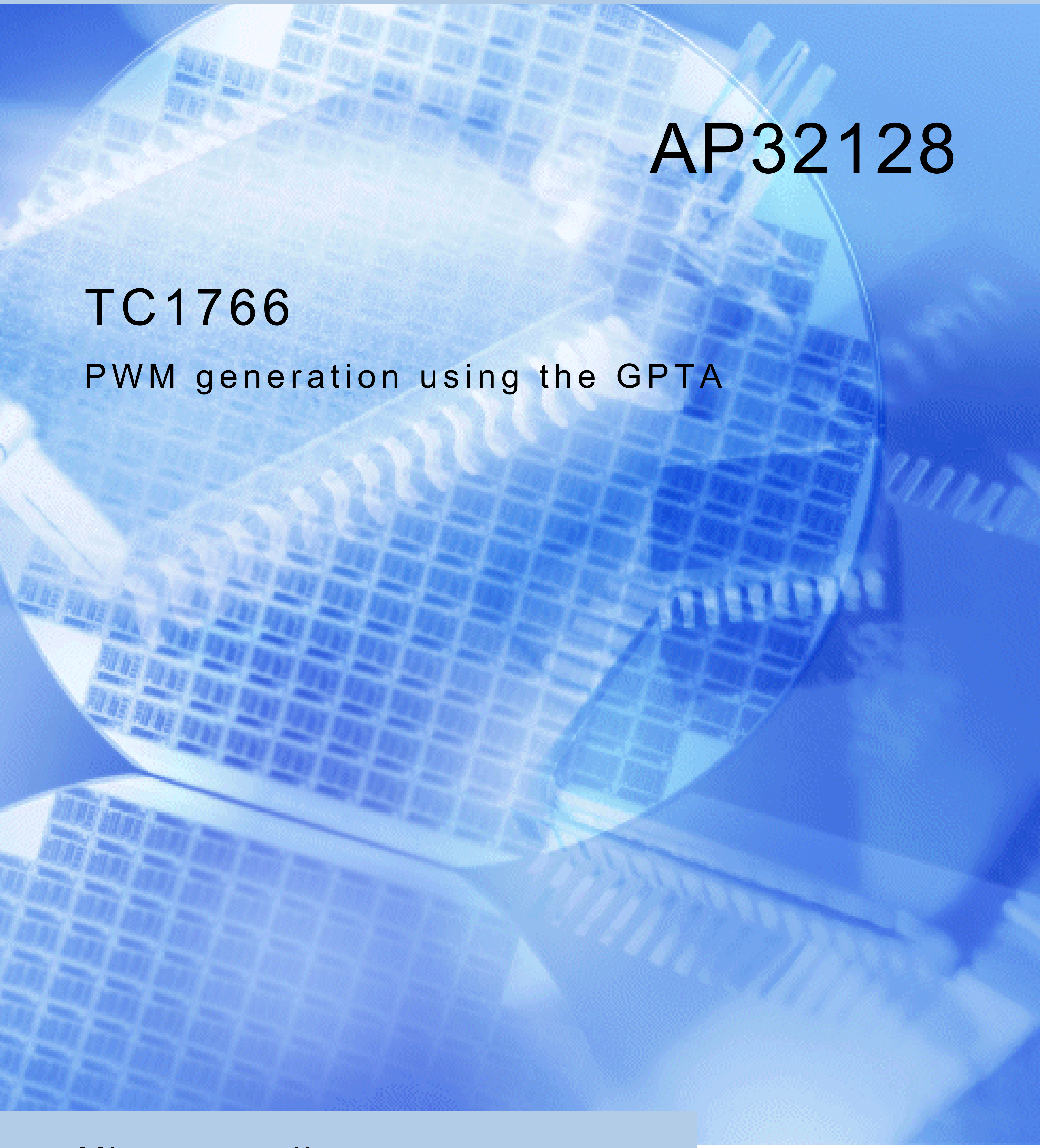
Application Note, V1.0, August 2008

AP32128

TC1766

PWM generation using the GPTA



Microcontrollers

Never stop thinking

Edition 2008-11-17

Published by
Infineon Technologies AG
81726 München, Germany

AP32128		
Revision History:	2008-08	V1.0
Previous Version:	none	
Author:	Alann Denais	

Page	Subjects (major changes since last revision)

We Listen to Your Comments
Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Table of Contents **Page**

1 Scope

Dear Reader,

Thanks for using this PWM generation appnote! This appnote will help you to implement simple to complex PWM on your TC1766 target microcontroller[1].

This appnote aims to give an example on how to use and configure the GPTA's LTCs of the TC1766 microcontroller in order to generate PWMs.

The following PWM function will be implemented:

- PWM 1: Edge aligned basic PWM channels
- PWM 2: Edge aligned coherent update PWM channels
- PWM 3: Center aligned 3*2 channels coherent update PWM (multi-channel PWM) for Inverter application
- Interrupt generation

Have fun with Infineon's PWM generation!

1.1 Micro controller resources used

- GPTA
    - Clock distribution unit
    - LTC (Free running timer, Compare)
    - Input / output Multiplexer
- Port
    - Alternate function

[1] This appnote do not aim to list all possible configuration suitable for the PWM generation using the GPTA

2 Setup

2.1 Needed material

The following HW and SW material is required to run the application example:

- A PC with local administration rights.
- A functional TC1766 TriBoard with 15Mhz quartz (incl. cables, power supply).
- A debugger supporting TC1766.
- An oscilloscope.
- The PWM source code and executable (included in this package).
- Optional: Tasking VX-toolset for TriCore, v2.5.

It is strongly recommended to have the Tasking toolchain installed and running on the PC where the PWM software is running, so that the PMW's default parameters can be changed (recompilation needed). For more information about the toolchain, please visit www.tasking.com .

The PWM software delivered with this application note is made of:

- Source code
- Executable files (.elf and .hex)
- Tasking projects file for the toolchain v2.5.

2.2 Installation steps

It is assumed that the user knows how to operate the different tools mentioned above (Compiler, Debugger, TriBoard, etc.). The following steps may be followed to install operate the PWM application:

- Install all the needed SW tools (Tasking toolchain, debugger).
- Connect a TriBoard to the debugger.
- Connect an scope probe to the required PWM outputs[2]
    - PWM 1: pins GPTA8 and GPTA9
    - PWM 2: pins GPTA6 and GPTA7
    - PWM 3: pins GPTA14, GPTA15, GPTA16, GPTA17, GPTA24, GPTA25 and GPTA3 for the center signal
- Build the project.
- Start a debugger session.
- Run the program, and look for the global variable `System`. The structure members `System.Pwm_1.DutyCycle[0..1]`, `System.Pwm_2.DutyCycle[0..1]`, `System.Pwm_3.DutyCycle[0..3]` hold the current PWM duty cycles. Manual setting of the duty cycle can be done by setting to TRUE / 1 the variable `System.Pwm_x.ManualDutyCycle` with x = [1,3] and writing the required duty cycle in the `System.Pwm_x.DutyCycle variables.`

[2] A simple RC filter may convert the PWM pulses to the averaged signal (sine waves)

3 Introduction to the PWM

When one hear from the GPTA, the 1st think that come to the mind is “*Oh no! Such a complex module! I’ll never get anything out of it without a lot of work*”. That’s what this appnote will try to remove from everyone’s mind.

