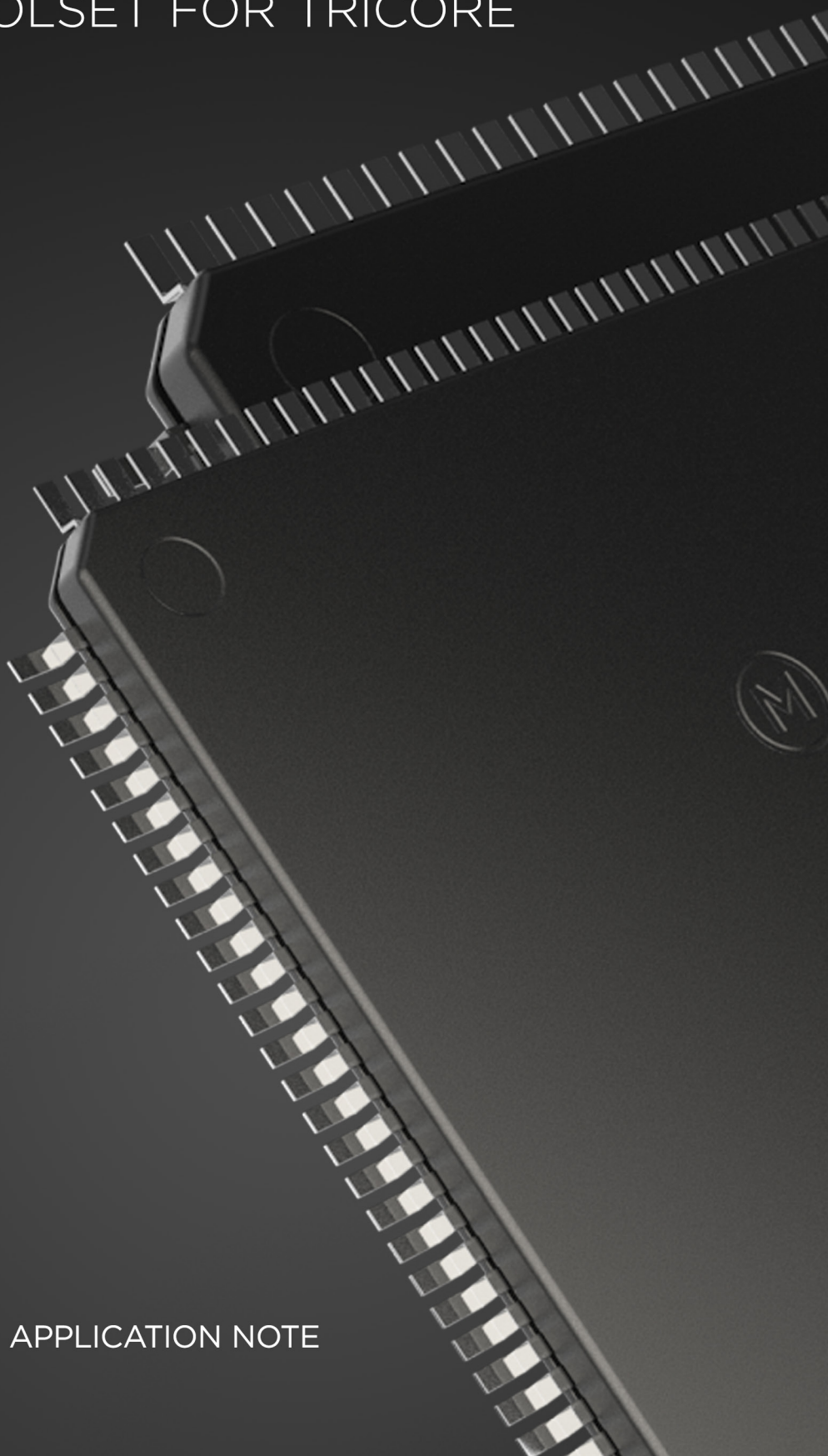
TASKING®

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING

VX-TOOLSET FOR TRICORE



APPLICATION NOTE

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

INTRODUCTION

The Low Level Driver library (iLLD) from Infineon provides access and configuration functions for the integrated peripherals of Infineon microcontrollers. This application note describes how you can use Infineon's iLLD together with the TASKING VX-toolset for TriCore to access and configure the integrated peripherals of Infineon TriCore derivatives.

