HIGHTEC

AURIX™ Development Studio

Guide for HighTec Toolchains

Version 1.1, October 2023:

Table of Contents

Introduction

1. Introduction

The following application note demonstrates how to configure and build AURIX™ Development Studio projects with Hightec toolchains.

2. Prerequisites

Prerequisite	Version
Development tool	AURIX™ Development Studio Limited 1.9.11-L (Limited version for TC4xx) or later
Infineon Low Level Drivers package	iLLD_TC4xx_2_0_1_2_19_Package
HighTec LLVM toolchain for TriCore™ applications	TriCore™ Development Platform version 8.0.0 or higher

Tab. 1. Prerequisites LLVM Toolchain

Prerequisite	Version
Development tool	AURIX™ Development Studio 1.9.16 or later
HighTec GCC toolchain for TriCore™ applications	TriCore™ Development Platform version 4.9.4.1

Tab. 2. Prerequisites GCC Toolchain

AURIX™ Development Studio

3. AURIX™ Development Studio

3.1. Download

The AURIX™ Development Studio can be downloaded from Infineon's pages using the link below.

https://www.infineon.com/cms/en/product/promopages/aurix-development-studio

It is available for the Windows platform as an executable installation file.

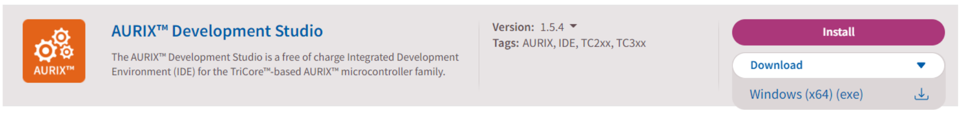
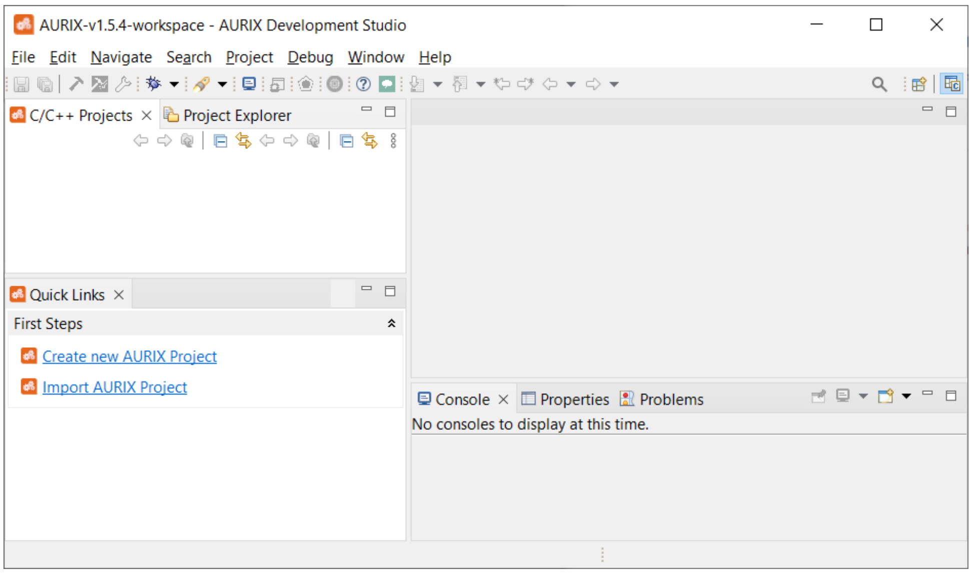

AURIX™
AURIX™ Development Studio
The AURIX™ Development Studio is a free of charge Integrated Development Environment (IDE) for the TriCore™-based AURIX™ microcontroller family.
Version: 1.5.4
Tags: AURIX, IDE, TC2xx, TC3xx
Install
Download
Windows (x64) (exe)

Fig. 1. Download of AURIX™ Development Studio

3.2. Installation and run

The process of installation is straightforward. When it finishes, the main screen appears.



No consoles to display at this time.

Fig. 2. Initial screen of the AURIX™ Development Studio

HighTec configurations in ADS

4. External LLVM for TriCore™ applications

Please ensure that you make all the necessary changes to the project before running it. Running the project without implementing all the changes can result in a corrupt build and may require starting the entire project from scratch. To avoid any issues, thoroughly review and implement all the proposed modifications before proceeding with the execution of the project. Please pay special attention to the **C/C++ Indexer**, see further down for more information.

4.1. Import existing project

To test if the LLVM HighTec compiler works correctly, let's import one of the existing Infineon projects.

Select **File→Import→Infineon→AURIX™ Development Studio Project** and click **`[Next]`**.

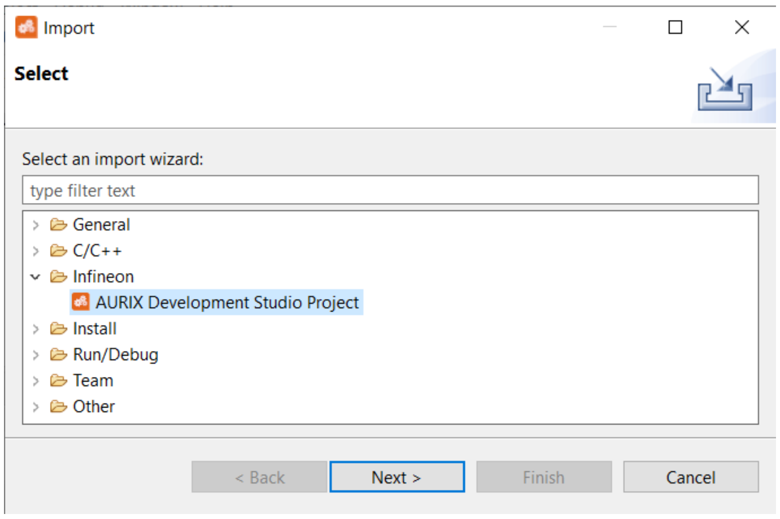
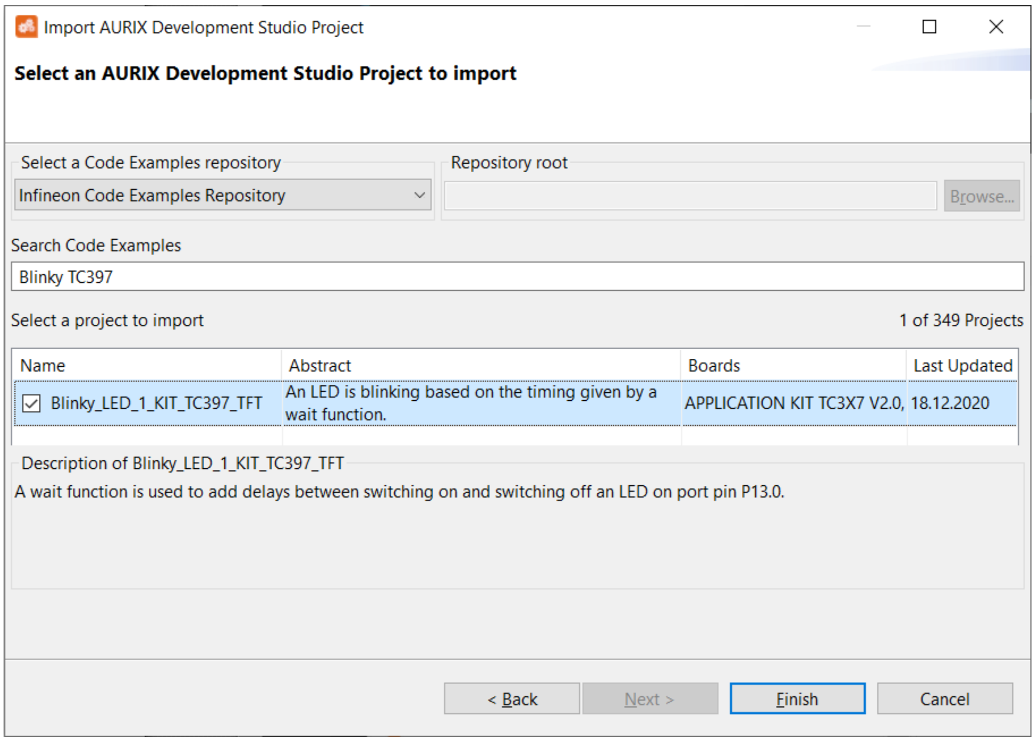


Fig. 3. Import Infineon example

Then select **Infineon TC4xx Code Examples Repository**, select an example of your choice (we will test the "blinky" example for STD Kit TC499) and click **`[Finish]`**.


Import AURIX Development Studio Project

Select an AURIX Development Studio Project to import

Select a Code Examples repository: Infineon Code Examples Repository

Repository root:

Search Code Examples: Blinky TC397

Select a project to import — 1 of 349 Projects

Name	Abstract	Boards	Last Updated
☑ Blinky_LED_1_KIT_TC397_TFT	An LED is blinking based on the timing given by a wait function.	APPLICATION KIT TC3X7 V2.0,	18.12.2020

Description of Blinky_LED_1_KIT_TC397_TFT

A wait function is used to add delays between switching on and switching off an LED on port pin P13.0.

Fig. 4. Select an example from the list (search is used as a filter)

4.2. Update iLLD

Update project libraries to the latest version and back up the current. Right-click on the project, select **Project updater→Update iLLD** and click **`[Finish]`**.

When the iLLD is updated, ADS executes the **C/C++ Indexer**. It needs to be finished before starting the build. Otherwise, there is a significant risk of the build failing.

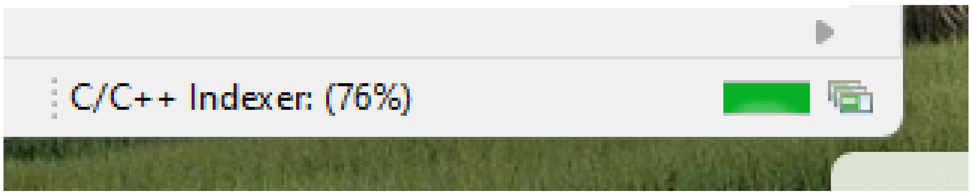


Fig. 5. C/C++ Indexer execution in the lower-right corner

4.3. Manage build configuration

To create a new HighTec LLVM toolchain configuration, right-click on the project and select **Build Configuration→Manage**.

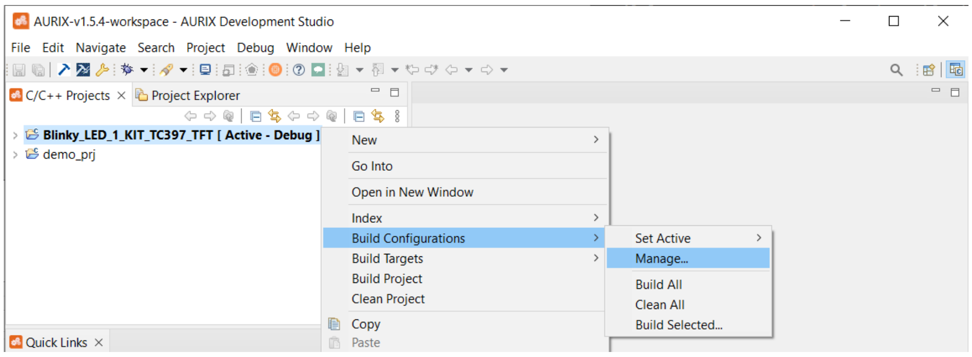
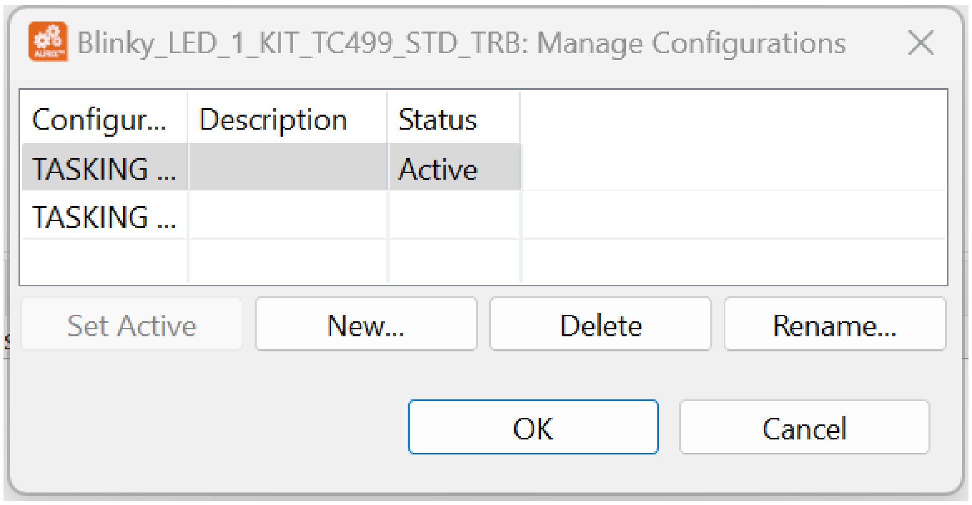


