

针对使用 NXP GCC10.2 编译程序的 优化操作方法

Peter Chen (Peter_Chen@wtmec.com)

1. 现状

就 2024 年年底而言，NXP 提供给客户用于 S32K3 开发软件是 S32DS for S32 Platform + RTD。区别如下：

Table 1. S32 Design Studio for S32 Platform 各版本区别

版本	S32K1 SDK 开发	S32K1/3 RTD 开发
3.4	✓	
3.5	✓	✓
3.6		✓

Table 2. Real Time Driver 各版本区别

版本	状态
1.0.0	版本旧，支持芯片有限，建议弃用
2.0.0	支持 3X4 版本，其他支持有问题，建议弃用
3.0.0	支持 8M 以外的版本，本身存在 Bug，需要修正以后才能使用
4.0.0	主力版本，建议使用
5.0.0	新版本，尚缺 Crypto 组件

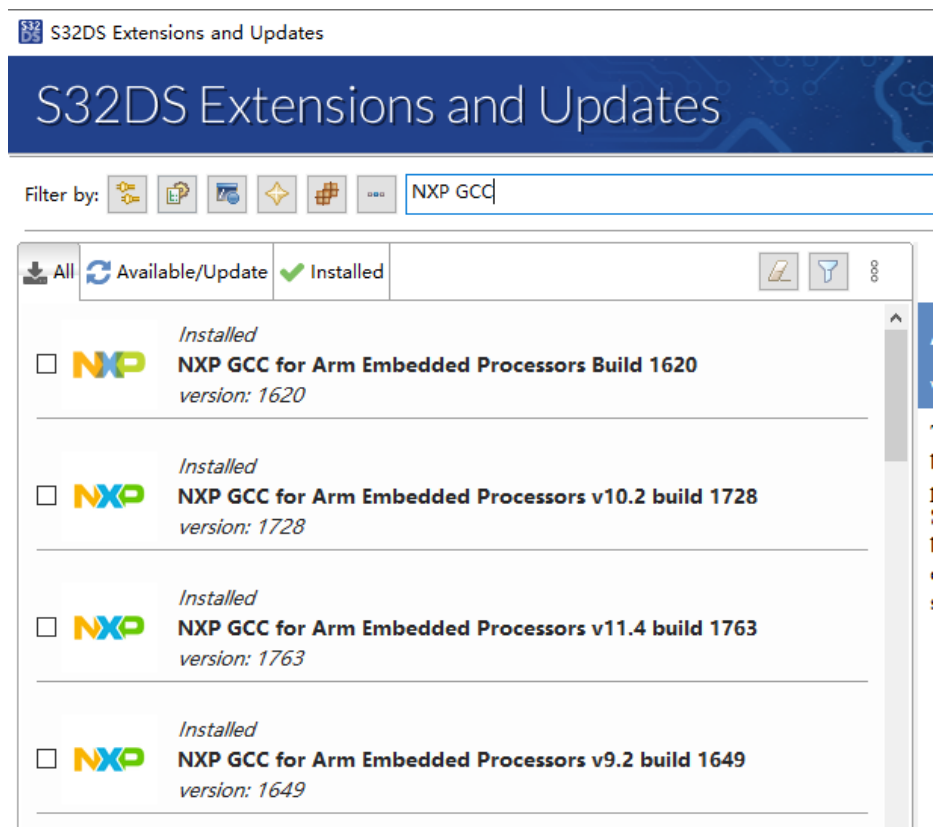
本文以 S32DS for S32 Platform 3.5 + RTD 4.0.0 作为例子基础展示优化方法。

