应用报告
ZHCA533A–April 2013–Revised May 2013

将软件项目从 StellarisWare®迁移到 TivaWare™ 上

David Wilson *Stellaris® Microcontrollers*

摘要

由于德州仪器 (TI) 发布了 Tiva™ 系列 32 位 ARM® Cortex™-M4 微控制器 (MCU)， StellarisWare® 综合软件工具套件也迁移到 TivaWare™上。 这个全新的软件平台提供了与之前版本中一样的特性，但是在迁移过程中做出了几处改动，需要对现有的 StellarisWare 应用进行简单修改，以使它们能够在全新的*针对 C 系列的 TivaWare*软件树内建立并运行。 本文档详述了用于 LM3S 和 LM4F MCU 的 StellarisWare 与用于 C 系列 MCU 的 TivaWare 之间的差异，并且对将现有用户代码从之前的平台迁移到全新平台所需做出的改动提供指导。

除非另外注明，术语*TivaWare*与*针对 C 系列的 TivaWare*是同义词。

内容

图表列表

1 背景情况

自从在 2006 年被引入以来，StellarisWare 软件开发平台一直为德州仪器 (TI) 的 Stellaris ARM Cortex-M3 和 Cortex-M4F 微控制器提供很好的服务。 随着德州仪器 (TI) 微控制器产品的发展壮大，现在的微控制器产品包括多个基于 ARM Cortex 的部件系列，此系列产品支持不同的市场划分和目标应用，然而，StellarisWare 正在成为 TivaWare。 这个迁移要求对平台架构和执行做出一些简单改动。 由于以下几个原因，已经做出了这些细小改动：

- 为了实现对同一 TivaWare 系列项下多个，不同 MCU 系列的支持；
- 为了提高不同处理器平台上的源代码可移植性；以及
- 为了纠正 StellarisWare 代码中已知架构缺陷和不一致的地方。

在针对 C 系列的 TivaWare 中保留了全部现有 StellarisWare 特性和功能性，并且外设驱动程序库、图形库以及 USB 库中适用于 Tiva 部件的应用程序接口 (API) 也被保留下来。 所需的改动并不全是向后兼容的，所以为了使之前用 StellarisWare 建立的应用能够移植成可由 TivaWare 建立的应用，某些源代码和项目文件修改是必须的。

程序设计工具值向后兼容性。 德州仪器 (TI) 一直设法平衡这些改动所带来的长期效益与迁移过程中，由于某些应用源代码修改所导致的不便之处之间的关系。 我们相信，迁移过程并不是一个困难的挑战，并且您的大多数应用可以与 TivaWare 一起运转正常，而无需大量的工作或昂贵的软件开发。

每个 Tiva 部件系列有其自身的软件安装，其内容已按照特定系列功能的需要进行了调整。 本文档主要解决将一个 StellarisWare 应用迁移至针对 C 系列环境的 TivaWare 中。 Tiva 系列软件新方法的一个目的就是在所有 Tiva 系列上提供一致的 API 和环境；因此，当其它 TivaWare 系列版本可用时，此处所介绍的信息也广泛适用于它们。

2 快速入门指南

由于 StellarisWare 与 TivaWare 之间的以下这些差异，需要对应用源代码进行改变：

- 目录结构和库命名规则已经发生变化。
- C99 数据类型现在已用于整个 TivaWare 中。
- 匈牙利前缀表示法已经发生变化，这导致很多结构字段的名称变化。
- 现有 Stellaris LM4F 器件部件号已经发生变化。
- USB 器件初始化已经被简化。
- StellarisWare 中过时不用的函数和标签已经从 TivaWare 中删除。

为了使您能够开始对您的项目进行改动，以下的步骤提供了在全新 TivaWare 树中建立并连接一个典型 StellarisWare 项目所需工作的概述。 不使用 USB 库的用户也许会发现，这个列表中描述的改动不足以使他们在全新的 TivaWare 树中建立并运行应用。 对于启用 USB 的应用，需要其它几个小改动；这些改变在 一个之后的部分中进行了描述。

1. 将您现有的项目目录复制到全新 TivaWare 树中相应的位置中。 例如，如果您的应用位于*C:\StellarisWare\boards\<您的电路板>\<您的应用>*中，将其复制到*C:\ti\TivaWare_C_Series_1.0\boards\<您的电路板>\<您的应用>*中。

