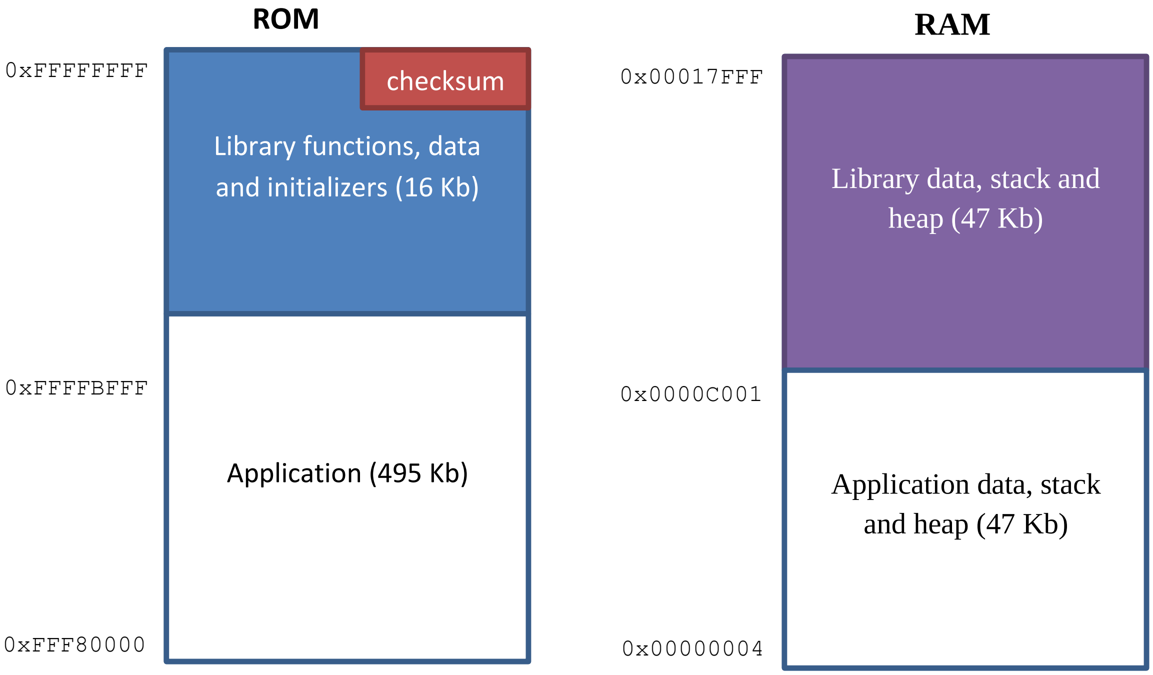
Creating an absolutely placed, checksum-protected library using IAR Embedded Workbench for RX

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.


ROM
0xFFFFFFFF
checksum
Library functions, data and initializers (16 Kb)
0xFFFFBFFF
Application (495 Kb)
0xFFF80000
RAM
0x00017FFF
Library data, stack and heap (47 Kb)
0x0000C001
Application data, stack and heap (47 Kb)
0x00000004

🟦 = Checksummed area (0xFFFFC000 – 0xFFFFFF7D)