Fig. 6. Setting the manage build configuration

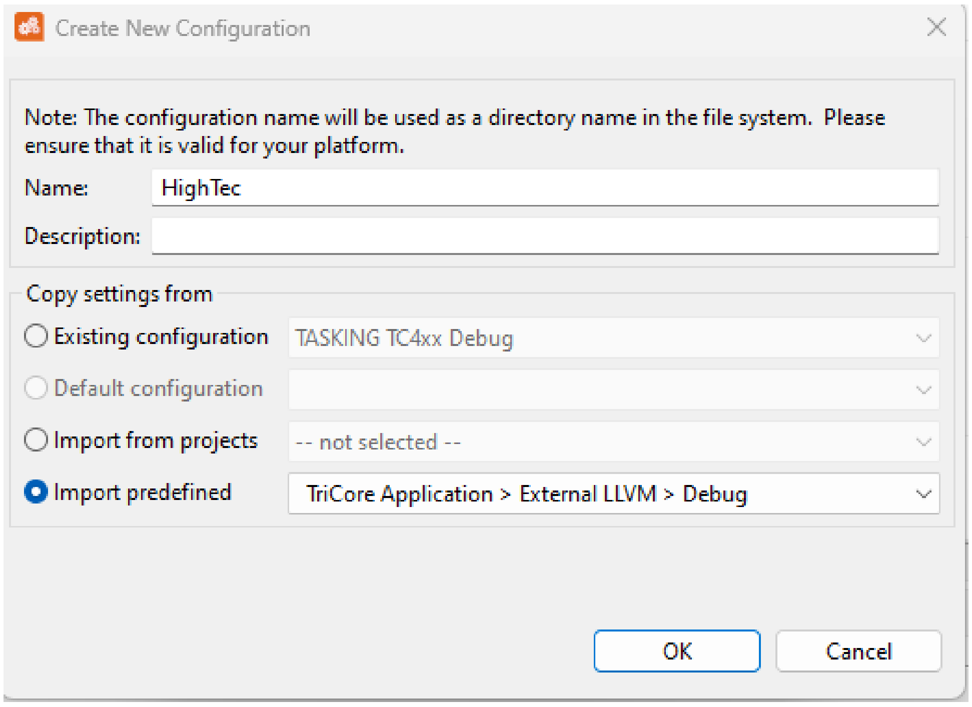
The Manage Configurations tab will be opened.



Configur...	Description	Status
TASKING ...		Active
TASKING ...		

Fig. 7. manage configuration tab

Select **[New]** and give a name to the new configuration. Then select **Import predefined**, choose **TriCore Application→External LLVM→Debug** and click **[OK]**.



Create New Configuration

Note: The configuration name will be used as a directory name in the file system. Please ensure that it is valid for your platform.

Name: HighTec

Description:

Copy settings from

- ( ) Existing configuration: TASKING TC4xx Debug
- ( ) Default configuration
- ( ) Import from projects: -- not selected --
- (●) Import predefined: TriCore Application > External LLVM > Debug

OK | Cancel

Fig. 8. new configuration

4.4. Activate build configuration

To select a new active configuration, right-click on the project and select **Build Configuration→Set Active →HighTec (name of the new configuration created)**.

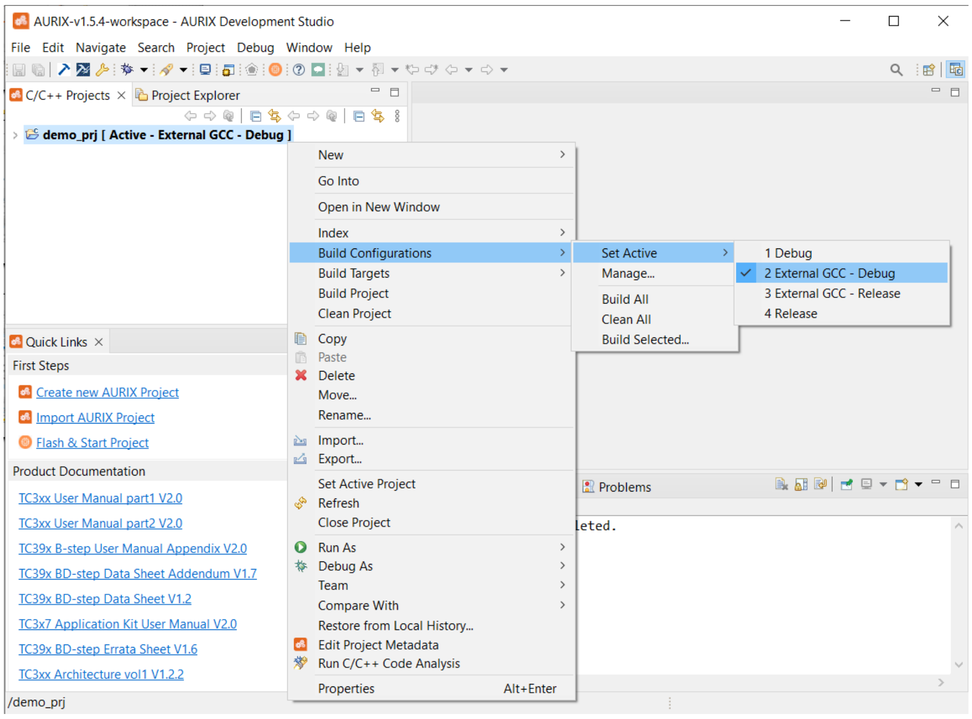


Fig. 9. Setting the active build configuration

4.5. Import linker script

After the iLLD update, the project linker script is replaced by a default one for the iLLD base project. If the linker script is not present after creating a new configuration, it is possible to get it from the latest version of the iLLD package and manually copy it into the project:

Target	Latest iLLD
TC3xx	iLLD_TC3xx_V1_0_1_17_0 _Package
TC4xx	iLLD_TC4xx_2_0_1_2_19_Package

Infineon Low-Level Drivers are available upon request from Infineon and contain a standardized linker script for LLVM toolchains.

The base TC49x linker script **Lcf_Hightec_Tricore_Tc.lsl**, and specialized linker scripts for the PPU and DMA examples, can also be downloaded via this link: HighTec Linker Scripts.

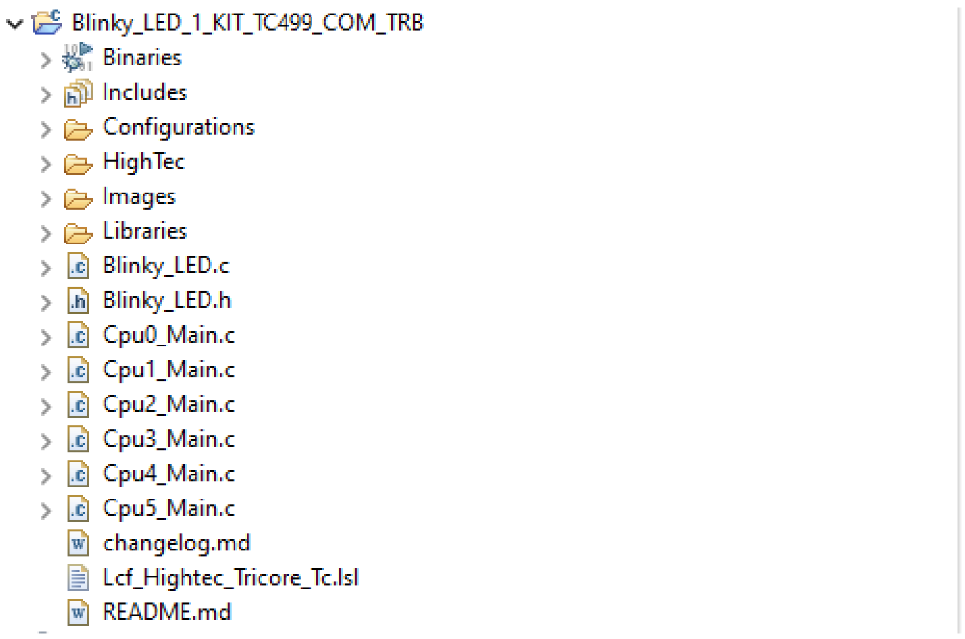

Blinky_LED_1_KIT_TC499_COM_TRB
Binaries
Includes
Configurations
HighTec
Images
Libraries
Blinky_LED.c
Blinky_LED.h
Cpu0_Main.c
Cpu1_Main.c
Cpu2_Main.c
Cpu3_Main.c
Cpu4_Main.c
Cpu5_Main.c
changelog.md
Lcf_Hightec_Tricore_Tc.lsl
README.md

Fig. 10. Imported Linker File

4.6. Build settings

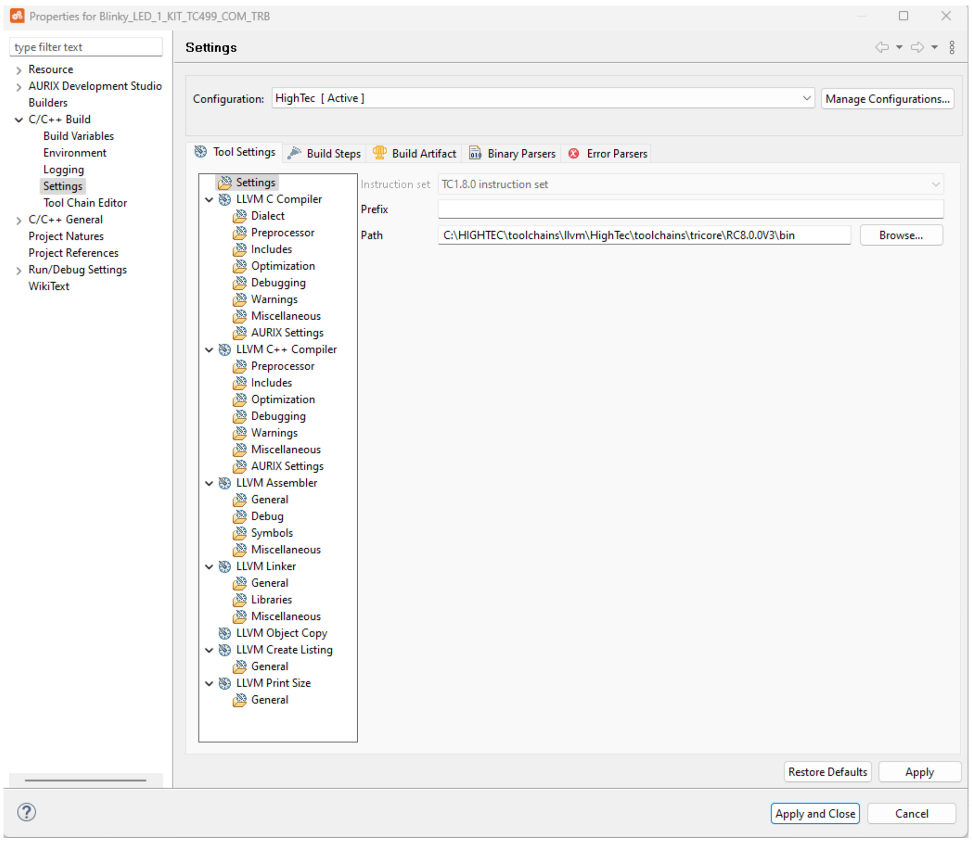
The build settings must be updated for the active build configuration. To access the settings, right-click on the project and select **Properties**→**C/C++ Build**→**Settings**.

4.6.1. Apply toolchain

Under the **Tool Settings**→**Settings** tab, change the Prefix and Path fields:

- The Prefix is always "empty" for the LLVM toolchain.
- The Path must be set to the "bin" folder of the LLVM toolchain.

After the change is done, click **`[Apply and Close]`**.



