Special Function Registers

32-bit TriCore™ AURIX™ TC3xx microcontroller family

About this document

Scope and purpose

This User Manual provides an overview of the Special Function Register headers and explains how to use the header files, in general, with any of the targeted microcontroller product.

Note: *This document contains examples, which may carry specific register name from specific product. These are to be treated as examples only and may or may not be correct to their actual hardware function in a product.*

Intended audience

This document is intended for anyone using the Special Function Register files.

Glossary of terms

Table 1 Glossary

Term	Description
IFX	Infineon Technologies
MCU	Micro Controller Unit
SFR	Special Function Registers
AoU	Assumptions of Use
µC	Micro Controller
UM	User Manual
TS	Target Specification

Reference document

This User Manual should be read in conjunction with the following document:

TC3xx User Manual / Target Specification [Corresponding version in Release Notes]

Table of contents

1 Introduction

The product, SFR header files, is the bridge to connect software domain with microcontroller hardware domain.

Software accessibility to microcontroller hardware is, always, only through the registers. These are commonly called as Special Function Registers (SFR). These registers in a way carry the commands, configuration or status to and from the microcontroller hardware. Each of the registers contains important attributes called as bit-fields. Each bit-field of a register represents specific functionality. The User Manual of microcontroller hardware product states the accessible registers along with their bit-field and their respective functionalities.

As the microcontroller hardware is very sensitive to the information these bit-fields carry the user commands or configuration. Software functionalities are also sensitive to these bit-fields, which carry the status information. Because of this sensitivity, the registers with their bit-fields are crucial for both the domains, that is, software and hardware to function together to fulfil the intended functionality.

SFR header files are C language representation of microcontroller registers because most of the software functionality are constructed with the C language (on the contrary the header files for assembly language are Register ASM headers).

Note: This product targets to represent only C language and not assembly language usage.

Table 356 Register Overview - ASCLIN (ascending Offset Address) (cont'd)

Short Name	Long Name	Offset Address	Access Mode		Reset
			Read	Write	
TXFIFOCON	TX FIFO Configuration Register	$00C_H$	U,SV	SV,U,P	Application Reset
RXFIFOCON	RX FIFO Configuration Register	010_H	U,SV	SV,U,P	Application Reset
BITCON	Bit Configuration Register	014_H	U,SV	SV,U,P	Application Reset
FRAMECON	Frame Control Register	018_H	U,SV	SV,U,P	Application Reset
DATCON	Data Configuration Register	$01C_H$	U,SV	SV,U,P	Application Reset
BRG	Baud Rate Generation Register	020_H	U,SV	SV,U,P	Application Reset
BRD	Baud Rate Detection Register	024_H	U,SV	SV,U,P	Application Reset
LINCON	LIN Control Register	028_H	U,SV	SV,U,P	Application Reset

User Manual representation of registers memory map

```
/** \brief C, TX FIFO Configuration Register */
#define ASCLIN0_TXFIFOCON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_TXFIFOCON*)0xF000060Cu)

/** \brief 10, RX FIFO Configuration Register */
#define ASCLIN0_RXFIFOCON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_RXFIFOCON*)0xF0000610u)

/** \brief 14, Bit Configuration Register */
#define ASCLIN0_BITCON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_BITCON*)0xF0000614u)

/** \brief 18, Frame Control Register */
#define ASCLIN0_FRAMECON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_FRAMECON*)0xF0000618u)

/** \brief 1C, Data Configuration Register */
#define ASCLIN0_DATCON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_DATCON*)0xF000061Cu)

/** \brief 20, Baud Rate Generation Register */
#define ASCLIN0_BRG /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_BRG*)0xF0000620u)

/** \brief 24, Baud Rate Detection Register */
#define ASCLIN0_BRD /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_BRD*)0xF0000624u)

/** \brief 28, LIN Control Register */
#define ASCLIN0_LIN_CON /*lint --e(923, 9078)*/ (*(volatile Ifx_ASCLIN_LIN_CON*)0xF0000628u)
/** Alias (User Manual Name) for ASCLIN0_LIN_CON.
* To use register names with standard convension, please use ASCLIN0_LIN_CON.
*/
#define ASCLIN0_LINCON (ASCLIN0_LIN_CON)
```

C header file representation of registers memory map

Figure 1 Register memory map notations

2 Including SFR headers

This section provides an overview of the package structure, file naming conventions and including the register headers in different use-cases.