从 IDE 附带的包管理界面（扩展与更新）中可以安装的 GCC 有以下 4 个版本，如图 1，对比如下：

Table 3. NXP GCC 各版本区别

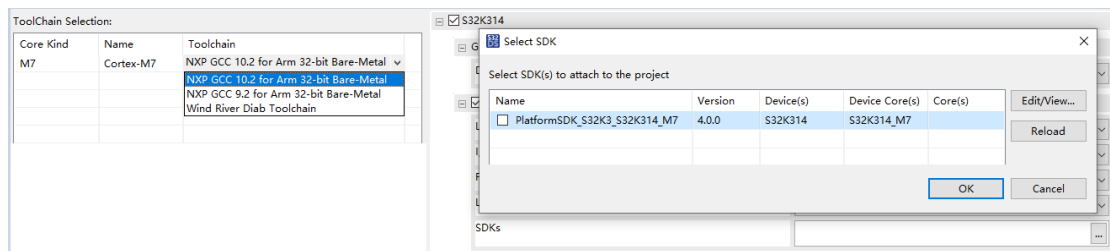
版本	说明
6.3.1 build 1620	用于编译 S32K1 SDK 4.x 版本
9.2 build 1649	用于编译 S32K3 RTD 1.0.0 版本
10.2 build 1728	主力版本，用于编译 S32K3 RTD 2.0.0 – 5.0.0 版本
11.4 build 1763	新版本，尚未与特定版本 RTD 相关联

图 1. S32DS for S32 Platform 自带的包管理软件



现在开发 S32K3 基于 RTD 的程序，一般是使用 v10.2 build 1728 版本的 GCC 编译器。因为如果使用 GCC 开发程序，不选择这个版本的 GCC，在新建工程的界面里无法选择 RTD 库，如下图：

图 2. 新建工程时，只有选择 GCC10.2 才能在 SDK 选择中选择 RTD 版本



现在使用 IDE 自带的样例 Siul2_Port_Ip_Example_S32K344 作为演示。为排除由于优化等级选项导致的变量，演示中将优化等级关闭。

1. 新建一个样例 Siul2_Port_Ip_Example_S32K344，关闭默认的针对大小优化的选项（图 3）以后，编译并查看输出的代码量。大约为 43K 左右（图 4）。

图 3. 关闭工程的优化选项

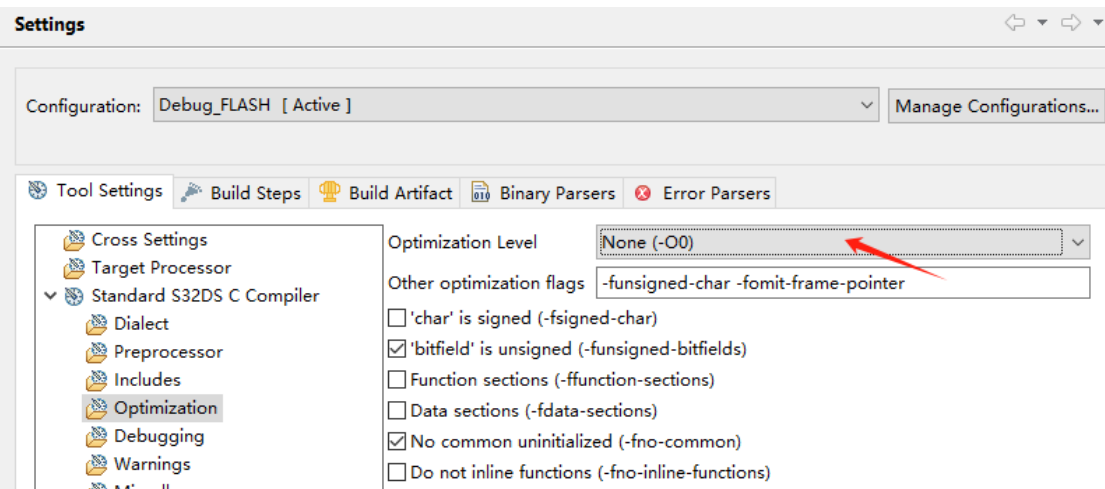


图 4. 默认设置的编译结果

```
Invoking: Standard S32DS Print Size
arm-none-eabi-size --format=berkeley Siul2_Port_Ip_Example_S32K344.elf
text      data      bss      dec      hex filename
43908      4      10944      54856      d648 Siul2_Port_Ip_Example_S32K344.elf
Finished building: Siul2_Port_Ip_Example_S32K344.siz
```

2. 在 Debug 配置中，一般不建议打开优化选项。如果打开优化选项，当你在调试时，会有变量无法查看中间结果，程序的运行也会与 C 代码的顺序有所出入，这样不便于调试。接下来按照传统的去除无用代码的方法，打开编译器中的 -ffunction-sections 选项（图 5）以及链接器中的 --gc-sections 选项（图 6）。然后编译并查看结果，代码量大约变成 21K 左右（图 7）。

