针对使用NXP GCC10.2编译程序的优化操作方法

Peter Chen（[Peter\_Chen@wtmec.com](mailto:Peter_Chen@wtmec.com)）

# 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 |

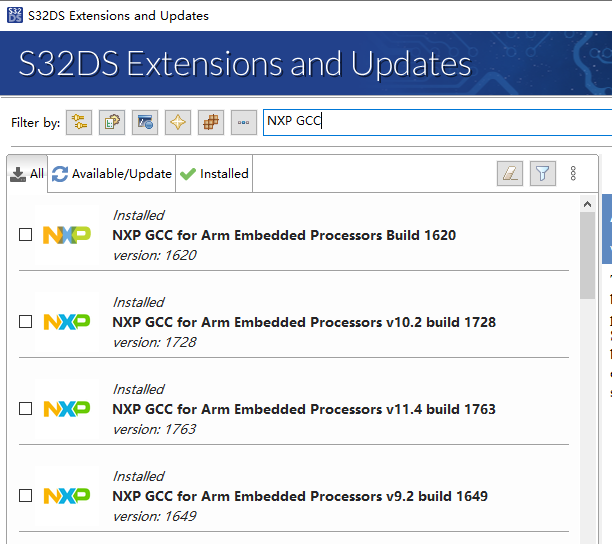
本文以S32DS for S32 Platform 3.5 + RTD 4.0.0作为例子基础展示优化方法。

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

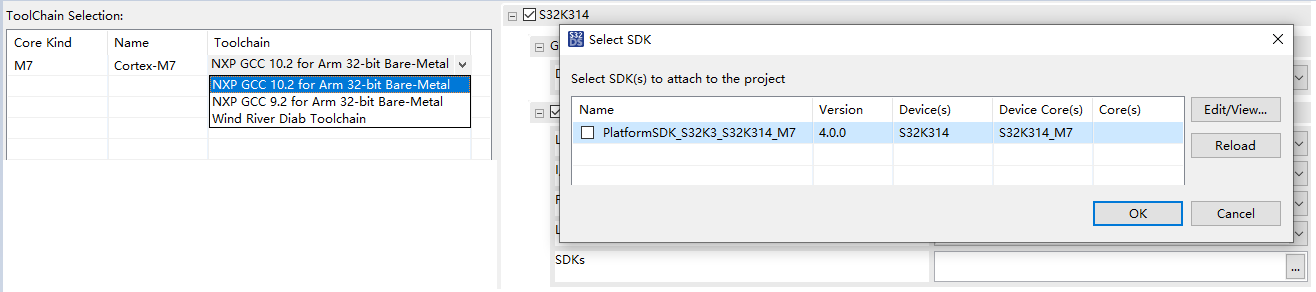
|  |  |
| --- | --- |
| 版本 | 说明 |
| 6.3.1 build 1620 | 用于编译S32K1 SDK 4.x版本 |
| 9.2 build 1649 | 用于编译S32K3 RTD 1.0.0版本 |
| 10.2 build 1728 | 主力版本，用于编译S32K3 RTD 2.0.0 – 5.0.0版本 |
| 11.4 build 1763 | 新版本，尚未与特定版本RTD相关联 |

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

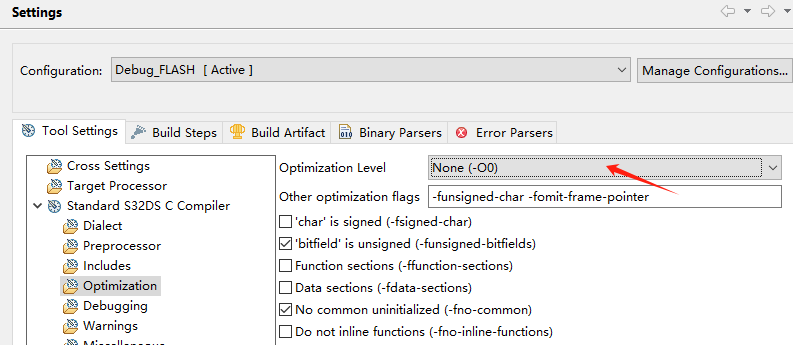
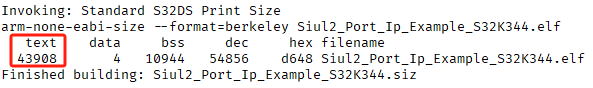


