从ADS到RealView MDK

作　者:
姜宁
ARM中国

1　RealView MDK——ARM微控制器开发的新工具

Keil是业界最好的51单片机开发工具之一，它拥有流畅的用户界面与强大的仿真功能。ARM将Keil公司收购之后，正式推出了针对ARM微控制器的开发工具RealView Microcontroller Development Kit（简称为RealView MDK或者MDK），它将ARM开发工具RealView Development Suite（简称为RVDS）的编译器RVCT与Keil的工程管理、调试仿真工具集成在一起，是一款非常强大的ARM微控制器开发工具。

很多嵌入式系统开发工程师对ARM的老版本开发工具ADS非常熟悉，而RealView MDK与ADS相比较，从外观、仿真流程以及内部二进制编译链接工具上都有了不少改进，用法稍有不同。本文的主旨是介绍通用的流程，以及一些注意事项，帮助ADS用户将老的、遗留的ADS工程转化成在RealView MDK中进行开发调试的工程。

2　工具结构的改进

作为ARM的新一代微控制器开发工具，RealView MDK不但包含ARM的最新版本编译链接工具，即RVDS3.0的编译链接工具，而且根据微控制器调试开发的特点采用了与ADS，RVDS完全不同的调试、仿真环境，μVision debugger与simulator。因此，MDK与ADS在工具架构组成上有一些不同，这些区别包括：不同的工程管理器，不同版本的ARM编译器（compiler），不同的调试器（debugger），不同的仿真器（simulator），以及不同的硬件调试单元，详见表1。

工具元件组成	RealView MDK	ADS
工程管理	uVision IDE	CodeWarrior IDE
编译器	ARM C/C++ Compiler RVCT	ARM C compiler for ADS
调试器	uVision Debugger	ARM Extention Degugger（AXD）
仿真器	uVision CPU & Peripheral Simulation	ARMulator
硬件调试单元	uLink	Multi-ICE

▲　表1　工具结构对比

2.1　编译工具例化形式

在ADS中，当用户需要将高级语言代码编译成目标文件时，需要根据目标机器码的不同（16位的Thumb代码或者32位的ARM代码），以及高级语言的不同（C代码或者C++代码）选择不同的编译器可执行文件；RVCT3.0编译器则将它们全部统一为armcc，仅仅通过不同的编译选项进行区分。表2较为详细的列出了其中的差别。

ADS1.2	REALVIEW MDK3.0	默认的编译选项
armcc	armcc	--c90 -arm
Tcc	armcc -thumb	--c90
armcpp	armcc --cpp	--arm
Tcpp	armcc --thumb --cpp	

▲　表2　MDK与ADS编译器的例化形式对比

2.2　POSIX格式

MDK集成了RVDS的编译工具RVCT，与ADS相比，除去编译、链接工具的可执行二进制文件不同之外，两个不同版本编译器的很多编译链接选项也有所不同。有关编译链接选项的变化用户可以参考ARM工具文档“RVCT Compiler and Libraries Guide中Table E-2 Mapping of compiler options。

RVCT采用了POSIX格式的编译链接选项，所有的多字符选项前必须使用双中划线。例如：ADS的编译选项-cpu，在MDK中需要改写成--cpu，否则用户在MDK中直接使用ADS的makefile时，工具会产生一个如下警告：

Warning: L3910W: Old syntax, please use '--cpu'

2.3 ARM ABI的变化

ARM ABI是Application Binary Interface for the ARM Architecture的简称，是一系列ARM体系架构标准的集合，囊括了ARM二进制代码交互、开发工具以及操作系统等等方面。

对目标文件进行链接之前，MDK工具的链接器会严格检查各个目标文件（objects），判断它们是否复合ARM体系结构的ABI标准。而MDK与ADS编译链接工具所遵循的ARM ABI是不同版本的，所以将ADS的遗留工程直接移植到MDK并进行链接时，用户可能会遇到如下的错误或者警告：

Error: L6238E: foo.o(.text) contains invalid call from '~PRES8' function to 'REQ8' function

Warning: L6306W: '~PRES8' section foo.o(.text) should not use the address of 'REQ8' function foobar