图 5. 编译器的优化选项中打开 -ffunction-sections 选项

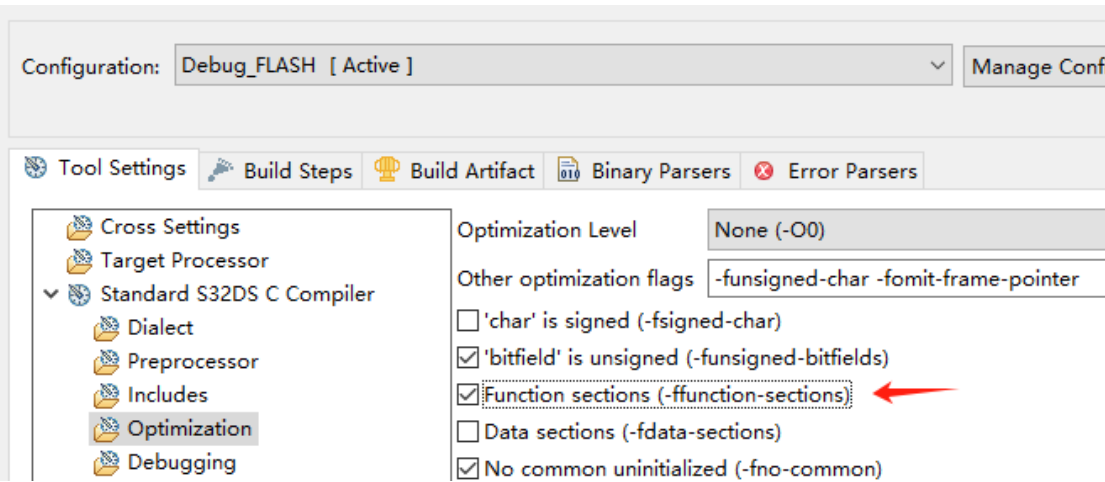


图 6. 链接器的杂项中打开 --gc-sections 选项

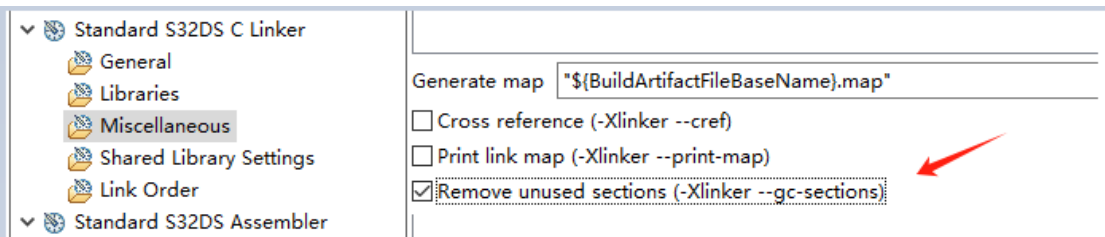
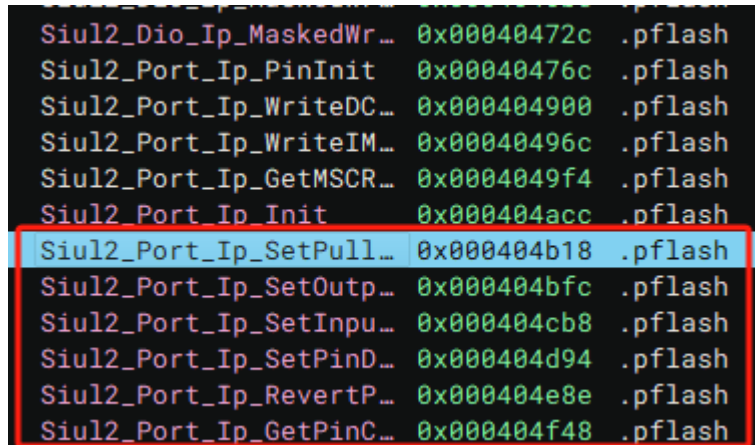


