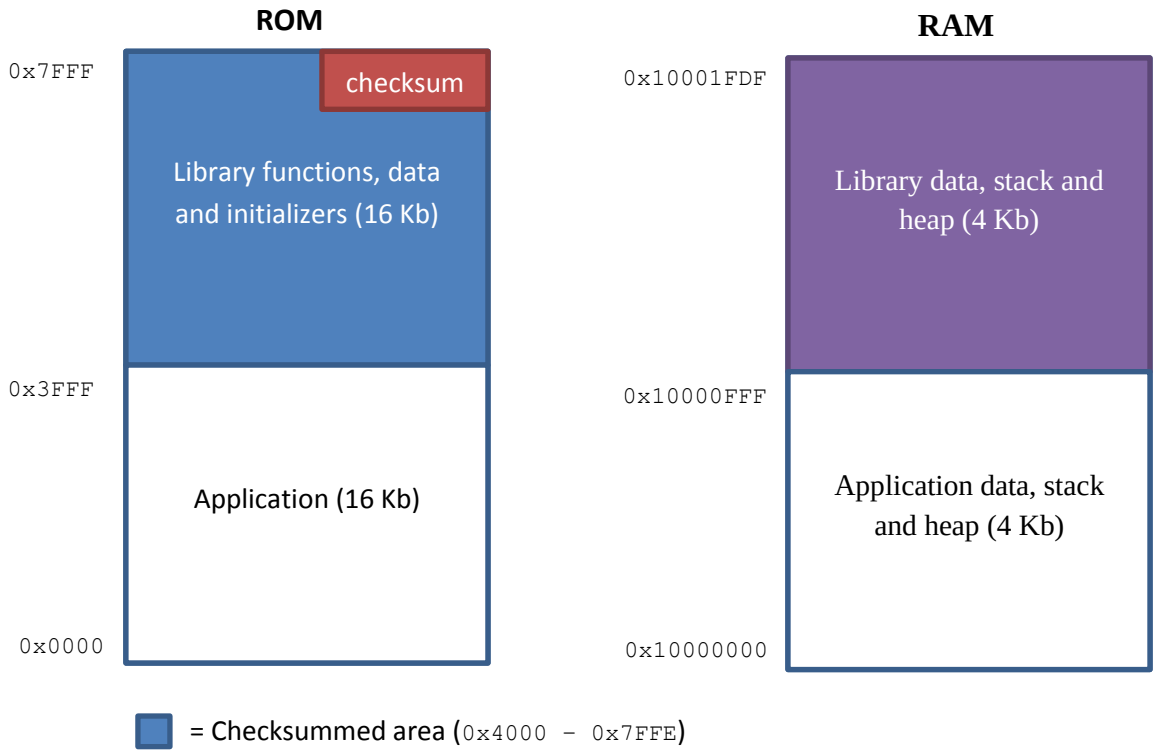
Creating an absolutely placed, checksum-protected library using IAR Embedded Workbench for ARM (Cortex-M3)

This article shows how to create an absolutely placed library (functions and data), that can be integrity-checked using a checksum. The idea is that the library can be separately verified and possibly certified once, and later on used by other applications. The library is compiled and linked in a separate Embedded Workbench project. The output is one ordinary ELF (or HEX) file, and one output file containing the exported symbols. The symbols are exported using the "isymexport"-tool, described in the C/C++ Development Guide, chapter "The IAR Absolute Symbol Exporter - isymexport".

The image below shows how the library is placed in ROM and RAM, and how it is separated from the application.