3.1 Object approach (Hardware)

The GPTA should be seen as a module made of very flexible objects like timers, capture, compare modules and other objects to suits nearly every application.

Let’s have a look from the application point of view, the requirements could be:

- “I need a centered aligned PWM output with a constant period and a variable duty cycle.”

If we look closer at the requirement and match it with the GPTA objects, the following objects will be required:

- A timer using a defined input clock ➔ We’ll use a **reset timer** with one of the GPTA available module clocks (GPTA input multiplexer)
- A period value ➔ We’ll use one **compare** cell to reset the timer (Period match)
- An output signal having a variable duty cycle ➔ We’ll use **2 compare cells**, one for the leading and one for the trailing edge
- A connection from the PWM output of the GPTA to the microcontroller output pin ➔ we’ll use the GPTA output multiplexer and the port alternate functions (signal routing).

The Figure 1 shows a 4 cells PWM configuration. The 1st cell is configured as a reset timer, and counts up. When the 2nd cell, configured as a compare cell, matches the timer value, the timer is reset to 0xFFFF and continues counting up. The 3rd and 4th cells produce the PWM output signal: the PWM leading edge (rising edge) and the PWM trailing edge (falling edge). They are configured as compare cells that change the signal output when the compare cell value match the timer value. Changing the compare value will change the PWM duty cycle. Note that the compare values are symmetrical to the middle of the period in order to produce a centered aligned PWM.

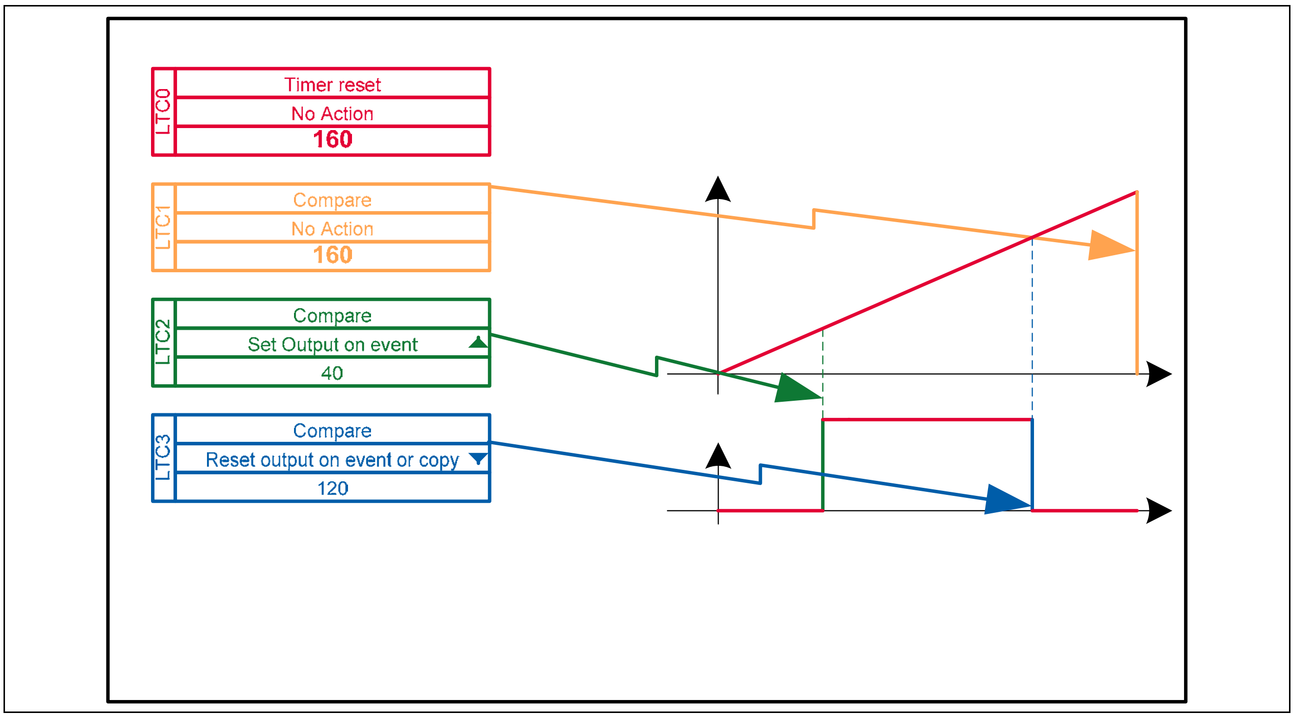


Figure 1 Basic PWM

3.2 Layered approach and hardware abstraction

The GPTA hardware abstraction can be done using 3 layers:

- The Low level driver layer
- The Driver layer
- The application layer

The Figure 2 shows a use case for creating and using the PWM objects described below. Every PWM generations are no more complex than such a modular assembly of basic objects: an input clock, a timer, some compare cell for toggling the output pin having a defined PWM period and duty cycle value. This application notes will describe and explain the different objects that can be implemented on the target microcontroller. Note that the source code of the object described is available as a concrete example.

