

# CCU6\_PWM\_Capture\_1

## CCU6 PWM signal capture

AURIX™ TC2xx Microcontroller Training  
V1.0.0



[Please read the Important Notice and Warnings at the end of this document](#)

## Scope of work

---

**The CCU6 unit is used to capture an external PWM signal and calculate its frequency and duty cycle.**

A simple PWM signal is generated by toggling a port pin. The resulting PWM frequency and duty cycle is measured by the Capture/Control Unit 6 (CCU6).

# Introduction

---

- › The Capture/Compare Unit 6 (CCU6) is a high resolution 16 bit capture and compare unit specially designed for motor control purposes.
- › The CCU6 unit is made up of a timer T12 Block with three capture/compare channels and a timer T13 Block with one compare channel.
- › Among other features, the CCU6 has the capability to capture external input signals. In this example, the Input Capture Unit and the timer T12 of the CCU6 module are used to capture a PWM signal and calculate its frequency and duty cycle.

# Hardware setup

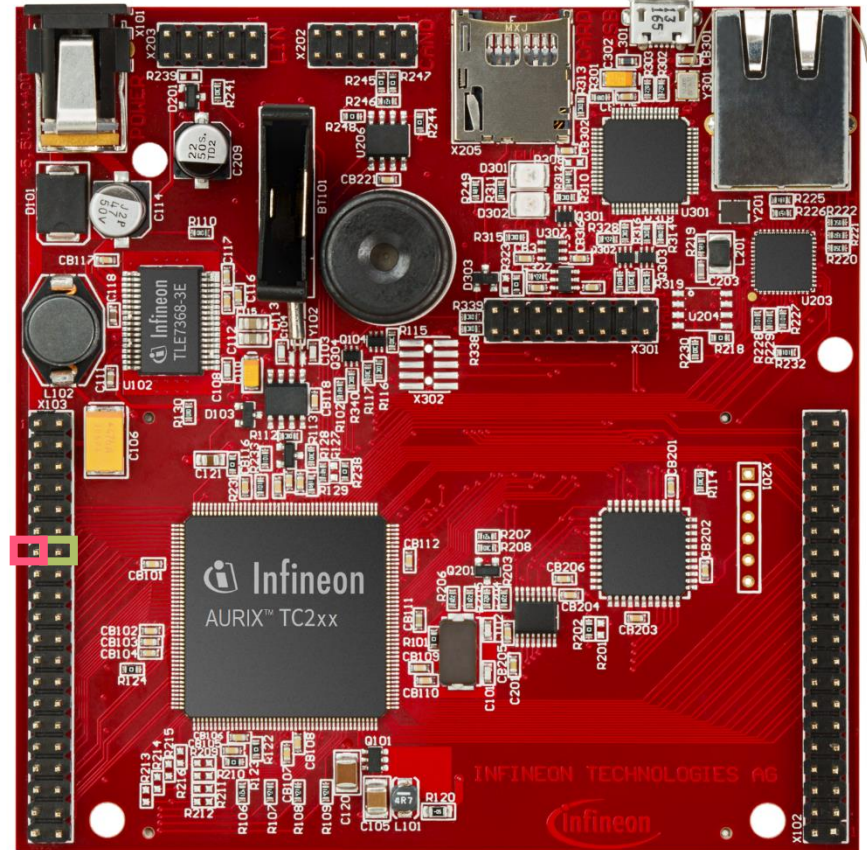
This code example has been developed for the board  
KIT\_AURIX\_TC297\_TFT\_BC-Step.

Connect the two pins  
P02.1 (PWM signal) and P02.0  
(CC60 input) to each other.

input pin A  
of CC60

	X103		
VCC_IN	1	2	V_UC(+5V)
GND	3	4	GND
P33.10	5	6	P33.9
P14.8	7	8	P14.7
P14.6	9	10	P10.6
P10.7	11	12	P10.4
P02.0	13	14	P02.1
P02.2	15	16	P02.3
P02.4	17	18	P02.5
P02.6	19	20	P02.7
P02.8	21	22	P00.0
P00.1	23	24	P00.2
P00.3	25	26	P00.4
P00.5	27	28	P00.6
P00.7	29	30	P00.8
P00.9	31	32	P00.10
P00.11	33	34	P00.12
AN45	35	36	AN44
AN17	37	38	AN16
AN25	39	40	AN24

PWM signal



# Implementation

---

## Configuring the CCU6 Input Capture Unit

To properly configure the CCU6 module for capturing a PWM signal, the module itself and one of its channels need to be configured.