DOWNLOAD THE ILLD SOURCES AND EXAMPLES

The iLLD sources and examples are available for download from the Infineon website.

https://myicp.infineon.com

Please note that for access you will need to register with Infineon.

1. Access the website and search for "iLLD".
2. Download the file:

   `iLLD_1_0_1_8_0_TC2xx_Drivers_And_Demos_Release.zip`
   or:
   `iLLD_1_0_1_8_0_TC3xx_Drivers_And_Demos_Release.zip`

   depending on the TriCore derivative you want to use. In this application note we will use TC2xx as an example.

EXPAND THE ARCHIVES TO A DIRECTORY OF YOUR CHOICE

The iLLD software release package is an archive, and it may be extracted to any drive on the computer. There is no specific directory structure needed.

1. Expand the main archive.
   After expansion of the main archive, the derivative dependent sources will be in a separate ZIP file.
2. Expand the ZIP files containing the sources for the TriCore derivative you are using.
   After expansion the directory tree contains iLLD sources for each derivative, for example:

   Drive`\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC23A`
   Drive`\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC27D`
   Drive`\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC29B`

   These directories contain subdirectories (example for the TC23x)

   Drive`\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC23A\Src\BaseSw`

   You will later need to copy the complete source tree of Src\BaseSw to your project directory in the Eclipse workspace.
3. Expand the ZIP file containing the iLLD examples.

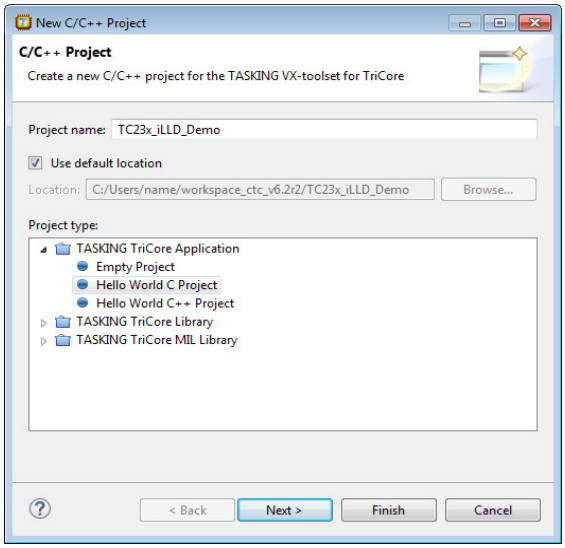
   Drive`\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC2xx_Demos.zip`

   Please note that the `AppSw` directory of an iLLD example will have to go into the project's `Src\AppSw` directory.

CREATE A PROJECT USING TRICORE ECLIPSE IDE

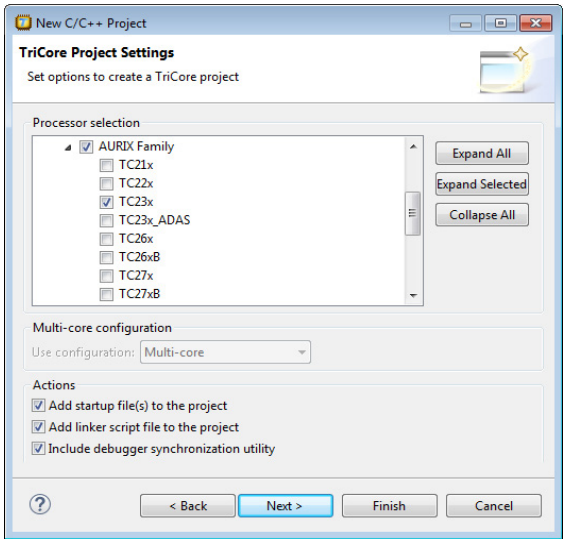
1. Start the TriCore Eclipse IDE.
2. From the **File** menu, select **New » TASKING TriCore C/C++ Project**.