Properties for Blinky_LED_1_KIT_TC499_COM_TRB

Settings

Configuration: HighTec [Active]

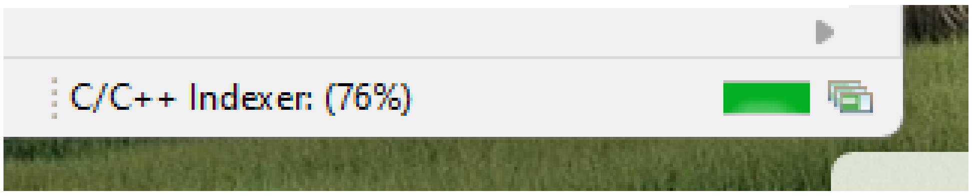
Instruction set: TC1.8.0 instruction set

Prefix:

Path: C:\HIGHTEC\toolchains\llvm\HighTec\toolchains\tricore\RC8.0.0V3\bin

Fig. 11. Setting the prefix and path to the HighTec LLVM compiler

When the new toolchain is set, ADS executes the **C/C++ Indexer**. It needs to be finished before starting the build. Otherwise, there is a considerable risk of the build failing.



C/C++ Indexer: (76%)

Fig. 12. C/C++ Indexer execution in the lower-right corner

4.6.2. Clang driver options

The C/C++ compiler driver clang is a complete control program for a large part of the toolchain. It can orchestrate the entire build of an executable with a single command-line invocation.

In ADS, the compiling and linking steps are done separately. For this reason, the following options must be used in both: **LLVM C Compiler options** and **LLVM Linker options**.

Architecture: **`-march=<arch>`**

In compilation, it affects the available instructions to emit. In linking, it affects the selection of target libraries. Allowed values are:

<arch>	Device
`tc161`	AURIX™ platform 1st generation (TC2xx)
`tc162`	AURIX™ platform 2nd generation (TC3xx)
`tc18`	AURIX™ platform 3rd generation (TC4xx)

For compilation, set the field **`-march`** in **LLVM Compiler→AURIX Settings**, i.e., **`tc18`**.
For the linking, set the field **`-march`** in **LLVM Linker→General**, i.e., **`tc18`**.

Errata: **`-merrata=<bug>`**

Some derivatives contain silicon bugs, also known as errata. In such cases, the workaround has to be applied to avoid triggering them. This option also affects the selection of target libraries during the linking. Allowed values:

<bug>	Description
`cpu141`	*[CPU_TC.141] Instructions not implemented* in TC49A derivative [1]

Please refer to the *Errata* chapter in [2] for more details.

For compilation, extend the Other flags field in **LLVM Compiler→Miscellaneous**, i.e., **`-merrata=cpu141`**.
For the linking, extend the Command field in **LLVM Linker**, i.e., **`clang -merrata=cpu141`**.

Floating point strategy: **`mfloat-abi=<float-abi>`**

Select the floating point handling strategy for code generation: *software* function calls or *hardware* FPU instructions. The <float-abi> keyword has the following pattern:
<float-strategy><double-strategy><size-of-double> , i.e., **`hs64`**.
The letters represent the handling strategy for **`float`** and **`double`** floating-point types. The number is the size of the **`double`** type in bits. The possible combinations can be selected from the following table according to the **`-march=<arch>`**

<float-abi>	`float` strategy	`double` strategy	`double` size	Supported archs	Default for
`ss32`	function calls	function calls	32	**`tc18`**	
`hh32`	FPU instructions	FPU instructions	32	**`tc161`**, **`tc162`**, **`tc18`**	
`ss64`	function calls	function calls	64	**`tc18`**	
`hs64`	FPU instructions	function calls	64	**`tc161`**, **`tc162`**, **`tc18`**	**`tc161`**, **`tc162`**
`hh64`	FPU instructions	FPU instructions	64	**`tc18`**	**`tc18`**

For more details, please refer to the *Multilib variants* chapter in [2].

For compilation, extend the Other flags field in **LLVM Compiler→Miscellaneous**, i.e., **`-mfloat -abi=hh64`**
For the linking, extend the Command field in **LLVM Linker**, i.e., **`clang -mfloat-abi=hh64`**

Exceptions: **`-f[no-]exceptions`**
Enable or disable the support for C++ exception handling. This flag also controls which libraries are linked. By default, exceptions are enabled.

For compilation, extend the Other flags field in **LLVM Compiler→Miscellaneous**, i.e., **`-fno-exceptions`**.
For the linking, extend the Command field in **LLVM Linker**, i.e., **`clang -fexceptions`**

The exhaustive list of the compiler driver options can be found in the *Compiler Driver* chapter in [2].

4.6.3. Compiler-specific options

- The Command is pre-set to **`clang`** in **LLVM C Compiler**.

Optimizations: **`-O<level>`**
The compiler accepts the following optimization options: **`-O0`**, **`-O1`**, **`-O2`**, **`-O3`**, **`-Ofast`**, **`-Os`**, **`-Oz`**, **`-Og`**

Choose from the offered options or set the Other optimization flags field in **LLVM Compiler →Optimization**. In the second case, there are two optimization options in the build command, and the compiler will use the last one in the sequence, i.e., in **`clang -O0 -Ofast`**, **`-Ofast`** will be used.

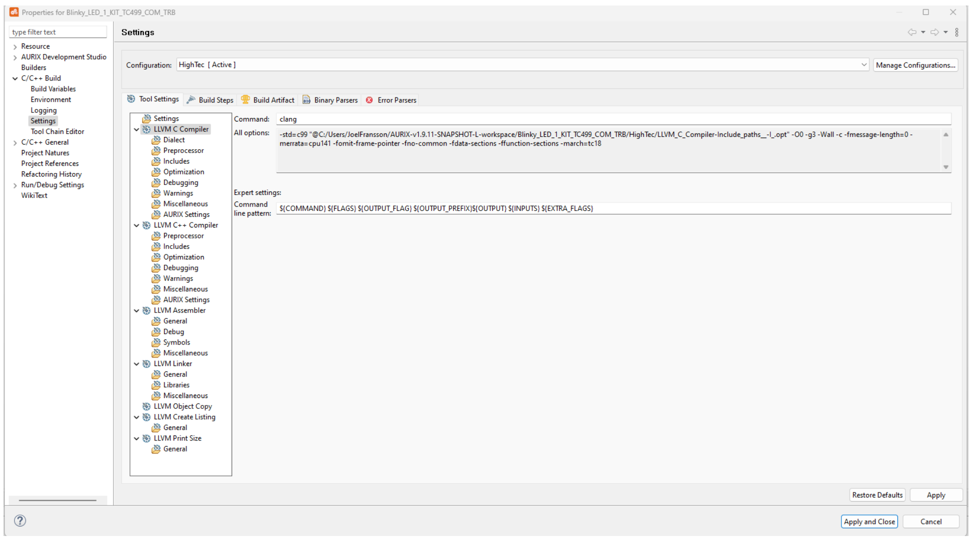

Command: clang
All options: -std=c99 "@C:/Users/JoelFransson/AURIX-v1.9.11-SNAPSHOT-L-workspace/Blinky_LED_1_KIT_TC499_COM_TRB/HighTec/LLVM_C_Compiler-Include_paths__-I_.opt" -O0 -g3 -Wall -c -fmessage-length=0 -merrata=cpu141 -fomit-frame-pointer -fno-common -fdata-sections -ffunction-sections -march=tc18
Expert settings:
Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS} ${EXTRA_FLAGS}

Fig. 13. AURIX™ LLVM C Compiler Options

Certain options, like **`-fstrict-volatile-bitfields`** in **LLVM C Compiler→AURIX**, are not supported by the LLVM toolchain. These options must be unchecked for the successful compilation of the project. The exhaustive list of the compiler options can be found in the *Compiler* chapter of [2].

4.6.4. Linker-specific options

- The Command is pre-set to **`clang`** in **LLVM Linker**.
- Although clang understands and forwards most of the common linker options, some need to be passed directly to the linker. To pass an option to the linker, you should prefix the option with **`-Xlinker`** or **`-Wl,`**.

For example, option **`--gc-sections`** should be passed to the linker as **`-Xlinker --gc-sections`** or **`-Wl, --gc-sections`**.

Linker script: **`-T<linker-script>`**

We use the linker script containing the *Hightec* keyword. The information on how to select and import it is described in chapter Linker script.

Set the field Linker Script in **LLVM Linker→General**, i.e., **`../Lcf_Hightec_Tricore_Tc.lsl`**.

Libraries

Internal libraries

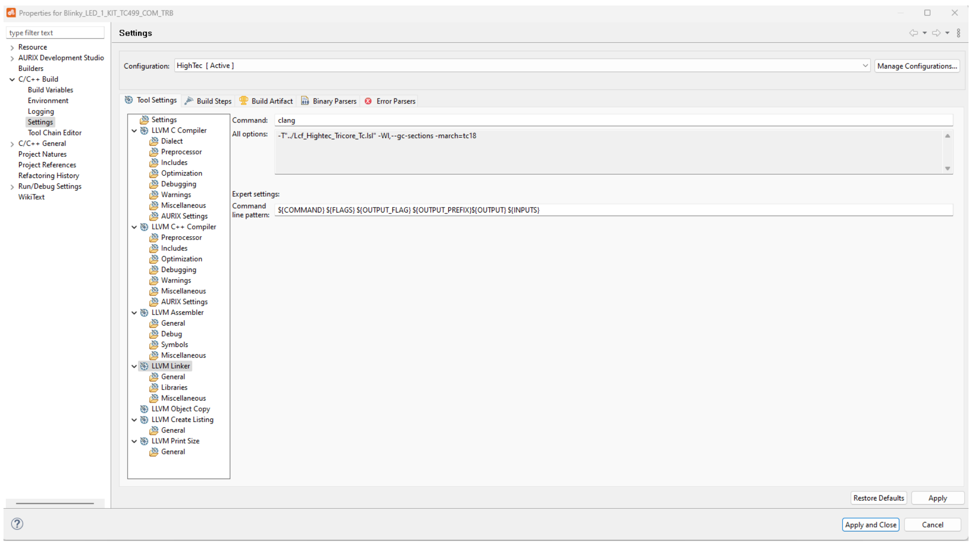
When using the clang compiler driver, the library paths are chosen based on the **`-march`**, **`-merrata`**, **`-mfloat-abi`**, and **`-f[no-]exceptions`** options. The libraries like C standard library, or C++ library, are automatically linked from these paths. Some functions from the C standard library reference the file IO functions. To prevent the linker error: **`ld.lld: error: undefined symbol`**, the **`libhtcos.a`** library has to be linked as well.

Add the option **`-lhtcos`** in **LLVM Linker→Miscellaneous→Add**.

External libraries

When linking external libraries, it is necessary to pass the library name with **`-l`** prefix: **`-l<lib>`** as well as the library path with **`-L`** prefix: **`-L<libpath>`**. To link, e.g., the "C:\tricore\libs\libbsp.a" library, the following parameters have to be passed to the linker: **`-lbsp`** and **`-LC:\tricore\libs`**.

Add options **`-l<lib>`** and **`-L<libpath>`** in **LLVM Linker→Miscellaneous→Add** to link an external library.



Configuration: HighTec [Active]

Command: clang

All options: -T"../Lcf_Hightec_Tricore_Tc.lsl" -Wl,--gc-sections -march=tc18

Expert settings:

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}

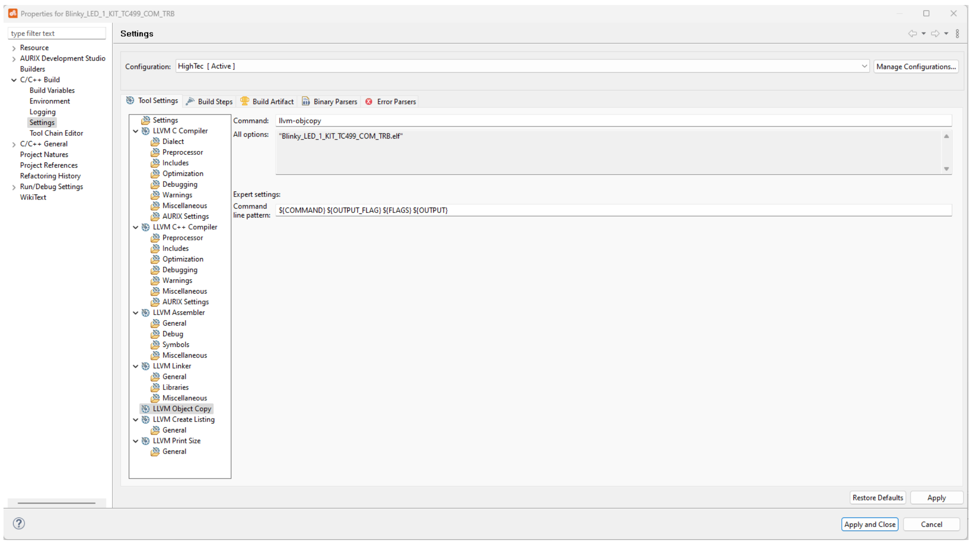
Fig. 14. LLVM Linker Options

Certain options like **`-nostartfiles`** in **LLVM Linker→General**, are not supported by the LLVM toolchain. These options must be unchecked for the successful linking of the project. An exhaustive list of linker options can be found in the *Linker* chapter of [2].