For the configuration of the module, the following steps are required:

- › An instance of the structure ***IfxCcu6\_Icu\_Config*** needs to be created and default values are assigned to the configuration with the function ***IfxCcu6\_Icu\_initModuleConfig()***
- › For capturing a PWM signal, no changes to the default configuration are required, therefore the module can be initialized with the default configuration by calling the function ***IfxCcu6\_Icu\_initModule()***
- › After the successful initialization of the CCU6 module, its handle structure (***IfxCcu6\_Icu***) contains the configured capture frequency of the timer. For further usage, this parameter is stored in the global variable ***g\_CCU6captureFrequency\_Hz***.

All used functions and structures can be found in the iLLD header ***IfxCcu6\_Icu.h***.

# Implementation

## Configuring the CCU6 Input Capture Unit

For the configuration of the channel, an instance of the structure *IfxCcu6\_Icu\_ChannelConfig* is created and default values are assigned to the configuration with the function *IfxCcu6\_Icu\_initChannelConfig()*. Then, the following changes to the default configuration are required:

- › **channelId** – Select the channel of timer T12. In this example, channel 0 is selected
- › **channelMode** – Select the operating mode for the channel. In this example, the rising and falling edges of an external signal are captured using the double register capture mode
- › **Interrupt1** structure – Configure an interrupt by selecting the source, the service request output, its priority and the service provider. In this example, the CCU6 service request 0 is configured to trigger an interrupt on every rising edge at the input pin of capture/compare channel 0 (CC60). To calculate the duty cycle, a shadow register filled when a falling edge occurs, is used. Therefore, no interrupt is required for the falling edge.
- › **Interrupt2** structure – Configure another interrupt similar with the previous, where the CCU6 service request 1 is configured to trigger an interrupt on every period match (match of the timer T12 counter value with the period value)

**Note:** An interrupt is needed to capture the PWM frequency, while the other counts the number of timer overflows (for more details, see [slides 8-12](#)).

# Implementation

---

## Configuring the CCU6 Input Capture Unit

- › ***trigger.extInputTrigger*** - Select the internal start controlled by the run bit **T12R** by setting a null pointer (***NULL\_PTR***) to this field
- › ***pins*** – A structure to set the used port pins for the CCU6 configuration. Only CC60In with input mode ***IfxPort\_InputMode\_pullUp*** is selected
- › ***multiInputCaptureEnabled*** – Disable the multiple input capture mode

After the initialization of the channel with the user configuration (which is applied by calling the function ***IfxCcu6\_Icu\_initChannel()***), the capture process is started by setting the run bit **T12R** through the function ***IfxCcu6\_Icu\_startCapture()***.

All functions and structures used for the configuration of the CCU6 channel can be found in the iLLD header ***IfxCcu6\_Icu.h***.

# Implementation

---

## The Interrupt Service Routines (ISR)

For capturing of PWM signals with the CCU6 module, two ISRs are required:

- › ***CCU6\_ICU\_Period\_Match\_Int\_Handler()*** - Interrupt on every period match, used for counting the timer overflows
- › ***CCU6\_ICU\_Rising\_Edge\_Int\_Handler()*** - Interrupt on every rising edge at the input pin of capture/compare channel 0, used to calculate
  - the time between two rising edges and the PWM frequency
  - the time between a rising edge and the falling edge and the PWM duty cycle

The method implementing each ISR needs to be assigned a **priority** and a **CPU core** responsible for its execution. This is done with the macro ***IFX\_INTERRUPT(isr, vectabNum, priority)***.



