# How to Use TC39x Example for Blinking LED

## For TC39x BIFACES Base Template Projects

## About this document

### Scope and purpose

Objectives of this example project are:

1. To show the user blinking 6 LEDs on Triboard
2. How to synchronize the CPUs to start execution with IfxCpu_emitEvent and IfxCpu_waitEvent APIs in case of disabled CPUs
3. How to enable/disable CPUs individually
4. How to enable/disable CPU cache individually

### Intended audience

This template is intended for the users of Base Template Projects for TC3xx microcontrollers. These template projects are made available as BaseFramework_<micro-controller derivate> and are buildable with ***BIFACES*** build environment.

## Table of contents



# 1 Blinking LED Example Files

Following are the files required for blinking LED example, which are required to be merged with ***Base Template Project***.

**Table 1 Files for Blinking LED Example**

| Files | Folder (project relative folder pah) | Purpose |
|---|---|---|
| BlinkeyLedExample.c | 0_Src\AppSw\Tricore \Demo_BlinkeyLed | Example code to implement the initialization settings and interrupt routines |
| CpuX_Main.c | 0_Src\AppSw\Tricore\Main | Calls to example functions are made here |
| Ifx_Cfg.h | 0_Src\AppSw\Config\Common | Configuration file for the project. This file holds the #define macros to enable/disable different features |

# 2 Building Blinking LED Example Files

This example is buildable only with BaseFramework_TC39A/B ***Base Template Project***s (which has 6 TriCore™ CPUs). However, you could easily modify the code to make it work for other derivates.

The example code is available with folder _Example/BlinkyLed_TC39x. Files provided with this example are only sub-set of required files to build, which are incremental to the BaseFramework_TC39A/B. You need to correctly merge the files to already working ***Base Template Project***. To build example code, you don't need extra project configuration.

## 2.1 Pre-requisites to Build Example Files

To build Blinking LED example,

- you must have required ***Base Template Project*** configured to build and generate the executable for the required compiler ***Toolchain***
- You must also check that required ***Base Template Project*** actually runs on the hardware and stays in infinite loop of coreX_main functions for all available TriCore™ CPUs

## 2.2 Merging Examples to Base Template

When you install ***Base Template Project***s, you get a folder with folder name: "**_Example_TC3xx**". This folder contains several sub-folders, each of them are sub set of project source files and/or project configuration files. By the folder-name of such sub-folders, you could figure out whether an example is directly compatible with targeted template project. As examples, if the folder-name ends with TC3xx, the example could be used with any of the ***Base Template Project***s (each corresponds to a derivate). If a folder-name ends with TC39x then such example could be used with ***Base Template Project***s of TC3xx.

This section details how to merge the examples with the ***Base Template Project***s with Eclipse CDT environment and outside Eclipse environment (normally for non Eclipse users).

### 2.2.1 Working Outside Eclipse Environment

Following are the steps to build project, when you want to do the merge examples outside Eclipse environment:

1. At your workspace, copy the folder tree of BaseFramework_TC3xx project (you must be sure that this project is correctly setup for used compiler, is buildable successfully and it runs on the hardware)
2. Paste it as new folder-name which is meaningful for the application you are going to work on (e.g. BlinkeyLed_TC39A/B)

Tip: Always suffix the root folder name of project to represent the microcontroller derivate name and its step,because this information is not easily found from the source files/ configuration inside your project.

3. Merge and overwrite the folder tree at project root level, with **_Examples_TC3xx/BlinkeyLed_TC39x** folder (overwrite the files/folders which are same). Take care to copy the folder structure one to one.