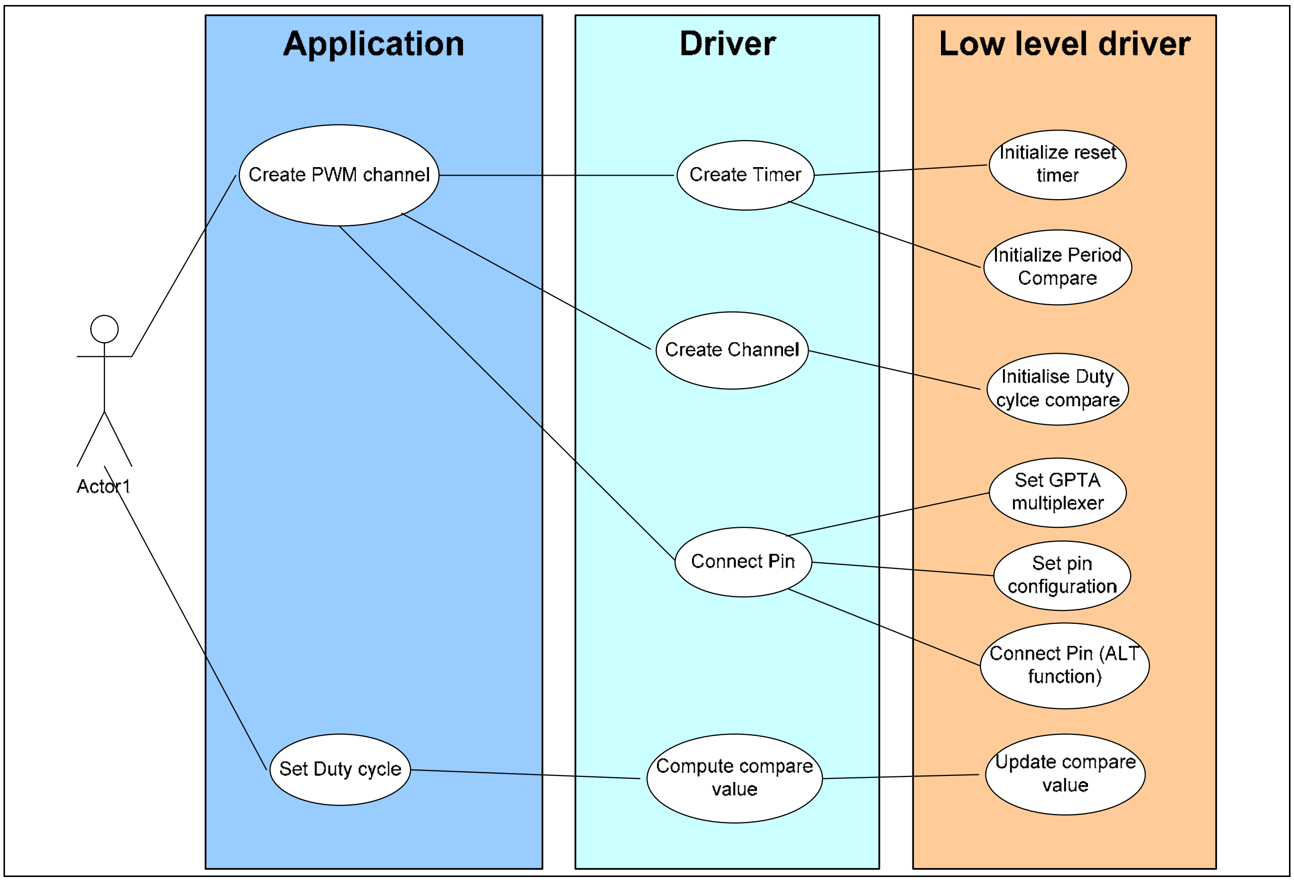


Figure 2 PWM use case

4 Low level driver: The basic GPTA objects

This appnote names the "basic GPTA objects", functionalities that are achieved using the simple GPTA elements like LTC, GTC, GT, FPC, DCM, PDL, digital PLL, …

For the PWM generation of this application note, only the *LTC Timer reset* and *LTC Compare* objects are used.

4.1 The GPTA LTC modes

The GPTA LTC is a flexible module that can operate in the following 4 modes:

Mode	**Description**	**Trigger Event**	**Action on event**
Free running timer	The timer is incremented by the selected input clock[3], and overflow as the value 0xFFFF is reached.	The timer reaches the overflow value 0xFFFF	• Raise interrupt[4] • Modify the LTC output[4,5] • Transfer the action request to the next LTC
Reset timer	The timer is incremented by the selected input clock[3], and is reset to 0xFFFF by an event from the next LTC.	The timer reaches the overflow value 0xFFFF	• Raise interrupt[4] • Modify the LTC output[4,5] • Transfer the action request to the next LTC • Toggle the select line SO[4]
Capture	The reference timer value is copied to the LTC value register on capture event.	The capture event occurs	• Save the reference timer value • Raise interrupt[4] • Modify the LTC output[4,5] • Transfer the action request to the next LTC • Reset the previous LTC if configured as Reset timer
Compare	Compare with the reference timer. If enabled by the select line SI.	The reference timer matches the compare value	• Raise interrupt[4] • Modify the LTC output[4,5] • Transfer the action request to the next LTC • Reset the previous LTC if configured as Reset timer

For more details on each mode, see the TC1766 User Manual.

4.2 Cascading the LTCs

Cascading cells consist in grouping LTCs in order to produce complex output signals. Applying the cell cascading method to the PWM generation leads to the following configuration:

A timer reset cell followed by m compare cells. Any number of adjacent cells can be cascaded[6], each cell having the ability to change the state of the output signal by an action like set, reset or toggle. Furthermore each cell is able to raise an interrupt on event and / or forward the output signal to a port pin.

[3] Level or edge sensitive. In level sensitive mode, the prescaler clock from the CDU can be used to reduce the timer frequency

[4] Optional

[5] Set, reset, toggle, unchanged, copy from previous cell

4.2.1 The reset timer cell

The reset timer cell generates the timer value which is incremented on each input clock according to the selected sensitivity. The timer value is propagated to the compare cell through the timer bus.

The reset timer cell has also the ability to toggle the timer Select Output line (SO line) in order to enable / disable selected compare cells. This is detailed later in the chapter 4.3 Coherent update (Shadow register) mode.

4.2.2 The cells actions & output signal

Each cell including the reset timer cell has the possibility to modify the output signal by an action on event: set, reset or toggle.

Whereas the *cascaded cell group* includes the reset timer with all the compare cells linked to it (see Figure 3); the *signal cascaded cell group* includes only the cells that are able to modify a given output signal by an action[7].

For the PWM application, the 1st cell of a *cascaded signal cell group* will either set or reset the output signal, whereas the next cells of the group will **propagate** or modify the output signal. An action done by a cell N is always overwritten by an action done by the cell N+1 in case the actions occur simultaneously. The last cell of the *cascaded signal group* is used to output the PWM signal to a port pin.