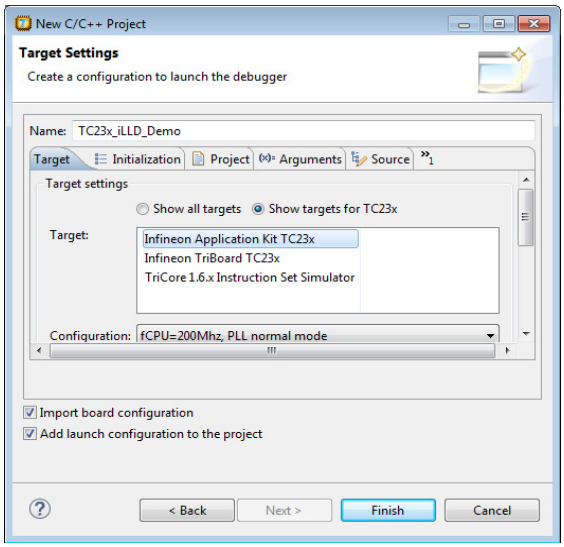
HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE



a. Specify a project name (for example TC23x_iLLD_Demo), select **TASKING TriCore Application » Hello World C Project** and click **Next**.

b. Select a processor and click **Next**. In this application note we select TC23x.





c. Select a target and click **Finish**. In this application note we select Infineon Application Kit TC23

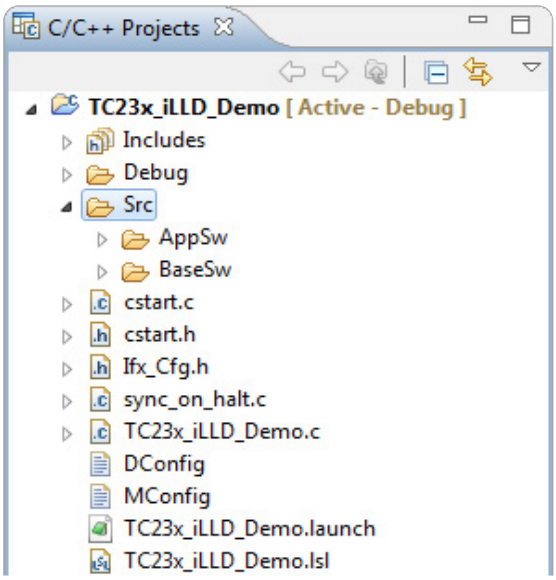
HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

3. Import the iLLD sources from the unpacked ZIP archive into the project by copying `\Src\BaseSw` for the desired derivative into the Eclipse project directory.

   ```
   copy Drive\IFX_iLLD_10180\iLLD_1_0_1_8_0__TC23A\Src workspace\TC23x_iLLD_Demo
   ```

4. For this specific sample we will use the `GtmTomPwmHlDemo` example, located in:

   Drive`\iLLD_1_0_1_8_0_TC2xx_Demos\GtmTomPwmHlDemo`
   Copy *Drive*`\iLLD_1_0_1_8_0_TC2xx_Demos\GtmTomPwmHlDemo\0_Src\AppSw`
   to the `Src\AppSw` directory of your workspace project directory: *workspace*`\TC23x_iLLD_Demo`



   Note: when you use a TC3xx ZIP archive and example, the `AppSw` and `BaseSw` directories contain two extra directories which should be removed. Remove the following two directories from your project:

   ```
   Src\AppSw\Tricore\Cfg_Ssw
   Src\BaseSw\Infra\Ssw
   ```

5. Set all include paths for iLLD. As the structure of the iLLD directory remains the same for all derivatives it is possible to use the same include paths for each project.