2.1 Package structure

A separate SFR Header files package is delivered for each microcontroller derivative. User must choose the correct package for the target microcontroller.

The release contains individual packages for each microcontroller. These are an installer packages with the name same as their version tag.

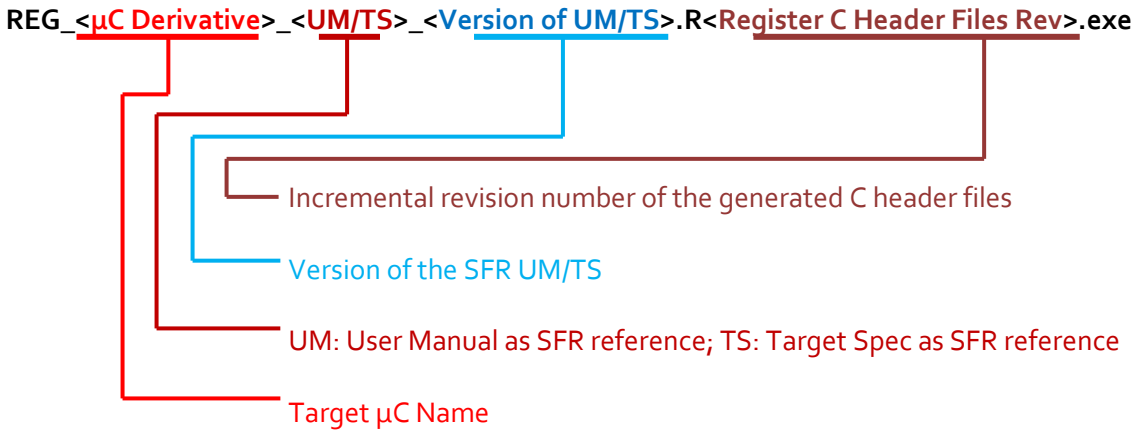


Figure 2 SFR Header release package name tag

For example:

Package `REG_TC39XB_UM/TS_V2.3.0.R0.exe`

- Contains registers for TC39xB derivative
- Registers are generated from UM/TS SFR reference file version 2.3.0
- Ro represents the first incremental release for the generated registers from same file

Once the package is installed, the following folders are created:

Table 2

Folder	Description
`_Reg`	Contains actual C header files, which are to be included in to the applications
`SupportDocuments`	Contains documents to support user during development and project quality audits. User Manual is also a part of the SupportDocuments.

2.2 SFR header files naming

For each peripheral within the microcontroller, SFR representation is realized with three different files.

- REGDEF files: definition of Register/Peripheral Module Types: Representation of hardware register memory map or bit-field map for any peripheral are done through structure and union type definitions

Naming convention: `Ifx<peripheral name>_regdef.h`

  - where, peripheral name represents the peripheral IP name of the microcontroller. For example, `IfxCpu_regdef.h`, `IfxCan_regdef.h` and `IfxGeth_regdef.h`
  - for peripheral IPs such as MCMCAN, the generic name CAN is used

- REG files: definition of register memory map: Register memory map representation of any peripheral is done through #define macros assigning the register names to their memory address.

Naming convention: `Ifx<peripheral name>_reg.h`

  - where, peripheral name represents the peripheral IP name of the microcontroller. For example, `IfxCpu_reg.h`, `IfxCan_reg.h` and `IfxGeth_reg.h`

- BF files: definitions of register bit-fields to use the masked way of register access are done in BF files. These files contain the register bit-field mask, bit-field length and bit-field position.

Naming convention: `Ifx<peripheral name>_bf.h`

  - where, peripheral name represents the peripheral IP name of the microcontroller. For example, `IfxCpu_bf.h`, `IfxCan_bf.h` and `IfxGeth_bf.h`

- SUPERSET REG file: includes all the individual peripherals registers in one file. This is useful when an application use the registers from many different modules.