4.6.5. LLVM Object Copy options

- **`llvm-objcopy`** is pre-set.

An exhaustive list of Binutils options can be found in the *Binutils* chapter of [2].


Properties for Blinky_LED_1_KIT_TC499_COM_TRB

Settings

Configuration: HighTec [Active]

Command: llvm-objcopy

All options: "Blinky_LED_1_KIT_TC499_COM_TRB.elf"

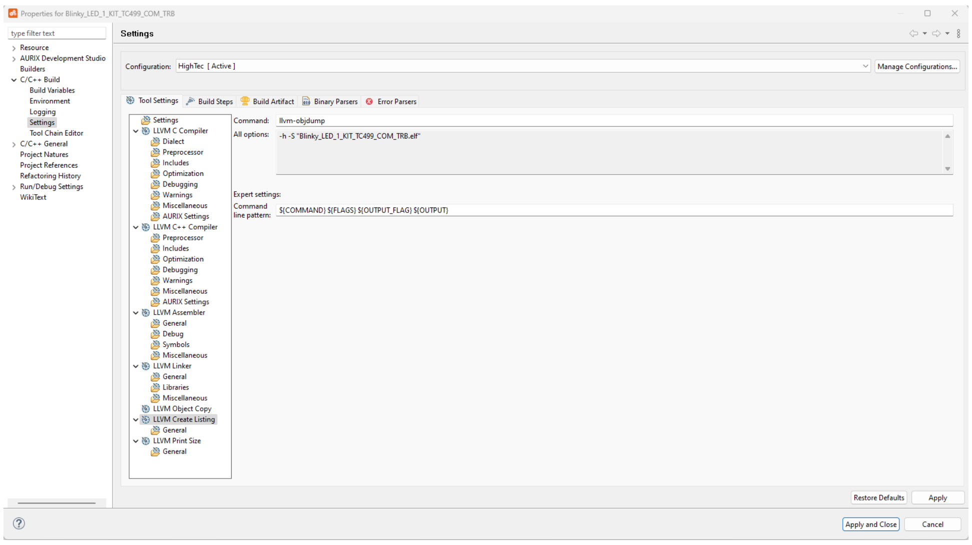
Expert settings:

Command line pattern: ${COMMAND} ${OUTPUT_FLAG} ${FLAGS} ${OUTPUT}

Fig. 15. LLVM Object Copy Options

4.6.6. LLVM Create Listing options

- **`llvm-objdump`** is pre-set.


Properties for Blinky_LED_1_KIT_TC499_COM_TRB

Settings

Configuration: HighTec [Active]

Command: llvm-objdump

All options: -h -S "Blinky_LED_1_KIT_TC499_COM_TRB.elf"

Expert settings:

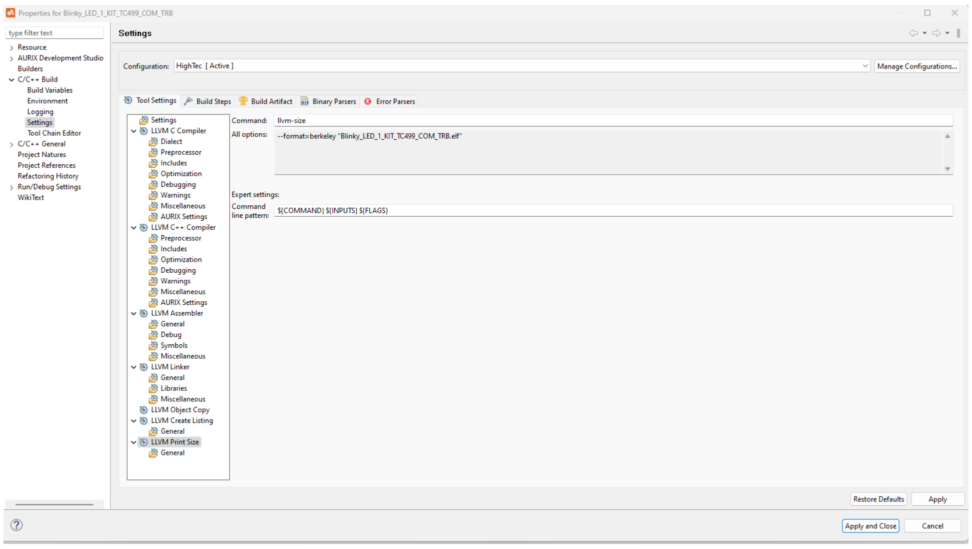
Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT}

Fig. 16. LLVM Listing Options

An exhaustive list of Binutils options can be found in the *Binutils* chapter of [2].

4.6.7. LLVM Print Size options

- **`llvm-size`** is pre-set.



Settings

Configuration: HighTec [Active]

Command: llvm-size

All options: --format=berkeley "Blinky_LED_1_KIT_TC499_COM_TRB.elf"

Expert settings:

Command line pattern: ${COMMAND} ${INPUTS} ${FLAGS}

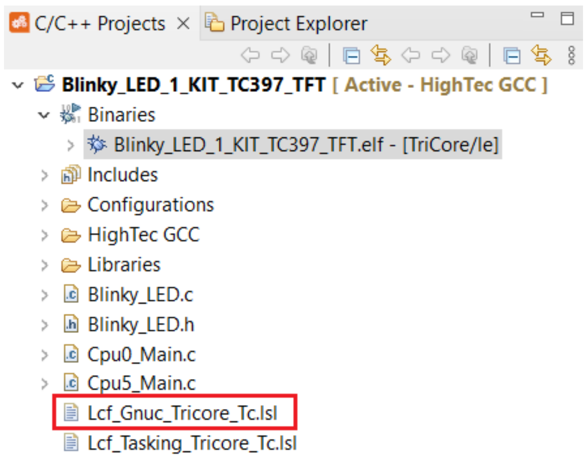
Fig. 17. LLVM Print Size Options

An exhaustive list of Binutils options can be found in the *Binutils* chapter of [2].

4.7. Build the project

Now, the project will use the HighTec LLVM compiler to build the application.

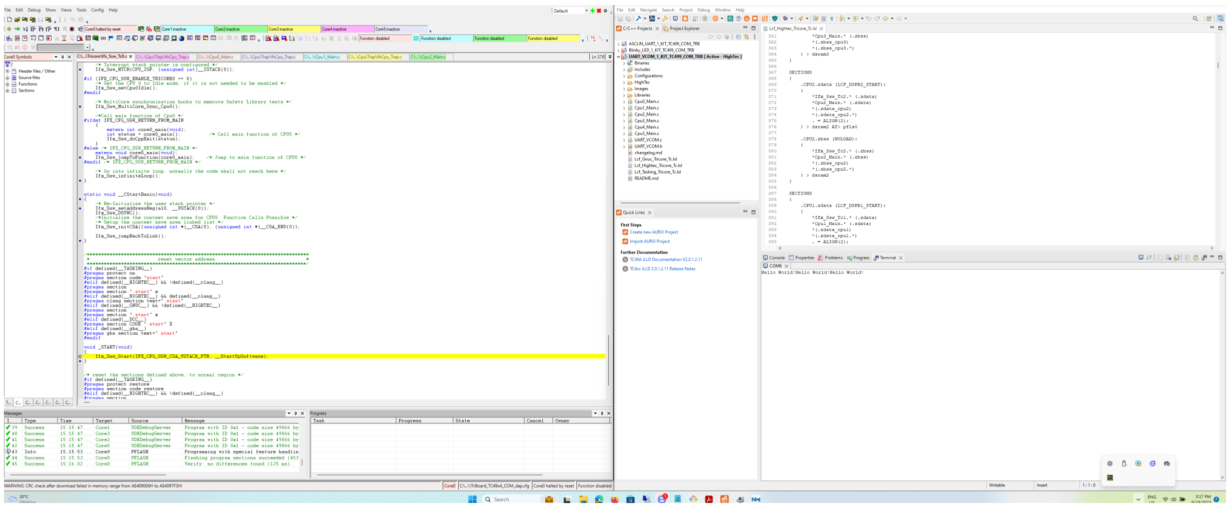
The final project content after a successful build is shown in the below figure (the copied linker file and the generated **`.elf`** file).



```
Blinky_LED_1_KIT_TC397_TFT [ Active - HighTec GCC ]
  Binaries
    Blinky_LED_1_KIT_TC397_TFT.elf - [TriCore/le]
  Includes
  Configurations
  HighTec GCC
  Libraries
  Blinky_LED.c
  Blinky_LED.h
  Cpu0_Main.c
  Cpu5_Main.c
  Lcf_Gnuc_Tricore_Tc.lsl
  Lcf_Tasking_Tricore_Tc.lsl
```

Fig. 18. Content of the imported Infineon project after a successful build

A complete project, run on a board.



```
    /* Interrupt stack pointer is configured */
    Ifx_Ssw_MTCR(CPU_ISP, (unsigned int)__ISTACK(0));

#if (IFX_CFG_SSW_ENABLE_TRICORE0 == 0)
    /* Set the CPU 0 to Idle mode, if it is not needed to be enabled */
    Ifx_Ssw_setCpu0Idle();
#endif

    /* MultiCore synchronization hooks to execute Safety Library tests */
    Ifx_Ssw_MultiCore_Sync_Cpu0();

    /*Call main function of Cpu0 */
#ifdef IFX_CFG_SSW_RETURN_FROM_MAIN
    {
        extern int core0_main(void);
        int status = core0_main();          /* Call main function of CPU0 */
        Ifx_Ssw_doCppExit(status);
    }
#else /* IFX_CFG_SSW_RETURN_FROM_MAIN */
    extern void core0_main(void);
    Ifx_Ssw_jumpToFunction(core0_main);    /* Jump to main function of CPU0 */
#endif /* IFX_CFG_SSW_RETURN_FROM_MAIN */

    /* Go into infinite loop, normally the code shall not reach here */
    Ifx_Ssw_infiniteLoop();
}


static void __CStartBasic(void)
{
    /* Re-Initialize the user stack pointer */
    Ifx_Ssw_setAddressReg(a10, __USTACK(0));
    Ifx_Ssw_DSYNC();
    /*Initialize the context save area for CPU0. Function Calls Possible */
    /* Setup the context save area linked list */
    Ifx_Ssw_initCSA((unsigned int *)__CSA(0), (unsigned int *)__CSA_END(0));

    Ifx_Ssw_jumpBackToLink();
}


/******************************************************************************
 *                          reset vector address                              *
 ******************************************************************************/
#if defined(__TASKING__)
#pragma protect on
#pragma section code "start"
#elif defined(__HIGHTEC__) && !defined(__clang__)
#pragma section
#pragma section ".start" x
#elif defined(__HIGHTEC__) && defined(__clang__)
#pragma clang section text=".start"
#elif defined(__GNUC__) && !defined(__HIGHTEC__)
#pragma section
#pragma section ".start" x
#elif defined(__DCC__)
#pragma section CODE ".start" X
#elif defined(__ghs__)
#pragma ghs section text=".start"
#endif

void _START(void)
{
    Ifx_Ssw_Start(IFX_CFG_SSW_CSA_USTACK_PTR, __StartUpSoftware);
}


/* reset the sections defined above, to normal region */
#if defined(__TASKING__)
#pragma protect restore
#pragma section code restore
#elif defined(__HIGHTEC__) && !defined(__clang__)
#pragma section
```

```
I...  Type     Time         Target  Source           Message
 39   Success  15:15:47...  Core1   UDEDebugServer   Program with ID 0x1 - code size 49866 by
 40   Success  15:15:47...  Core3   UDEDebugServer   Program with ID 0x1 - code size 49866 by
 41   Success  15:15:47...  Core2   UDEDebugServer   Program with ID 0x1 - code size 49866 by
 42   Success  15:15:47...  Core5   UDEDebugServer   Program with ID 0x1 - code size 49866 by
 43   Info     15:15:53...  Core0   PFLASH           Programming with special feature handlin
 44   Success  15:15:53...  Core0   PFLASH           Flashing program sections succeeded (453
 45   Success  15:16:02...  Core0   PFLASH           Verify: no differences found (125 ms).
```