图4. 默认设置的编译结果



1. 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

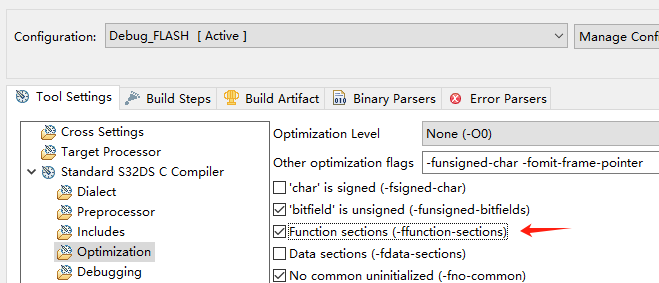


图6. 链接器的杂项中打开--gc-sections选项

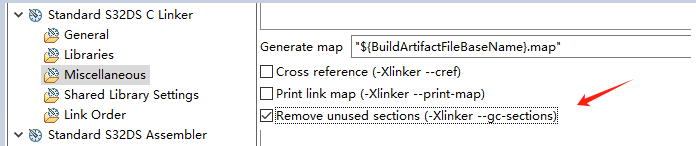
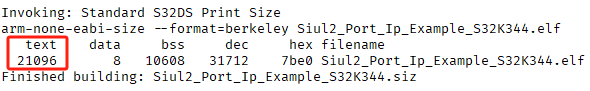
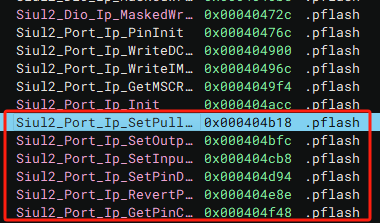


图7. 重新编译以后的结果



1. 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

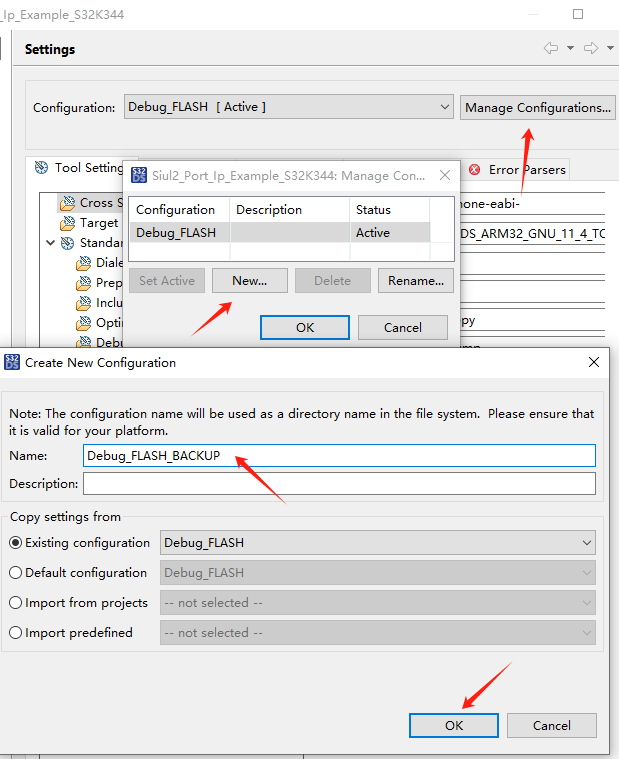


1. 通过查看map文件可知，原因是在于NXP GCC v10.2中对于GCC编译器的优化选项-ffunction-sections并未实现，从而导致段的最小单位是文件。一个C文件中所有的函数都会编译在一个text段中，只要其中有一个函数被调用过，那么链接器的--gc-sections就不会将这个段删除，从而增加了空间占用。加上RTD的代码中会把大量函数实现放在为数不多的几个文件中，这样也加剧了这种情况。

# 优化方法

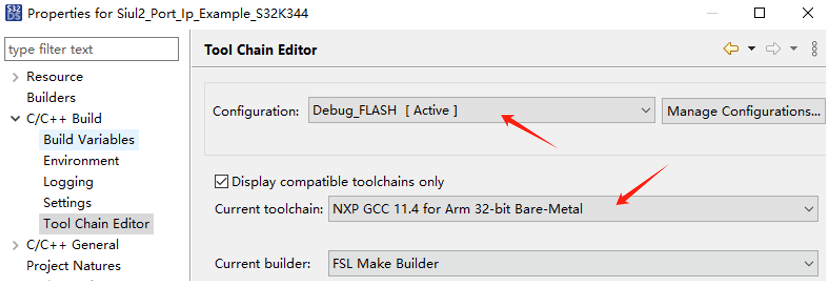
1. -ffunction-sections选项在NXP GCC v11.4 build 1763中已经实现了功能。第一步需要安装v11.4版本的编译器。在S32DS 3.5中可以通过IDE自带的包管理工具在线安装，在S32DS 3.6中该版本编译器是随安装包一起安装的。
2. 安装完以后，需要更改项目的编译器。在更改之前需要先备份设置里的配置内容，更改编译工具会重置配置。可以使用图9的方法复制一份配置。

图9. 从现有的配置复制并新建一个备份配置



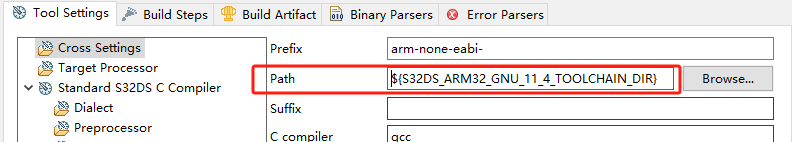
1. 在Tool Chain Editor设置中，选择原始配置，并将当前编译器选择为NXP GCC 11.4，如下图：