File name: `Ifx_reg.h`

- BASICTYPES: define the basic data types for the Registers bit-fields. Some of the registers need the special keyword to tell the compiler that, such registers bit-fields can only be accessed as 32- or 16-bit. Compiler will not optimize access to such register bit-fields.

File name: `Ifx_TypesReg.h`

2.3 Include structure within the SFR header files

SFR headers files have fixed include structure hierarchy in such a way that user has flexibility and ease of use across different use cases. The following figure depicts the hierarchy.

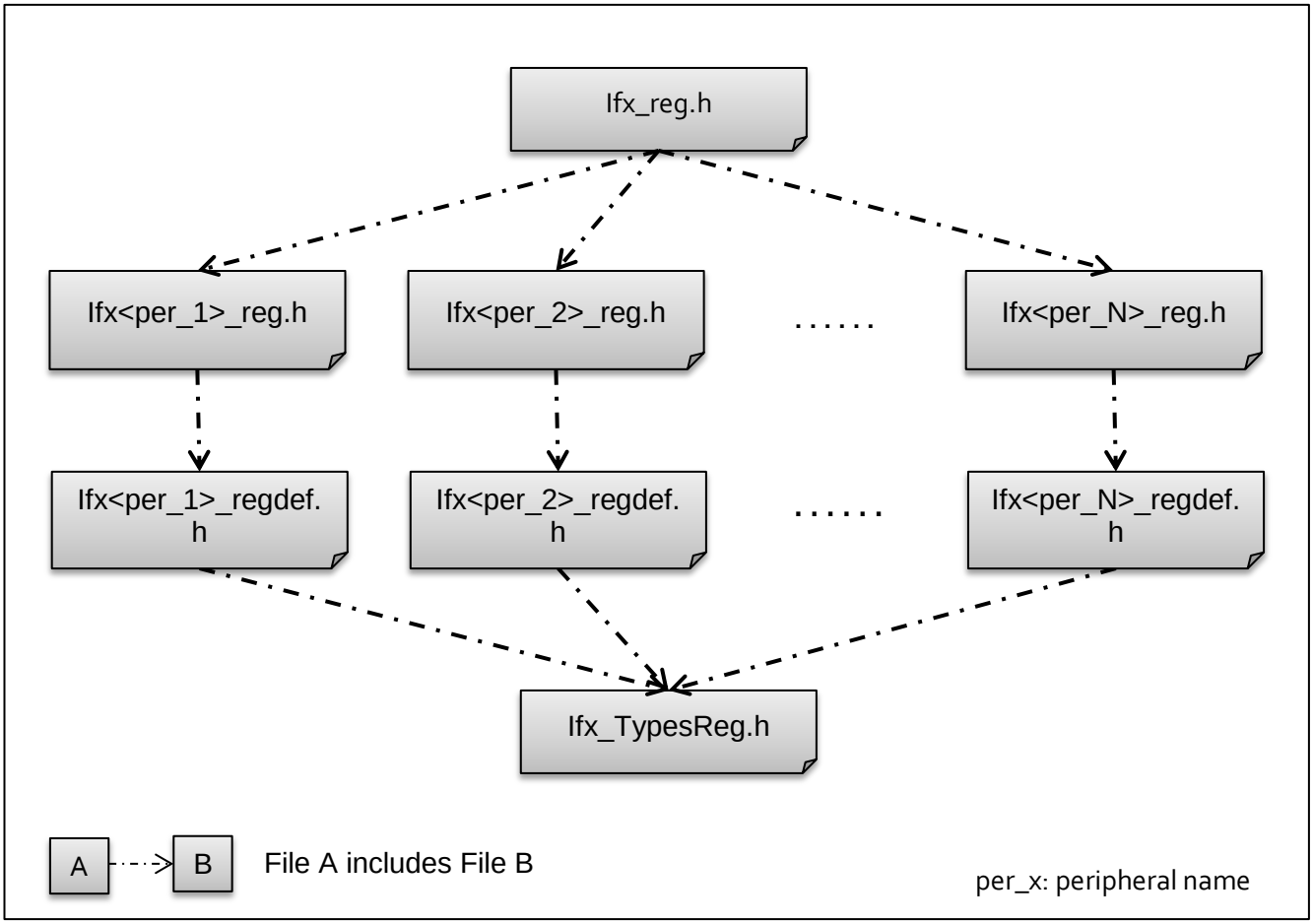


Figure 3 Include structure within SFR header files

In the figure, file names, `Ifx<peripheral name>_bf.h`, are individual files, which are independently included based on the need in the application.