# Implementation

## The Interrupt Service Routines (ISR)

- › When the rising edge is detected at the input pin of capture/compare channel 0 (CC60), the interrupt handler (***CCU6\_ICU\_Rising\_Edge\_Int\_Handler()***) is triggered
- › The interrupt status flags of the CC60 interrupt have to be cleared inside the ISR
- › The PWM frequency is calculated by dividing the CCU6 capture frequency (***g\_CCU6captureFrequency\_Hz***) by the total amount of increments of the timer T12 (the time between two rising edges)
- › The PWM duty cycle is calculated as a percentage value between the amount of increments of the timer T12 between the rising and falling edge (high level time) and the total amount of increments between two rising edges (period).
- › An overflow of timer T12 triggers the second interrupt (***CCU6\_ICU\_Period\_Match\_Int\_Handler()***) which is used for counting:
  - the overflows between two rising edges for PWM frequency calculation
  - the overflows between the rising edge and the falling edge for PWM duty cycle calculation

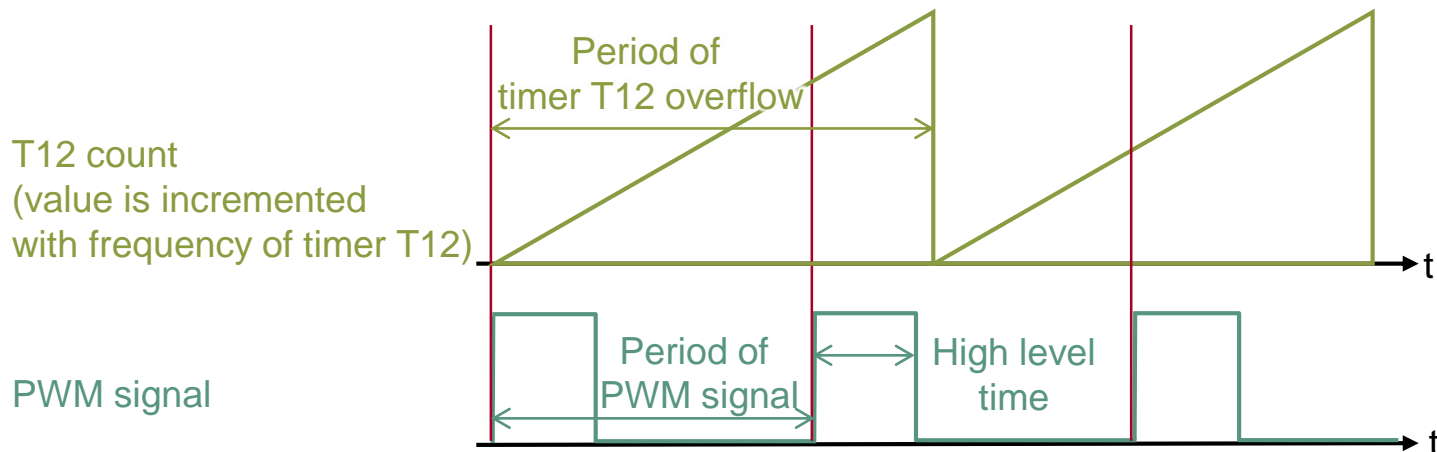
# Implementation

## Calculation example of the PWM frequency

Background knowledge:

- › Maximum value of timer T12 is **65534** (16 bit – 1; default configuration, configured inside function *IcxCcu6\_Icu\_initModuleConfig()*)
- › Frequency of timer T12 is **781250 Hz** (stored inside variable *g\_CCU6captureFrequency\_Hz*; default configuration), which means that the value of timer T12 is incremented every **1.28 μs** ( $1 / 781250 \text{ Hz}$ )
- › Overflow of timer T12 occurs after  $\approx$  **0.084 s** ( $65534 * 1.28 \mu\text{s}$ )
- › The total amount of increments can be calculated by comparing the current value of timer T12 with the value of timer T12 one PWM period ago e.g. **1000**

Interrupt on each rising edge:



# Implementation

## Calculation example of the PWM frequency

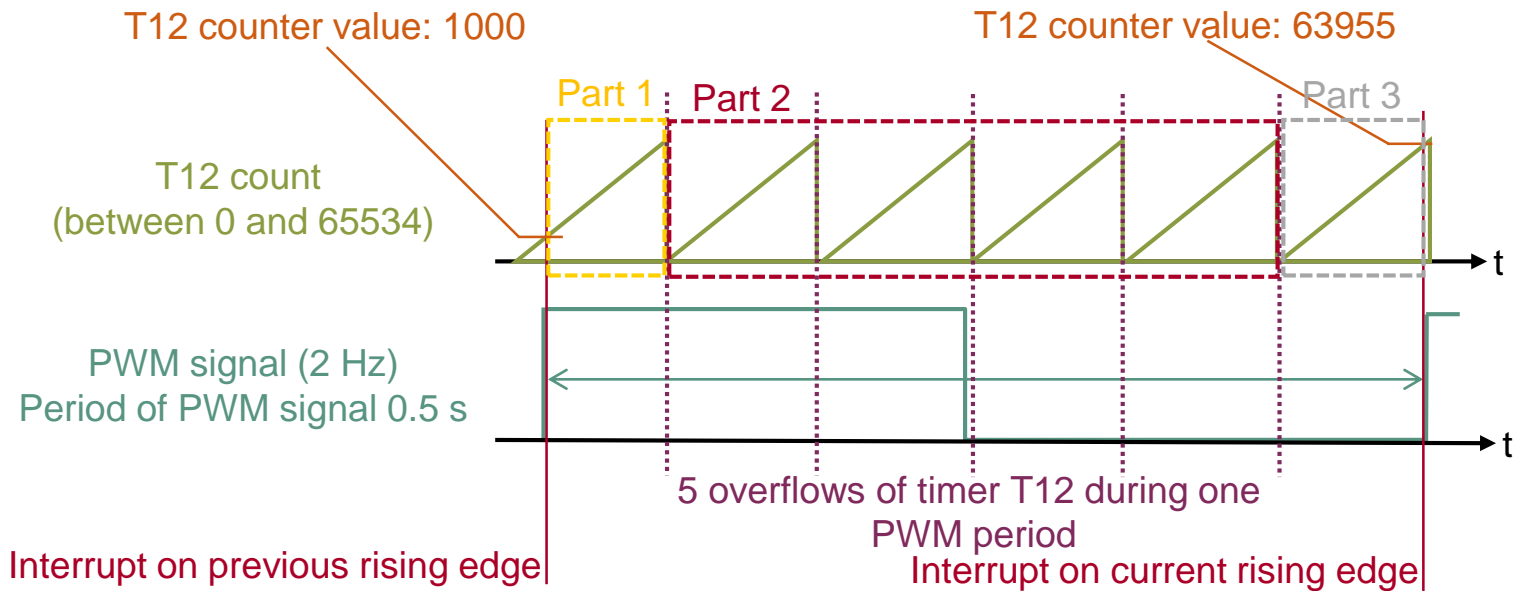
Example for 2 Hz PWM signal with 50% duty cycle:

- › Period time of 2 Hz signal is **0.5 s**
- › High level time is **0.25 s** (period \* duty cycle)
- › Within **0.5 s**, the timer T12 is incremented **390625** times ( $0.5 \text{ s} / 1.28 \mu\text{s}$ ) and has **5** overflows
- › Within **0.25 s**, the timer T12 is incremented **195312** times ( $0.25 \text{ s} / 1.28 \mu\text{s}$ ) and has **2** overflows
- › Parameters available inside the ISR:
  - Counter value of timer T12 one period ago: **1000**
  - Current counter value of timer T12: **63955**
  - Counter value when the falling edge occurred: **65244**
  - Amount of overflows during one PWM period: **5**
  - Amount of overflows during PWM high level time: **2**
- › Calculation of total amount of increments:
  - Increments before the first overflow:  $65534 - 1000 = \mathbf{64534}$
  - Increments during the second and the last overflow:  $(5 - 1) * 65534 = \mathbf{262136}$
  - Increments after the last overflow: **63955**
  - Total amount of increments:  $\mathbf{64534} + \mathbf{262136} + \mathbf{63955} = \mathbf{390625}$
- › Calculation of increments during high level time:
  - Increments before the first overflow:  $65534 - 1000 = \mathbf{64534}$
  - Increments during the second and the last overflow:  $(2 - 1) * 65534 = \mathbf{65534}$
  - Increments after the last overflow: **65244**
  - Total amount of increments:  $\mathbf{64534} + \mathbf{65534} + \mathbf{65244} = \mathbf{195312}$
- › Calculation of the PWM frequency by dividing the frequency of timer T12 by the total amount of increments during one PWM period:
  - $781250 \text{ Hz} / 390625 = \mathbf{2 \text{ Hz}}$
- › Duty cycle (increments during high level time / increments during total period) =  $195312 / 390625 = \mathbf{50\%}$