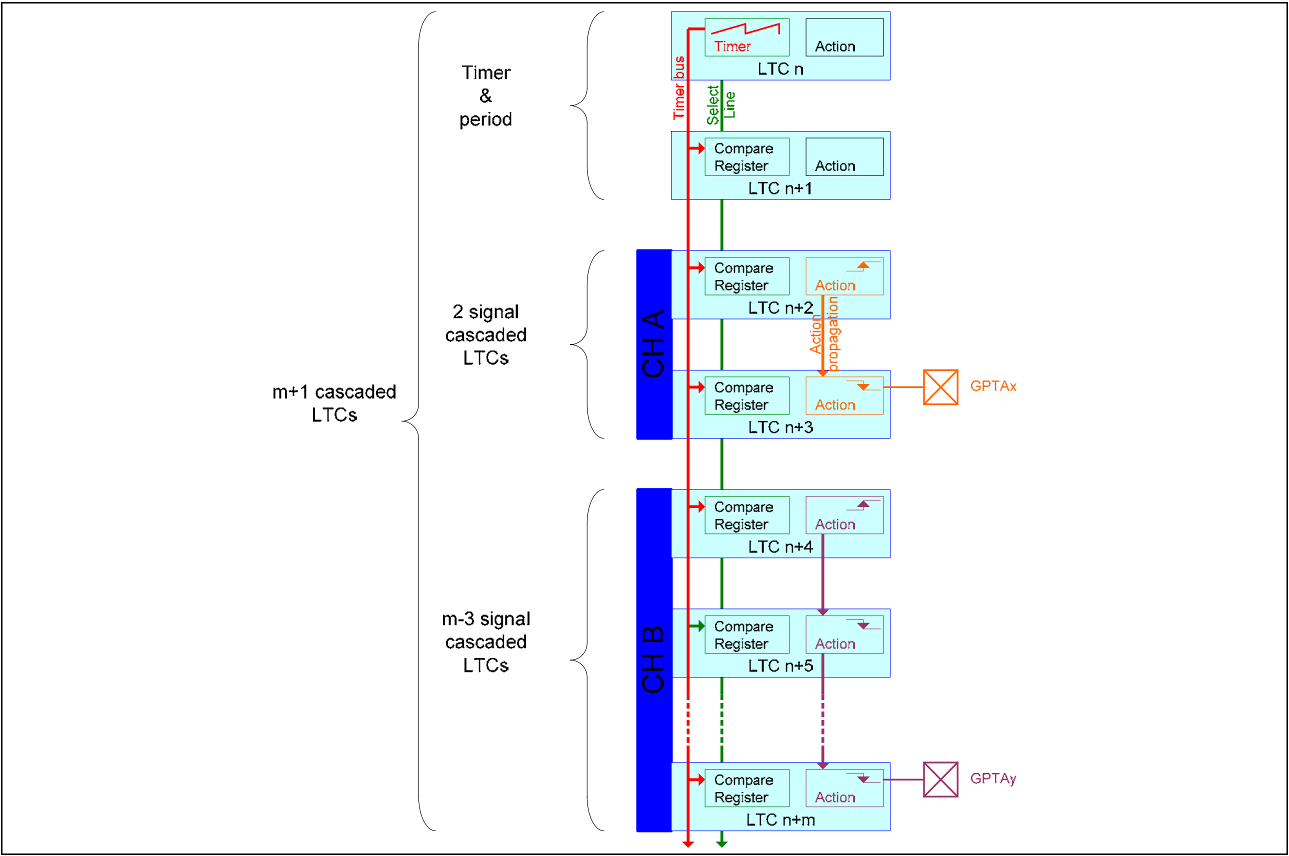


Figure 3 LTC cascading

The Figure 3 shows m+1 cascaded cells referring to the same reset timer cell n. The cell n+1 is a compare cell which reset the previous timer cell n on period match. The cell n+2 and n+3 is a *signal cascaded cell group* made of 2 cells where the action of the cell n+2 is only propagated to the cell n+3. The output of the

[6] On the TC1766, the number of cascaded cell is limited according to ratio f_{SYS}/f_{GPTA}. With f_{SYS}=80MHz, for 1≤ f_{SYS}/f_{GPTA} <2: max 20 LTCs; for 2≤ f_{SYS}/f_{GPTA} <3: max 40 LTCs; for 3≤ f_{SYS}/f_{GPTA}: No limits

[7] In case the reset timer modifies the output signal, the reset timer is also included in the signal cascaded cell group.

cell n+3 is routed to GPTAx pin. The group of cells n+4 to n+m is an other semi-independent more complex *signal cascaded cell group* made of m-3 cell and using the GPTAy output. The PWM is semi-independent because the period is the same for all the channels but the duty cycle is totally independent.

The Figure 4 shows the produced output using the configuration described in the Figure 3, for the LTC n to n+3 only. The period is set to 160, the leading edge to 40 and the trailing edge to 120 which produce a center aligned PWM with a duty cycle of 50%.

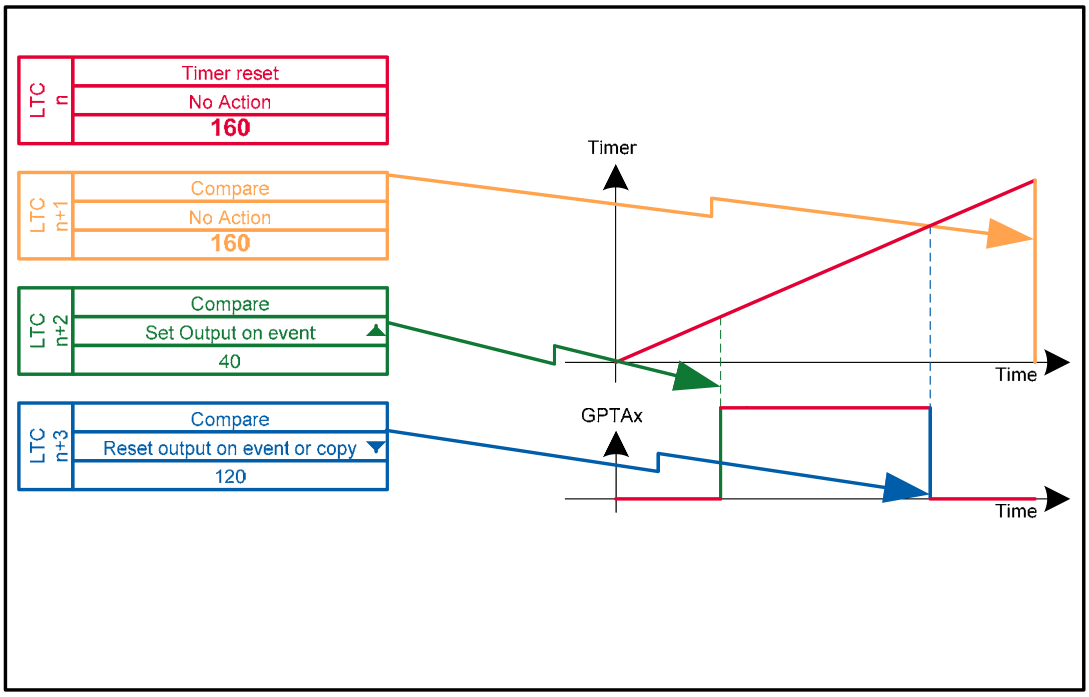


Figure 4 LTC cascading output example

4.3 Coherent update (Shadow register)

The coherent update or shadow register method consist of using two LTC compare objects for a single signal edges. Either the 1st or the 2nd LTC compare object is active at a time. As the timer is reset, the inactive cell is made active and the active one is made inactive simultaneously, by toggling the timer Select Output line[8] (SO line). This method ensures a coherent and synchronous update of all PWM edges and/or channels in case of multi-channels PWM.

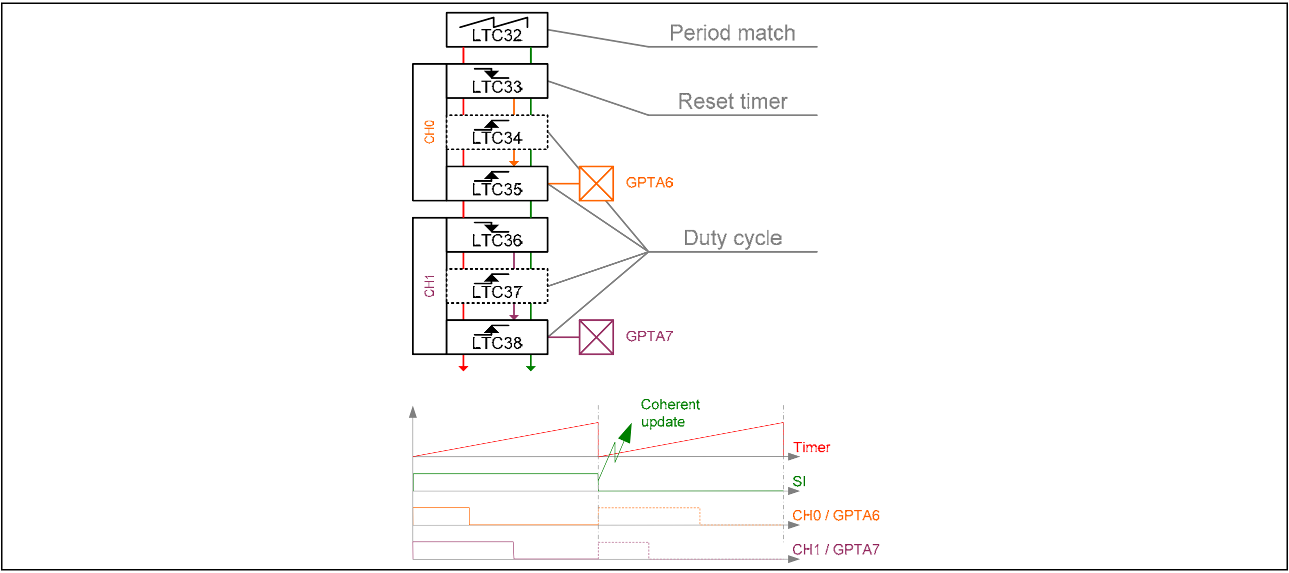


Figure 5 Single edge coherent update

The Figure 5 shows an example of an edge aligned PWM configured as follow:

- 1st cell: Reset timer cell.
- 2nd cell: Period match compare cell that resets the timer and the output signal training edge on event.
- 3rd and 4th cells: Coherent update for the leading edge (rising edge). Both cells are configured as compare cells but, the 3rd LTC is enabled on SI High and the 4 LTC is enabled on SI low. A software request[9] will toggle the SO line of the reset timer on the next timer event.
- The 5th to 7th cells are configured as the 2nd to 4th cells except that the 5th cell does not have any effect on the timer period.

