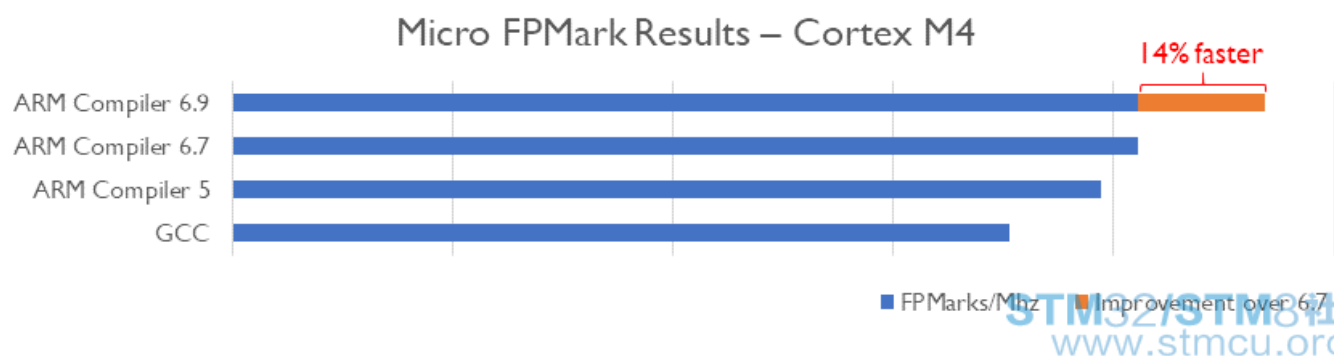


现在主流的 Cortex-M 开发环境和中间件代码都是基于 GNU 和 ArmCompiler v5 工具链写的。其实 ArmCompiler v6 工具链已经出来很久了。对比 ArmCompiler v5 工具链，Arm Compiler v6 做了很多改进。这篇文章就来研究下怎么快捷的把 ArmCompiler v5 的代码迁移到 Arm Compiler v6 上。为什么要用 Arm Compiler v6 的工具链呢？Keil 网站上有一张图解释了这个问题。



毕竟是 ARM 的官方工具链，对构架理解比较深，优化的好一点吧。

先介绍下 Arm Compiler v6 编译器。

Arm Compiler 6 是一个基于 LLVM 的工具链，那 LLVM 是什么呢？简单来说 LLVM 是把语法分析和机器码生成分开成两个独立部分。这样移植编译器到新构架就很容易了，你只要修改机器码生成部分，语法分析不用改。GCC 的编译器这两部分是混在一起的。LLVM 与 GCC 比较优点很多。比如错误信息详细啊，编译快啊等等。感兴趣的可以自行查一下。

一般来讲 LLVM 的语法分析部分是从 GCC 继承过来的。所以吗，语法规则和 GCC 是差不多的。说了这么多其实就像告诉你一件事，Arm Compiler 6 的 C 和汇编的语法和 GCC 是差不多的。（是不是完全相同？我没看到在文档里有写，也没有验证，脑补算是一样的。）

现有的很多中间件，像 HAL 库、FREERTOS 都没有为 Arm Compiler 6 做过适配的版本。但是都有 GNU 工具链的适配代码。那是不是可以直接用 Arm Compiler 6 编译 GNU 工具链的代码呢？那肯定是不行的。因为 Arm Compiler 6 和 GNU 工具链还是有不同的。

首先是链接器不同，Arm Compiler 6 用的是和 Arm Compiler 5 一样的 armlink 链接器，和 GNU 工具链的 LD 是不一样的，链接脚本也是不一样的。