   Note: the include directory listing directly depends on the project structure of iLLD which you you have in place. Many source files and include files of the iLLD package depend on this structure as they use hardcoded relative paths.

   Thus any change in the iLLD / project structure will cause a great number of include errors which are hard to resolve.

   The include file list is ready to use and contains information to use workspace/project relative include references. This allows to use these in any project and any location of the file system / workspace.

   a. From the **Project** menu, select **Properties for » C/C++ Build » Settings » Tool Settings » C/C++ Compiler » Include Paths**

   b. Click on the **Add** button and use the **Workspace** button in the **Add directory path** dialog to make the search paths relative to your project's workspace.

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

c. Add the following list (example for a TC23 project):

```
"${workspace_loc:/${ProjName}}"
"${workspace_loc:/${ProjName}/Src/AppSw/Tricore/Main}"
"${workspace_loc:/${ProjName}/Src/AppSw/Tricore/Cfg_Illd}"
"${workspace_loc:/${ProjName}/Src/AppSw/Tricore/Demo_Illd}"
"${workspace_loc:/${ProjName}/Src/BaseSw/Service/CpuGeneric}"
"${workspace_loc:/${ProjName}/Src/BaseSw/Infra/Platform}"
"${workspace_loc:/${ProjName}/Src/BaseSw/Infra/Sfr/TC23A/_Reg}"
"${workspace_loc:/${ProjName}/Src/BaseSw/Service}"
"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/TC23A/Tricore}"
"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/TC23A/Tricore/Scu/Std}"
"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/TC23A/Tricore/_Impl}"
"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/TC23A/Tricore/Cpu/Std}"
```

Note:

If you want to use iLLD BaseSw sources for a different chip, like the TC27x, replace all occurrences of the derivative name (in this sample TC23A) with TC27D (source directory name of iLLD for the TC27x). This will do the trick for the include directories.

Example:

`"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/`**`TC23A`**`/Tricore}"`

Change to:

`"${workspace_loc:/${ProjName}/Src/BaseSw/iLLD/`**`TC27D`**`/Tricore}"`

TO USE THE DEMO ILLD EXAMPLES ADAPT MAIN()

In order to use the demo iLLD examples, adapt your main source file, for example:

```
#pragma section all "APP_CODE"
extern int core0_main(void);

int main(void)
{
      core0_main();
}
```

This will make a call to the `core0_main()` function of the iLLD example code, without modifying the Infineon Application example code.

CREATE CONFIGURATION HEADER FILE IFX_CFG.H

Create a configuration header file with the name `"Ifx_Cfg.h"` in your application project root to configure the iLLD library components. You can place this header file in the root directory of your product, as the include path for the project root is provided to the compiler.

Currently it only requires to select the external oscillator frequency (XTAL) and the desired system frequency as well as the `IFX_CFG_USE_COMPILER_DEFAULT_LINKER` macro.

In addition, this file allows you to specify how assertions should be print out.

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

Minimal `Ifx_Cfg.h` file:

```
#ifndef IFX_CFG_H
#define IFX_CFG_H
/* necessary to set up iLLD properly */
#define IFX_CFG_USE_COMPILER_DEFAULT_LINKER  (1)
// XTAL/PLL frequency
#define IFX_CFG_SCU_XTAL_FREQUENCY (20000000)
#define IFX_CFG_SCU_PLL_FREQUENCY (200000000)
// Assertions
#define IFX_ASSERT(level, expr)
#endif /* IFX_CFG_H */
```

Example for assertion output via printf:

```
#ifdef __cplusplus
extern "C" {
#endif
// defined in $VERIF_SRC/lib/main.c
extern const char* verboseLevelStr[6];
#define IFX_ASSERT(level, expr) (((expr) || (level > IFX_VERBOSE_LEVEL_ERROR)) ? ((void)0) : (void)
printf("[ASSERT:%s] '%s' in %s:%d (function '%s')\n", verboseLevelStr[level], #expr, __FILE__, __
LINE__, __func__))
#endif
```

And in your `main.c` file (or somewhere else):

```
//! this global array is used by IFX_ASSERT of iLLDs
const char* verboseLevelStr[6] = {
"OFF",
"FAILURE",
"ERROR",
"WARNING",
"INFO",
"DEBUG"
};
```

DISABLE AUTOMATIC SFR FILE INCLUSION

The settings below refer to the **C/C++ Build » Settings » Tool Settings** tab of the **Properties** dialog.