# Implementation

## Period calculation example

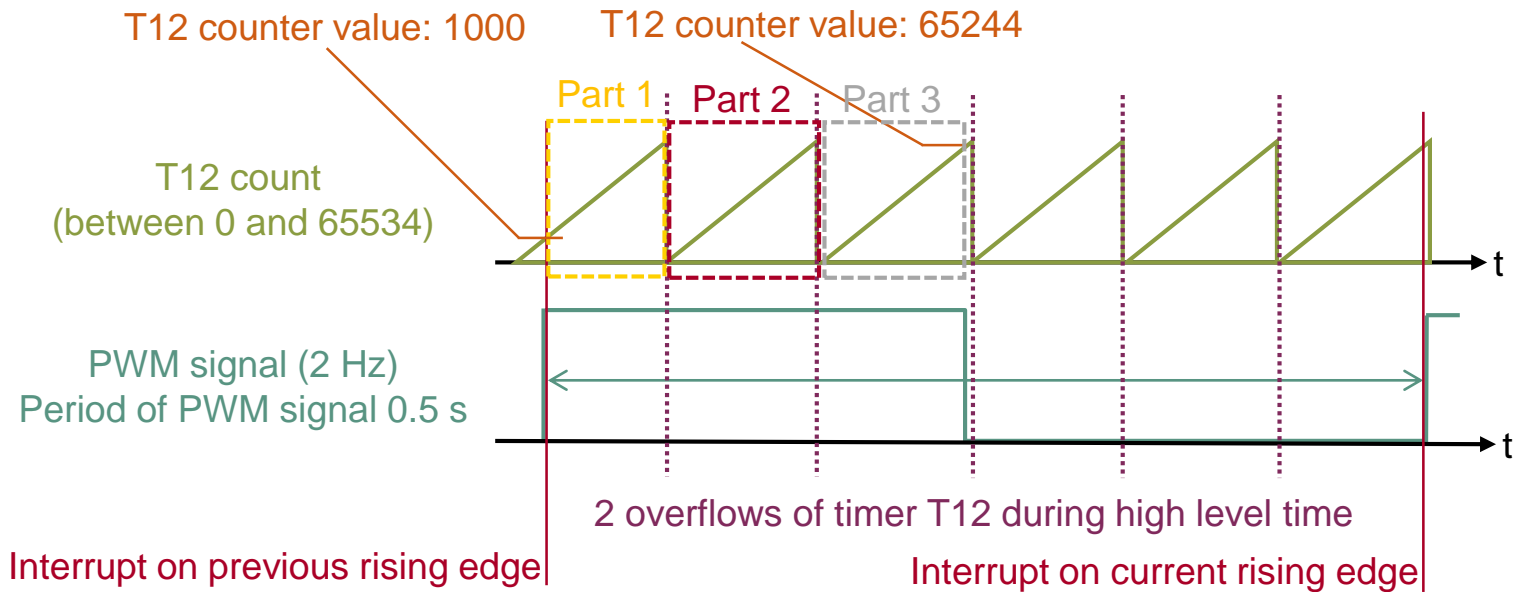
- › Part 1: 64534 increments before the first overflow:  $65534 - 1000 = 64534$   
(Maximum value of T12 - counter value at rising edge = total increments)
- › Part 2: 262136 increments between the first and last overflow:  
 $4 * 65534 = 262136$  (4 overflows \* maximum value of T12 = total increments)
- › Part 3: 63955 increments after the last overflow: 63955 (counter value at rising edge)
- › Total increments between two rising edges:  $64534 + 262136 + 63955 = 390625$



# Implementation

## High Level time calculation example

- › Part 1: 64534 increments before the first overflow:  $65534 - 1000 = 64534$   
(Maximum value of T12 - counter value at rising edge = total increments)
- › Part 2: 65534 increments between the first and last (before falling edge) overflow:  
 $1 * 65534 = 65534$  (1 overflows \* maximum value of T12 = total increments)
- › Part 3: 63955 increments after the last overflow: 63955 (counter value at rising edge)
- › Total increments between the rising and falling edges:  $64534 + 65534 + 65244 = 195312$



# Implementation

---

## Generation of PWM signal

After initializing the time constants (*initTime()*) and setting the port pin P02.1 as push-pull output (*IfxPort\_setPinMode()*), a simple PWM signal is generated by toggling a pin with the function *generate\_PWM\_signal()*.