WARNING: CRC check after download failed in memory range from AE409000H to AE4097F3H!

```
                *Cpu3_Main.* (.zbss)
                *(.zbss_cpu3)
                *(.zbss_cpu3.*)
            } > dsram3
        }

        SECTIONS
        {
            .CPU2.zdata (LCF_DSPR2_START):
            {
                *Ifx_Ssw_Tc2.* (.zdata)
                *Cpu2_Main.* (.zdata)
                *(.zdata_cpu2)
                *(.zdata_cpu2.*)
                . = ALIGN(2);
            } > dsram2 AT> pfls0

            .CPU2.zbss (NOLOAD):
            {
                *Ifx_Ssw_Tc2.* (.zbss)
                *Cpu2_Main.* (.zbss)
                *(.zbss_cpu2)
                *(.zbss_cpu2.*)
            } > dsram2
        }

        SECTIONS
        {
            .CPU1.zdata (LCF_DSPR1_START):
            {
                *Ifx_Ssw_Tc1.* (.zdata)
                *Cpu1_Main.* (.zdata)
                *(.zdata_cpu1)
                *(.zdata_cpu1.*)
                . = ALIGN(2);
```

```
Hello World!Hello World!Hello World!
```

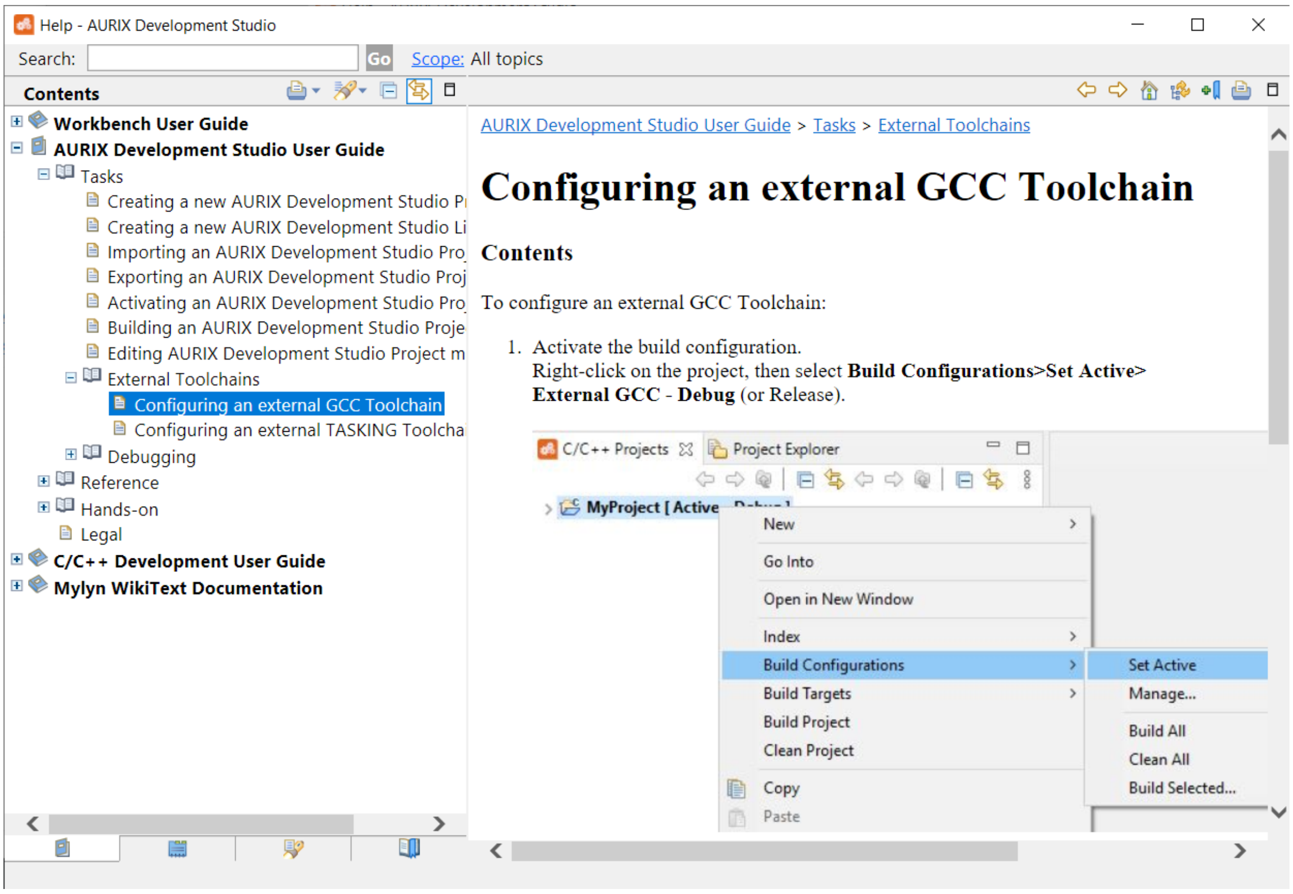
Fig. 19. UART_VCOM_1_ example run on board TC400_COM_TRB

5. External GCC for TriCore™ applications

5.1. Documentation

The configuration of an external GCC toolchain is described in the documentation of the AURIX Development Studio.

Choose **Help→Help Contents**, and navigate to **AURIX Development Studio User Guide→Tasks→External Toolchains→Configuring and external GCC Toolchain**.



Help - AURIX Development Studio
Search:
Go
Scope: All topics
Contents
Workbench User Guide
AURIX Development Studio User Guide
Tasks
Creating a new AURIX Development Studio P
Creating a new AURIX Development Studio Li
Importing an AURIX Development Studio Pro
Exporting an AURIX Development Studio Proj
Activating an AURIX Development Studio Pro
Building an AURIX Development Studio Proje
Editing AURIX Development Studio Project m
External Toolchains
Configuring an external GCC Toolchain
Configuring an external TASKING Toolcha
Debugging
Reference
Hands-on
Legal
C/C++ Development User Guide
Mylyn WikiText Documentation
AURIX Development Studio User Guide > Tasks > External Toolchains
C/C++ Projects
Project Explorer
MyProject [Active
New
Go Into
Open in New Window
Index
Build Configurations
Build Targets
Build Project
Clean Project
Copy
Paste
Set Active
Manage...
Build All
Clean All
Build Selected...

Configuring an external GCC Toolchain

Contents

To configure an external GCC Toolchain:

1. Activate the build configuration.
   Right-click on the project, then select **Build Configurations>Set Active> External GCC - Debug** (or Release).

Fig. 20. Navigation inside Help Content window

5.2. Import existing Infineon example

To test if the HighTec compiler works correctly, let’s import one of the existing Infineon projects.

Select **File→Import→Infineon AURIX Development Studio Project** and click **Next**.

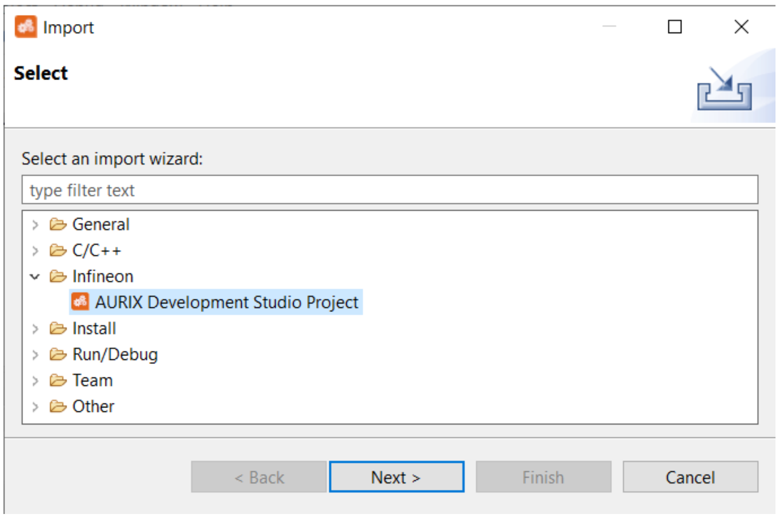
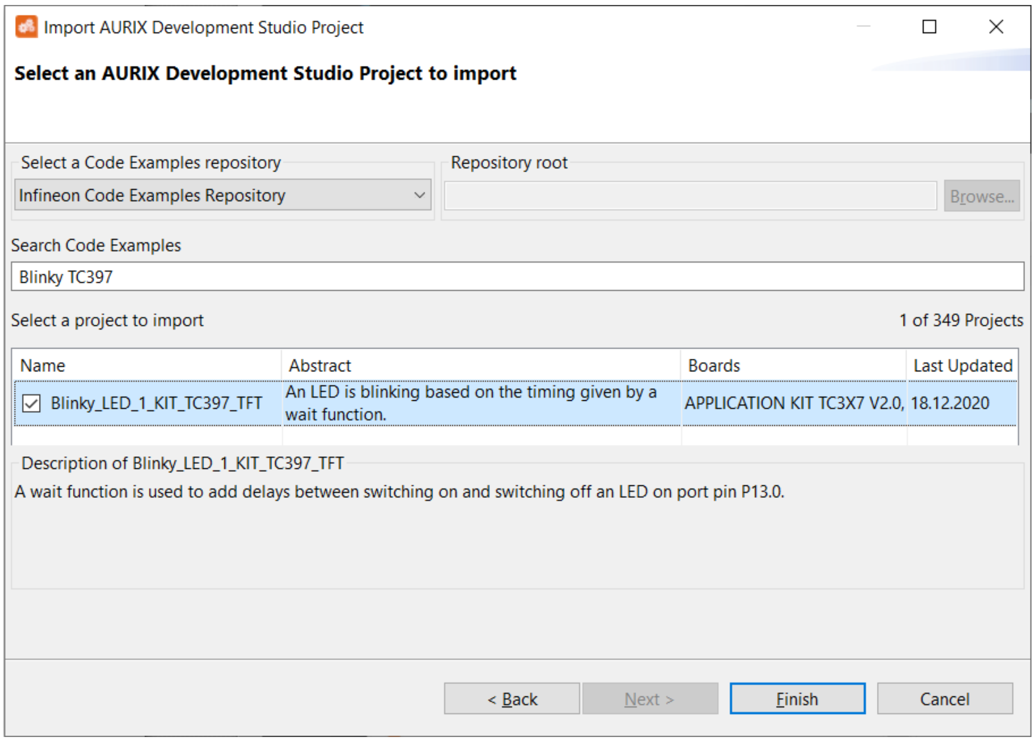


Fig. 21. Import Infineon example

Then select an example of your choice (we will test the "blinky" example for Application Kit TC397).


Import AURIX Development Studio Project
Select an AURIX Development Studio Project to import
Browse...
< Back
Next >
Finish
Cancel

Select a Code Examples repository: Infineon Code Examples Repository

Repository root:

Search Code Examples: Blinky TC397

Select a project to import — 1 of 349 Projects

Name	Abstract	Boards	Last Updated
☑ Blinky_LED_1_KIT_TC397_TFT	An LED is blinking based on the timing given by a wait function.	APPLICATION KIT TC3X7 V2.0,	18.12.2020

Description of Blinky_LED_1_KIT_TC397_TFT

A wait function is used to add delays between switching on and switching off an LED on port pin P13.0.

Fig. 22. Select an example from the list (search is used as a filter)

5.3. Manage build configuration

To create a new HighTec GCC toolchain configuration, right-click on the project and select **Build Configuration→Manage**.