   请注意：确保您复制已开发的任一与电路板相关的驱动程序代码，通常位于 C:\StellarisWare\boards\<您的电路板>\drivers 中，并且已经知道您必须在 TivaWare 树中创建*电路板*目录。 **预先建立的电路板示例已经移动到新位置。**
2. 在所有包含 StellarisWare 或 TivaWare 头文件的源文件顶部的 <stdint.h> 和 <stdbool.h> 中添加 #includes。 这些头文件定义现在在 TivaWare 中使用的标准数据类型。
3. 如果您的源文件包括*utils/ustdlib.h*，在文件顶部的 <time.h> 内添加一个 #include。 这个添加的内容被要求用来获取*struct tm*的标准定义，现在它被用来替代私有的*tTime*类型
4. 在您源代码的全局范围内，用标准*bool*类型替代*tBoolean*。
5. 在全局范围内，用*struct tm*替代*tTime*。 如果您的应用使用*tTime*，那么您重新编写代码以使用 C 标

准*struct tm*的等值字段。 这个修改是名称的直接替换，例如将**ucMon**或**ucYear**替换为**tm_mon**或**tm_year**。

6. 如果应用使用的 StellarisWare API 需要从其中一个 StellarisWare 头文件中获得一个结构，那么请检查您的源代码中所使用的字段名称，这是因为这些名称中的很多名称已经发生变化以符合匈牙利前缀表示法规范以及 C99 数据类型的使用。
7. 如果使用**GPIOPinConfigure**函数，请修改您的项目设置或 Makefile 来将现有的**PART_LM4Fxxx**标签替换为替代部件号（请参考表 5）。
8. 如果应用使用格式*inc/lm4f*.h*的任一特定部件头文件，将这个名称替换为等值的*tm4c*部件头文件，方法如3.4 节中所述。
9. 修改您的项目设置或 Makefile 来连接至 TivaWare 树内新位置上的驱动程序库、图形库和/或 USB 库。TivaWare 库的放置位置与它们在 StellarisWare 树中的位置一致，但是从 toolchain 目录名称和库名称中删除了*-cm3*和*-cm4*后缀。 更多信息请参阅 库命名部分。

3 特定 API 和源代码变化

3.1 目录结构变化

TivaWare 目录结构除了以下两个例外情况，几乎完全映射至 StellarisWare 发布版本内使用的结构：

- 缺省安装目录已经从*C:\StellarisWare*变成了*C:\ti\TivaWare_C_Series_<version>*。
- 特定电路板示例应用已经从*boards*子目录移动至*examples\boards*子目录中。

当使用多个 TivaWare 安装或者针对不同 TivaWare MCU 系列的安装时，第一个改动可实现更加整齐干净的总体目录结构。 如果在安装过程中选择了缺省目录，这个全新的缺省目录还可确保不同的版本能够并排安装，而不会相互写覆盖。 当然，在安装过程中可轻松写覆盖安装目录；用户可以随意选择将 TivaWare 树安装在特定应用所需的任一目录中。 TivaWare 内的软件或目录节点不会假定根目录的名称或位置。

第二个改动减少了因代码示例的位置所造成的混淆。 在 StellarisWare 中，可在两个不同的子目录中找到代码示例：**boards**针对特定代码示例应用，而**examples**针对特定外设示例。 现在，所有示例源代码可在单个**examples**目录中找到，这一目录中的更低级子目录用于电路板和外设。

StellarisWare 用户已经可以在他们各自文件系统中的任一位置创建他们自己的应用项目。 TivaWare 内也提供了这个灵活性。 例如，如果您的项目驻留在*C:\StellarisWare*目录中，将其移动到*C:\TI\TivaWare-C-Series-<版本>*下的同一位置可对其进行编译，而无需包含任何与路径相关的改动，这是因为所有头文件的相对路径保持不变。 一个被存储在*C:\StellarisWare\boards\<你的电路板>\<您的应用>*中的之前的项目可被移动至*C:\TI\TivaWare-C-Series\boards\<你的电路板>\<您的应用>*（即使基本 TivaWare 发布版本在这个级别上不包含*boards*目录也是如此）并且仍能够被编译，而无需在代码中进行任何路径改变。

对于存储在*C:\StellarisWare*子树之外的项目，或者那些存储在树内，但是引用树外条目的项目，有必要对 toolchain 项目或 makefile 进行修改以解决这样一个事实，那就是由于缺省安装目录发生改变，TivaWare 库和头文件现在处于不同的位置。