图 7. 重新编译以后的结果

```
Invoking: Standard S32DS Print Size
arm-none-eabi-size --format=berkeley Siul2_Port_Ip_Example_S32K344.elf
text      data      bss      dec      hex filename
21096      8      10608    31712    7be0 Siul2_Port_Ip_Example_S32K344.elf
Finished building: Siul2_Port_Ip_Example_S32K344.siz
```

3. 通过反汇编程序查看可知，此时仍然有很多未被使用的函数被编译进了最终的 elf 文件中，如下图：

图 8. 使用反汇编工具检查生成物中，仍然有未被使用的代码



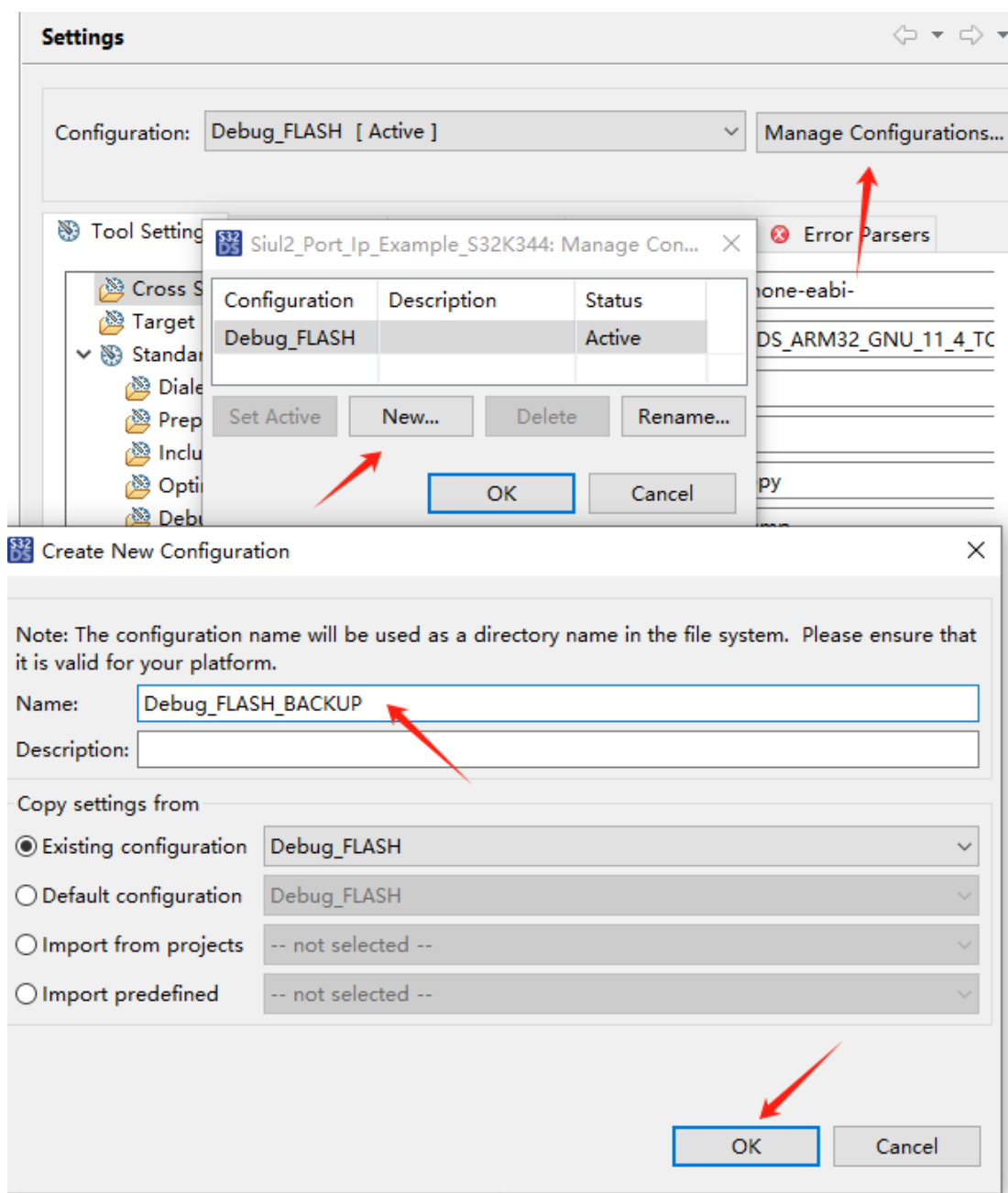
Function Name	Address	Section
Siul2_Dio_Ip_MaskedWr...	0x00040472c	.pflash
Siul2_Port_Ip_PinInit	0x00040476c	.pflash
Siul2_Port_Ip_WriteDC...	0x000404900	.pflash
Siul2_Port_Ip_WriteIM...	0x00040496c	.pflash
Siul2_Port_Ip_GetMSCR...	0x0004049f4	.pflash
Siul2_Port_Ip_Init	0x000404acc	.pflash
Siul2_Port_Ip_SetPull...	0x000404b18	.pflash
Siul2_Port_Ip_SetOutp...	0x000404bfc	.pflash
Siul2_Port_Ip_SetInpu...	0x000404cb8	.pflash
Siul2_Port_Ip_SetPinD...	0x000404d94	.pflash
Siul2_Port_Ip_RevertP...	0x000404e8e	.pflash
Siul2_Port_Ip_GetPinC...	0x000404f48	.pflash

