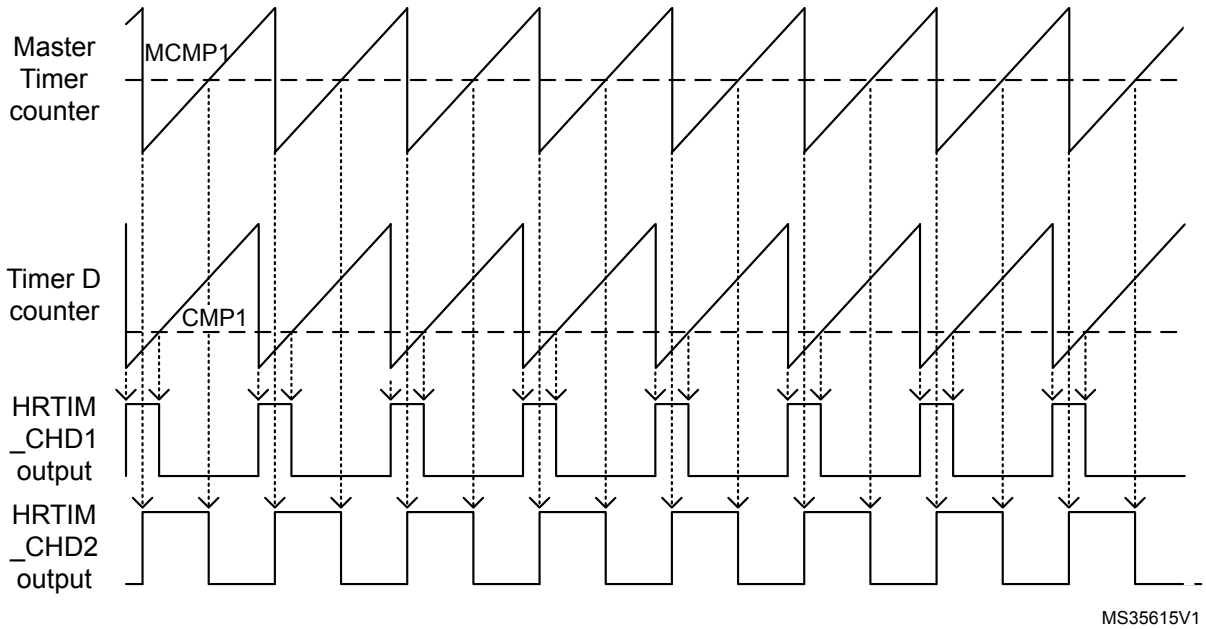
Note: It is mandatory to have the same prescaling factors for all timers sharing resources (for instance master timer and Timer A must have identical CKPSC[2:0] values if master timer is controlling HRTIM_CHx1 or HRTIM_CHx2 outputs of Timer x).

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 to demonstrate that the signals are completely asynchronous.

Figure 4. PWM generation with the master timer



The code here below is provided as example.

```
/*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 200kHz and duty cycle to 50% */
HRTIM1->sMasterRegs.MPER = _200KHz_Plus_PERIOD;
HRTIM1->sMasterRegs.MCMP1R = _200KHz_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 200kHz and duty cycle (CMP1) to 50%*/
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _200KHz_PERIOD;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _200KHz_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;
```

6 Arbitrary waveform generation

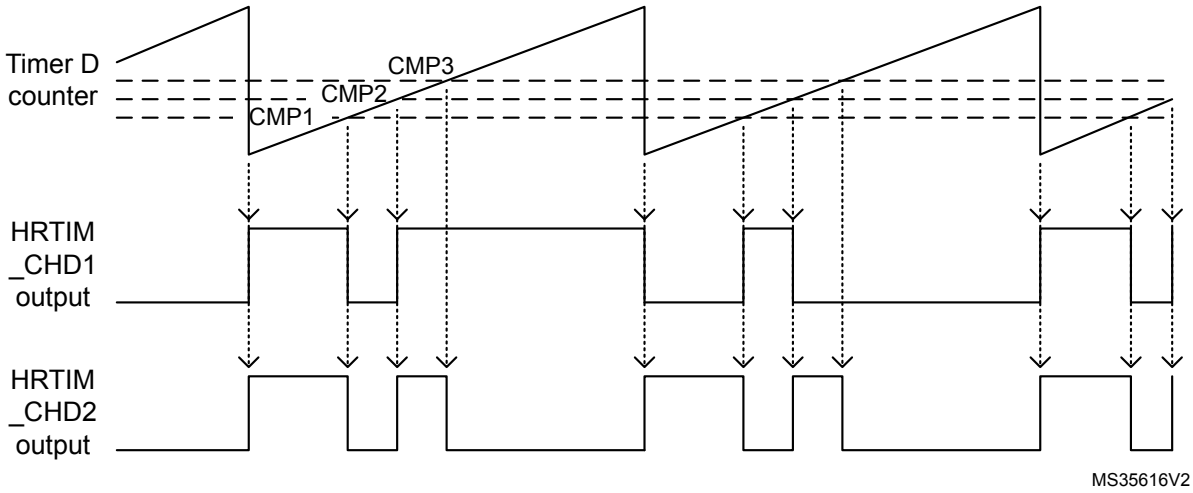
This section focuses on waveforms with multiple set/reset/toggle requests per period.

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 code here-below is provided as example.

```
/*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 200kHz and edge timings */
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].PERxR = _200KHz_PERIOD;
        HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP1xR = _200KHz_PERIOD/4;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP2xR =
      (3*_200KHz_PERIOD)/8;
HRTIM1->sTimerxRegs[HRTIM_TIMERINDEX_TIMER_D].CMP3xR = _200KHz_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;
```

Appendix A HRTIM main features

A.1 Features

This section highlights the main features and the usage of the HRTIM. For further details refer to the reference manual, Section Appendix B Reference documents.