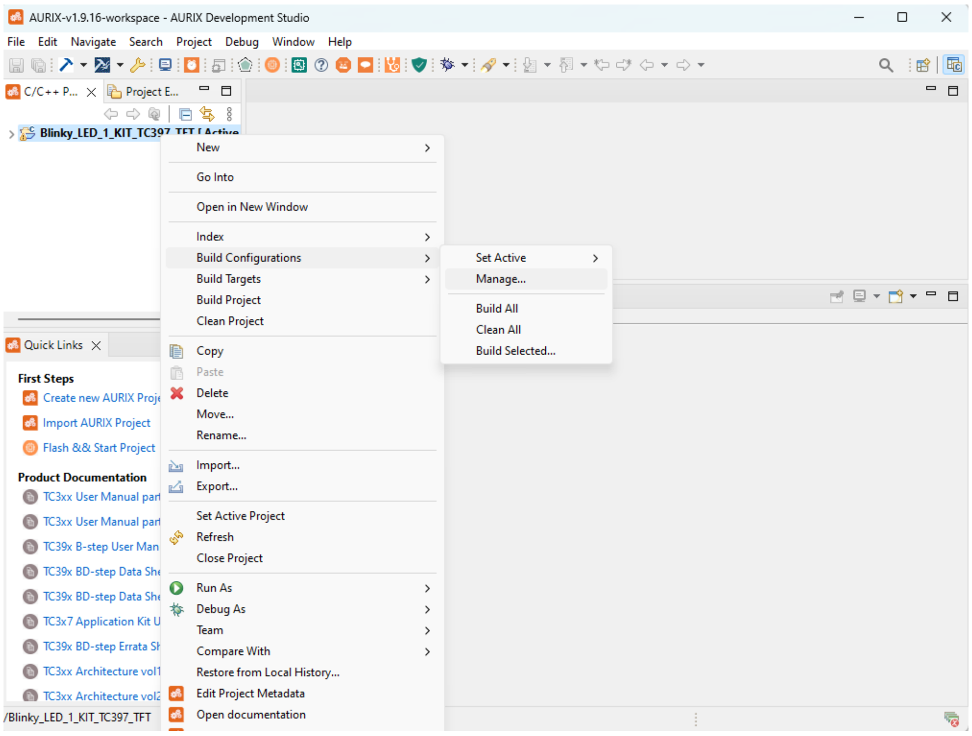
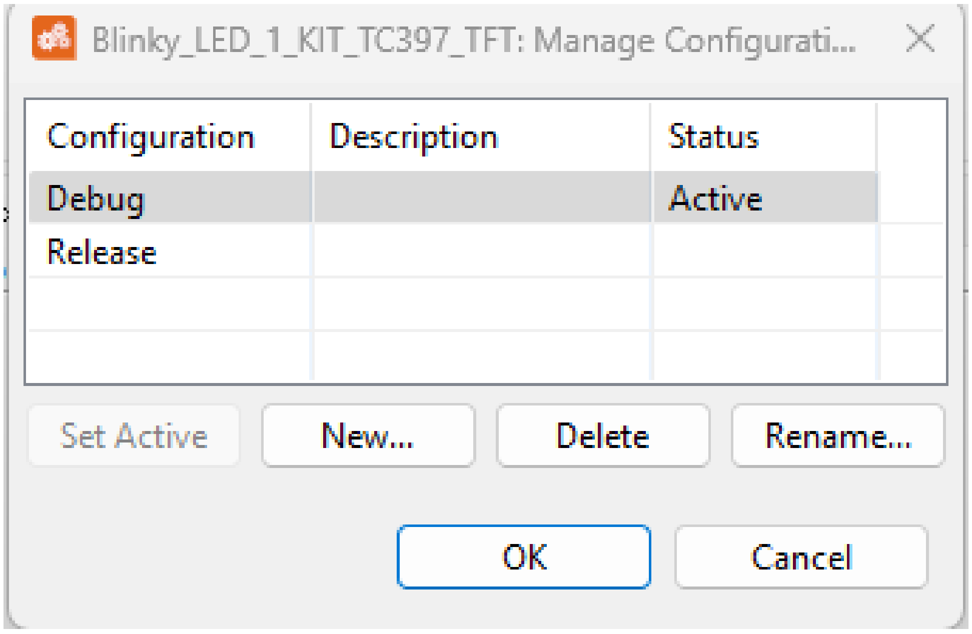


Fig. 23. Setting the manage build configuration

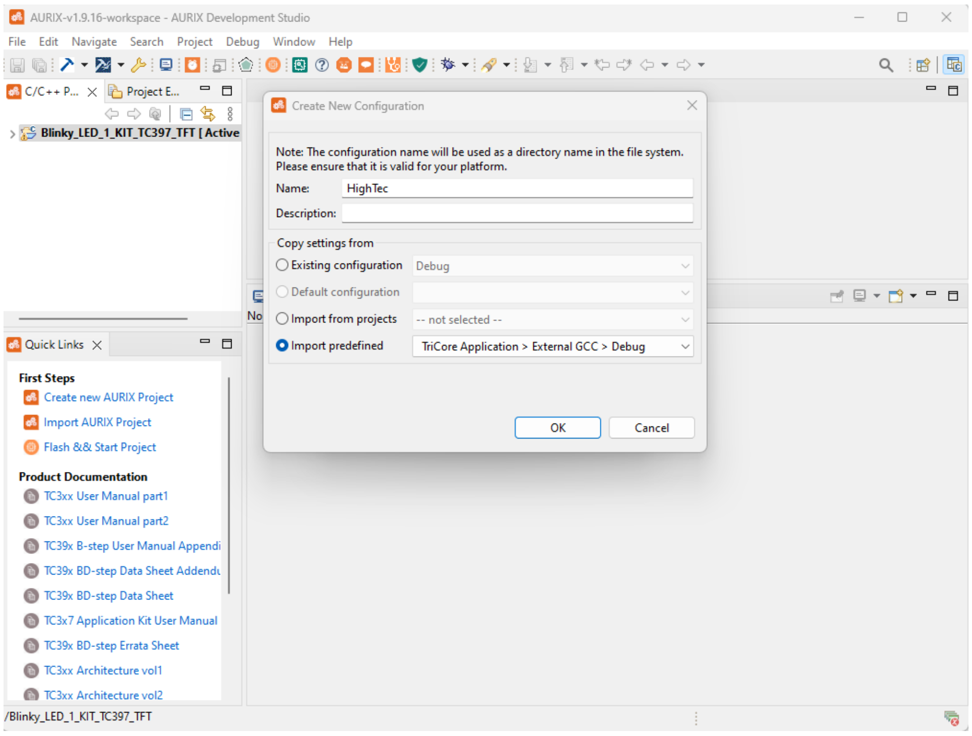
The Manage Configurations tab will be opened.



Configuration	Description	Status
Debug		Active
Release		

Fig. 24. manage configuration tab

Select **[New]** and give a name to the new configuration. Then select **Import predefined**, choose **TriCore Application→External GCC→Debug** and click **[OK]**.


Create New Configuration

Note: The configuration name will be used as a directory name in the file system.
Please ensure that it is valid for your platform.

Name: HighTec

Description:

Copy settings from

- Existing configuration: Debug
- Default configuration
- Import from projects: -- not selected --
- [✓ Import predefined]: TriCore Application > External GCC > Debug

AURIX-v1.9.16-workspace - AURIX Development Studio
File Edit Navigate Search Project Debug Window Help
C/C++ P...
Project E...
Blinky_LED_1_KIT_TC397_TFT [Active
Quick Links
First Steps
Create new AURIX Project
Import AURIX Project
Flash && Start Project
Product Documentation
TC3xx User Manual part1
TC3xx User Manual part2
TC39x B-step User Manual Appendi
TC39x BD-step Data Sheet Addendu
TC39x BD-step Data Sheet
TC3x7 Application Kit User Manual
TC39x BD-step Errata Sheet
TC3xx Architecture vol1
TC3xx Architecture vol2
/Blinky_LED_1_KIT_TC397_TFT
OK
Cancel

Fig. 25. new configuration

5.4. Activate build configuration

To select a new active configuration, right-click on the project and select **Build Configuration→Set Active →HighTec (name of the new configuration created)**.

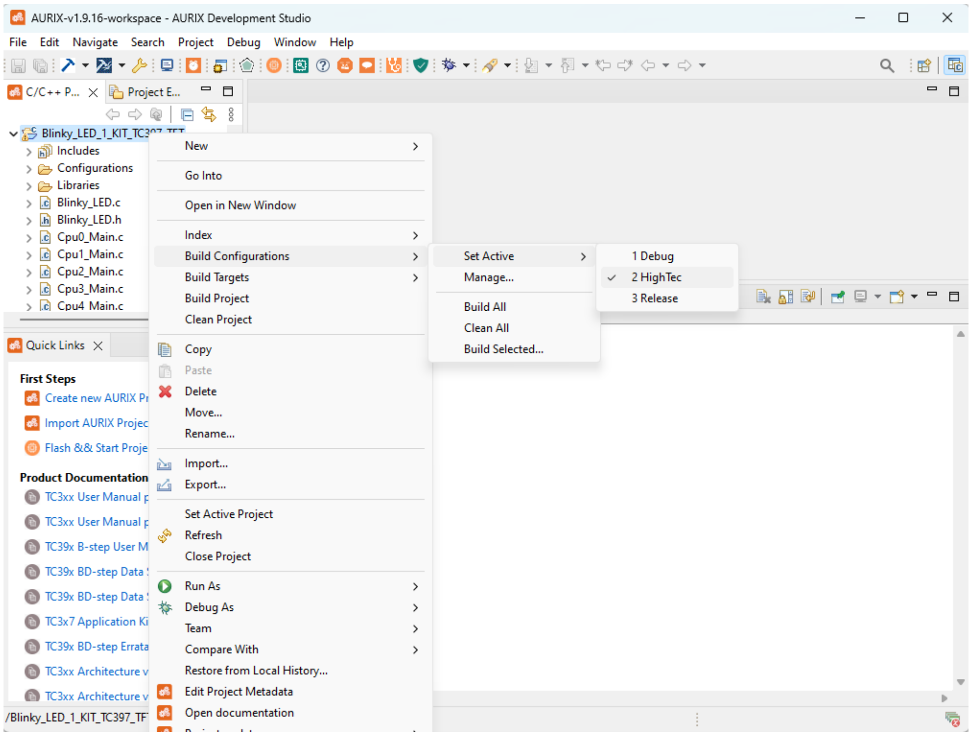


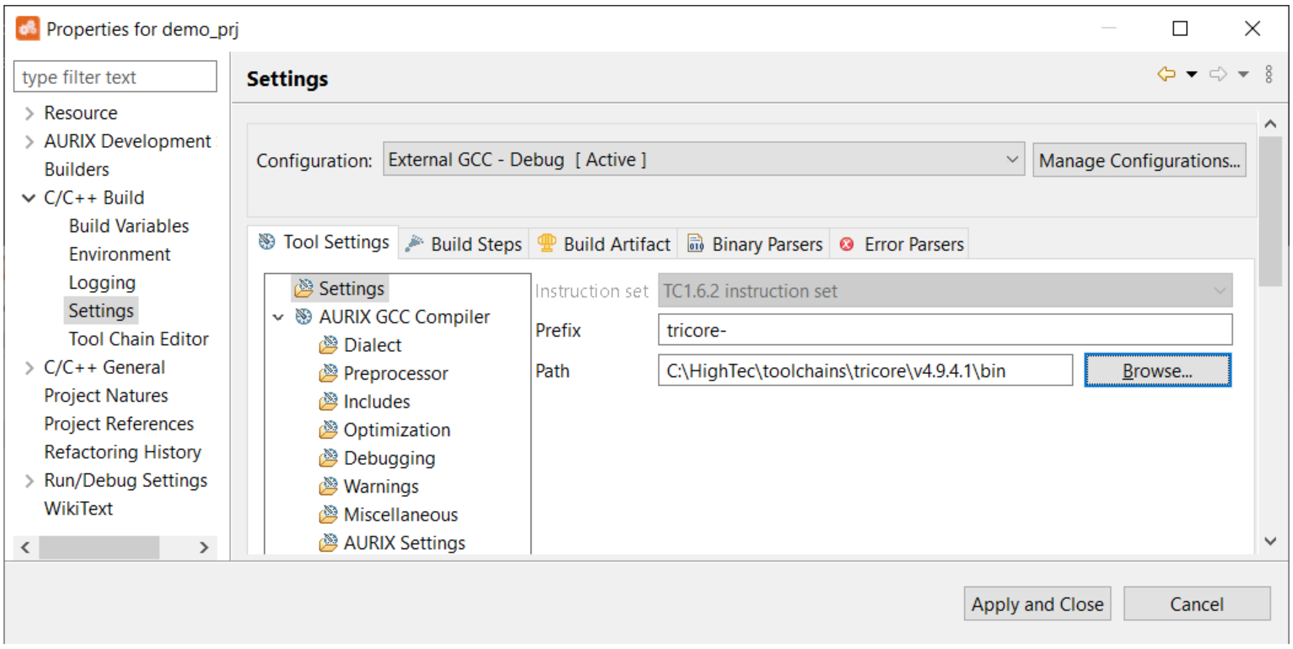
Fig. 26. Setting the active build configuration

5.5. Compiler-specific options

5.5.1. Path to the compiler

The path and name prefix of the Hightec compiler must be updated for the active build configuration.

Right-click on the project and select **Properties→C/C++ Build→Settings**. Change the **Prefix** and **Path** fields under the **Tool Settings** tab. The prefix is always "tricore-" for AURIX devices; the path might be different.