这是因为新工具的ABI要求在函数调用时，系统必须保证堆栈指针8byte对齐，即每次进栈或者出栈的寄存器数目必须为偶数。这是为了能够更加高效的使用STM与LDR指令对“double”或者“long long”类型的数据进行访问。而老的ARM开发工具ADS并没有考虑到新的ARM内核架构，其ABI对于堆栈的操作仅仅要求4byte对齐。所以当用户将在ADS中编译链接成功的工程代码移植到MDK上，或者将老的、ADS遗留的目标文件、库文件在新工具MDK中进行链接时，MDK的链接器就会报出以上的错误。

对于以上情况，用户可以通过简单修改代码并重新编译链接，或者使用特殊的编译选项来解决。

2.3.1 重新编译所有代码

当用户拥有该ADS遗留工程的所有源代码时，使用MDK重新编译链接全部代码是最好的解决方法。MDK中的新版本编译工具会重新生成满足堆栈8byte对齐要求的目标文件，避免由于堆栈不对齐引起的链接错误。

当工程中包含汇编代码时，用户可能还需要做少量的代码修改。这些修改包括：

1) 检查汇编源码中的指令，确保堆栈操作指令是8byte对齐的。

如Ex1中，ADS的遗留代码一次性将5个寄存器压栈，由于ARM的指令寄存器宽度为32位，即4byte，显然5个寄存器入栈之后，堆栈指针不能够满足64位，8byte对齐。为了解决这种情况，我们可以将另外一个并不需要压栈的寄存器，R12，同时压栈，这样当6个32位寄存器进栈之后，堆栈就能满足64位对齐了。

Ex.1

```
STMFD sp!, {r0-r3, lr ;
将R0,R1,R2,R3,LR（奇数）寄存器入栈
↓
STMFD sp!, {r0-r3, r12, lr} ;
将偶数个寄存器入栈
```

2)→ 在每个汇编文件的开头，添加“PRESERVE8”指令。见Ex2。

Ex.2

```
AREA Init, CODE, READONLY
↓
PRESERVE8
AREA Init, CODE, READONLY
```

2.3.2 使用--apcs /adsabi编译选项

当用户没有该ADS遗留工程的全部源码，只拥有库文件或者目标文件时，可以通过--apcs/adsabi编译选项强制MDK的编译器产生复合ADS ABI要求的目标文件，以达到与遗留的ADS库文件、目标文件兼容的目的。

注 ARM新工具将不会继续支持——apcs/adsabi选项。建议用户及时更新工具到最新版本。

2.4 分散加载注意事项

MDK同样支持ADS的分散加载文件，但是当分散加载文件中涉及到必须被放置ROOT Region中的C库函数时，有时用户需要作少量修改。

ROOT Region的load address与execution address相同，所以这部分代码在系统初始化时无需进行搬移操作，很多库函数，如__scatter*.o或者__dc*.o，必须被放置在Root Region中。

Ex.3 – 分散加载文件的修改

```
; ADS 中的分散加载文件
ROM_LOAD 0x0
{
    ROM_EXEC 0x0
    {   vectors.o (Vect, +First)
        __main.o (+RO)
    * (Region$$Table)
    * (ZISection$$Table)
    }
    RAM_EXEC 0x100000
    {   *.o (+RO,+RW,+ZI)   }
}
```

↓

```
; MDK中的分散加载文件1
ROM_LOAD 0x0
{
  ROM_EXEC 0x0
    {
      vectors.o (Vect, +First)
      * (InRoot$$Sections)
    }
  RAM_EXEC 0x100000
    {
      *.o (+RO,+RW,+ZI)
    }
}
```

```
; MDK中的分散加载文件2
ROM_LOAD 0x0
{
  ROM_EXEC 0x0
    {
    vectors.o (Vect, +First)
    __main.o(*)
    * (Region$$Table)
          __scatter*.o(*)
    __dc*.o(*)
    }
  RAM_EXEC 0x100000
  { *.o (+RO,+RW,+ZI)}
}
```

在ADS中，用户必须在分散加载文件中明确的将特定section代码放置在Root Region中。而MDK为了支持新的RW压缩机制，采用了新的region table格式，这种新的格式并不包含ZISection$$Table，而且新的scatter-loading (__scatter*.o)与 decompressor (__dc*.o)必须被放置在root region中。所以EX3中ADS的分散加载文件应该被修改成新的形式。例3中提供了两种修改分散加载文件的方法，分散加载文件1通过InRoot$$Sections自动将所有必须的库目标放至在root region中，而分散加载文件2则详细的注明了__scatter*.o与 __dc*.o的位置。