Now your example is ready to be used

### 2.2.2 Working within Eclipse Environment

Following are the steps to build project when you want to merge with Eclipse environment:

**Copy the working template project**

1. Under eclipse Project explorer: Copy BaseFramework_TC39A/B project with **Ctrl+c** keys (or right-mouse-click on project root folder and click **Copy**)

   ***Attention***: ***You need to be sure that this project is correctly setup for used compiler, is buildable successfully and it runs on the hardware.***

2. Paste **Ctrl+v** (or right-mouse-click at the end of folders can click **Paste**). You need to provide a new name in the window, which comes up. Give any new name which meaningful for the application you target (e.g. BlinkeyLed_TC39A/B). This is new project which has all the required settings which you have already done in template project.

**Merge the example sources:**

1. Right-click on newly copied project root folder (e.g. BlinkeyLed_TC39A/B) at eclipse project explorer, and click on **“Import”** as shown in the figure below:

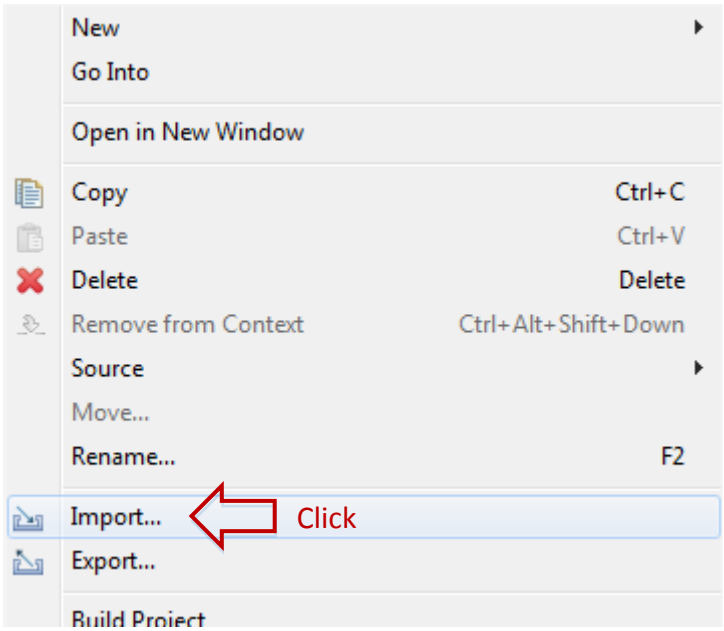


**Figure 1 Import Menu**

2. A window **“Import”** opens up, under **“Select an import source:”** expand the tab **“General”** and click on **“File System”** and click **“Next”** button as shown below:

## 2 Building Blinking LED Example Files

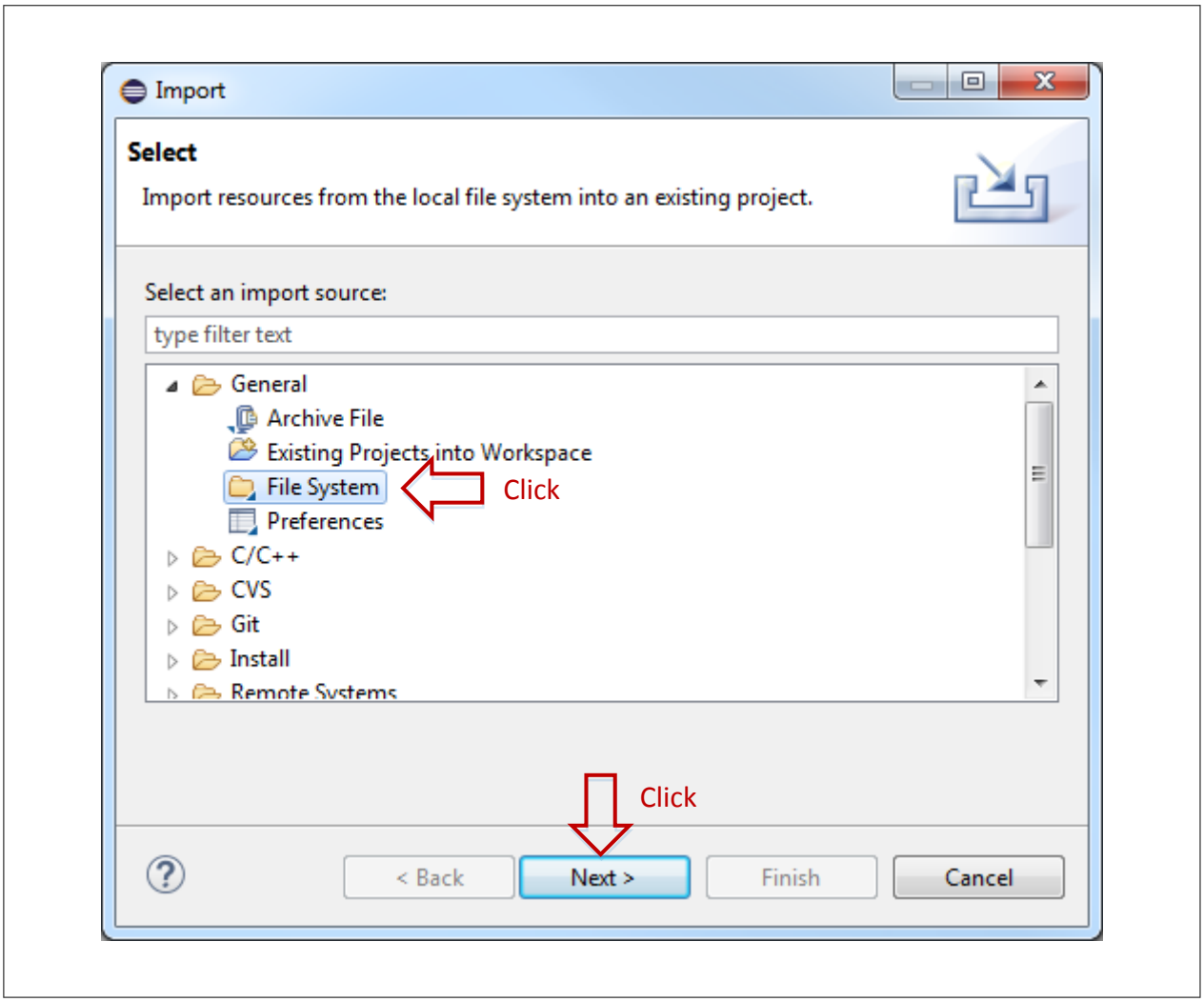


**Figure 2 Import Wizard**

**3.** Now this change to window as shown below, click on **"Browse"** button to browse to the workspace where the template projects are installed and select the folder **_Examples_TC3xx/BlinkeyLed_TC39x**.

*Note*: *Below picture is made for the example folder of BlinkeyLed_TC39x and that is merged to project MyDemo_TB39B. This looks different your project and the example you are selecting.*