A.2 Timing units (timer A,.., F)

Each HRTIM module has 12 outputs, to cover interleaved topologies, such as:

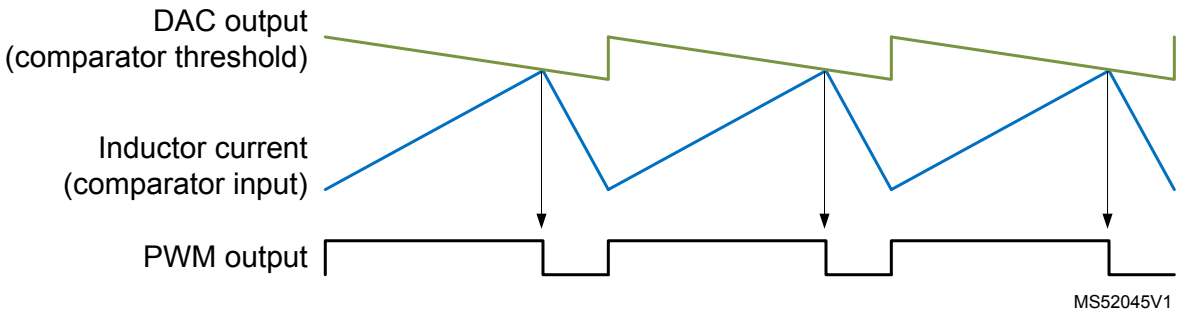
- triple-interleaved half-bridge LLC (3 x 4 outputs required, two at primary for half-bridge control and two at secondary for synchronous rectification),
- dual interleaved full-bridge LLC (2 x 6 outputs) and
- dual phase-shifted full bridge converters (2 x 6 outputs).

A.3 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 Arrow Electronics in Section Appendix B Reference documents, 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 6. Peak current-mode control with slope compensation



A.4 External event counter

The HRTIM 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.5 Fault features

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

A.6 Push-pull

The push-pull mode primarily aims at driving converters using push-pull topologies.

The push-pull mode with dead time insertion and in single-shot operating mode is possible.

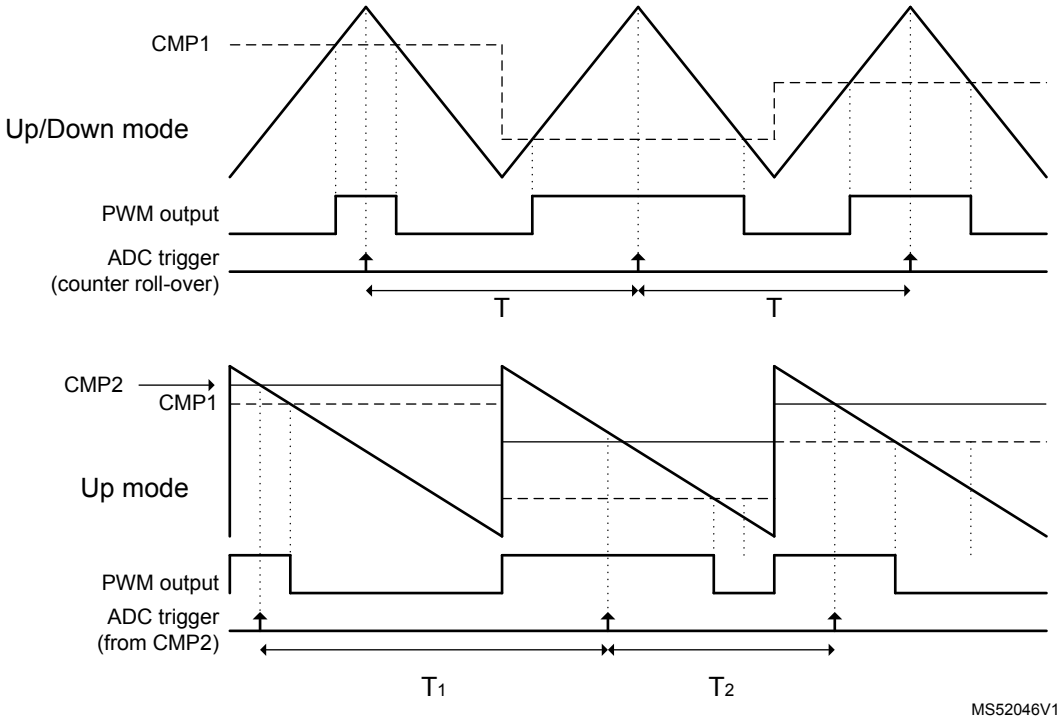
A.7 Classical PWM mode

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. By default, the preload mechanism is disabled.

A.8 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 the following figure, a constant sampling frequency and sampling at the middle of the pulse are needed.

Figure 7. Comparison of up-only and up-down modes



A.9 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.10 Null duty cycle mode

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

A.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 Reference documents

Document name	Title
DS13808	SR5 E1 line of Stellar electrification MCUs — 32-bit Arm® Cortex®-M7 automotive MCU 2x cores, 300 MHz, 2 MB Flash, rich analog, 104 ps 24 ch high-resolution timer, HSM, ASIL-D
RM0483	SR5E1x 32-bit Arm® Cortex®-M7 architecture microcontroller for electrical vehicle applications

Revision history

Table 6. **Document revision history**

Date	Revision	Changes
10-Jan-2023	1	Initial release.
02-Feb-2023	2	Confidentiality level changed from ST Restricted to ST Public.

Contents

List of tables

List of figures