2.5 C库函数的差异

为了与新的ABI一致，MDK中的库函数名称与ADS可能会有不同。ADS中的__rt_*库函数被替换为__aeabi_*。如果用户的ADS工程中曾经重定义(retarget)过这些库函数，那么在移植到MDK时，需要重新实现这些函数，以满足新ABI的要求。表3列出了部分函数的对应关系。

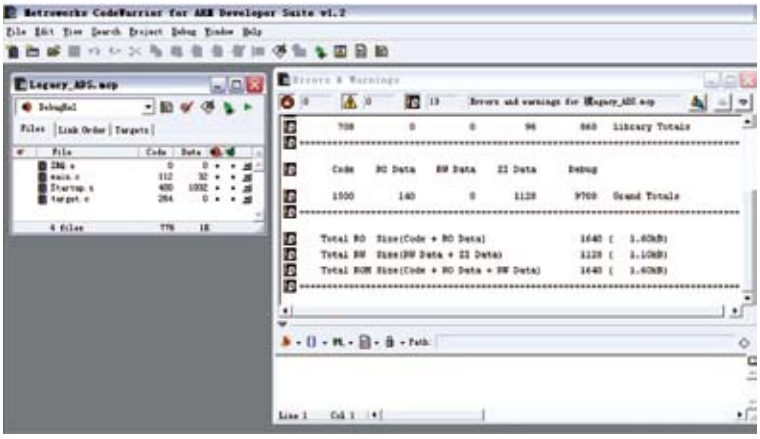
ADS库函数	RREALVIEW MDK库函数
__rt_memcpy_w	__aeabi_memcpy4
__rt_div0	__aeabi_idiv0
__rt_sdiv	__aeabi_idiv
__rt_udiv	__aeabi_uidiv
__rt_fp_status_addr	__aeabi_fp_status_addr
__rt_errno_addr	__aeabi_errno_addr

▲ 表3 部分库函数对比

3 移植实例

结合以上对MDK与ADS差异的描述，本小节将以实例的形式叙述如何将ADS1.2上的遗留代码移植到MDK上。

以Philip的LPC2294(ARM7TDMI)为处理器，将一个在ADS1.2上开发的由LPC2294控制LED闪烁的工程移植到MDK上来。由图1可以看出，该工程(Legacy_ADS.mcp)共有4个源文件(Startup.s、tartget.c、IRQ.s、main.c)，以及一个分散加载文件(Scatterload)。



◀ 图1 ADS遗留工程

使用ADS1.2编译器，

编译选项为：-O1 -g+

链接选项为：-info totals -entry 0x00000000 -scatter .\src\Scatterload.scf -info sizes

我们得到最终代码尺寸信息如下：

```
Total RO  Size(Code + RO Data)                 1640 (  1.60kB)
Total RW  Size(RW Data + ZI Data)              1128 (  1.10kB)
Total ROM Size(Code + RO Data + RW Data)       1640 (  1.60kB)
```

为了能够使用ARM新工具MDK的一系列特性，我们需要把ADS中的遗留工程移植到MDK上来。其具体步骤如下：

1) 在MDK中新建工程

打开MDK，在主菜单中选择Project→New…→uVision Project，并给新工程命名为New_MDK.uv2，单击“保存”，见图2。

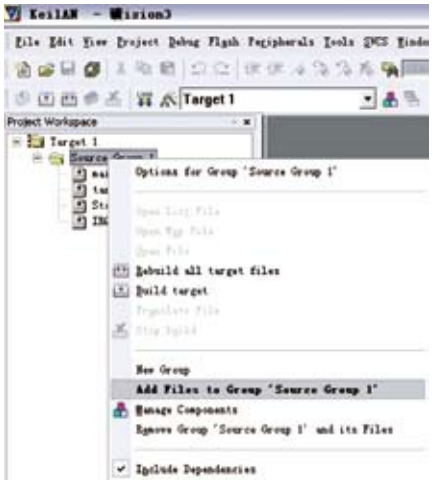
▲ 图2 在MDK中新建工程

在MDK自动弹出的器件选择窗口(Select Device for Target)中选择该工程所对应的处理器型号，“LPC2294”，并单击“确定”，见图3。当MDK提示用户是否自动添加启动代码时，选择“否”。