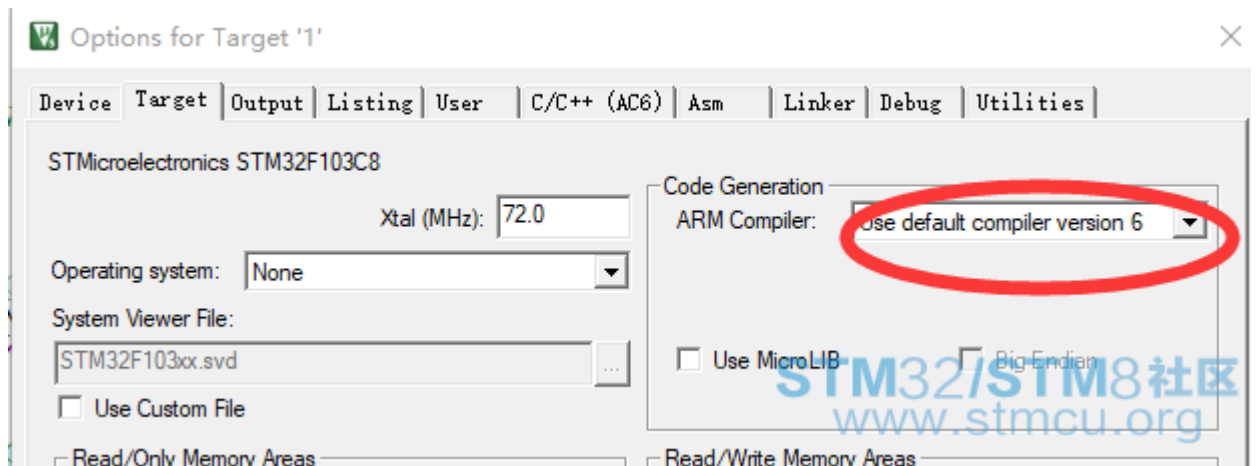
此外 Arm Compiler 6 和 GNU 编译工具链用的运行时库是不一样的。Arm Compiler 6 的运行时库除了提供标准 C 库的那些功能，还添加了 Semihosting 模式，支持 Scatter-loading 和初始化 HEAP、STACK 的代码。GNU 的运行时库是没有上面三个功能的，需要自己用汇编写（这一点是我脑补的）。

另外，Arm Compiler 6 里面有两种汇编语言的编译器，一种是使用 GNU 汇编格式的 armclang，一种是使用 ARM 汇编格式的 armasm。也就是说以前 Arm Compiler 5 的汇编启动文件还是可以用的。但是 C 的内联汇编必须使用 GNU 内联格式。

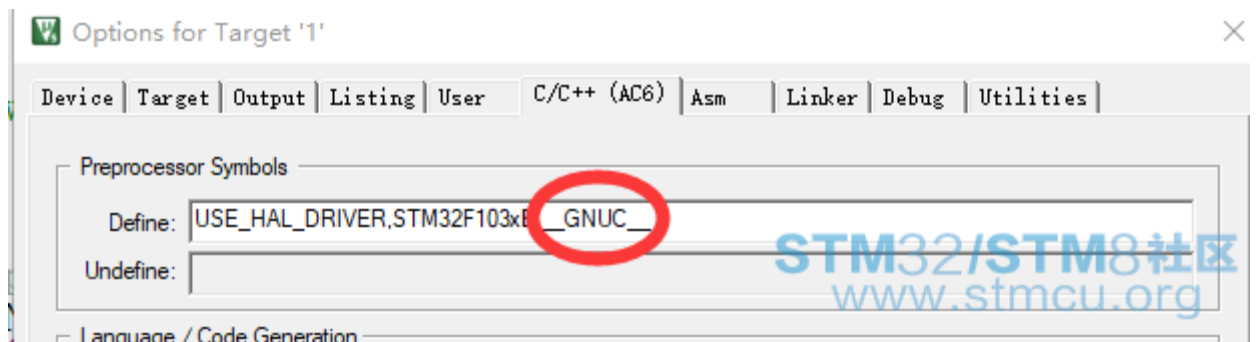
下面实际使用 Arm Compiler 6 工具链编译一下使用 HAL 库的程序。

看过上面说的，要把 GUN 工具链的项目或者老的 Arm Compiler 5 项目迁移到 Arm Compiler 6 工具链需要修改的地方就很明显了。Arm Compiler 6 帮助文档里专门有一个从 Arm Compiler 5 往 Arm Compiler 6 迁移的文档。可以参考。
把 Arm Compiler 5 的项目往 Arm Compiler 6 迁移很简单。用 STM32CubeMX 创建一个 MDK 项目。默认是 Arm Compiler 5 的编译器。直接用 Arm Compiler 6 编译肯定很多错误。成功迁移其实只需要简单的两步。