Configuration: External GCC - Debug [Active]

Instruction set: TC1.6.2 instruction set

Prefix: tricore-

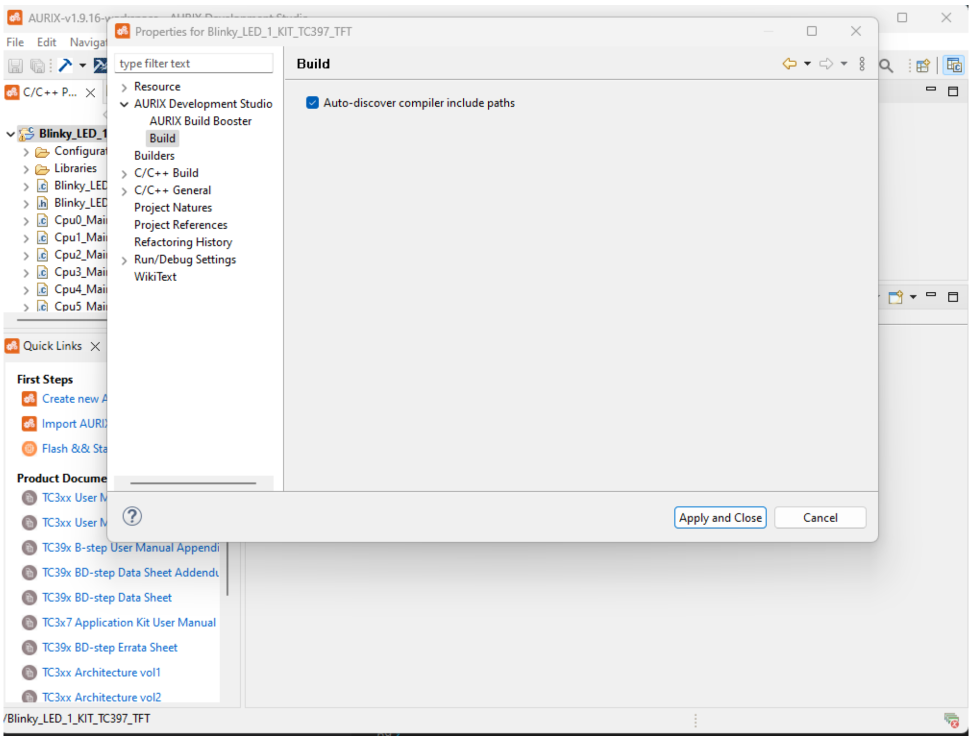
Path: C:\HighTec\toolchains\tricore\v4.9.4.1\bin

Fig. 27. Setting the prefix and path to the HighTec GCC compiler

Now, the project will use the HighTec GCC compiler to build the application.

5.5.2. Include paths

Right-click on the project and select **Properties→Aurix Development Studio→Build** check the box to auto-discover compiler include paths.



Auto-discover compiler include paths ✓

Fig. 28. Include paths

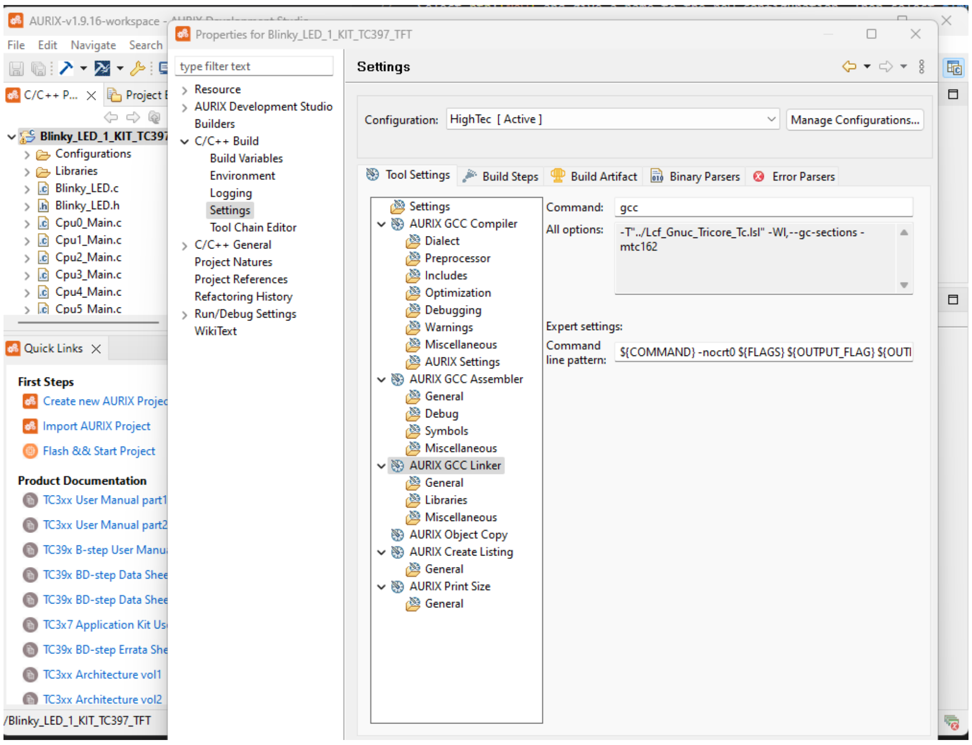
5.5.3. GCC linker options

Right-click on the project and select **Properties→C/C++ Build→Settings**, click on AURIX GCC Linker. Under **Command line pattern**, change:

From: **`${COMMAND} ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`**

To: **`${COMMAND} -nocrt0 ${FLAGS} ${OUTPUT_FLAG} ${OUTPUT_PREFIX}${OUTPUT} ${INPUTS}`**

Click on Apply and Close.



Configuration: HighTec [Active]

Command: gcc

All options: -T"../Lcf_Gnuc_Tricore_Tc.lsl" -Wl,--gc-sections -mtc162

Expert settings:

Command line pattern: ${COMMAND} -nocrt0 ${FLAGS} ${OUTPUT_FLAG} ${OUTI

Fig. 29. GCC linker options

5.6. Linker script

Before the imported Infineon example can be built with the HighTec compiler, the **`Lcf_Gnu_Tricore_Tc.lsl` linker script file must be copied** into the newly imported Infineon project.

You can find the linker script here: HighTec Linker Scripts.

After the linker script is in place, the project is ready to be built.

Appendix

Appendix A: DMA Example

This guide contains instructions for how to successfully compile the **DMA_Mem_to_Mem_1_KIT_TC499_COM_TRB** project using the HighTec LLVM toolchain version 8.0.0. It assumes that the project has been imported into the current workspace.

The following changes are required before the first project build attempt:

- Wait until the indexer finishes.
- Right-click on the project, then **Project updater**→**Update iLLD**→**Finish**.
- Right-click on the project and click on **Refresh**.
- Wait until the indexer finishes.
- Add **`Lcf_Hightec_Tricore_Dma.lsl`** linker script adapted specifically for the DMA use case.

The following changes are required to build the project:

- Right-click on the project, then **Build Configurations**→**Manage**→**New**
  - Give any name
  - Select the option **Import predefined**, then **TriCore Application**→**External**→**LLVM**→**Debug** and click **`[OK]`**.
- Set the newly created configuration to active by clicking on **`[Set Active]`**.
- Wait until the indexer finishes.
- Edit the build settings: right-click on the project **Properties**→**C/C++ Build**→**Settings**
  - Click on **Settings** and Update Path to the LLVM bin directory even though it is not used later on, i.e., "C:\HighTec\toolchains\tricore\v8.0.0\bin" and click on **`[Apply]`**.
  - Click on **LLVM Linker**, extend the command string with **`-merrata=cpu141`** to **`clang -merrata=cpu141`** and click on **`[Apply]`**.
  - Click on **LLVM Linker**→**General** and change Linker Script name to **`../Lcf_Hightec_Tricore_Dma.lsl`** and click on **`[Apply and close]`**.
- Update the file *DMA_Mem_to_Mem.c*:
  From:

```
/* DMA Source buffer for DMA transfer stored inside DSPR0: 0x70000000 */
uint32 g_dataForDmaTransfer[DATA_ARRAY_LENGTH] __at(0x70000000);

/* DMA Destination buffer stored inside DLMU RAM (of CPU0): 0xB0000000 */
uint32 g_dmaLmuDestination[DATA_ARRAY_LENGTH]  __at(0xB0000000);
```

  To:

```
#if defined(__TASKING__)
/* DMA Source buffer for DMA transfer stored inside DSPR0: 0x70000000 */
uint32 g_dataForDmaTransfer[DATA_ARRAY_LENGTH] __at(0x70000000);
/* DMA Destination buffer stored inside DLMU RAM (of CPU0): 0xB0000000 */
uint32 g_dmaLmuDestination[DATA_ARRAY_LENGTH]  __at(0xB0000000);

#elif defined(__HIGHTEC__) && defined(__clang__)
#pragma clang section bss=".dma.dspr0bss"
/* DMA Source buffer stored inside .dma.dspr0bss section */
uint32 g_dataForDmaTransfer[DATA_ARRAY_LENGTH];
#pragma clang section bss=".dma.dlmu0bss"
/* DMA Destination buffer stored inside .dma.dlmu0bss */
uint32 g_dmaLmuDestination[DATA_ARRAY_LENGTH];
#pragma clang section bss=""

#else
#error "Unknown compiler"
#endif
```

- Build the project.

Appendix B: PPU examples

This guide contains instructions for successful compilations of the following projects using HighTec LLVM toolchain version 8.0.0:

- **iLLD_TC49xA_ADS_PPU-IPC**
- **iLLD_TC49xA_ADS_PPU-IRQ**
- **iLLD_TC49xA_ADS_PPU-SIMD**
- **iLLD_TC49xA_ADS_PPU-STU**

It assumes that one of the projects has been imported into the current workspace.

The following changes are required before the first project build attempt:

- Wait until the indexer finishes.
- Right-click on the project, then **Project updater**→**Update iLLD**→**Finish**.
- Right-click on the project and click on **Refresh**.
- Wait until the indexer finishes.
- Add the **`Lcf_Hightec_Tricore_Ppu.lsl`** linker script specifically adapted for the PPU use case.

B.1. PPU MetaWare build configuration

The following changes are required to build the PPU project for the existing PPU Debug (Ext. MetaWare) build configuration:

- Right-click on the project, then **Build Configurations**→**Set Active**→**PPU Debug (Ext. MetaWare)**.
- Right-click on the project, then **Properties**→**C/C++ Build**→**Settings**→**Settings**. Update MetaWare Installation Path to, i.e., "C:\HighTec\toolchains\mwaurix\v0.2.0" and click on **`[Apply and close]`**.
- Click on the project, then **Properties**→**C/C++ Build**→**Behavior**. Disable parallel build by unchecking the "enable parallel build" checkbox, then click on **`[Apply and close]`**. Optionally, configure the option "use optimal jobs" to "use parallel jobs" and set the value to be lower or equal to the physical cores of the used laptop. **This is not a fail-proof option, and one might encounter a compiler crash during the build.**
- One by one, right-click on the following directories/files, then **Resource Configuration**→**Exclude from build**→**Select your build configuration** and click **`[OK]`** (some may already have been disabled).
  /Libraries/iLLD/TC49A/CpuGeneric/Asclin
  /Libraries/iLLD/TC49A/Csrm, /Libraries/iLLD/TC49A/Scr
  /Libraries/Infra/Platform/Tricore
  /Libraries/Infra/Ssw/TC49A/Csrm, /Libraries/Infra/Ssw/TC49A/Tricore
  AllowAccess.c, PpuInterface.c
  Cpu0_Main.c, Cpu1_Main.c, Cpu2_Main.c, Cpu3_Main.c, Cpu4_Main.c, Cpu5_Main.c
- Build the PPU project.