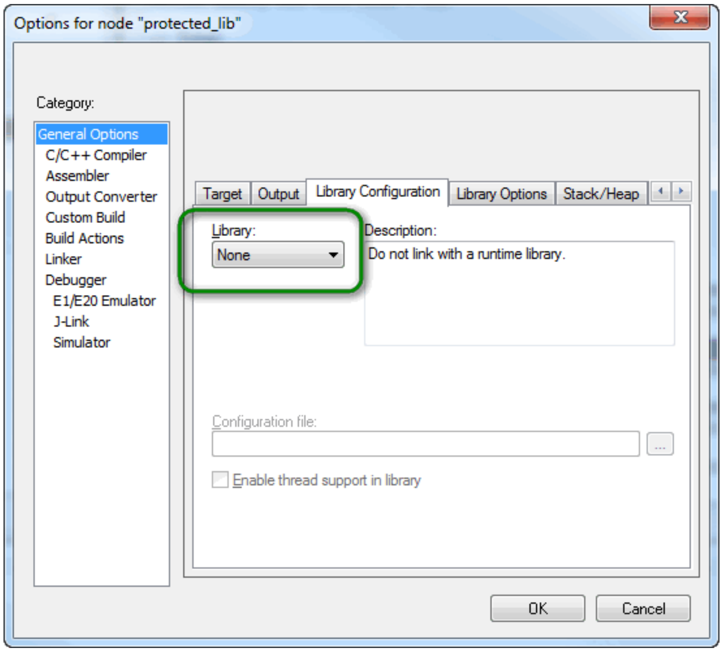
Creating the Library

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 (RX62N > R5F562N8).

3. Configure the linker to use an address range separate from the application.
   In this example project, the library uses the range 0xFFFFC000 to 0xFFFFFFFF. See the linker configuration file "`lnkr5f562n8_lib.icf`".

4. Select General Options -> Library Configuration -> Library: None



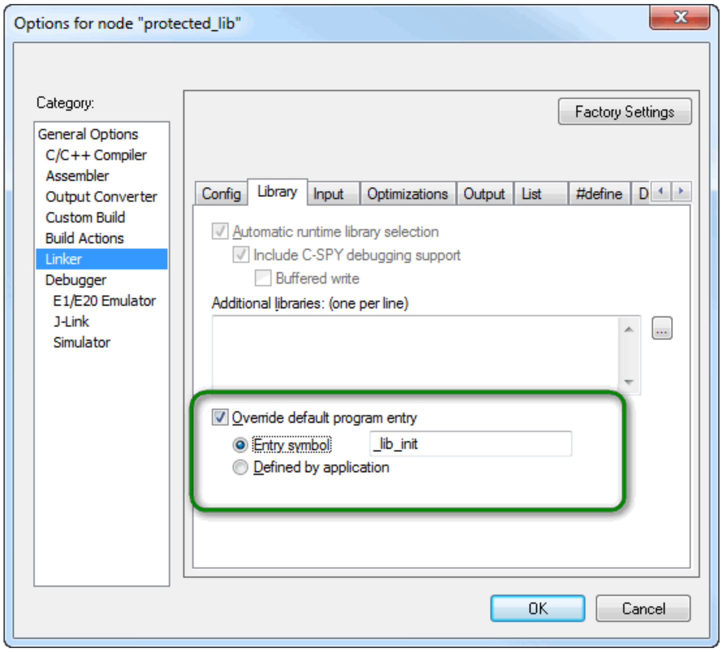
Library: None

Description: Do not link with a runtime library.

□ Enable thread support in library

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_init2`" function, provided by the C-files in "`<EWRX>\rx\src\lib\rx`". In the example code, see the file "`lib_func.c`".

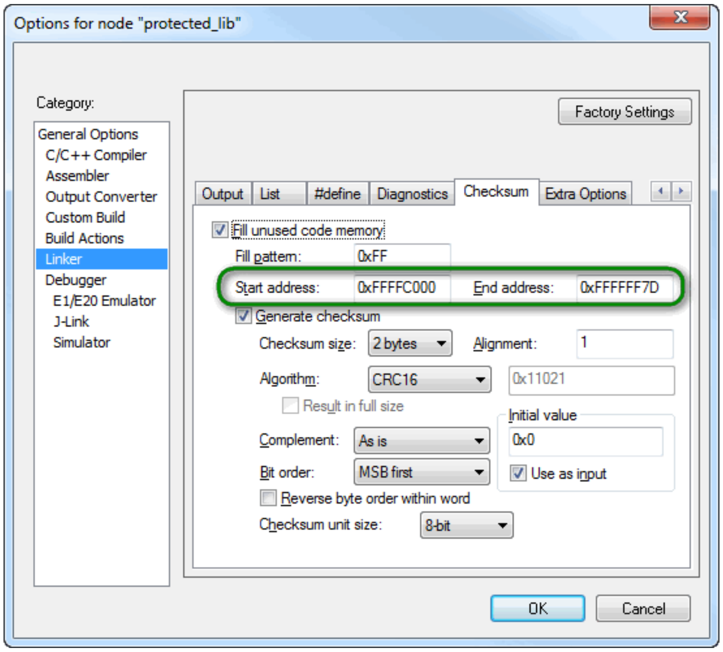
6. Set the default program entry to "`lib_init`" in Linker -> Library options.