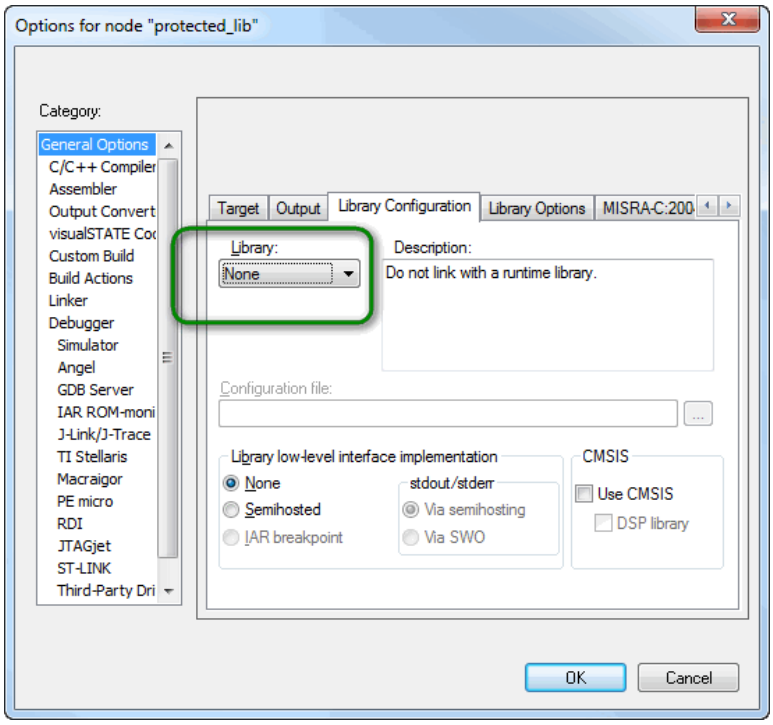


Creating the Library

1. Create a project for the **library** (functions and data). Note that Options -> Output should be set to "Executable" (i.e. this is not a Library project).

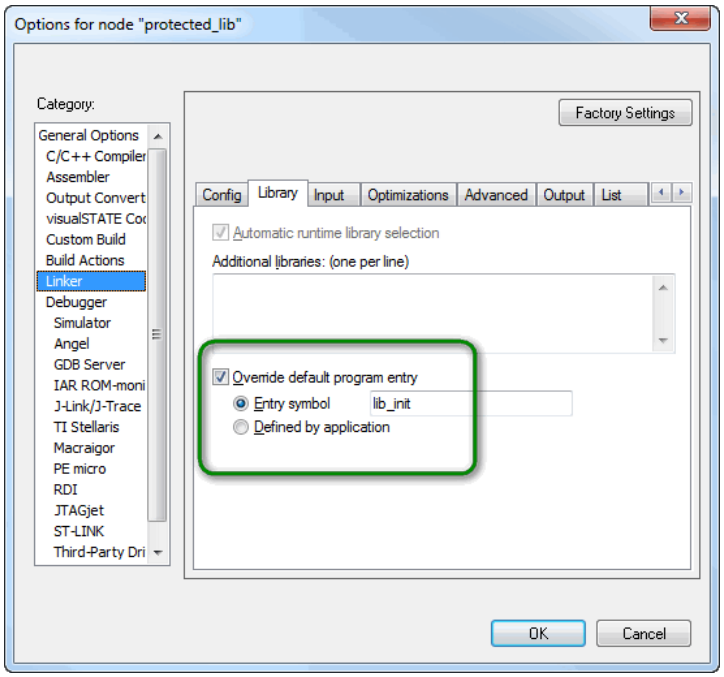
2. Configure the target device (Cortex-M3).

3. Configure the linker to use an address range separate from the application.
   In this example project, the library uses the range 0x4000 to 0x7FFF. See the linker configuration file "`protected_lib.icf`".
4. Select General Options -> Library Configuration -> Library: None