4.4 Connecting the input clock

The timer input clock is connected using the GPTA multiplexer. See the GPTA user manual for more details.

4.5 Connecting to output pin

The LTC output is connected to the port pin using the GPTA multiplexer and the port alternate function. See the GPTA user manual for more details.

[8] The reset timer Select Output line (SO line) is connected to the next compare cells Select Input line (SI line)

[9] Setting the bits LTCCTR.CUD and LTCCTR.CUDCLR of the timer simultaneously will request the toggling of the SO line on the next timer event.

4.6 GPTA and port initialization sequence

At reset the microcontroller GPTA ports are 3-stated; the output signal level is at that time defined by the external circuit as for example with pull up or pull down resistors. When programming the multiplexer, the GPTA outputs are disconnected internally from the LTCs and set to a default low level. Therefore glitches may appear on the GPTA outputs while the GPTA multiplexer is being programmed. In order to avoid such glitch the port alternate function must be set last as in the following initialization sequence example:

- Enable and set the GPTA module clock & enable the GPTA module
- Setup the LTC, LTC interrupts, and LTC default output state
- Program the GPTA I/O multiplexer
- In case the emergency stop functionality is used, set the port output default state and enable the emergency outputs
- Set the port pad driver mode
- Set the port alternate function to connect the GPTA output to the uC ports (port behavior)

4.7 API

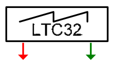
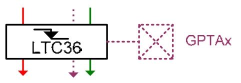
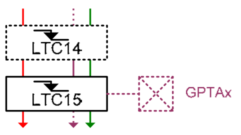
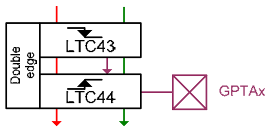
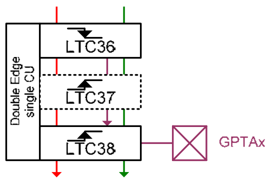
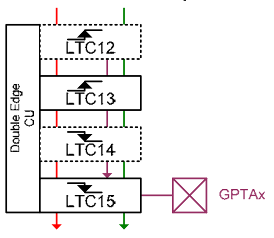
Mode	**API**	**Brief description**
Reset timer	`GPTA_LTC_CellTimerReset()`	Configure a reset timer cell
Compare	`GPTA_LTC_CellCompare()`	Configure a compare cell
Multiplexer input	`GPTA_LTC_ConnectCLK()`	Route an input signal to an LTC
Multiplexer output	`GPTA_LTC_ConnectPin()`	Route the LTC output signal to a pin
Multiplexer programming	`GPTA_InitMultiplexerArrays()`	Initialize the GPTA multiplexer
	`GPTA_ProgramMultiplexerArrays()`	Program the GPTA multiplexer
GPTA initialization	`GPTA_InitializeClock()`	Initialize the GPTA clock
	`GPTA0_ClockEnable()`	Enable the GPTA clock

For more details on each function, see the software documentation.

5 Driver: The Basic PWM objects

The PWM modular approach requires the definition of the basic PWM objects. Those objects will be put together to build complete single to multi-channel PWM object. The basic PWM objects are the PWM timer, the PWM channel single edge, and the PWM channel double edge.

5.1 The PWM objects

Object	Property	Action on event	Resources	
Timer	• Counter • Input clock frequency	• The internal counter is reset • Interrupt generation[10]	1 LTC as reset timer 	
Single Edge	• Edge value	• Interrupt generation[10] • Output signal modification	1 LTC as compare 	non coherent
			2 LTCs as compare 	Coherent update
Double Edge	• Leading edge value • Trailing edge value	• Interrupt generation[10] • Output signal modification	2 LTC as compare 	non coherent update
			3 LTC as compare 	Half coherent upadte
			4 LTCs as compare 	Coherent update

Note that the complete timer object (timer + period match) can be built either independently from the next channel as on the Figure 4 where there is a dedicated compare cell for the period match or part of the next

[10] Optional

channel as on the Figure 5 where the period compare cell is also used in the generation of the PWM output signal.

5.2 Coherent update vs. non coherent update

	Coherent update	**Non coherent update**
Resource used	☹ Costly in resources (2 LTC per edge)	☺ Less resources used (1 LTC per edge)
Small duty cycle	☺ No restriction, 0% duty cycle possible	☹ Limited by the compare value update latency
Big duty cycle	☺ No restriction, 100% duty cycle possible	☹ Limited by the compare value update latency
Correct pulses	☺ Proper pulses guarantied, no glitches	☹ Glitches may appear on compare value update. Depends on the update method used
Multi-channel	☺ Multi-channel update is synchronous	☹ Channel update might not be synchronous. Depends on the update method used
Update timing	☺ The compare value update can be done at any time. The new value is taken in account for the next period (SO line)	☹ The update time has an influence on the generated pulse. Unexpected duty cycle might be generated in case the update timing is not respected.
Jitter of the output signal	☺ No Jitter	☹ Jitter may appear according to the update method used
Period	☺ Period is constant	☹ Period may vary according to the update method used

5.3 API

Mode	API[11]	Brief description
Timer	`Pwm_Timer_Init()`	Initialize a timer object
	`Pwm_Timer_Update()`	Transfer the shadow value (Toggle SO line)
Single edge	`Pwm_Channel1Edges_Init()`	Initialize a PWM channel with a configurable single edge
	`Pwm_Channel1Edges_Update()`	Update the PWM channel single edges
Single edge (Coherent update)	`Pwm_Cu_Channel1Edge_Init()`	Initialize a PWM channel with a configurable single edge
	`Pwm_Cu_Channel1Edge_Update()`	Update the PWM channel single edges
Double edge (Coherent update)	`Pwm_Cu_Channel2Edges_Init()`	Initialize a PWM channel with configurable leading and trailing edges
	`Pwm_Cu_Channel2Edges_Update()`	Update the PWM channel leading and trailing edges

For more details on each function, see the software documentation.

[11] _Cu_ refers to Coherent Update API

6 Application examples: The PWM objects

6.1 General resource assignment check list

The configuration of the PWM hardware is quite flexible but requires a dedicated attention during the resource assignment phase. One should care of the following:

- Check for the possible connection between the LTC and the output pin
- Check that the number of the cascaded cell do not exceed the maximum allowed
- Check the routing of the trigger signal to the output ports, ADC, DMA, … as required
- Check the LTC interrupt source register in case interrupt are needed. It is better if the service request control register is not shared between different interrupt sources (faster handling because there is no need to check the interrupt source).