## 2 Building Blinking LED Example Files

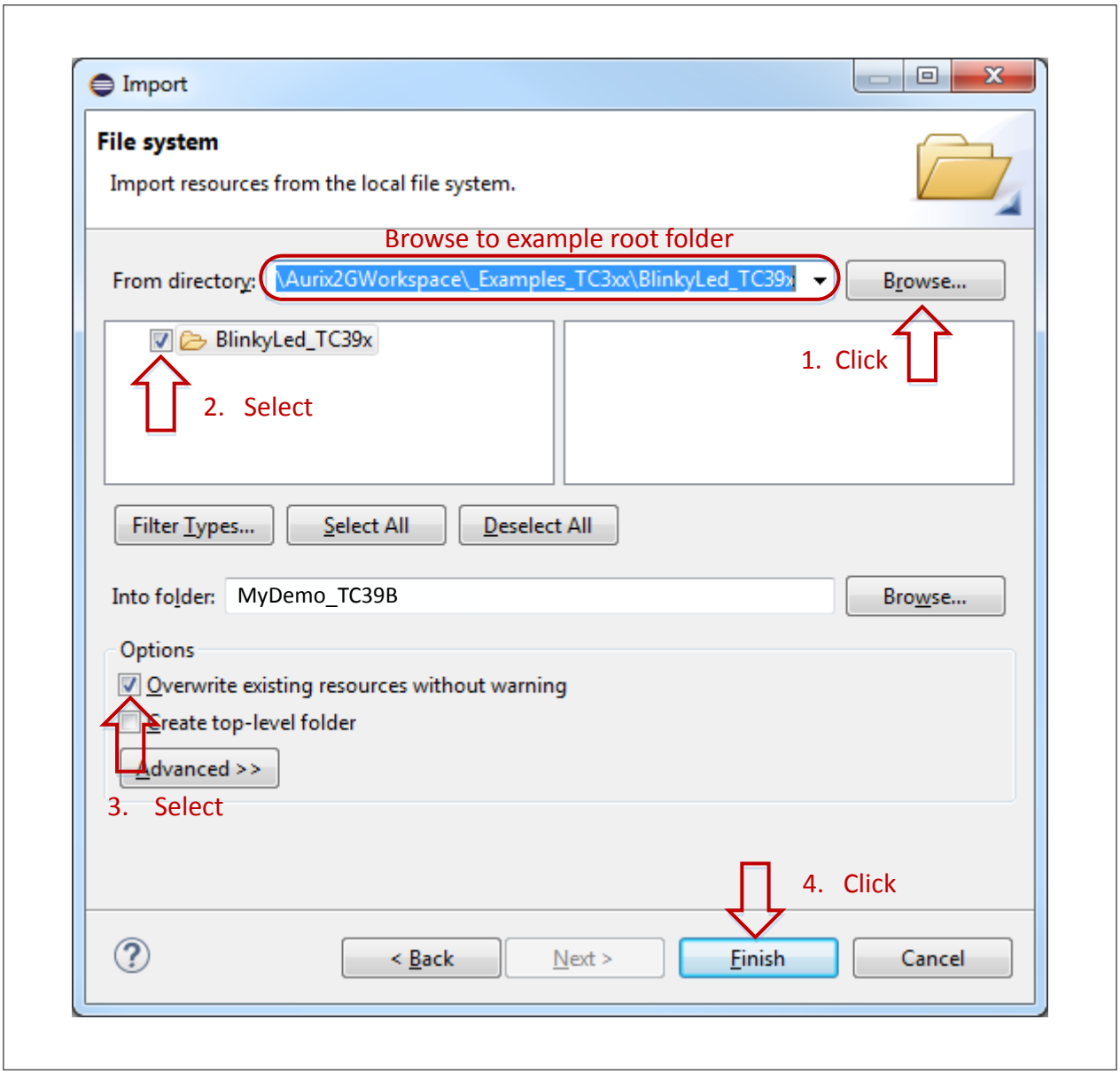


**Figure 3 Import Folders**

4. Path for folder: BlinkeyLed_TC39x appears in **"From directory:"** field as above figure.
5. Select the check-box against BlinkeyLed_TC39x in the selection field.
6. Also check the check-box under **"Options"**, "**Overwrite existing resources without warning**"
7. Click "**Finish**"
8. Now example folders are merged and the example is ready to be used

## 2.3 Invoke Build for Example Project

Once all the required setups are done, you can build the project.

To build the project, you must invoke:

- **make** target **clean**
- **make** target **all**

The result of build shall produce the elf file as TriCore™ image. .

Tip: You need to build once before your modifications, just to be sure that the integration is fine. After successful build, you could proceed with your own modifications.

*Note*: *Please remember to build your examples every time after your modifications.*

# 3 LEDs Blinking

When the example is built, downloaded to the flash and run, the effect is blinking LEDs which are blinking at 1 second's rate (On for 1 s and Off for 1 s). Each LED corresponds to a TriCore™ CPU. Below table maps the LEDs on Triboard HW to the CPU id.