In order to prevent build errors, it is necessary to disable **C/C++ Compiler » Preprocessing » Automatic inclusion of '.sfr' file**

`cstart.c` and `cstart_tc1.c` etc will take care of that since they have the include directive.

Congratulations, you are now done with the basics.
The project should now be buildable using all default options.

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

ADDITIONAL TWEAKS AND MODIFICATIONS

The settings and tweaks below are not mandatory, but greatly help improve the behavior on hardware and also help organize the many bits and pieces of code that come together in an iLLD based Software project.

STARTUP CODE MODIFICATIONS

Add `#pragma optimize 0` to startup code `cstart.h` at the very beginning of the file.

This helps to keep the startup code transparent and well to understand, in case something should go wrong with the chip initialization. The code which is now not optimized and shuffled around by the compiler to gain some speed - is much easier to understand and debug.

CONFIGURE STARTUP REGISTERS

In the **Properties** dialog, select **C/C++ Build » Startup Registers**. Set `PLLCON`, `CCUCON` and other startup registers. You can also do this in `cstart.h` directly.

Values for PWM examples etc:

```
#define __SCU_PLLCON0_INIT       1
#define __SCU_PLLCON0_VALUE      0x02013a00
#define __SCU_SYSPLLCON0_INIT    0
#define __SCU_SYSPLLCON0_VALUE   0x00000000
#define __SCU_PLLCON1_INIT       1
#define __SCU_PLLCON1_VALUE      0x00020000
#define __SCU_SYSPLLCON1_INIT    0
#define __SCU_SYSPLLCON1_VALUE   0x00000000
#define __SCU_OSCCON_INIT        1
#define __SCU_OSCCON_VALUE       0x00070018
#define __SCU_CCUCON0_INIT       1
#define __SCU_CCUCON0_VALUE      0x12120101
#define __SCU_CCUCON1_INIT       1
#define __SCU_CCUCON1_VALUE      0x50002211
#define __SCU_CCUCON2_INIT       1
#define __SCU_CCUCON2_VALUE      0x00000002
```

This configuration of the PLLCON and CCUCON registers should be done to match main system clock and the individual peripheral subclocks to the settings assumed for the iLLD examples and will allow to observe correct PWM frequencies etc. It also aligns the system clocks with those settings assumed by the debugger.

CONFIGURE BOOT MODE HEADER

Select **C/C++ Build » Memory » Boot Mode Headers** and set Boot Mode Header 0 to **Generate Boot Mode Header**.

This will allow to run the application standalone without the debugger, after disconnecting the debug interface and reset.

If you disconnect the board from the debug system and power cycle the board - the application will run standalone.

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

COMPILER AND LINKER OPTION SETTINGS

All settings below refer to the **C/C++ Build » Settings » Tool Settings** tab of the **Properties** dialog. The following options are not mandatory, but they make your life easier when inspecting the generated output files.

1. Enable **Global Options » Verbose mode of control program**
2. Set **C/C++ Compiler » Allocation » Threshold for putting data in __near** to 0.
3. Set **C/C++ Compiler » Optimization » Optimization level** to **Custom Optimization** and from **C/C++ Compiler » Optimization » Custom Optimization** disable **Code compaction**.
4. From **Linker » Optimization** disable all linker optimizations. Linker optimizations can be enabled later.
5. From **Linker » Map File** select **Generate map file (.map)**

RENAME SECTIONS OF APPLICATION CODE AND ILLD LIBRARY CODE

To match the section naming of the linker script file for the application and the iLLD components, rename all application code sections of this part of the project to `APP_CODE`:

1. Select and right-click on AppSw and select **Properties**.
2. Add option `-RAPP_CODE` to **C/C++ Build » Settings » Tool Settings » C/C++ Compiler » Miscellaneous » Additional options**.