6.2 PWM 1: Edge aligned basic PWM channels

6.2.1 Features

This example implements two edge aligned basic PWM channels (no coherent update). For this a PWM timer object is used (LTC40), followed by the 1st channel and the 2nd channel.

The 1st channel is made of two single edge PWM object (LTC41 & LTC42) where LTC 41 is also used for the timer period match. The duty cycle update is done by writing the new compare value directly to the active compare cells LTC42. The LTC41 compare value is never modified. Therefore the duty cycle of 100% is not possible for this channel.

The 2nd channel is made of two single edge PWM object (LTC43 & LTC44). The duty cycle update is done by writing the new compare value directly to the active compare cells LTC44. The LTC43 compare value is only updated in case the duty cycle is set to 100%. In this case the compare value for LTC43 is set out of the timer range so that the compare event never occurs and the output remains set.

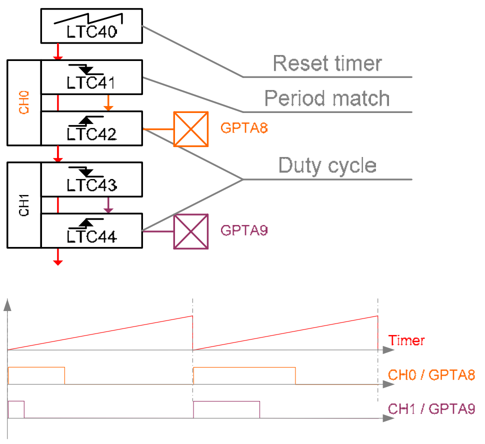


Figure 6 Basic PWM channels

6.2.2 Configuration summary

Resource	Configuration	Info
GPTA8	• Direction : output • Behavior : Push-pull ALT1 • Default : Low	CH0
GPTA9	• Direction : output • Behavior : Push-pull ALT1 • Default : Low	CH1

	Mode	Input	Input sensitivity	Initial Value	Default SO line	Action on event	Output	Interrupt
LTC40	Reset timer	GPTA clock bus 0	Level sensitive (high level)	0x0000	low	Hold	No	Yes
	Mode	**Input**	**Input sensitivity**	**Initial Value**	**SI enable state**	**Action on event**	**Output**	**Interrupt**
LTC41	compare	-	-	Period-2	Always	Reset	No	No
LTC42	compare	-	-	0xFFFE	Always	Copy or Set	No	No
LTC43	compare	-	-	Period-2	Always	Reset	No	No
LTC44	compare	-	-	0xFFFE	Always	Copy or Set	No	No

6.2.3 Duty cycle formula

Find below the pseudo-code used for the leading and trailing edge computation:

```
tTrailingEdge = Period-2;
if (Ton == 0)                // 0% => 2nd compare never occurs (1st edge)
       tLeadingEdge = 0xFFFE;
else if (Ton >= Period)      // 100% => 2nd compare (1st edge) overwrite the 1st compare (2nd edge)
{
       if (Channel->FirstCell != Channel->Timer->Cell+1)
       {                     // The channel is allowed for 100%
              tLeadingEdge = Period - 2;
              tTrailingEdge = Period-1;
       }
       else
       {                     // The channel is not allowed for 100%
              tLeadingEdge = 0xFFFF;
       }
}
else                         // x% =>  -1 < compare value <= Period-3
       tLeadingEdge = Period - Ton-2;
```

Where:

- `Period` is the period in ticks `Period` = F_{Timer} / f_{PWM} with f_{Timer} the reset timer input frequency ($f_{Timer} = f_{GPTA}$)
- `Ton` is the “ON” time of the PWM in ticks with `Ton` = DutyCycle * `Period`

6.2.4 API

Mode	API	Brief description
Reset timer Initialization	`Pwm_Timer_Init()`	Initialize the reset timer
Channel initialization	`Pwm_EdgeAligned_Init()`	Initialize a PWM channel
PWM update	`Pwm_EdgeAligned_Update()`	Update the PWM duty cycle
	`Pwm_EdgeAligned_Off()`	Set the PWM off

For more details on each function, see the software documentation.

6.3 PWM 2: Edge aligned coherent update PWM channels

6.3.1 Features

This example implements two edge aligned coherent update PWM channels. For this a PWM timer object is used (LTC32), followed by the 1st channel and the 2nd channel.

The 1st channel is made of a single edge PWM object (LTC33) and a single edge coherent update PWM object (LTC34 & LTC35) where LTC 33 is also used for the timer period match. The duty cycle update is done by writing the new compare value to the shadow compare cells LTC34 or LTC35 whichever is inactive. The LTC33 compare value is never modified. Therefore the duty cycle of 100% is not possible for this channel.

The 2nd channel is made of a single edge PWM object (LTC36) and a single edge coherent update PWM object (LTC37 & LTC38). The duty cycle update is done by writing the new compare value to the shadow compare cells LTC37 or LTC38 whichever is inactive. The LTC36 compare value is only updated in case the duty cycle is set to 100%. In this case the compare value for LTC36 is set out of the timer range so that the compare event never occurs and the output remains set.

In order to reflect the update of the shadow registers on the PWM outputs the SO line of the timer must be toggled by a software request. Note that once the request has been done, no further modification to the compare values should be made before the SO line has been toggles. In case this is not respected, glitches or incorrect pulses might be produced on the output.

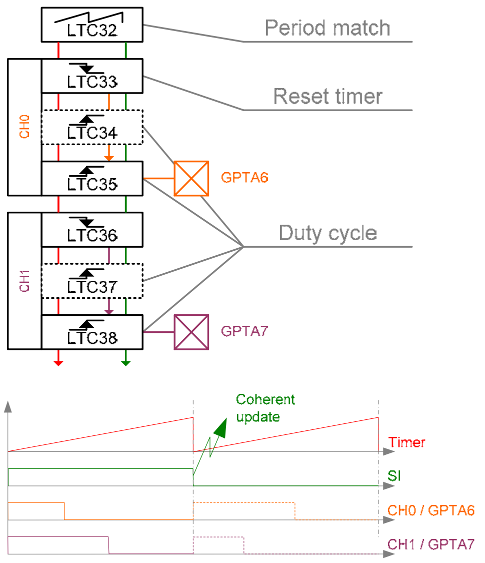


Figure 7 Coherent update PWM channels

6.3.2 Configuration summary

Resource	Configuration		Info
GPTA6	• Direction • Behavior • Default	: output : Push-pull ALT1 : Low	CH0
GPTA7	• Direction • Behavior • Default	: output : Push-pull ALT1 : Low	CH1

	Mode	Input	Input sensitivity	Initial Value	Default SO line	Action on event	Output	Interrupt
LTC32	Reset timer	GPTA clock bus 0	Level sensitive (high level)	0x0000	low	Hold	No	Yes
	Mode	**Input**	**Input sensitivity**	**Initial Value**	**SI enable state**	**Action on event**	**Output**	**Interrupt**
LTC33	compare	-	-	Period-2	Always	Reset	No	No
LTC34	compare	-	-	0xFFFE	High	Copy or Set	No	No
LTC35	compare	-	-	0xFFFE	Low	Copy or Set	GPTA6	No
LTC36	compare	-	-	0xFFFE	Always	Reset	No	No
LTC37	compare	-	-	0xFFFE	High	Copy or Set	No	No
LTC38	compare	-	-	0xFFFE	Low	Copy or Set	GPTA7	No