☑ Automatic runtime library selection
☑ Include C-SPY debugging support
□ Buffered write
Additional libraries: (one per line)
☑ Override default program entry
● Entry symbol _lib_init
○ Defined by application

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).

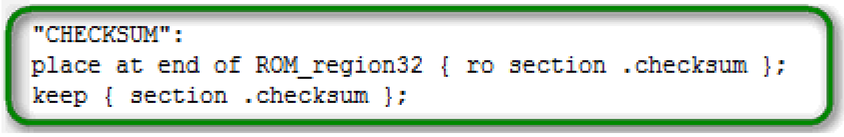
8. Enable the checksum option in the linker options (CRC16 with range 0xFFFFC000 to 0xFFFFFF7D).



☑ Fill unused code memory
Fill pattern: 0xFF
Start address: 0xFFFFC000 End address: 0xFFFFFF7D
☑ Generate checksum
Checksum size: 2 bytes Alignment: 1
Algorithm: CRC16 0x11021
□ Result in full size
Initial value 0x0
Complement: As is
Bit order: MSB first ☑ Use as input
□ Reverse byte order within word
Checksum unit size: 8-bit

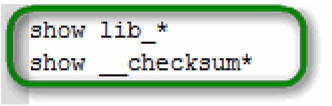
9. Place the checksum at the end of the ROM region (i.e. address 0xFFFFFF7E), using "`place at end of ROM_region32`" and "`keep {section .checksum}`" in the linker configuration file. Note

that it is important that the checksum value itself is not placed inside the checksummed area. (Therefore, the calculation range stopped at 0xFFFFFF7D in the previous step).



```
"CHECKSUM":
place at end of ROM_region32 { ro section .checksum };
keep { section .checksum };
```