1、你打开一个 MDK 项目。然后在 Target 选项卡里选择 Arm Compiler 6。



2、在 C/C++选项卡里添加一个定义__GNUC__



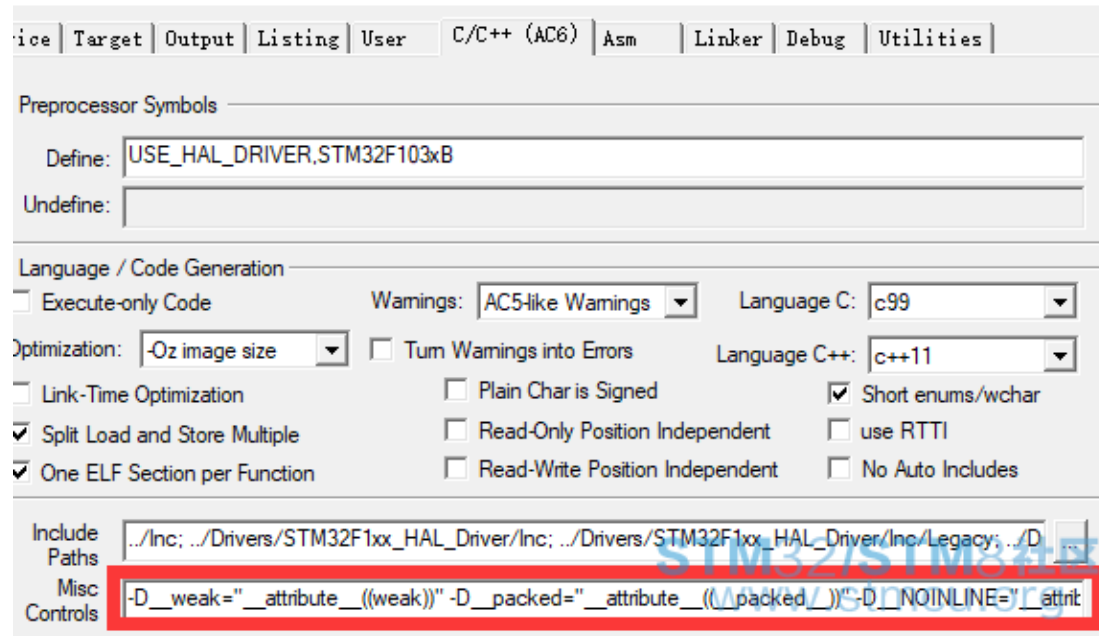
然后点编译就可以了。没有任何错误和警告，顺利通过编译。

这个添加的定义导致编译时候的同时引用了 CMSIS 的 GNU 实现接口 (cmsis_gcc.h) 和 Arm Compiler 6 的实现接口 (cmsis_armcc_V6.h)，可能会出现问
题。那我们严谨一点，不搞这种投机取巧的办法。
Arm Compiler 6 和 Arm Compiler 5 的 C 语言区别主要在扩展的 C 语言属性关键字上。
就是__weak,__packed 这类关键字。那我们预处理定义里面自己添加转换规则就行了。
使用-D 这个选项。比如-D__weak="__attribute__((weak))" -

D_packed="__attribute__((packed__))" -D_NOINLINE="__attribute__((noinline))"

根据编译的时候出现的错误，按照迁移手册上说明，定义一个预编译的转换规则就行。

Options for Target 'ct8test'



ice | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities

Preprocessor Symbols

Define: USE_HAL_DRIVER,STM32F103xB

Undefine:

Language / Code Generation

☐ Execute-only Code Warnings: AC5-like Warnings Language C: c99

Optimization: -Oz image size ☐ Turn Warnings into Errors Language C++: c++11

☐ Link-Time Optimization ☐ Plain Char is Signed ☒ Short enums/wchar

☒ Split Load and Store Multiple ☐ Read-Only Position Independent ☐ use RTTI

☒ One ELF Section per Function ☐ Read-Write Position Independent ☐ No Auto Includes

Include Paths: ../Inc; ../Drivers/STM32F1xx_HAL_Driver/Inc; ../Drivers/STM32F1xx_HAL_Driver/Inc/Legacy; ../D

Misc Controls: -D__weak="__attribute__((weak))" -D_packed="__attribute__((packed__))" -D_NOINLINE="__attribute__((noinline))"

前面的添加转换规则对于纯 C 程序肯定是完全够用了。但是碰到有内联 ARM 汇编的代码怎么办呢?没办法。只有按照 GNU 的格式改吧。

不过这个时候你可以找一下有没有 GNU 版本的源码直接用 GNU 工具链的源码就可以了。

比如 FreeRTOS 源码里面很关键的两个文件: port.c 和 portmacro.h。这两个文件里面都有大量的内联汇编。不同构架不同编译器的代码是不一样的。要用 Arm Compiler 6 工具链编译，你直接用 GNU 工具链版本的代码就可以了。另外 STM32CubeMX 是用了 CMSIS-RTOS 的接口。所以要改一下 cmsis_os.c 文件。

```
/*
 * ARM Compiler 4/5
 */
#if defined ( __CC_ARM )

// #define __ASM __asm
// #define __INLINE __inline
// #define __STATIC_INLINE static __inline
// #include "cmsis_armcc.h"
```

```
#define __ASM __asm
#define __INLINE inline
#define __STATIC_INLINE static inline
#include "cmsis_armcc_V6.h"
```

最后，对于预编译的库怎么选择呢？比如 emwin，针对不同的工具链提供了不同的库版本。但是没有提供 Arm Compiler 6 工具链的版本，那你直接用 GNU 的版本的预编译库就行了。一点小小的研究，如果有错误的地方，请回复指出。载请注明出处
<http://smallcsduck.blog.163.com>