2.4 Hints on including the register header files

To access registers, only `Ifx_reg.h` and `Ifx<peripheral name>_bf.h` files are required to be included by the user. Other files are indirectly included as shown in Figure 3.

This section provides details about including the SFR header files based on the use cases.

2.4.1 Usage with peripheral driver files

When a driver is developed for a specific peripheral hardware module, registers from only few of the peripherals are accessed. It is recommended to include only the files for the needed peripheral modules. For example, for the ADC driver, for EVADC peripheral include only include `IfxEvadc_reg.h` file and additionally `IfxScu_reg.h` file.

Note: This is recommended for optimizing the compile time of individual driver files.

2.4.2 Access SFRs from more than one peripheral

When an application or a test/start up functionality needs to access multiple peripherals, then it is easy to include `Ifx_reg.h` file.

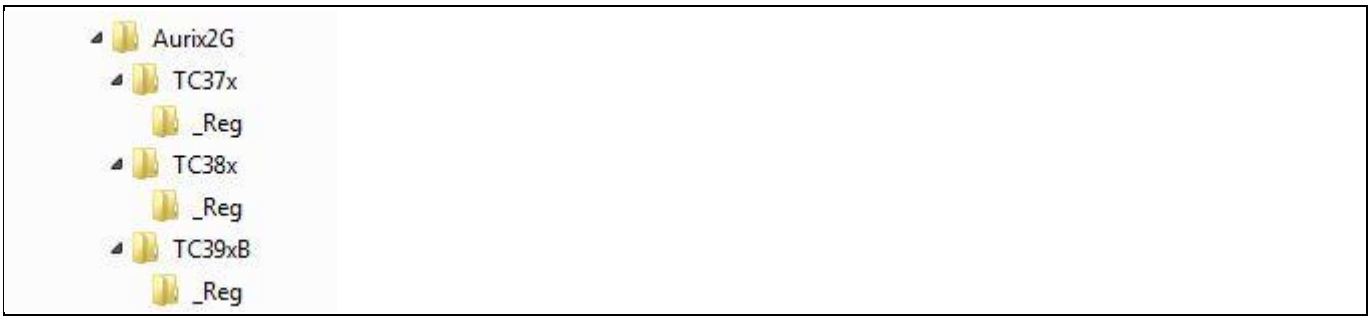
Note: Compile time is affected only for the individual application file.

2.4.3 Handling different derivatives

SFR header files are available as specific package for each microcontroller derivative. When a generic driver targeted for multiple microcontroller derivatives, it is important to take care that during build process correct set of files for the targeted microcontroller are included.

To handle this, the following is recommended:

1. Place `_Reg` folders in peripheral-specific folders as follows:



2. Include the register header files without their paths. For example,

Code Listing 1

```
001        #include IfxEvadc_reg.h
002        #include IfxScu_reg.h
```

3. For target-specific build, configure the include path to `./< Microcontroller target>/_Reg`.
   For example, for the target TC39xB controller compiler option should be: `-I./<path>/TC39xB/_Reg`.

3 Accessing individual registers

SFR headers provide an interface to access the hardware. Most of the cases require accessing the register as volatile access (volatile is compiler directive for defining a data to direct the compiler not to optimize the code, but for each read/write to SFR directly access the memory).

Such an access of registers for a peripheral draws different scenarios. These scenarios are explained in the subsequent sections.

3.1 Accessing the register targeting an individual bit-field

For a driver function if only one of the bit-fields need to be write-accessed, this can be achieved by directly assigning the value to the bit-field.

The syntax is as follows:

```
<SFR Name>.B.<Bit-field Name>= <value to be assigned>
```

For example, the driver is required to write to the EDIS bit of STMo_CLC.