**Table 2 CPU to LED Mapping for Triboard (Valid for Type TC3xx)**

| LED number | Mapped TriCore™ CPU | Port Pin to which LED is connected |
|---|---|---|
| 0 | CPU0 | P33.4 |
| 1 | CPU1 | P33.5 |
| 2 | CPU2 | P33.6 |
| 3 | CPU3 | P33.7 |
| 4 | CPU4 | P20.11 |
| 5 | CPU5 | P20.12 |

# 4 Enable/Disable CPUs at Startup

Enable or Disable of CPUs are done at the startup software based on the following #define constants. Please refer to ***iLLD*** startup code, which is available in folder 0_Src\BaseSw\Infra\Ssw\<derivate>, to know how this is implemented. When you disable a CPU with this configuration, user startup software would keep the CPU in halt mode. In case of CPU0 is disabled with this configuration, it goes to halt mode at the end of user startup software after starting other enabled CPUs.

**Table 3 Configuration constants for CPU enable / disable**

| #define Constant | Purpose | Value |
|---|---|---|
| IFX_CFG_SSW_ENABLE_TRICORE**x** | Enable/ disable TriCore™ CPU **x** (where x=0-6) during user startup software | 0: Disables<br>1: Enables |

You need to define these #define constants explicitly only when you want to change the default value. If these are not defined explicitly, by default, all the TriCore™ CPUs are enabled by startup software. You need to define these constants at file **“0_Src/AppSw/CpuGeneric/Config/Ifx_Cfg.h”**.

When a CPU is not enabled during startup software, obviously, this would not participate in CPU synchronization event. CPU synchronization is done with Cpu-Emit and Cpu-Wait response calls. To make Cpu-Wait bypassed for a disabled CPU, following #define constant need to be configured:

**Table 4 CPUs' emit event constant**

| #define Constant | Purpose | Value |
|---|---|---|
| IFXCPU_CFG_ALLCORE_DONE | **IfxCpu_emitEvent** API, which is called by each CPU during synchronization. This API updates a variable parameter corresponding to bit position represented by CPU ID.<br>This constant value is the final value once all CPUs are synchronized. The variable is polled against this value by the API **IfxCpu_waitEvent** | Bit 0 = 1 : CPU0 is at synch point<br>Bit 1 = 1 : CPU1 is at synch point<br>Bit 2 = 1 : CPU2 is at synch point<br>Bit 3 = 1 : CPU3 is at synch point<br>Bit 4 = 1 : CPU4 is at synch point<br>Bit 6 = 1 : CPU5 is at synch point |

You need to define this #define constant explicitly only when you want to change the default value. If this is not defined explicitly, by default, a constant is defined that expect all TriCore™ CPUs to participate in CPU synchronization event. You need to define the above constant at file **"0_Src/AppSw/CpuGeneric/Config/Ifx_Cfg.h"**

*Note*: *In the worst case, if the above constant is not defined with appropriate value, the wait for CPU synchronization event will timeout and goes further. No error notification done in this case. However, is such a case, CPUs are not synchronized.*

Following is an example to disable CPU1 (as described earlier, by default all the CPUs are enabled). You need to modify the file Ifx_Cfg.h as shown below.

```
//File: 0_Src/AppSw/CpuGeneric/Config/Ifx_Cfg.h

#define IFX_CFG_SSW_ENABLE_TRICORE0  1
#define IFX_CFG_SSW_ENABLE_TRICORE1  0
#define IFX_CFG_SSW_ENABLE_TRICORE2  1
#define IFX_CFG_SSW_ENABLE_TRICORE3  1
#define IFX_CFG_SSW_ENABLE_TRICORE4  1
#define IFX_CFG_SSW_ENABLE_TRICORE5  1


//Constant for CPU Sync:
#define IFXCPU_CFG_ALLCORE_DONE                \
        (IFX_CFG_SSW_ENABLE_TRICORE0 << 0U)  | \
        (IFX_CFG_SSW_ENABLE_TRICORE1 << 1U)  | \
        (IFX_CFG_SSW_ENABLE_TRICORE2 << 2U)  | \
        (IFX_CFG_SSW_ENABLE_TRICORE3 << 3U)  | \
        (IFX_CFG_SSW_ENABLE_TRICORE4 << 4U)  | \
        (IFX_CFG_SSW_ENABLE_TRICORE5 << 6U)
```