6.3.3 Duty cycle formula

Find below the pseudo-code used for the leading and trailing edge computation:

```
tTrailingEdge = Period-2;
if (Ton == 0)               // 0% => 2nd compare never occurs (1st edge)
        tLeadingEdge = 0xFFFE;
else if (Ton >= Period)     // 100% => 2nd compare (1st edge) overwrite the 1st compare (2nd edge)
{
        if (Channel->FirstCell != Channel->Timer->Cell+1)
        {                   // The channel is allowed for 100%
                tLeadingEdge = Period - 2;
                tTrailingEdge = Period-1;
        }
        else
        {                   // The channel is not allowed for 100%
                tLeadingEdge = 0xFFFF;
        }
}
else                        // x% =>  -1 < compare value <= Period-3
        tLeadingEdge = Period - Ton-2;
```

Where:

- `Period` is the period in ticks `Period` = F_{Timer} / f_{PWM} with f_{Timer} the reset timer input frequency ($f_{Timer} = f_{GPTA}$)
- `Ton` is the “ON” time of the PWM in ticks with `Ton` = DutyCycle * `Period`

6.3.4 API

Mode	API	Brief description
Reset timer Initialization	`Pwm_Timer_Init()`	Initialize the reset timer
Channel initialization	`Pwm_Cu_EdgeAligned_Init()`	Initialize a PWM channel
PWM update	`Pwm_Cu_EdgeAligned_Update()`	Update the PWM duty cycle
	`Pwm_Cu_EdgeAligned_Off()`	Set the PWM off
Toggle the SO line	`Pwm_Timer_Update()`	Toggle the SO line after PWM update

For more details on each function, see the software documentation.

6.4 PWM 3: Center aligned 3*2 channels coherent update PWM (multi-channel PWM)

6.4.1 Features

This example implements 3 center aligned coherent update PWM channels with top and bottom PWM as used in inverter application. The PWM has the following features:

- 2*3 PWM channels
- Dead time generation
- Variable duty cycle 0-100%, coherent update for all 3 channels
- Min pulse cancellation
- Variable period, coherent update

For this a 29 LTC are used. The PWM object is made of:

- A PWM timer object (LTC3)
- A double edge coherent update PWM object for the period and period center signals (LTC4 to LTC7)
- Three double edge coherent update PWM objects for the 3 top PWM channels (LTC8 to LTC11, LTC16 to LTC19 and LTC24 to LTC27)
- Three double edge coherent update PWM objects for the 3 bottom PWM channels(LTC12 to LTC15, LTC20 to LTC23 and LTC28 to LTC31)

When updating the duty cycles, the channel shadow registers are updated for all 3 channels and then the software request to toggle the timer SO line is done.

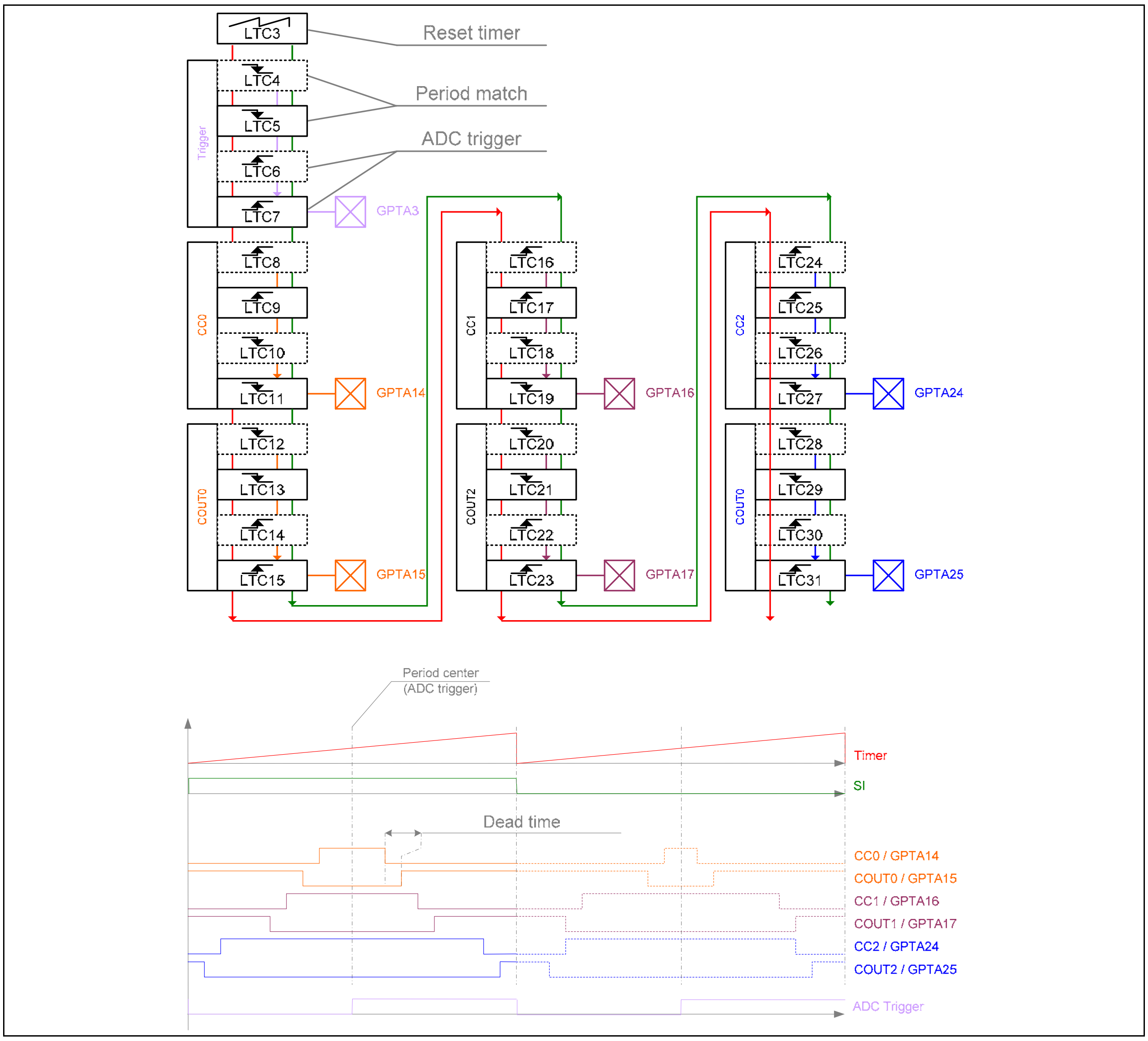


Figure 8 3*2 channels coherent update PWM

6.4.2 Configuration summary

Resource	Configuration	Info
GPTA3	• Direction : output • Behavior : Push-pull ALT1 • Default : High	Timer pin
GPTA8 GPTA17 GPTA26	• Direction : output • Behavior : Push-pull ALT1 • Default : Low	Top PWM
GPTA9 GPTA19 GPTA27	• Direction : output • Behavior : Push-pull ALT1 • Default : Low	Bottom PWM