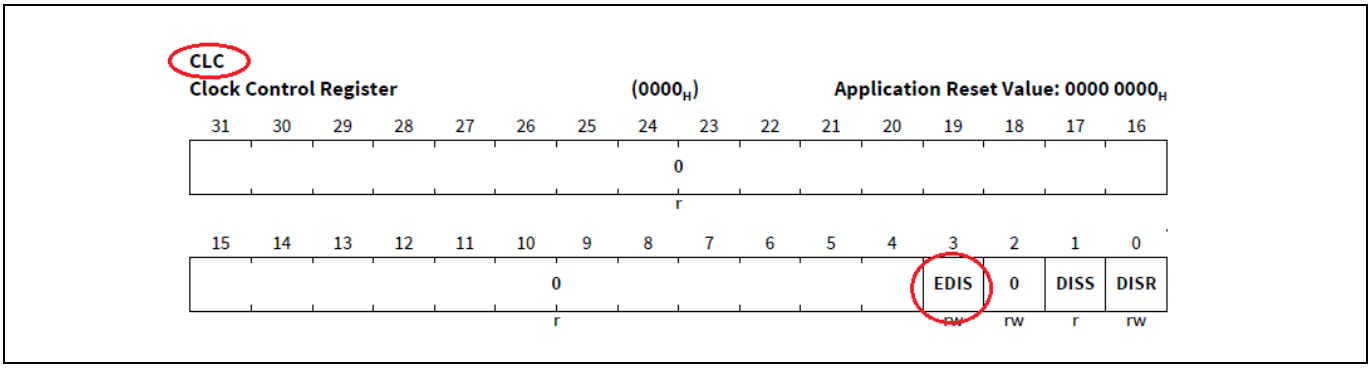


Figure 4 User Manual representation of STMo_CLC register

The code snippet is as follows:

Code Listing 2

```
001:  #include IfxStm_reg.h
002:
003:  /*Example function*/
004:  void myExample_accessReg_01(void)
005:  {
006:     STM0_CLC.B.EDIS= 1;
007:  }
```

3.2 Accessing the register targeting multiple bit-fields

For a driver function if multiple bit-fields need to be write-accessed, this can be achieved in many ways:

- Assigning the values to the all bit-fields individually
- Use the bit-field mask and position to make a read, modify and write operations

Both the approaches have disadvantages as the first approach is not optimized to access the SFR memory each time. The second approach does not produce a readable code.

To solve both the problems, the recommendations are as follows:

Create a local function variable of targeted register type. As type-define of target register is not with volatile keyword, all the bit-fields access will be local register accesses.

Note: The type-define without volatile keyword has this specific used case in mind. To access register as volatile access one must use the register name directly.

Assign values to individual bit-fields in the same way as writing to register bit-field, but with local variable.

After all the bit-fields are written, assign to the actual register accessing it as unsigned integer.

For example, the driver needs to write to SCU_CCUCON1 register bit fields.

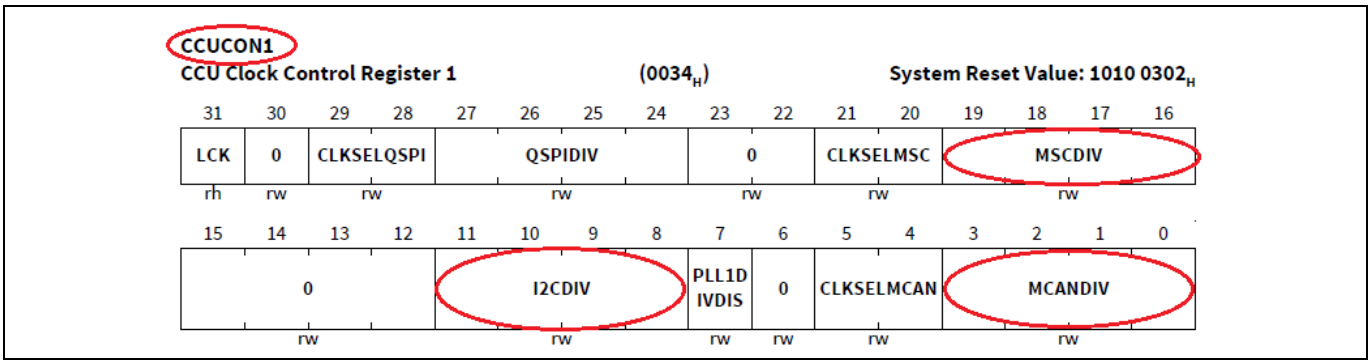


Figure 5 User Manual representation of SCU_CCUCON1 register

The code snippet is as follows:

Code Listing 3

```
008:  #include IfxScu_reg.h
009:
010:  /*Example function*/
011:  void myExample_accessReg_02(void)
012:  {
013:    Ifx_SCU_CCUCON1 myLocalReg;          /*nonvolatile definition*/
014:    myLocalReg.U = SCU_CCUCON1.U;        /*Read the register value
      locally*/
015:
016:    /*Write to local variable*/
017:    myLocalReg.B.MSCDIV= 1;
018:    myLocalReg.B.I2CDIV= 1;
019:    myLocalReg.B.MCANDIV= 1;
020:
        SCU_CCUCON1.U= myLocalReg.U;   /*volatile access*/
021:  }
```

3.3 Accessing the registers with its bit mask values

Registers could be accessed also with the bit mask values. This approach could be used for the following use cases

- Use-case required efficient access of the registers

- Target register has some reserved bitfields, which are to be written with non-zero values
- Use-case requires accessing the registers with atomic load-modify-store instructions

Note: The Ifx<peripheral name>_bf.h file should be individually and explicitly included, and they are not implicitly included by Ifx<peripheral name>_reg.h or Ifx_reg.h.

For the use-case 3, the following code snippet provides an example. This example accesses the same register with same values, as explained in section 3.2.

Code Listing 4

```
#include IfxScu_bf.h

/*Example function*/
void myExample_accessReg_03(void)
{
  uint32 myMask=IFX_SCU_CCUCON1_I2CDIV_MSK <<
IFX_SCU_CCUCON1_I2CDIV_OFF |
                      IFX_SCU_CCUCON1_MCANDIV_MSK <<
IFX_SCU_CCUCON1_MCANDIV_OFF |
                      IFX_SCU_CCUCON1_MSCDIV_MSK <<
IFX_SCU_CCUCON1_MSCDIV_OFF |

  uint32 myVal= 1 << IFX_SCU_CCUCON1_I2CDIV_OFF |
               1 << IFX_SCU_CCUCON1_MCANDIV_OFF |
               1 << IFX_SCU_CCUCON1_MSCDIV_OFF;

  __ldmst((void *)(&SCU_CCUCON1), myMask, myVal);
}
```

4 Accessing register through grouped structures

Linear memory map of the hardware registers can be translated to logical groups with the same hierarchy as they appear in the hardware. Such grouping is represented in C as structures instances and array of either (such) instances or array of individual registers themselves. This grouping eases the development of drivers without errors, which results in optimized code.

This kind of usage is explained with the following example.

Accessing multiple registers of the STM module and the instance is configurable. Here, the module instance can be passed as a pointer of type of that module.

Code Listing 5

```
#include IfxStm_reg.h

/*Example function*/
void myExample_initStmReg_01(Ifx_STM *stm)
{
  stm->CMP[0].B.CMPVAL= 0xFFFF; /*Update compare value*/
  stm->ICR.B.CMP0EN= 1; /*Enable compare*/
}
```

The same approach can be used for modules that have logical grouping for module, sub-modules and sub-sub modules and so on.

As another example, the group can also be used to iterate through the array of logical instance such as CAN nodes.

5 Specific hints

Following are some of the hints when register header files are used for safety applications:

- Include only the needed register header files for the target peripheral.
- Do not rely only on the comments against the registers/ bit-fields to understand the behavior of the hardware. The comments are currently not verifiable. Instead, refer to the user manual of the target hardware.
- Before accessing the register, user must check if the register or bitfield is implemented in the used instance of the target peripheral. The structures defined normally contain the superset of registers / bitfields. For example, `IOCR8` registers are not available for `P10`. However, if the user tries to access `IOCR8` register with `MODULE_P10.IOCR8.U`, then `IOCR8` register may become accessible by the software.
- SFR Headers are tightly linked to the hardware user manual, based on which it is generated, as detailed in section 2.1. Any Errata corresponding to the hardware user manual may have impact on the register or bit field definitions. Users must take due care while using such registers.