4. 通过查看 map 文件可知，原因是在于 NXP GCC v10.2 中对于 GCC 编译器的优化选项 -ffunction-sections 并未实现，从而导致段的最小单位是文件。一个 C 文件中所有的函数都会编译在一个 text 段中，只要其中有一个函数被调用过，那么链接器的 --gc-sections 就不会将这个段删除，从而增加了空间占用。加上 RTD 的代码中会把大量函数实现放在为数不多的几个文件中，这样也加剧了这种情况。

2. 优化方法

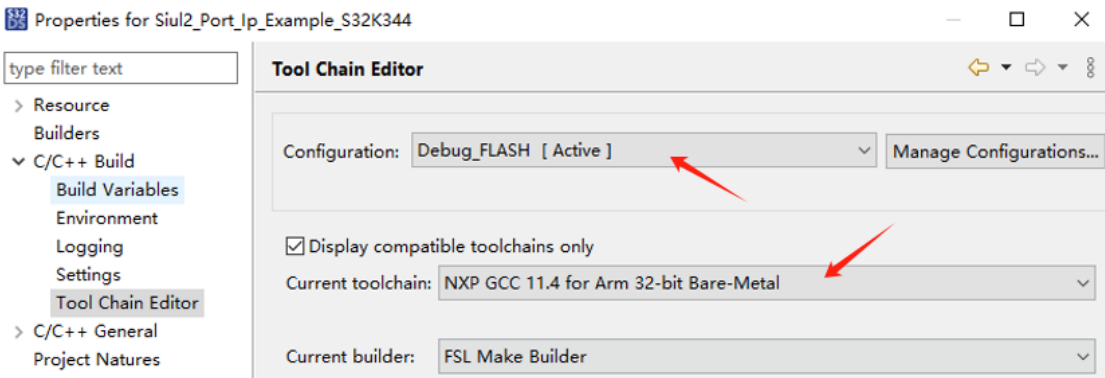
1. -ffunction-sections 选项在 NXP GCC v11.4 build 1763 中已经实现了功能。第一步需要安装 v11.4 版本的编译器。在 S32DS 3.5 中可以通过 IDE 自带的包管理工具在线安装，在 S32DS 3.6 中该版本编译器是随安装包一起安装的。
2. 安装完以后，需要更改项目的编译器。在更改之前需要先备份设置里的配置内容，更改编译工具会重置配置。可以使用图 9 的方法复制一份配置。