	Mode	Input	Input sensitivity	Initial Value	Default SO line	Action on event	Output	Interrupt
LTC3	Reset timer	GPTA clock bus 0	Level sensitive (high level)	0x0000	low	Hold	No	No
	Mode	**Input**	**Input sensitivity**	**Initial Value**	**SI enable state**	**Action on event**	**Output**	**Interrupt**
LTC4	compare	-	-	Period-2	High	Reset	No	No
LTC5	compare	-	-	Period-2	Low	Copy or Reset	No	No
LTC6	compare	-	-	(Period-2)/2 - 1	High	Copy or Set	No	Yes
LTC7	compare	-	-	(Period-2)/2 - 1	Low	Copy or Set	GPTA3	Yes
LTC8 LTC16 LTC24	compare	-	-	Period/2-1	High	Set	No	No
LTC9 LTC17 LTC25	compare	-	-	Period/2-1	Low	Copy or Set	No	No
LTC10 LTC18 LTC28	compare	-	-	Period/2-1	High	Copy or Reset	No	No
LTC11 LTC19 LTC27	compare	-	-	Period/2-1	Low	Copy or Reset	GPTA8 GPTA17 GPTA26	No
LTC12 LTC20 LTC28	compare	-	-	Period/2-1	High	Reset	No	No
LTC13 LTC21 LTC29	compare	-	-	Period/2-1	Low	Copy or Reset	No	No
LTC14 LTC22 LTC30	compare	-	-	Period/2-1	High	Copy or Set	No	No
LTC15 LTC23 LTC31	compare	-	-	Period/2-1	Low	Copy or Set	GPTA9 GPTA19 GPTA27	No

6.4.3 Duty cycle formula and output example

Find below the pseudo-code used for the leading and trailing edge computation:

```
Min = MinPulse;
Max = Period - MinPulse;
if (Ton < Min)
        Ton = 0;
else if (Ton > Max)
        Ton = Period;
tTopLeadingEdge = (Period -Ton)/2-1
tTopTrailingEdge = (Period +Ton)/2-1
if (Ton > 0)
{
        Ton += 2 * DeadTime;
        if (Ton > Max)
                Ton = Period;
}
tBottomLeadingEdge = (Period -Ton)/2-1
tBottomTrailingEdge = (Period +Ton)/2-1
```

Where:

- `Period` is the period in ticks `Period` = F_{Timer} / f_{PWM} with f_{Timer} the reset timer input frequency ($f_{Timer} = f_{GPTA}$)
- `Ton` is the “ON” time of the PWM in ticks with `Ton` = DutyCycle * `Period`
- `MinPulse` is the smallest allowed pulse in ticks (0 in the example)

The table below shows in the 1st part the timer output for a period of T=8 ticks, DeadTime=1, MinPulse=0. The output is high for 4 ticks and low for 4 ticks. The 2nd part shows the output for one of the top channels for the duty cycle 0%, 100%, 50% and 12.5%. Note that:

- For dc[12]=0%, the top output is set at t=3 but immediately reset by the trailing cell so that no pulse appear on the output. The bottom output is reset at t=3 but immediately set by the trailing cell so that no pulse appears on the output.
- For dc=100% the top output is set as the timer is reset to t=-1 (0xFFFF). As the trailing edge is set to t=7 the falling edge never occurs and the output remain set. The bottom output is reset as the timer is reset to t=-1 (0xFFFF). As the trailing edge is set to t=7 the falling edge never occurs and the output remain set.
- For dc=50% the top output is set at t=1 and reset at t=5, the output have 4 ticks high and 4 ticks low. The bottom output is reset at t=0 and set at t=6. A dead time of 1 tick have been inserted.
- For dc=12.5%, the output is set for 1 tick only at t=2. The bottom output is reset at t=1 and set at t=4. A dead time of 1 tick have been inserted.

Timer Value														
					7									
					6								X	
					5							X		
					4						X			
					3					X				
					2				X					
					1			X						
					0		X							
					-1	X								X
Timer output														
dc/Ton	Lead-ing value	Trail-ing value	Leading formula	Trailing formula										
50%/4	6	2	T-2	(T-2)/2 - 1		■	■	■	□	□	□	□	■	
Channel output														
dc/Ton	Lead-ing value	Trail-ing value	Leading formula	Trailing formula										
0%/0	3	3	$t_{TopLeadingEdge}$	$t_{TopTrailingEdge}$	Top	□	□	□	□	▣	□	□	□	
	3	3	$t_{BottomLeadingEdge}$	$t_{BottomTrailingEdge}$	Bottom	■	■	■	■	▩	■	■	■	
100%/8	-1	7	$t_{TopLeadingEdge}$	$t_{TopTrailingEdge}$	Top	■	■	■	■	■	■	■	■	
	-1	7	$t_{BottomLeadingEdge}$	$t_{BottomTrailingEdge}$	Bottom	□	□	□	□	□	□	□	□	
50%/4	1	5	$t_{TopLeadingEdge}$	$t_{TopTrailingEdge}$	Top	□	□	■	■	■	■	□	□	
	0	6	$t_{BottomLeadingEdge}$	$t_{BottomTrailingEdge}$	Bottom	■	□	□	□	□	□	□	■	
12.5%/1	2	3	$t_{TopLeadingEdge}$	$t_{TopTrailingEdge}$	Top	□	□	□	■	□	□	□	□	
	1	4	$t_{BottomLeadingEdge}$	$t_{BottomTrailingEdge}$	Bottom	■	■	□	□	□	■	■	■	

Legend		
	High	■
	Low	□
	High overwritten by low	▣
	Low overwritten by High	▩

[12] Duty cycle

6.4.4 API

Mode	API[13]	Brief description
Initialization	`Pwm_Cu_3ChannelsCenterAligned_Init()`	Initialize the 3 channels center aligned PWM object
Update	`Pwm_Cu_3ChannelsCenterAligned_Update()`	Update the top and bottom PWM duty-cycle of all 3 channels
	`Pwm_Cu_3ChannelsCenterAligned_SetPeriodAndUpdate()`	Set the PWM period and update the top and bottom PWM duty-cycle of all 3 channels
	`Pwm_Cu_3ChannelsCenterAligned_Off()`	Switch off the PWM outputs
Settings	`Pwm_Cu_3ChannelsCenterAligned_SetDeadTime()`	Set the dead time
	`Pwm_Cu_3ChannelsCenterAligned_SetMinPulse()`	Set the min pulse
Status	`Pwm_Cu_3ChannelsCenterAligned_GetPeriod()`	Return the PWM period
	`Pwm_Cu_3ChannelsCenterAligned_GetDeadTime()`	Return the dead time
	`Pwm_Cu_3ChannelsCenterAligned_GetMinPulse()`	Return the time min pulse

For more details on each function, see the software documentation.

[13] _Cu_ refers to Coherent Update API

7 Source code example

7.1 Documentation

The source code documentation can be found in HTML format, see the file\Doc\html\index.html.

The documentation structure is as follow:

Tab	Description
Main page	General information
Module	Structured source code documentation.
Data structures	Information on data structures
Files	File description

Note that the default configuration can be modified in the file “configuration.h”.

7.2 Building the executable

Load and compile the tasking project AP32128_PwmGeneration(TC1766_RAM).pjt.

8 Abbreviations

LTC:	Local timer cell
GTC:	Global timer cell
GT:	Global timer
FPC:	Filter and prescaler cell
DCM:	Duty cycle measurement
PDL:	Phase discrimination logic
GPTA:	General purpose timer array
PWM:	Pulse width modulation