3.2 库命名

基于 Cortex-M3 和 Cortex-M4F 架构的 MCU 在内的 Stellaris 系列；所有 Tiva 器件是基于 M4F 的。 因此，TivaWare 内库的名称已经被简化以删除最近 StellarisWare 建立内使用的特定内核后缀。表 1列出了 TivaWare 和 StellarisWare 库的名称。

表 **1.** 库命名

说明	**Toolchain**	**StellarisWare** 文件	**TivaWare** 文件
外设驱动程序库	CCS	driverlib/ccs-m3/Debug/driverlib-cm3.lib	driverlib/ccs/Debug/driverlib.lib
		driverlib/ccs-m4f/Debug/driverlib-cm4f.lib	
	Keil RVMDK	driverlib/rvmdk-cm3/driverlib-cm3.lib	driverlib/rvmdk/driverlib.lib
		driverlib/rvmdk-cm4f/driverlib-cm4f.lib	
	IAR EWARM	driverlib/ewarm-cm3/Exe/driverlib-cm3.a	driverlib/ewarm/Exe/driverlib.a
		driverlib/ewarm-cm4f/Exe/driverlib-cm4f.a	
	gcc 和 Code Bench	driverlib/gcc-cm3/libdriver-cm3.a	driverlib/gcc/libdriver.a
		driverlib/gcc-cm4f/libdriver-cm4f.a	
图形库	CCS	grlib/ccs-m3/Debug/ grlib-cm3.lib	grlib/ccs/Debug/grlib.lib
		grlib /ccs-m4f/Debug/ grlib-cm4f.lib	
	Keil RVMDK	grlib /rvmdk-cm3/grlib-cm3.lib	grlib/rvmdk/grlib.lib
		grlib /rvmdk-cm4f/grlib-cm4f.lib	
	IAR EWARM	grlib /ewarm-cm3/Exe/grlib-cm3.a	grlib/ewarm/Exe/grlib.a
		grlib /ewarm-cm4f/Exe/grlib-cm4f.a	
	gcc 和 Code Bench	grlib /gcc-cm3/libgr-cm3.a	grlib/gcc/libgr.a
		grlib /gcc-cm4f/libgr-cm4f.a	
USB 库	CCS	usblib/ccs-m3/Debug/ usblib-cm3.lib	usblib/ccs/Debug/usblib.lib
		usblib/ccs-m4f/Debug/usblib-cm4f.lib	
	Keil RVMDK	usblib/rvmdk-cm3/usblib-cm3.lib	usblib/rvmdk/usblib.lib
		usblib/rvmdk-cm4f/ usblib-cm4f.lib	
	IAR EWARM	usblib/ewarm-cm3/Exe/ usblib-cm3.a	usblib/ewarm/Exe/usblib.a
		usblib/ewarm-cm4f/Exe/ usblib-cm4f.a	
	gcc 和 Code Bench	usblib/gcc-cm3/libusb-cm3.a	usblib/gcc/libusb.a
		usblib/gcc-cm4f/libusb-cm4f.a	

3.3 C99 类型和匈牙利前缀表示法变化

从 StellarisWare 移动至 TivaWare 使我们能够更正一个几年前就希望解决的 StellarisWare 架构问题。StellarisWare API 使用诸如*无符号长整型 (unsigned long)*的简单 C 数据类型；然而，这些数据类型通常会产生严重的问题：每个类型的大小会因它们所使用的 CPU 的不同而不同。 虽然当把 StellarisWare 用在简单微处理器系列上这个差别不是什么大问题，它仍然会产生几个移植性问题并且使得 API 会发生更多的问题，这是因为基本微处理器架构会随着时间的推移而不断发展。

为了解决这一潜在问题，并且使得能够被移植到其它架构中，而又无需重大的源代码重新编写工作，现在 TivaWare 使用标准的、明确尺寸的 C99 数据类型。 通过使用这些数据类型，例如，一个 32 位实体类型可保证保持在 32 位，而与代码运行的处理器无关。

虽然 StellarisWare API 已经迁移至 TivaWare，但是，如果应用之前使用 StellarisWare API，而现在使用 TivaWare 的话，那么这个数据类型变化要求重写现有应用。 首先，C99 类型不是编译器所固有的；反之，这些类型通过标准 C 运行时间头文件进行定义：**stdint.h**针对基本类型，而**stdbool.h**针对*bool*布尔数据类型。 StellarisWare 和 TivaWare 头文件不嵌套其它头文件，所以这两个标准头文件现在必须被添加到所有源文件中，这些源文件包含所有 TivaWare 头文件以确保所需的基本数据类型可用。

```
#include <stdbool.h>
#include <stdint.h>
```