图 9. 从现有的配置复制并新建一个备份配置



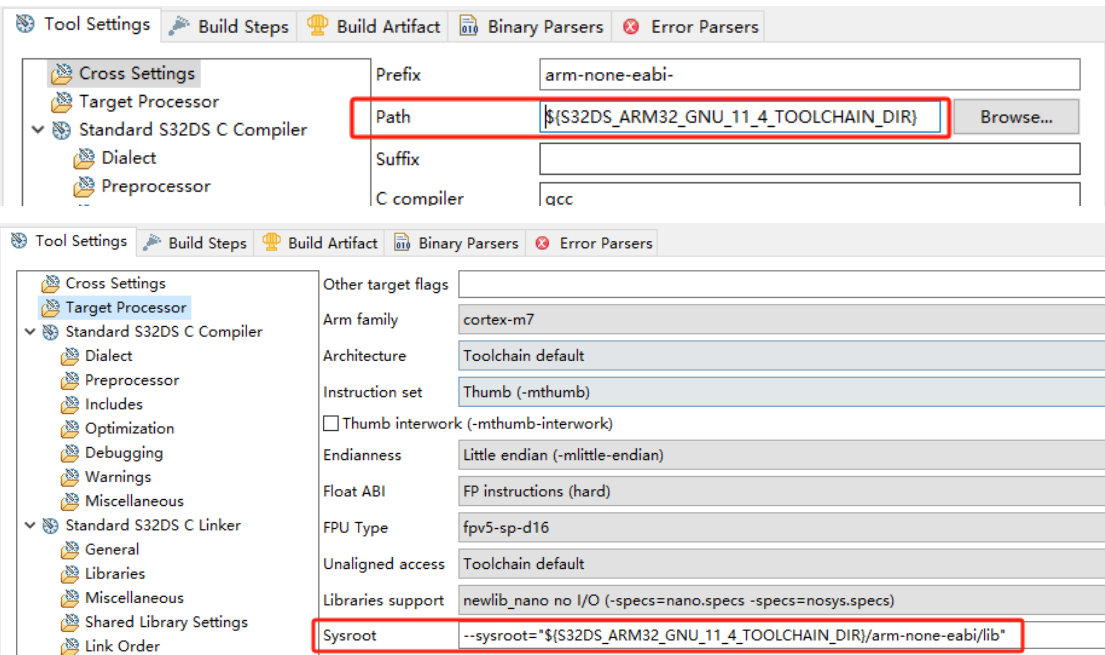
3. 在 Tool Chain Editor 设置中，选择原始配置，并将当前编译器选择为 NXP GCC 11.4，如下图：

图 11. 将编译器选择为 11.4 版本



4. 回到设置中，通过切换现有配置和之前备份的配置，将重置的配置根据需要进行恢复。但是 Cross Settings 不需要修改，Target Processor 最后一项 Sysroot 中的 `$(S32DS_K3_ARM32_GNU_10_2_TOOLCHAIN_DIR)`，需要修改成 Cross Settings 中 Path 的设置，`$(S32DS_ARM32_GNU_11_4_TOOLCHAIN_DIR)`

图 12. Cross Settings 中的 Path 是编译器的路径，Sysroot 中的设定需要手动更新



5. 重新编译以后，发现代码变成了 13K 左右，如下图：

图 13. 删除无用代码以后的代码大小

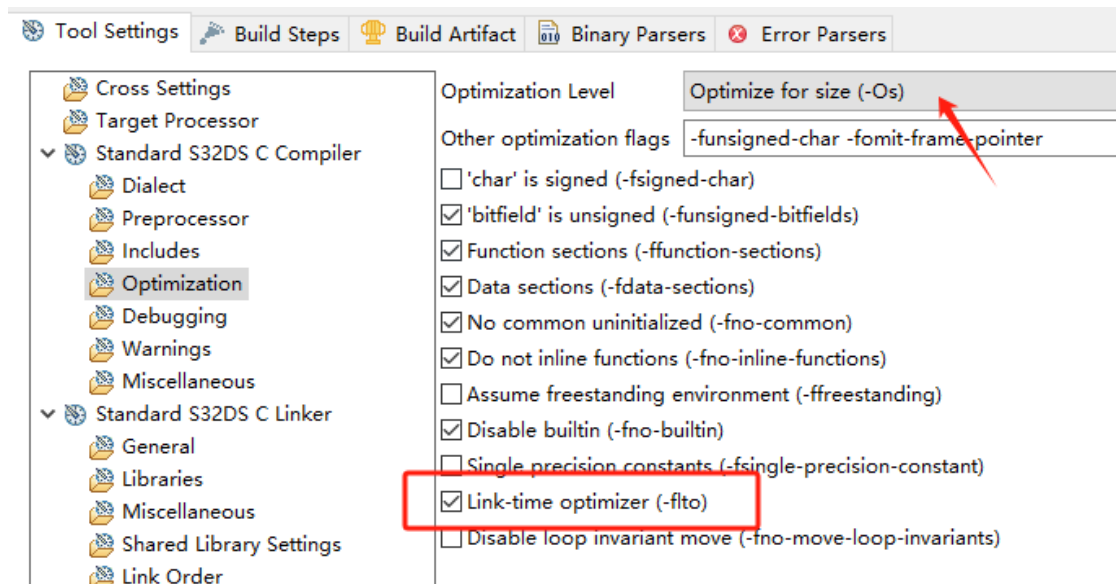
```
Invoking: Standard S32DS Print Size
arm-none-eabi-size --format=berkeley Siul2_Port_Ip_Example_S32K344.elf
text    data    bss    dec    hex filename
13532    4      9712  23248  5ad0 Siul2_Port_Ip_Example_S32K344.elf
Finished building: Siul2_Port_Ip_Example_S32K344.siz
```