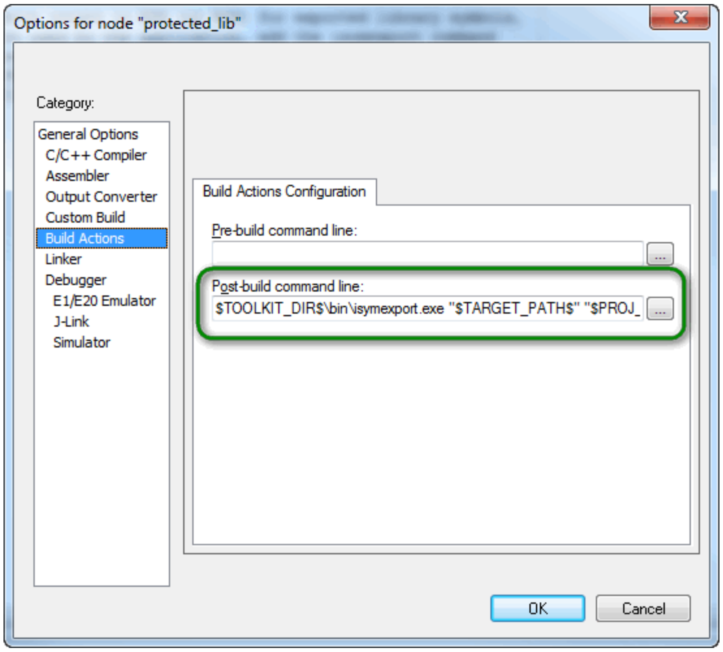
10. Create an isymexport steering file that specifies which symbols that are included in the isymexport output file. It is important not to export all symbols, especially the "`__iar_data_init2`" and other compiler-specific ("`__iar*`") functions may otherwise cause conflicts with the application later 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"
```



Options for node "protected_lib"

Pre-build command line:

Post-build command line: $TOOLKIT_DIR$\bin\isymexport.exe "$TARGET_PATH$" "$PROJ_

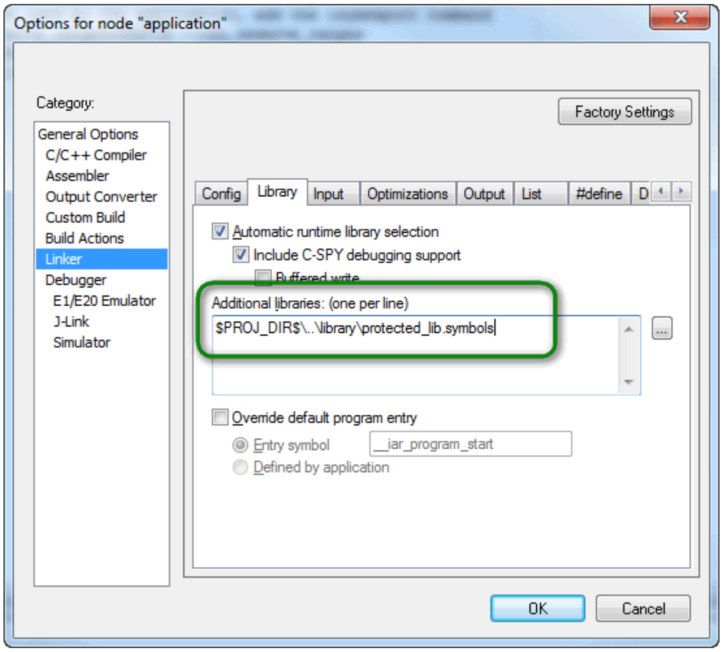
Creating the Application

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

2. Configure the target device (RX62N > R5F562N8).

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 0xFFF80000 to 0xFFFFBFFF. See the linker configuration file “`lnkr5f562n8_app.icf`”.

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



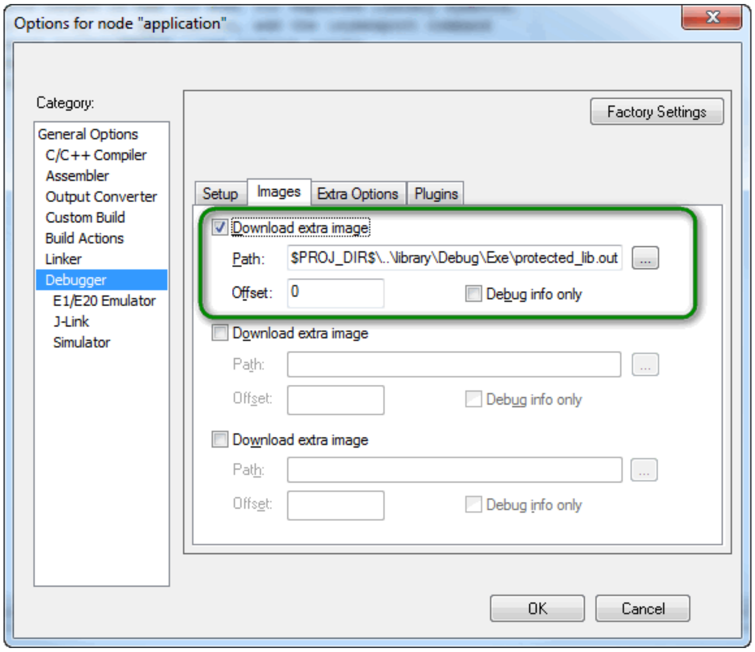
☑ Automatic runtime library selection
☑ Include C-SPY debugging support
□ Buffered write
Additional libraries: (one per line)
```
$PROJ_DIR$\..\library\protected_lib.symbols
```
□ Override default program entry
● Entry symbol __iar_program_start
○ Defined by application

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. 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:
   `$PROJ_DIR$\..\library\Debug\Exe\protected_lib.out`

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



☑ Download extra image
Path: $PROJ_DIR$\..\library\Debug\Exe\protected_lib.out
Offset: 0
□ Debug info only

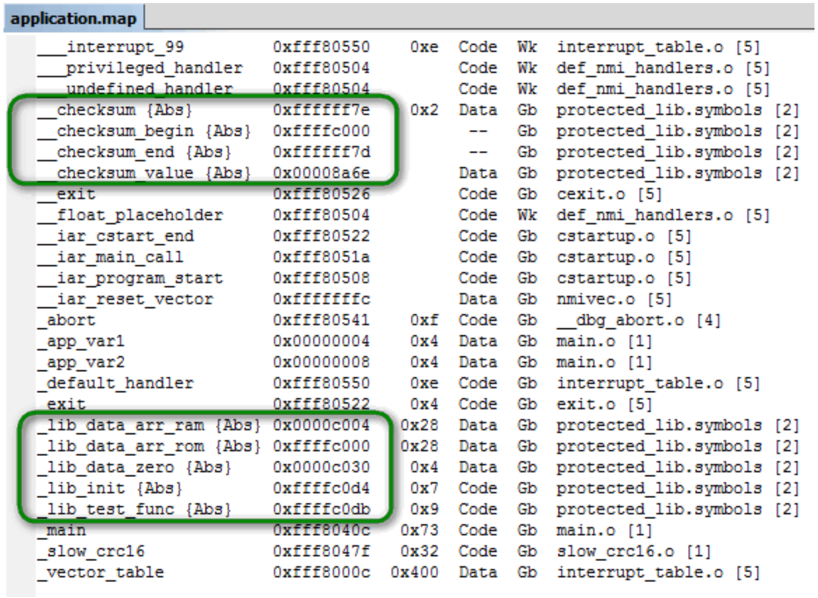
□ Download extra image
Path:
Offset:
□ Debug info only

□ Download extra image
Path:
Offset:
□ Debug info only

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 (0xFFFFFF7E), and also the library functions and data. Verify that the library functions are separated from the application (using the address range 0xFFFFC000 to 0xFFFFFF7D).

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).