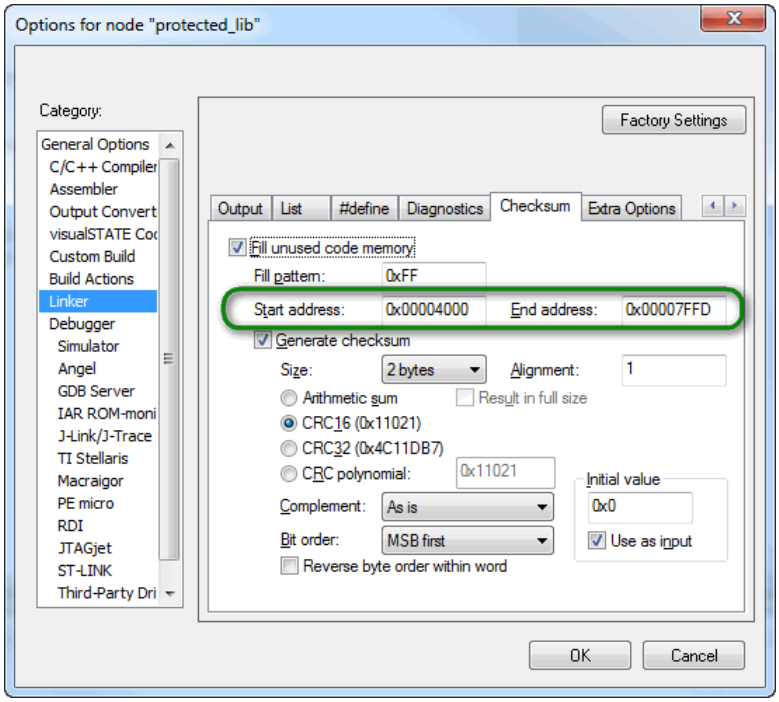
5. Create a `lib_init()` function, for the C initialization. This function will copy the initial values for static and global initialized variables from ROM to RAM, and initialize zero-initialized data to 0. This is done by calling the "`__iar_data_init3`" function, provided by the C-files in "`<EWARM>\arm\src\lib\init`". In the example code, see the file "`lib_func.c`".

6. Set the default program entry to "`lib_init`" in Linker -> Library options.



7. Make sure to add the "`__root`" keyword to the library functions and data, so that they are not removed from the linked output file (since the functions are not used by the library itself). In this example project, see the files "`lib_func.c`" and "`lib_data.c`". (It is also possible to use the linker option "`--no_remove`" to keep all symbols in the library).

8. Enable the checksum option in the linker options (CRC16 with range 0x4000 to 0x7FFD).



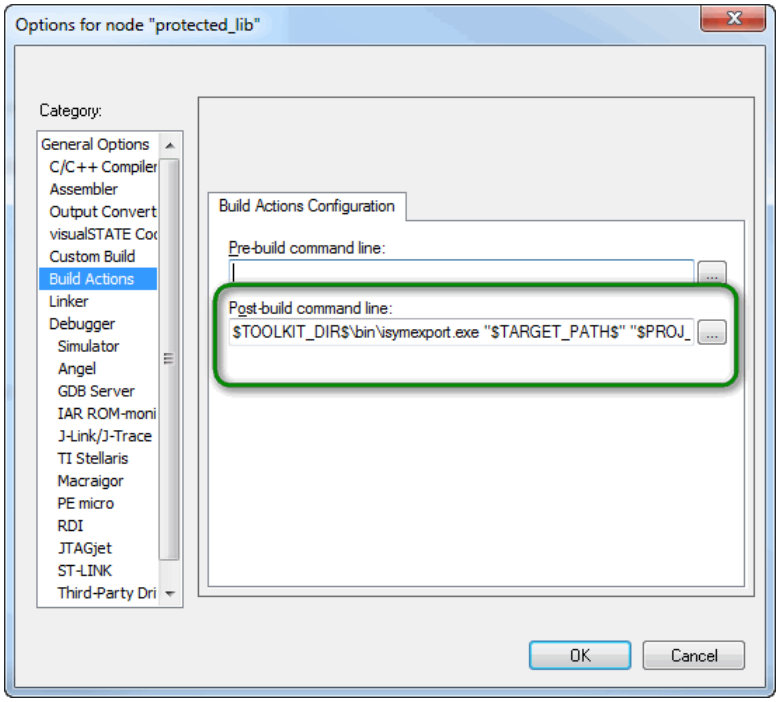
9. Place the checksum at the end of the ROM region (i.e. address 0x7FFE), using "`place at end of ROM_region`" and "`keep {section .checksum}`" in the linker configuration file. Note that it is important that the checksum value itself is not placed inside the checksummed area. (Therefore, the calculation range stopped at 0x7FFD in the previous step).

```
"CHECKSUM":
place at end of ROM_region { ro section .checksum };
keep { section .checksum };
```