The state of the output port pin P02.1 is toggled by calling the function *IfxPort\_setPinState()* with the parameters *IfxPort\_State\_high* and *IfxPort\_State\_low*.

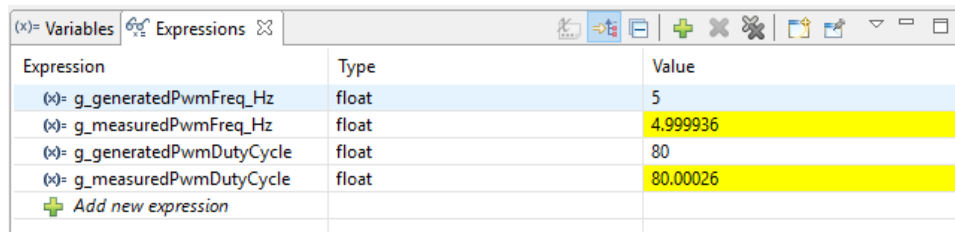
For changing the frequency and duty cycle of the generated PWM signal the global parameter *g\_generatedPwmFreq\_Hz* and *g\_generatedDutyCycle* can be modified. Depending on the frequency and duty cycle (set to 50%), two timeout values are calculated by software and passed to the *wait()* function. The two *wait()* function calls are succeeding the two calls of the function *IfxPort\_setPinState()*.

The parameters and functions used for the port pin control and timing are provided by the two iLLD headers *IfxPort.h* and *Bsp.h*.

# Run and Test

After code compilation and flashing the device, perform the following steps:

1. Connect the two pins P02.1 (PWM signal) and P02.0 (CC60 input) to each other
2. Check the parameter ***g\_measuredPwmFreq\_Hz*** in the debugger (the debug session should be suspended previously). Its value should be similar to the parameter ***g\_generatedPwmFreq\_Hz***
3. Change the parameter ***g\_generatedPwmFreq\_Hz*** and check if ***g\_measuredPwmFreq\_Hz*** changes accordingly
4. Check the parameter ***g\_measuredPwmDutyCycle*** in the debugger (the debug session should be suspended previously). Its value should be similar to the parameter ***g\_generatedPwmDutyCycle***
5. Change the parameter ***g\_generatedDutyCycle*** and check if ***g\_measuredDutyCycle*** changes accordingly



Expression	Type	Value
(x)= g_generatedPwmFreq_Hz	float	5
(x)= g_measuredPwmFreq_Hz	float	4.999936
(x)= g_generatedPwmDutyCycle	float	80
(x)= g_measuredPwmDutyCycle	float	80.00026
+ Add new expression		

# References



- › AURIX™ Development Studio is available online:
- › <https://www.infineon.com/aurixdevelopmentstudio>
- › Use the „*Import...*“ function to get access to more code examples.



- › More code examples can be found on the GIT repository:
- › [https://github.com/Infineon/AURIX\\_code\\_examples](https://github.com/Infineon/AURIX_code_examples)



- › For additional trainings, visit our webpage:
- › <https://www.infineon.com/aurix-expert-training>



- › For questions and support, use the AURIX™ Forum:
- › <https://www.infineonforums.com/forums/13-Aurix-Forum>



## Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

## Edition 2020-01

### Published by

**Infineon Technologies AG**  
**81726 Munich, Germany**

**© 2020 Infineon Technologies AG.**  
**All Rights Reserved.**

**Do you have a question about this document?**

**Email: [erratum@infineon.com](mailto:erratum@infineon.com)**

## Document reference

**CCU6\_PWM\_Capture\_1**

## IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office ([www.infineon.com](http://www.infineon.com)).

## WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.