   This will set the option to rename the sections to `APP_CODE` for all source files in the project subdirectory with just a few mouse clicks.

Likewise rename all iLLD sections of this part of the project to `IFX_ILLD`:

1. Select and right-click on BaseSw and select **Properties**.
2. Add option `-RIFX_ILLD` to **C/C++ Build » Settings » Tool Settings » C/C++ Compiler » Miscellaneous » Additional options**.

   This will set the option to rename the sections to IFX_ILLD for all source files in that project subdirectory with just a few mouse clicks.

   Renaming the section allows you to keep the TriCore's memory organized and allows for an easy overview of memory allocation and consumption of memory of the many modules being linked. The linker script file below will sort this out for you.

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

MEMORY ORGANIZATION USING THE LINKER SCRIPT FILE

To group sections and receive a fairly organized mapping add the following lines to the template linker script file in your project:

```
/*==== grouping for better overview in map file ===*/
/* allocate cinit: section = .text._c_init.libcs_fpu to a specific location */
section_layout mpe:vtc:linear{
    group CSTART_NOT_CACHED(fill,ordered, contiguous, run_addr = mem:mpe:pflash0  / not_cached
[0x400]) // / not_cached)
    {
       select ".text.cstart.*";
       select ".text._c_init.libcs_fpu";
    }
}
/* allocate near data */
section_layout mpe:vtc:abs18
{
    group APPLICATION_NEAR_DATA (ordered, contiguous, fill, run_addr=0x70000000)
    {
       select ".zbss.APP_CODE";
       select ".zdata.APP_CODE";
    }
}

section_layout mpe:vtc:linear
{    /* use offset 0x5000 to keep near area untouched */
    group APPLICATION_FAR_DATA(ordered, contiguous, fill, run_addr = mem:mpe:dspr0[0x5000])
    {
       select ".bss.APP_CODE";
       select ".data.APP_CODE";
    }

    group APPLICATION_ROM (ordered, contiguous, fill, run_addr=0x80010000)
    {
       group APPLICATION_CODE(ordered, contiguous, fill)
       {
              select ".text.APP_CODE";
       }

       group APPLICATION_ROM_DATA(ordered, contiguous, fill)
       {
              select ".rodata.APP_CODE";
       }
    }
}
```

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

```
section_layout mpe:vtc:linear
{    /* use no offset just create a group */
    group INSTRUMENTATION_FAR_DATA(ordered, contiguous, fill, run_addr = mem:mpe:dspr0[0x6000])
    {
       select ".bss.INSTRUMENT_CODE";
       select ".data.INSTRUMENT_CODE";
    }
}

/** use only if you want to place interrupts in RAM */
/* Interrupt Service Routines in RAM - everyone wants speed !
 * These routines need to be copied from their storage location in Flash to their designated RAM
location
 * The cstart initialization automatically takes care of that
 */
section_layout mpe:vtc:linear
{
      group APP_ISR_RAMCODE_group ( run_addr = mem:mpe:pspr0, ordered, copy ) /* run fromm pspr0
*/
      {
      select ".text.APP_ISR_RAMCODE";
      }
}

/* put the TASKING runtime library functions into their place - the address is not mandatory but
helps to keep things tidy */
section_layout mpe:vtc:linear
{
    group TASKING_LIB(ordered, contiguous, fill, run_addr=0x80001000)
    {
       group TASKING_LIB_CODE(ordered, contiguous, fill)
       {
               select ".text.*.libcs_fpu";
               select ".text.*.libfp";
               select ".text.*.libc";
//             select ".text.libc.*"; .text.libc.reset as a fixed address. do not touch !
               select ".text.librt";
               select ".text.*.librt";
       }
      group TASKING_LIB_ROMDATA (fill,ordered, contiguous)
      {
              select ".rodata.*.libcs_fpu";
              select "\[.data._end.libcs_fpu\]";
              select ".rodata.*.libfp";
              select ".rodata.librt";
              select ".rodata.libc";
      }
    }
}
```