这时打开 map 文件可以看到，原来段名，比如.mcal_text，已被修改成了.mcal_text.[函数名]。编译器在编译时给每一个函数都建立了一个段，这样在链接时 gc-sections 就可以以函数的粒度来删除没有调用的函数，而不是以文件为最小粒度来处理了。

6. 因为段的名称更改了。所以对应的 ld 文件内的段名也需要跟着更改。.mcal_text 段都变成了.mcal_text.[函数名]段，因此需要在 ld 文件中的*(.mcal_text)后新增一行*(.mcal_text.*)用于包含所有.mcal_text 段。需要检查生成的 map 文件中的每一个段的位置，用以确定是否需要在 ld 文件中使用相同的方法增加新的更改。

```
80      *(.text)
81      *(.text*)
82      . = ALIGN(4);
83      *(.mcal_text)
84      *(.mcal_text.*)
85      . = ALIGN(4);
```

3. 最大优化



最终生产时如果空间不够的情况下，可以将优化等级设置成针对 size 优化。打开优化以后可以将代码减小到 12K 左右。

```
[39/39] Linking C executable Port_Example_S32K344.elf
text    data    bss    dec    hex filename
12008    4    9680    21692    54bc Port_Example_S32K344.elf
```

打开 Link-time optimizer 的话可以开启最大优化，但是由于现在 NXP 提供的 GCC 编译器 v11 对于 Link-time optimizer 支持还有问题。我制作了一个 GCC 的插件（https://github.com/chenzch/GCC_Section_Plugin）可以应用于标准版的 arm-none-eabi-gcc。打开 -flto 优化以后的代码可以减小到 11.8K 左右。

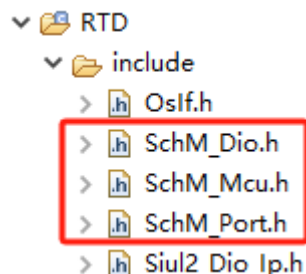
```
text    data    bss    dec    hex filename
11848    4    9672    21524    5414 Port_Example_S32K344.elf
```