5.1 Dos and Don'ts

Following are the list of Dos and Don'ts.

Table 3

Dos	Don'ts
Include `Ifx<per>_reg.h`	Do not include `Ifx<per>_regdef.h` This is an internal file. The File name/ internal include could change.
Use Register Data Types only for local variables	Do not define your own registers using register typedefs. Portability is affected as the memory map changes derivative to derivative. The register typedefs are not defined with volatile keyword explicitly.
Use local variables to write multiple bitfields (if it is allowed from targeted function)	Do not use volatile write unnecessarily It is un-optimized
Use register groups with `MODULE_<per>` structure where it is necessary	Do not define your own groups Portability is affected as the memory map changes derivative to derivative.
Be choosy to include only required header file	Do not include `Ifx_reg.h` file or do not include an unnecessary header file. It affects the compilation time
To use the SFR header for accessing individual registers from module pointer : Access by pointing to the base of the particular instance rather than the first instance of any IP. This is because, not all IP's are contiguous in	-

Specific hints

Dos	Don'ts

memory.
Example, to write `0x01` to CLC in module ASCLIN instance 2:
`Ifx_ASCLIN *asc_ptr = &MODULE_ASCLIN2;`
`/* point to the specific instance */`
`asc_ptr->CLC.U = 0x01u;`	

6 Assumptions of Use (AoU)

SFR header files are intended to be used in safety related applications. SFR headers are developed based on the following assumptions of use. It is the responsibility of the user to take care that these assumptions are handled with due care, while using them.

- When a register bitfield is accessed implicitly or explicitly, user must be aware that accessed bitfields are either
  - Available at the hardware target
  - If not available, they are written with right values as defined by the hardware target user manual.
- SFR header files do not implement the hardware errata and hardware documentation errata. Any applicable errata should be respected by the user while using the affected registers.

7 PC Lint + MISRA

This section explains the deviations for register header files against the MISRA. For SFR headers, MISRA 2012 compliance is checked with the PC Lint tool.

The warnings are analyzed for register header files and deviations are noted in a compiled report. User must be aware of such deviations and verify if the justifications are correct from the project perspective. The lint reports are available at the following locations:

- `<install folder>/SupportDocuments/PcLint_Messages.txt` (raw file for messages)
- `<install folder>/SupportDocuments/PCLint_Report.xls` (compiled report with justifications)

Table 4 Known deviations

PC Lint No.	MISRA No.	Generic violation message	Justification
46	6.1	Field type should be int	Such message is for following #define macro Ifx_Strict_16Bit <bit field name>:<bit position>; The bit-fields of Aurix Mc registers are 16 bit width where as Aurix controllers are 32 bit controllers By defining "int" in place of "Ifx_Strict_16Bit" will make the register offset different than that of HW. Such definitions are allowed with compilers, which support Aurix microcontrollers.
537		Repeated include file 'File'	Each of the Ifx<peripheral name>_reg.h includes the Ifx_TypesReg.h Ifx_TypesReg.h is multiple include protected and there is no harm to the user program due to this
621	5.1	Identifier clash (Symbol 'Name' with Symbol 'Name' at String)	Compilers which support Aurix have no limitation of 31 characters
658		Anonymous union assumed	Anonymous unions are defined for the multi-view registers. As these multi-view registers are different registers but point to same address in the register memory map for the better readability of the code where these registers are used, this is defined this way Aurix compilers support such feature
9058	2.4	Unused outside of typedefs	These Tags are meant for C++ use of the SFR files The SFR generator tool and the SFR validation tool makes sure that each tag has a unique name This warning can be ignored
766		Not used in module	These header files contain only the includes of SFR headers They do not contain any macro, typedef, struct, union or enum tag or component, or declaration This warning can be ignored

Revision history

Major changes since the last revision

Date	Version	Description
14-01-2019	5.0	Updates: • Reference document : UM/TS version to be referenced by user from Release notes
27-09-2018	4.0	Updated for Release as per comments
19-04-2018	3.0	Updated for Release as per comments
15-03-2018	2.0	Formatting updates done as per the document review
12-12-2017	1.0	Initial version

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

Edition 2019-01-14
Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
Z8F62138655

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**).

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.