10. Create an isymexport steering file that specifies which symbols that are included in the isymexport output file. It is important not to export all symbols, especially the "`__iar_data_init3`" and other compiler-specific ("`__iar*`") functions may otherwise cause conflicts with the application later on.
    In this example, the steering file is called "`sym_export.txt`" and contains the following (i.e. only the `lib_` and `__checksum` symbols are exported):

```
show lib_*
show __checksum*
```

11. Add the export of library symbols in Build Actions -> Post-build command line:
    `$TOOLKIT_DIR$\bin\isymexport.exe "$TARGET_PATH$" "$PROJ_DIR$\protected_lib.symbols" --edit "$PROJ_DIR$\sym_export.txt"`



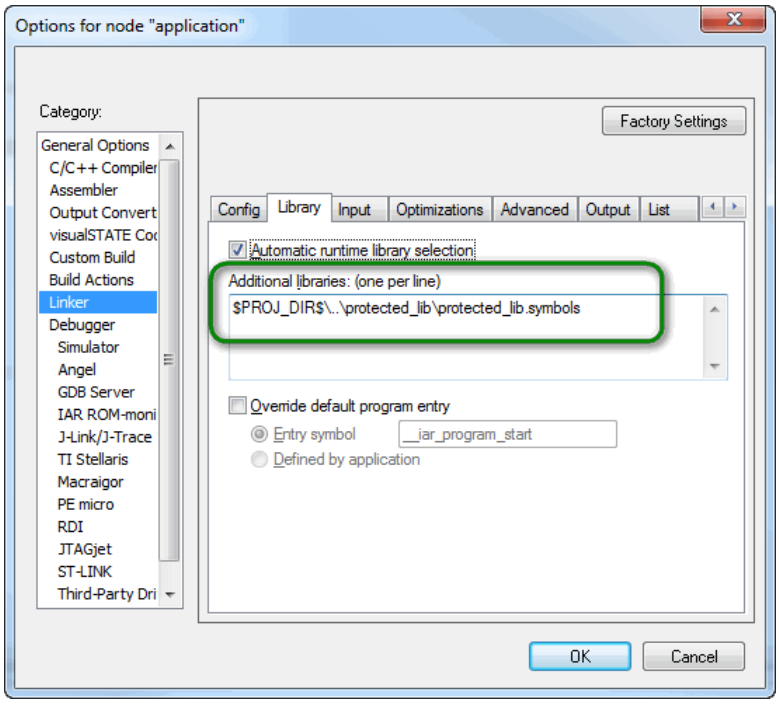
Creating the Application

1. Create a project for the **application**.

2. Configure the target device (Cortex-M3).

3. Configure the linker to use an address range separate from the address range of the library. In this example project, the application uses the range 0x0000 to 0x3FFF. See the linker configuration file "`application.icf`".

4. Add the exported library symbols to Options -> Linker -> Library -> Additional libraries: `$PROJ_DIR$\..\protected_lib\protected_lib.symbols`