在进行此改动时，我们还决定不使用私有数据类型，*tTime*，来代表数据和时间值。 此外，utils/ustdlib 文件中的函数已经被重写以使用标准*struct tm*结构，而非*tTime*。 因此，包含 utils/ustdlib.h 的源文件也必须包含标准*time.h*头文件。

```
#include <time.h>
```

虽然由于*struct tm*的使用而要求进行源代码修改，*struct tm*内的字段名几乎完全映射至*tTime*中的字段名，这使得这些修改变得简单直接。 *struct tm*het*Time*内的字段名的映射方式如表 2中所示。

表 **2.** 针对*struct tm*和*tTime*的字段名映射

tTime 字段名	**struct tm** 字段名	注释
usYear	tm_year	*tm_year*被定义为“1900 年之后的年份”，而*usYear*包含实际年份。当重写包含此字段的代码时，必须将这个差异考虑在内。
ucMon	tm_mon	
ucMday	tm_mday	
ucWdat	tm_wday	
ucHour	tm_hour	
ucMin	tm_min	
ucSec	tm_sec	

C99 数据类型变化的第二个结果-更有可能需要进行源代码更新的结果-是匈牙利前缀表示法的相关改变。 虽然这个改变不会对参数命名产生影响，但是它的确改变了大多数字段的名称，并因此影响了 USBLib 和 GrLib 的使用。 除了因全新的数据类型而改变所使用的前缀外，德州仪器 (TI) 已经尽可能地确保整个代码库中匈牙利前缀表示法使用的一致性。 这一改变已经使得某些附加结构文件名发生变化，这些文件名在之前并未采用正确的前缀。

在表 3中给出了 StellarisWare 中使用的数据类型列表连同它们的匈牙利前缀表示法以及全新的 TivaWare 替代。

表 **3. StellarisWare** 和 **TivaWare** 中使用的数据类型

StellarisWare 类型	之前的前缀	**TivaWare** 类型	新前缀	示例
tBoolean	b	bool	b	bFoo
字符型(1)	c	字符型	c	cFoo
字符型(2)	c	int8_t	i8	i8Foo
短整型	s	int16_t	i16	i16Foo
长整型	l	int32_t	i32	i32Foo
超长整型	ll	int64_t	i64	i64Foo
无符号字符 (unsigned char)	uc	uint8_t	ui8	ui8Foo
无符号短整型	us	uint16_t	ui16	ui16Foo
无符号长整型	ul	uint32_t	ui32	ui32Foo

(1) 何时被用来代表一个文本字符型编码。
(2) 何时被用来代表一个 8 位有符号数。

表 3. StellarisWare 和 TivaWare 中使用的数据类型 (continued)

StellarisWare 类型	之前的前缀	TivaWare 类型	新前缀	示例
无符号超长整型	ull	uint64_t	ui64	ui64Foo

表 4中列出了 StellarisWare 中经常使用不一致的匈牙利前缀表示法。 已经在 TivaWare 中改变了受影响的变量和字段名称：

表 4. TivaWare 中被改变的匈牙利前缀表示法

类型	前缀	示例
指针	p<prefix>	pcFoo，pui32Foo
typedef	t	tFoo
枚举值	e	eFoo
函数指针	pfn	pfnFoo
结构变量	s	sFoo
联合变量	u	uFoo
枚举值	i	iFoo
数组	p<prefix>	pcFoo[]，pui32Foo[]
指针到指针或二维数组	pp<prefix>	ppcFoo，ppui32Foo

根据使用中的 toolchain 和分支语句，使用之前数据类型的传递参数对 TivaWare 函数的影响会有所不同。 当使用具有*–Wall-pedantic*的 GCC 4.3.6 时，例如，*传递无符号长整型*变量至函数并希望*unit32_t*参数不产生警告。 然而，当在*所有警告*被启用，但是*严格 ANSI*被禁用时使用 Keil RVMDK 进行同样的操作时，会产生一个报警，但是生成连接和运行正确的输出。

3.4 部件号变更

生产前的**LM4F**Stellaris 部件号分别被改为**TM4C**（Tiva C 系列）部件号，这是因为部件完全符合生产状态。 之前的部件和全新的部件功能完全一样，但是部件变更会影响某些使用特定部件头文件的软件，这些头文件可在 StellarisWare 或 TivaWare 发布版本的*inc*目录中找到。 这些头文件按照使用中的特定部件号来命名，并且包含所有与那些特定部件相关的寄存器定义。 如果您的项目代码使用这些头文件中的一个，那么使用表 5来确定您 MCU 的全新部件号，并用相等的*tm4c*版本来替代*lm4f*头文件。 标签定义从之前头文件延续至新的头文件，所以除了文件名变换外，这个变更不会修改源代码。