```
 ___interrupt_99         0xfff80550   0xe  Code  Wk  interrupt_table.o [5]
 ___privileged_handler   0xfff80504        Code  Wk  def_nmi_handlers.o [5]
 ___undefined_handler    0xfff80504        Code  Wk  def_nmi_handlers.o [5]
 __checksum {Abs}        0xffffff7e   0x2  Data  Gb  protected_lib.symbols [2]
 __checksum_begin {Abs}  0xffffc000         --   Gb  protected_lib.symbols [2]
 __checksum_end {Abs}    0xffffff7d         --   Gb  protected_lib.symbols [2]
 __checksum_value {Abs}  0x00008a6e        Data  Gb  protected_lib.symbols [2]
 __exit                  0xfff80526        Code  Gb  cexit.o [5]
 __float_placeholder     0xfff80504        Code  Wk  def_nmi_handlers.o [5]
 __iar_cstart_end        0xfff80522        Code  Gb  cstartup.o [5]
 __iar_main_call         0xfff8051a        Code  Gb  cstartup.o [5]
 __iar_program_start     0xfff80508        Code  Gb  cstartup.o [5]
 __iar_reset_vector      0xfffffffc        Data  Gb  nmivec.o [5]
 _abort                  0xfff80541   0xf  Code  Gb  __dbg_abort.o [4]
 _app_var1               0x00000004   0x4  Data  Gb  main.o [1]
 _app_var2               0x00000008   0x4  Data  Gb  main.o [1]
 _default_handler        0xfff80550   0xe  Code  Gb  interrupt_table.o [5]
 _exit                   0xfff80522   0x4  Code  Gb  exit.o [5]
 _lib_data_arr_ram {Abs} 0x0000c004  0x28  Data  Gb  protected_lib.symbols [2]
 _lib_data_arr_rom {Abs} 0xffffc000  0x28  Data  Gb  protected_lib.symbols [2]
 _lib_data_zero {Abs}    0x0000c030   0x4  Data  Gb  protected_lib.symbols [2]
 _lib_init {Abs}         0xffffc0d4   0x7  Code  Gb  protected_lib.symbols [2]
 _lib_test_func {Abs}    0xffffc0db   0x9  Code  Gb  protected_lib.symbols [2]
 _main                   0xfff8040c  0x73  Code  Gb  main.o [1]
 _slow_crc16             0xfff8047f  0x32  Code  Gb  slow_crc16.o [1]
 _vector_table           0xfff8000c 0x400  Data  Gb  interrupt_table.o [5]
```

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).