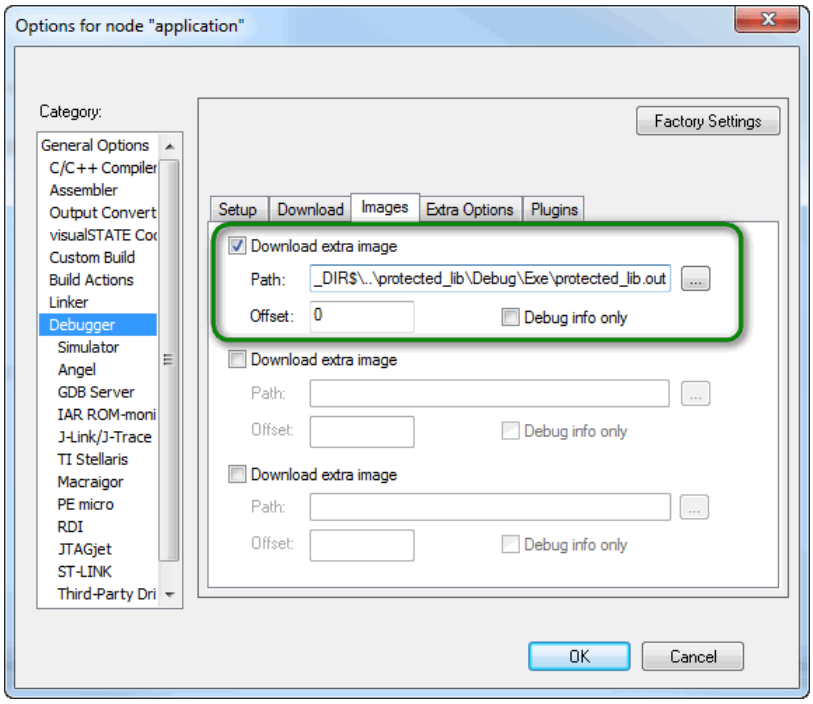


5. In the application's main function, check the value of the `__checksum` variable in the library. In this example project, see the "`main.c`" file.

6. In the application's main function, make sure to call "`lib_init`" to initialize the data in the library. In this example project, see the "`main.c`" file.

7. If the flash (where the library is located) is supported by an EWARM flash loader and debug probe, you can download the library to the target device (needed at least once) by adding the output file to Options -> Debugger -> Images -> Download extra image -> Path: `$PROJ_DIR$\..\protected_lib\Debug\Exe\protected_lib.out`

   (Note that for some devices, you may need to download the library ELF or HEX file separately).



Conclusion

Using the settings above, and the example project called “application”, it is now possible to debug the application and library using the C-SPY Debugger. The linker map file for the application shows the absolute location of the `__checksum` variable (0x7FFE), and also the library functions and data. Verify that the library functions are separated from the application (using the address range 0x4000 to 0x7FFF).

After verification and certification of the library has been performed, the checksum ensures that the exact same library code is used (by possibly different applications).

application.map

```
SVC_Handler              0x000000e3        Code  Wk  vector_table_M.o [5]
SysTick_Handler          0x000000e3        Code  Wk  vector_table_M.o [5]
UsageFault_Handler       0x000000e3        Code  Wk  vector_table_M.o [5]
__checksum {Abs}         0x00007ffe   0x2  Data  Gb  protected_lib.symbols [2]
__checksum_begin {Abs}   0x00004000        --  Gb  protected_lib.symbols [2]
__checksum_end {Abs}     0x00007ffd        --  Gb  protected_lib.symbols [2]
__checksum_value {Abs}   0x0000f7c2        Data  Gb  protected_lib.symbols [2]
__cmain                  0x000000e5        Code  Gb  cmain.o [5]
__exit                   0x00000111  0x14  Code  Gb  exit.o [6]
__iar_program_start      0x00000125        Code  Gb  cstartup_M.o [5]
__iar_rom_use_HHKwmbLsxZ9 {Abs}
                         0x00000028        --  Gb  protected_lib.symbols [2]
__low_level_init         0x000000fb   0x4  Code  Gb  low_level_init.o [4]
__vector_table           0x00000000        Data  Gb  vector_table_M.o [5]
_call_main               0x000000f1        Code  Gb  cmain.o [5]
_exit                    0x00000105        Code  Gb  cexit.o [5]
_main                    0x000000f7        Code  Gb  cmain.o [5]
exit                     0x000000ff   0x4  Code  Gb  exit.o [4]
lib_data_arr_ram {Abs}   0x10001000  0x28  Data  Gb  protected_lib.symbols [2]
lib_data_arr_rom {Abs}   0x00004000  0x28  Data  Gb  protected_lib.symbols [2]
lib_init {Abs}           0x0000402d  0x18  Code  Gb  protected_lib.symbols [2]
lib_test_func {Abs}      0x00004029   0x4  Code  Gb  protected_lib.symbols [2]
main                     0x00000041  0x68  Code  Gb  main.o [1]
slow_crc16               0x000000a9  0x3a  Code  Gb  slow_crc16.o [1]
```

Notes

a) Note that it is not necessary to select “Library Configuration -> Library: None” in the library project. If you wish to use a C runtime library, it is possible to do so. Setting the Library to “None” ensures that you do not get any runtime library code in your project.

b) As a general recommendation, the library project should not contain static and global initialized variables. If the library project does not contain static and global initialized variables, there is no need for the “lib_init” C initialization copy routines (and the project is simpler to create).