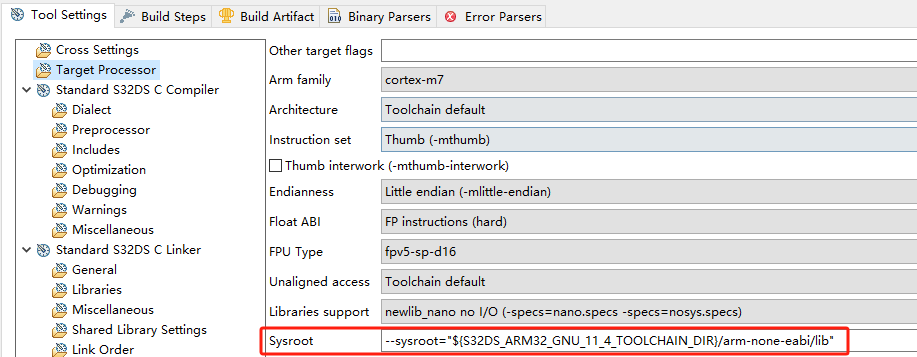
图11. 将编译器选择为11.4版本



1. 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}

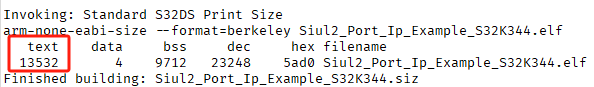
图12. Cross Settings中的Path是编译器的路径，Sysroot中的设定需要手动更新





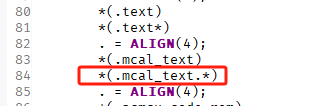
1. 重新编译以后，发现代码变成了13K左右，如下图：

图13. 删除无用代码以后的代码大小

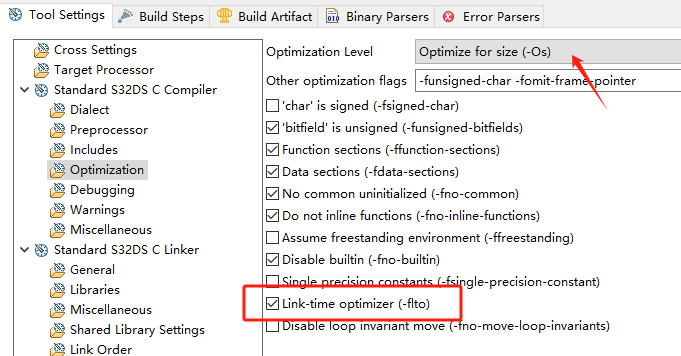


这时打开map文件可以看到，原来段名，比如.mcal\_text，已被修改成了.mcal\_text.[函数名]。编译器在编译时给每一个函数都建立了一个段，这样在链接时gc-sections就可以以函数的粒度来删除没有调用的函数，而不是以文件为最小粒度来处理了。

1. 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。



# 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。



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。

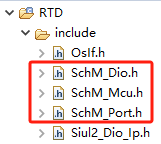


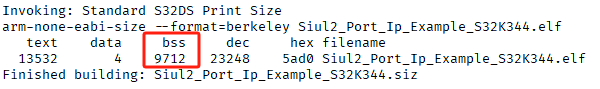
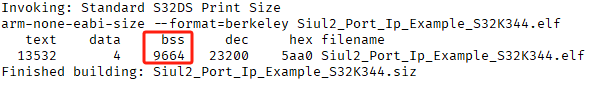
但是由于还没找到在Windows下编译插件的方法，暂时只能在Linux系统中进行编译。具体操作方法，可以参考[NXP GCC优化问题的总结.pdf](https://chenzch.github.io/NXPMCUFAQ/S32K3/NXP%20GCC%E4%BC%98%E5%8C%96%E9%97%AE%E9%A2%98%E7%9A%84%E6%80%BB%E7%BB%93.pdf)

# RTE模块中内存使用的优化

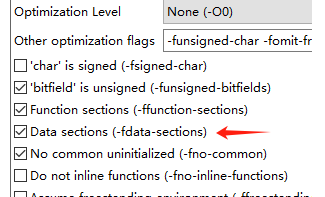
这个修改请不要合并到RTD的原始安装目录中，避免影响其他项目的编译

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只是为了让客户能在仅使用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。



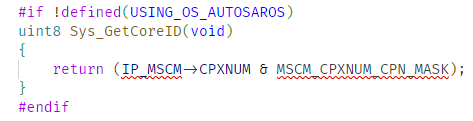
 

1. 打开-fdata-sections还可以下降几个字节，但是相同的问题是数据段的名称都会被修改，需要大量地修改ld文件，实际用量减小的效果并不明显，所以一般不需要实施。



# 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OsIf\_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。

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。