B.2. TriCore™ external LLVM build configuration

The following changes are required to build the Tricore project part:

- Right-click on the project, then **Build Configurations→Manage→New**
  - Give any name
  - Select the option **Import predefined**, then **TriCore Application→External→LLVM→Debug** and click **`[OK]`**.
- Set the newly created configuration to active by clicking on **`[Set Active]`**.
- Wait until the indexer finishes.
- Edit the build settings: right-click on the project **Properties→C/C++ Build→Settings**
  - Click on **Settings** and Update Path to the LLVM bin directory even though it is not used later on, i.e., "C:\HighTec\toolchains\tricore\v8.0.0\bin" and click on **`[Apply]`**.
  - Click on **LLVM Linker**, extend the command string with **`-merrata=cpu141`** to **`clang -merrata=cpu141`** and click on **`[Apply]`**.
  - Click on **LLVM Linker→Miscellaneous→Add `-lhtcos`**. Then click on **`[Ok]`** and **`[Apply]`**.
  - Click on **LLVM Linker→General** and change Linker Script name to **`../Lcf_Hightec_Tricore_Ppu.lsl`** and click on **`[Apply and close]`**.
- Right-click on the project, then **Properties→AURIX Development Studio→AURIX Build Booster →Libraries paths→Delete */Libraries/iLLD* if present** and click on **`[Apply and close]`**.
- Right-click on the project, then **Properties→AURIX Development Studio→AURIX Build Booster →Ignore paths→Add the following folders** and click on **`[Apply and close]`**.
  /Libraries/iLLD/TC49A/ArcEV, */Libraries/iLLD/TC49A/Csrm*, */Libraries/iLLD/TC49A/Scr*
  /Libraries/Infra/Platform/ArcEV
  /Libraries/Infra/Ssw/TC49A/Csrm
  /PPU
- Right-click on the project, then **Properties→C/C++ General→Paths and Symbols→Includes→GNU C**, and ensure that the include paths do not contain any of the following directories and their subdirectories, click on **`[Apply and close]`**.
  /Libraries/iLLD/TC49A/ArcEV, */Libraries/iLLD/TC49A/Csrm*, */Libraries/iLLD/TC49A/Scr*
  /Libraries/Infra/Platform/ArcEV
  /Libraries/Infra/Ssw/TC49A/Csrm
  /PPU
- One by one, Right-click on the following directories, then **Resource Configuration→Exclude from build →Select your build configuration** and click **`[OK]`** (some may already have been disabled).
  /Libraries/iLLD/TC49A/ArcEV, */Libraries/iLLD/TC49A/Csrm*, */Libraries/iLLD/TC49A/Scr*
  /Libraries/Infra/Platform/ArcEV
  /Libraries/Infra/Ssw/TC49A/Csrm
  /PPU
- Change the definition of uncached data/bss sections in *Cpu0_Main.c* due to the warning: unknown pragma ignored [**`-Wunknown-pragmas`**]

* Change *uncached.lmubss* section definition (applies to **iLLD_TC49xA_ADS_PPU-IPC**, **iLLD_TC49xA_ADS_PPU-IRQ**, and **iLLD_TC49xA_ADS_PPU-STU**)
  From:

  ```
  #pragma section all "uncached.lmubss"
  /* Declaration of global uninitialized variables */
  #pragma section all
  ```

  To:

  ```
  #if defined(__HIGHTEC__) && defined(__clang__)
  #pragma clang section bss=".uncached.lmubss"
  #else #pragma section all "uncached.lmubss"
  #endif
  /* Declaration of global uninitialized variables */
  #if defined(__HIGHTEC__) && defined(__clang__)
  #pragma clang section bss=""
  #else
  #pragma section all "uncached.lmubss"
  #endif
  ```

* Change *uncached.lmudata* section definition (applies to **iLLD_TC49xA_ADS_PPU-STU**):
  From:

  ```
  #pragma section all "uncached.lmudata"
  /* Definition of global initialized variables */
  #pragma section all "uncached.lmudata"
  ```

  To:

  ```
  #if defined(__HIGHTEC__) && defined(__clang__)
  #pragma clang section data=".uncached.lmudata"
  #else
  #pragma section all "uncached.lmudata"
  #endif
  /* Definition of global initialized variables */
  #if defined(__HIGHTEC__) && defined(__clang__)
  #pragma clang section data=""
  #else
  #pragma section all "uncached.lmudata"
  #endif
  ```

* Change the following code in *AllowAccess.c* due to the warning: non-portable path to file; specified path differs in case from file name on disk [**`-Wnonportable-include-path`**]:
  From:

  ```
  #include "IfxCpu_Cfg.h"
  #include "IfxGtm_Cfg.h"
  #include "IfxEgtm_Cfg.h"
  #include "IfxQspi_Cfg.h"
  #include "IfxCan_Cfg.h"
  #include "IfxAsclin_Cfg.h"
  #include "IfxSent_Cfg.h"
  #include "IfxGpt12_Cfg.h"
  #include "IfxI2c_Cfg.h"
  #include "IfxGeth_Cfg.h"
  ```

To:

```
#include "IfxCpu_cfg.h"
#include "IfxGtm_cfg.h"
#include "IfxEgtm_cfg.h"
#include "IfxQspi_cfg.h"
#include "IfxCan_cfg.h"
#include "IfxAsclin_cfg.h"
#include "IfxSent_cfg.h"
#include "IfxGpt12_cfg.h"
#include "IfxI2c_cfg.h"
#include "IfxGeth_cfg.h"
```

- Build TriCore™ project.

Appendix C: SCR example

This guide contains instructions for how to successfully compile the **iLLD_TC49x_ADS_SCR_Blinky_LED_1** project using the HighTec LLVM toolchain version 8.0.0. It assumes that the project has been imported into the current workspace.

The following changes are required before the first project build attempt:

- Wait until the indexer finishes.
- Right-click on the project, then **Project updater**→**Update iLLD**→**Finish**.
- Right-click on the project and click on **Refresh**.
- Wait until the indexer finishes.

C.1. SCR SDCC build configuration

The following changes are required to build the SCR project with the SDCC toolchain:

- Right-click on the project, then **Build Configurations**→**Manage**→**New**
  - Give any name.
  - Select the option **Import predefined**, then **SCR Application**→**External**→**SDCC**→**Debug** and click **`[OK]`**.
- One by one, Right-click on the following directories, then **Resource Configuration**→**Exclude from build** →**Select your build configuration** and click **`[OK]`** (some may already have been disabled).
  Configurations, *Libraries*
  CPU0_Main.c, *CPU1_Main.c*, *CPU2_Main.c*, *CPU3_Main.c*, *CPU4_Main.c*, *CPU5_Main.c*
  SCR_AURIX_TC4x.c
- Edit the build settings: right-click on the project **Properties**→**C/C++ Build**→**Settings**
  - SDCC Compiler
    - Update Command with the full compiler path to:
      `"C:\HighTec\toolchains\scr\v1.0.0/bin/scr-cc" -c -mtc4xx-scr --model-large -MMD -o "SCR/main.rel"`
  - SDCC Linker
    - Update Command with the full compiler path to:
      `"C:\HighTec\toolchains\scr\v1.0.0/bin/scr-cc" -mtc4xx-scr --model-large --xram -size 0x400 -o 'aurix_scr.ihx' ./SCR/main.rel`
    - Update Command Line pattern to: **`${COMMAND}`**
  - Update SDCC Object copy CArray command with the full compiler path to:
    `"C:\HighTec\toolchains\scr\v1.0.0/bin/scr-objcopy" -I ihex -O binary 'aurix_scr.ihx' 'aurix_scr.bin'`
    - Update Command Line pattern to: **`${COMMAND}`**
- Build the SCR project.

C.2. TriCore™ external LLVM build configuration

The following changes are required to Build the Tricore project part:

- Right-click on the project, then **Build Configurations→Manage→New**
  - Give any name
  - Select the option **Import predefined**, then **TriCore Application→External→LLVM→Debug** and click **`[OK]`**.
- Set the newly created configuration to active by clicking on **`[Set Active]`**.
- Wait until the indexer finishes.
- Edit the build settings: right-click on the project **Properties→C/C++ Build→Settings**
  - Click on **Settings** and Update Path to the LLVM bin directory even though it is not used later on, i.e., "C:\HighTec\toolchains\tricore\v8.0.0\bin" and click on **`[Apply]`**.
  - Click on **LLVM Linker**, extend the command string with **`-merrata=cpu141`** to **`clang -merrata=cpu141`** and click on **`[Apply]`**.
  - Click on **LLVM Linker→Miscellaneous→Add `-lhtcos`**. Then click on **`[Ok]`** and **`[Apply]`**.
- Right-click on the project, then **Properties→AURIX Development Studio→AURIX Build Booster →Libraries paths→Delete *\/Libraries/iLLD* if present** and click on **`[Apply and close]`**.
- Right-click on the project , then **Properties→AURIX Development Studio→AURIX Build Booster →Ignore paths**, add the following folders, and click on **`[Apply and close]`**.
  /Libraries/iLLD/TC49A/ArcEV
  /Libraries/iLLD/TC49A/Csrm
  /Libraries/iLLD/TC49A/Scr
- Right-click on the project, then **Properties→C/C++ General→Paths and Symbols→Includes→GNU C**, ensure that the Include paths do not contain */Libraries/iLLD/TC49A/Scr* or its subdirectories, and click on **`[Apply and close]`**.
- One by one, Right-click on the following directories, then **Resource Configuration→Exclude from build →Select your build configuration** and click **`[OK]`** (some may already have been disabled).
  /Libraries/iLLD/TC49A/ArcEV
  /Libraries/iLLD/TC49A/Csrm
  /Libraries/iLLD/TC49A/Scr
  /SCR
- Change the following code in *Cpu0_Main.c*:
  From:

```
if (PMS_SCR_CON0.B.SCREN == 0)
{
    IfxPmsPm_MemoryConfig memCfg = {&scr_xram, PMS_XRAM, SIZE_scr_xram};
    IfxPmsPm_initScr(IfxPmsPm_ScrBootMode_userMode, FALSE, &memCfg);
}
```

  To:

```
#if defined(__HIGHTEC__) && defined(__clang__)
     /*
---------------------------------------------------------------------------------------------
      * Include the binary file of the SCR program into the TriCore build

*--------------------------------------------------------------------------------------------
*/
     extern const unsigned char _bin_file_start[];
     extern const unsigned char _bin_file_end[];
     __asm__(
      ".section \".rodata\", \"a\", @progbits\n"
      "_bin_file_start:\n"
      ".incbin \"../../iLLD_TC49x_ADS_SCR_Blinky_LED_1/Debug_SCR/aurix_scr.bin\"\n"
      "_bin_file_end:\n"
      ".previous\n"
     );
#endif
     if (PMS_SCR_CON0.B.SCREN == 0)
     {
#if defined(__TASKING__)
          IfxPmsPm_MemoryConfig memCfg = {&scr_xram, PMS_XRAM, SIZE_scr_xram};
#elif defined(__HIGHTEC__) && defined(__clang__)
          IfxPmsPm_MemoryConfig memCfg = {(void*)&_bin_file_start, PMS_XRAM, (unsigned long
int)((unsigned long int)_bin_file_end-(unsigned long int)_bin_file_start)};
#endif
          IfxPmsPm_initScr(IfxPmsPm_ScrBootMode_userMode, FALSE, &memCfg);
     }
```

- Build TriCore™ project.

Exclude folders from the build

Some examples contain superfluous folders that can interrupt or contaminate a build.

Building such examples results in compiler errors like:
`error: unknown type name 'sint64'`,
`error: unknown type name 'uint64'` or
`error: use of undeclared identifier 'uint64'`.

The following changes are required to build these projects for LLVM Build configuration:

- Right-click on the project, then **Properties→AURIX Development Studio→AURIX Build Booster →Libraries paths**, delete */Libraries/iLLD* if it is present, and click on **`[Apply and close]`**.
- Right-click on the project , then **Properties→AURIX Development Studio→AURIX Build Booster →Ignore paths**, add the following folders, and click on **`[Apply and close]`**.
  /Libraries/iLLD/TC49A/ArcEV, */Libraries/iLLD/TC49A/Csrm*, */Libraries/iLLD/TC49A/Scr*
  /Libraries/Infra/Platform/ArcEV
  /Libraries/Infra/Ssw/TC49A/Csrm
- Right-click on the project, then **Properties→C/C++ General→Paths and Symbols→Includes→GNU C**, ensure that the Include paths do not contain any of the following directories and their subdirectories, click on **`[Apply and close]`**.
  /Libraries/iLLD/TC49A/ArcEV, */Libraries/iLLD/TC49A/Csrm*, */Libraries/iLLD/TC49A/Scr*
  /Libraries/Infra/Platform/ArcEV
  /Libraries/Infra/Ssw/TC49A/Csrm
- One by one, Right-click on the following directories, then **Resource Configuration→Exclude from build**, select your build configuration, and click **`[OK]`** (some may already have been disabled).
  /Libraries/iLLD/TC49A/ArcEV
  /Libraries/iLLD/TC49A/Csrm
  /Libraries/iLLD/TC49A/Scr

Manually excluding these folders doesn't always prevent them from being included again upon rebuilding.

Document references

[1] "AURIX™ TC49x errata sheet, Infineon Technologies AG, v1.16, 2023-08-07"

[2] "TriCore™ Development Platform: User Guide, HighTec EDV-Systeme GmbH, Version 8.0.0, October 2023"

Disclaimer

Document history

Version	Date	Changes to the previous version
1.0	June 2023	Initial version
1.1	October 2023	Changes to accommodate newer version of ADS
1.2	November 2023	Added section: External toolchain GCC

HIGHTEC

HighTec EDV-Systeme GmbH
Europaallee 19, D-66113 Saarbrücken
info@hightec-rt.com
+49-681-92613-16
www.hightec-rt.com