▲ 图3 在MDK中选择合适的处理器

2) 添加源文件，并设置工程属性

将Legacy_ADS.mcp工程中所有的源文件都添加到新的New_MDK.uv2工程中来，见图4。



▲ 图4 将ADS工程的遗留源代码全部添加到新工程中

单击工程属性快捷键，打开工程属性设置窗口，并选择C/C++标签页，设置编译器属性。用户可以根据以前ADS工程的编译属性设置，也可以根据当前具体需求重新设置编译属性。在本例中，我们将ADS遗留工程的编译属性，“-O1

-g+”修改为“-O1 -g -W”后，拷贝到“Misc Controls”栏中来，见图5。这是因为由于编译器版本的变化，其对应的编译选项也有所变化的缘故。

注　-W选项可以抑止所有的warning。

▲ 图5　编译选项的设置

对ADS工程中的链接选项作适当修改如下，使其复合POSIX格式。

--info totals --entry 0x00000000 --scatter .\src\Scatterload.scf --info sizes

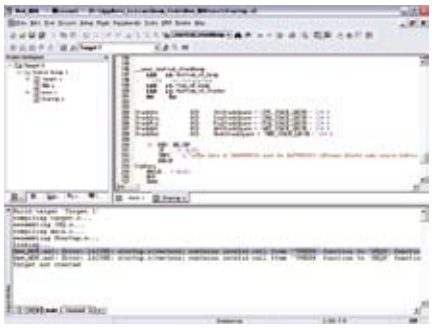
选择Linker标签，将修改过的链接选项拷贝至MDK工程属性的Linker属性中，并单击“确定”，见图6。

▲ 图6　链接选项的设置

3)　Build工程并适当修改代码

当所有的工程属性都设置好之后，单击“Build all target file”快捷键，对整个工程进行编译链接。在MDK窗口的build输出一栏中，我们会发现系统出现了一个链接错误L6238E，见图7，这是由于MDK中新版本编译链接工具与ADS的老版本build工具采用不同的ABI造成的（详见本文2.3小节）。

为此我们打开该工程中的汇编文件startup.s，在该程序第55行添加PRESERVE8指令，如下所示：



▲ 图7　链接错误L6238E

```
CODE32
PRESERVE8
AREA    vectors,CODE,READONLY
```

4)　重新编译链接该工程

代码修改完毕之后，单击“Build all target file”快捷键，对该工程进行二次编译链接。MDK将成功生成New_MDK.axf文件，并显示其代码尺寸信息为：

Program Size: Code=1576 RO-data=64 RW-data=0 ZI-data=1128

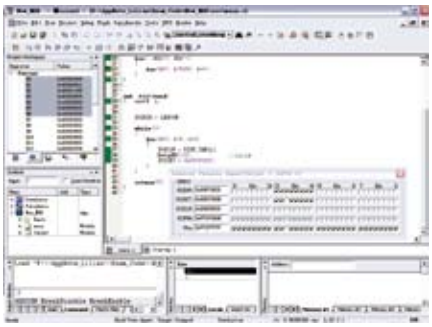
这些信息同样可以从链接生成的New_MDK.map文件中得到。

5)　代码调试与固化

与其它ARM开发工具相比较，MDK拥有非常出色的仿真功能，可以帮助用户在纯软件的平台上进行较为精确的调试。用户可以在工程属性设置窗口选择simulator调试（见图8）或者通过硬件调试工具（uLink）进行调试。

▲ 图8　选择uVision Simulator作为调试平台

当选择simulator调试时，单击debug快捷键，打开simulator调试窗口，见图9。为了验证该程序在LPC2294硬件平台上是否能够正确执行，通过GPIO口驱动LED进行循环闪烁，用户可以单击Peripherals->GPIO->Port2，将GPIO端口2的仿真界面打开，见图9。



▲ 图9　RVDMK调试环境

单击运行快捷键，可以看到在GPIO端口2的仿真调试窗中，IO口的输出在不停的循环变化。

当程序通过了仿真调试之后，用户就可以通过MDK的硬件调试工具，uLink，将最终代码固化在非易失性的存储器中了。uLink的设置见图10与图11。

▲ 图10 设置使用uLink调试

▲ 图11 选择MDK默认的LPC2294 Flash烧写代码

4　总结

作为ARM新推出的微控制器开发工具，MDK集成了业界最好的ARM编译链接工具RVCT，拥有出色的仿真调试功能，集成了代码固化工具Flash Programming Utilities，支持ARM7、ARM9以及Cortex-M3等多种ARM内核，是ARM微控制器开发的首选。