但是由于还没找到在 Windows 下编译插件的方法，暂时只能在 Linux 系统中进行编译。

4. RTE 模块中内存使用的优化

这个修改请不要合并到 RTD 的原始安装目录中，避免影响其他项目的编译


1. 在 RTD 代码中使用了很多互锁机制用于代码的安全，这样就需要使用很多的内存用于保存锁状态，这些状态是保存在 RTE 模块中的。为了代码能编译成功，NXP 提供的 RTD 本身也提供了一个简易的 RTE 模块实现。这个模块实现的目的是为了客户能在仅使用 RTD 的情况下能将项目编译成功，所以功能并不完善。最终量产的产品，客户需要使用从第三方购买的 RTE 层代码来替换 RTD 中提供的简易 RTE 模块。
2. 在 RTD 的代码中搜索“#define NUMBER_OF_CORES”会发现，RTE 模块下的以 SchM 开头的头文件中基本都定义为了数字 4，还会发现有 1、2 个文件是按照 2 来定义的。
3. 按照 2 来定义的这些模块是从 0.8.0 开始遗留下来的代码已经被弃用了。这个头文件以及相对应的 c 文件可以一并删除。
4. 按照 4 来定义的模块是现在使用的文件，客户可以根据自己的实际目标芯片修改为实际的内核数量。比如之前例子中使用的 S32K344，实际对软件来说只有一个核心，那么这里的数字 4 就可以修改成 1。将 RTD/include 目录下的 3 个以 SchM 开头的头文件中的 NUMBER_OF_CORES 定义修改成 1 并重新编译以后，BSS 的用量从 9712 下降为 9664。3 个模块下降了 48 个字节，如果模块多的话，下降得会多一些。



```
Invoking: Standard S32DS Print Size
arm-none-eabi-size --format=berkeley Siul2_Port_Ip_Example_S32K344.elf
  text    data    bss     dec     hex filename
 13532     4    9712    23248    5ad0 Siul2_Port_Ip_Example_S32K344.elf
Finished building: Siul2_Port_Ip_Example_S32K344.siz
```

```
Invoking: Standard S32DS Print Size
arm-none-eabi-size --format=berkeley Siul2_Port_Ip_Example_S32K344.elf
  text    data    bss     dec     hex filename
 13532     4    9664    23200    5aa0 Siul2_Port_Ip_Example_S32K344.elf
Finished building: Siul2_Port_Ip_Example_S32K344.siz
```

5. 打开 -fdata-sections 还可以下降几个字节，但是相同的问题是数据段的名称都会被修改，需要大量地修改 ld 文件，实际用量减小的效果并不明显，所以一般不需要实施。

Optimization Level	None (-O0)
Other optimization flags	-funsigned-char -fomit-fr
<input type="checkbox"/> 'char' is signed (-fsigned-char)	
<input checked="" type="checkbox"/> 'bitfield' is unsigned (-funsigned-bitfields)	
<input checked="" type="checkbox"/> Function sections (-ffunction-sections)	
<input checked="" type="checkbox"/> Data sections (-fdata-sections) 	
<input checked="" type="checkbox"/> No common uninitialized (-fno-common)	
<input type="checkbox"/> Do not inline functions (-fno-inline-functions)	
<input type="checkbox"/> ...	

5. Sys_GetCoreID 的优化

这个修改请不要合并到 RTD 的原始安装目录中，避免影响其他项目的编译

1. 在 RTD 代码中有很多地方使用了 Sys_GetCoreID 函数来获取当前内核的 ID 号，从 0-3。当用户在使用单核处理器来做开发的时候，这个函数应该永远返回 0。这种情况下，客户可以直接修改 Oslf_GetCoreID 的定义，将其修改为 0，或者将 Sys_GetCoreID 函数的实现修改为 return 0;
2. Sys_GetCoreID 函数的定义如下。这个实现在普通的芯片上是没有问题的，CPXNUM 返回 0-3 表示当前是第 0 到第 3 个核心。

```
#if !defined(USING_OS_AUTOSAROS)
uint8 Sys_GetCoreID(void)
{
    return (IP_MSCM->CPXNUM & MSCM_CPXNUM_CPN_MASK);
}
#endif
```

但是如果是类似于 S32K358 这种芯片这个函数就需要注意一下。S32K358 对软件来说有 2 个核心，一个是锁步核 0，一个是单核 2。所以这里的 CPXNUM 返回的是 0 和 2。根据 AUTOSAR 的定义，AUTOSAR 中的 GetCoreID 函数应该将内核的物理 ID 转换成逻辑 ID。所以对于 S32K358 来说 GetCoreID 应该返回 0 或者 1。但是 Vector 的代码就直接使用了 Sys_GetCoreID 的实现，导致所有关于第二个核心的代码都没办法正常运行。如果客户没有购买第三方的 BSW 软件，直接使用 NXP 的 RTD 代码，并且按照上述第三条的方案修改了 RTE 模块中的数组大小。那么针对 S32K358 这种特殊的芯片，需要将这个 Sys_GetCoreID 的实现修改为 return (IP_MSCM->CPXNUM & MSCM_CPXNUM_CPN_MASK) >> 1;，这样函数就能返回 0 和 1，数组的访问就不会越界。对于核心数更多的芯片的返回状况，因为还没遇到过，所以不确定这个函数应该如何修改。请根据实际情况做相应的改动。