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

```
/* put the TASKING library data into their place - the address is not mandatory but helps to keep
things tidy */
section_layout mpe:vtc:linear
{
      group TASKING_LIBRARY_DATA (fill,ordered, contiguous, run_addr = mem:mpe:dspr0)
      {
              select ".data.*.libcs_fpu";
              select ".data._end.libcs_fpu";
              select ".bss.*.libcs_fpu";
              select ".data.*.libfp";
              select ".bss.*.libfp";
              select ".data.librt";
              select ".bss.librt";
              select ".data.libc";
              select ".bss.libc";
              select ".data.librt";
      }
}

/* put the iLLD functions into their separate place - the address is not mandatory */
section_layout mpe:vtc:linear
{
    group ILLD_ROM(ordered, contiguous, fill, run_addr = 0x80040000)
    {
       group IFX_CODE(ordered, contiguous, fill)
       {
              select ".text.IFX_ILLD";
       }

       group IFX_ROM_DATA(ordered, contiguous, fill)
       {
              select ".rodata.IFX_ILLD";
              select "\[.data.IFX_ILLD\]";
       }
    }
}

section_layout mpe:vtc:linear
{
    group ILLD_FAR_DATA(ordered, contiguous, fill, run_addr = mem:mpe:dspr0)
    {
              select ".bss.IFX_ILLD";
              select ".data.IFX_ILLD";
    }
}

/* ==== end of linker script =====*/
```

HOW TO BUILD YOUR ILLD APPLICATION WITH TASKING VX-TOOLSET FOR TRICORE

FINAL NOTES

This concludes the Application Note on how to integrate the Infineon iLLD source code and examples into one working project.

The chosen sample code of `GtmTomPwmHlDemo` does not provide much feedback to the observer, unless you have a scope to observe the output pins of the GTM / PWM output. Therefore, an example project has been created for this Application Note. You can download the example project Zip file from this link. It contains some changes and additions to the original example code of Infineon.

These changes incorporate LED which provide an optical feedback on the behavior of the main loop (heartbeat about every 2 seconds) and the interrupt service routine for GTM TOM which controls the PWM duty cycle. This LED blinks with 1/1000 of the interrupt frequency (about 10Hz).

It is recommended to study the source code in

`Drive\workspace\TC23x_iLLD_Demo\Src\AppSw\Tricore\Demo_Illd`

The code is liberally commented to get a better understanding of the functions involved.

Source modules added and customized

- Additional source modules:

  ```
  <project root>\TC237_IFX_iLLD_PwmDemo_Pin_Toggles.c
  <project root>\TC237_IFX_iLLD_PwmDemo_Pin_Toggles.h
  ```

  This module and header file defines the I/O pin Toggles for LED and pins to be observed with a scope to see the signal level. They are implemented in a portable manner for easy change of the target hardware which may have other pins available for that purpose.

  The distributed sample code uses the TC237 AppKit. If you use a different board you will have to check availability and pin assignments using the Board User Manual

- The main application for the demo itself:

  `<project>\Src\AppSw\Tricore\Demo_Illd\GtmTomPwmHlDemo.c`

  This module was modified by adding additional duty cycles, introducing the STM Timer Ticker and interrupt service routine and adding the LED Toggles as well as the scoping pin outputs to check Interrupt service routine timing.

  The Demo will change the PWM Duty cycle in intervals of a few seconds from 0% to 100% in multiple steps.

  There is no need for modification of this part as all TC2x series use the same GTM and identical registers to control it.