# 5 Enable/Disable Caches at Startup

CPU data cache and program caches are independently enabled/ disabled. This is done at the startup software, based on the following #define constants. Please refer to ***iLLD*** driver code, which is available in folder "**0_Src\BaseSw\Infra\Ssw\<derivate>**" to know how this is done.

### Glossary

You need to define these #define constants explicitly only when you want to change the default value. If these are not defined explicitly, by default, all the caches with TriCore™ CPUs are enabled by startup software.

**Table 5 Configuration constants for CPU cache enable / disable**

| #define Constant | Purpose | Value |
|---|---|---|
| IFX_CFG_SSW_ENABLE_TRICORE**x**_PCACHE | Enable/ disable program cache for CPU **x** (where x=0-6) during user startup software | 0: Disables<br>1: Enables |
| IFX_CFG_SSW_ENABLE_TRICORE**x**_DCACHE | Enable/ disable data cache for CPU **x** (where x=0-6) during user user startup software | 0: Disables<br>1: Enables |

Above macros are imported by user startup software through file: "**0_Src/AppSw/CpuGeneric/Config/Ifx_Cfg.h**". You have to update this file to configure CPU enable or disable with user startup software.

Following is an example to disable CPU1 program cache and data cache (By default all the CPU caches are enabled). You need to modify the file as shown below.

```
//File: 0_Src/AppSw/CpuGeneric/Config/Ifx_Cfg.h

#define IFX_CFG_SSW_ENABLE_TRICORE0_PCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE0_DCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE1_PCACHE  0
#define IFX_CFG_SSW_ENABLE_TRICORE1_DCACHE  0
#define IFX_CFG_SSW_ENABLE_TRICORE2_PCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE2_DCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE3_PCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE3_DCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE4_PCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE4_DCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE5_PCACHE  1
#define IFX_CFG_SSW_ENABLE_TRICORE5_DCACHE  1
```

# Glossary

**API**
**A**pplication **P**rogram **I**nterface

**Architecture**
CPU Architecture. e.g. TriCore™, Arm, Intel e.t.c

**Base Template Project**
This is a ***BIFACES Project***, which is provided for each micro-controller derivative. This project contains

- Basic project configurations,
- Startup, SFR Headers and infrastructure driver files
- Template linker command files for each supported compiler.

The infrastructure drivers used with base projects are respective ***iLLD***s of corresponding microcontroller detivate. User could enhance this project with own application code.

## Glossary

**BIFACES**

**B**uild and **I**ntegration **F**ramework for **A**utomotive **C**ontroller **E**mbedded **S**oftware

**BIFACES Project**
It is a project environment or working directory, where source files and/or configuration files are stored. Files and folders in such working directory is organized as required by ***BIFACES***. ***BIFACES*** works only on such a working directory.

**iLLD**

**I**nfineon **L**ow **L**evel **D**river

**Tool**
Tool to build source files. Normally a tool is used for a particular phase of build process e.g. Compiler, Assembler, Linker, Archiver e.t.c. Tools could be internal or external.

**Toolchain**
Collection of ***Tool***s, for a specific ***Architecture***, to build source files. e.g. GNU C from Hightec, Tasking from Altium, Diab from Windriver e.t.c.

**Trademarks**

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

**Edition 2017-07-20**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**



**Do you have a question about any aspect of this document?**

**Email: erratum@infineon.com**

**Document reference**
**IFX-wbw1499174254285**

**IMPORTANT NOTICE**

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. 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) with respect to any and all information given in this application note.

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.

**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