除了使用 pin_map.h 中标签定义的**GPIOPinConfigure()**函数之外，使用 DriverLib API 的软件通常不使用特定部件头文件。 这个头文件包含针对每个特定部件而进行了调整的标签定义。 这些定义由格式 PART_X 的预处理器定义来控制，在这里，***X***是目标部件号。 由于 pin_map.h 现在使用 Tiva 部件号，所以必须修改应用 makefile 或项目中定义的标签以确保能够定义特定部件引脚复用标签的正确集合。

例如，如果您的应用之前是针对一个 LM4F232H5QD 部件，并且您的项目或 makefile 定义了标签*PART_LM4F232H5QD*，那么您必须将这个标签替换为*PART_TM4C123GH6PGE*以确保将针对目标器件的正确引脚复用标签包含在内。

表 5. LM4F 部件号的 Tiva C 系列替代部件号

LM4F 部件号	TM4C 部件号		LM4F 部件号	TM4C 部件号
LM4F110B2QR	TM4C1231C3PM		LM4F131E5QR	TM4C1236E6PM
LM4F110C4QR	TM4C1231D5PM		LM4F131H5QR	TM4C1236H6PM
LM4F110E5QR	TM4C1231E6PM		LM4F130C4QR	TM4C1237D5PM
LM4F110H5QR	TM4C1231H6PM		LM4F130E5QR	TM4C1237E6PM
LM4F111B2QR	TM4C1230C3PM		LM4F130H5QR	TM4C1237H6PM
LM4F111C4QR	TM4C1230D5PM		LM4F132C4QC	TM4C1237D5PZ
LM4F111E5QR	TM4C1230E6PM		LM4F132E5QC	TM4C1237E6PZ
LM4F111H5QR	TM4C1230H6PM		LM4F132H5QC	TM4C1237H6PZ
LM4F112C4QC	TM4C1231D5PZ		LM4F132H5QD	TM4C1237H6PGE
LM4F112E5QC	TM4C1231E6PZ		LM4F210E5QR	TM4C123BE6PM
LM4F112H5QC	TM4C1231H6PZ		LM4F210H5QR	TM4C123BH6PM
LM4F112H5QD	TM4C1231H6PGE		LM4F211E5QR	TM4C123AE6PM
LM4F120B2QR	TM4C1233C3PM		LM4F211H5QR	TM4C123AH6PM
LM4F120C4QR	TM4C1233D5PM		LM4F212E5QC	TM4C123BE6PZ
LM4F120E5QR	TM4C1233E6PM		LM4F212H5QD	TM4C123BH6PGE
LM4F120H5QR	TM4C1233H6PM		LM4F212H5QC	TM4C123BH6PZ
LM4F121B2QR	TM4C1232C3PM		LM4F212H5BB	TM4C123BH6ZRB
LM4F121C4QR	TM4C1232D5PM		LM4F231E5QR	TM4C123FE6PM
LM4F121E5QR	TM4C1232E6PM		LM4F231H5QR	TM4C123FH6PM
LM4F121H5QR	TM4C1232H6PM		LM4F230E5QR	TM4C123GE6PM
LM4F122C4QC	TM4C1233D5PZ		LM4F230H5QR	TM4C123GH6PM
LM4F122E5QC	TM4C1233E6PZ		LM4F232E5QC	TM4C123GE6PZ
LM4F122H5QC	TM4C1233H6PZ		LM4F232H5QD	TM4C123GH6PGE
LM4F122H5QD	TM4C1233H6PGE		LM4F232H5QC	TM4C123GH6PZ
LM4F131C4QR	TM4C1236D5PM		LM4F232H5BB	TM4C123GH6ZRB

3.5 *DriverLib 变更*

TivaWare 中的 DriverLib API 支持所有曾经在 Stellaris 器件上使用的外设和函数以及现在 Tiva 器件上使用的外设和函数。 函数名称和操作保持不变；总的来说，唯一使用 TivaWare DriverLib 所需的源代码改动与 C99 数据类型相关（请见3.3 节）。 根据您的 toolchain，这些改动可能也是不需要的。 如之前所述，某些 toolchain 允许 C99 类型与使用以前数据类型的已定义大小相等项目的混用，而不会生成警告。

3.5.1 被删除的外设

在从 StellarisWare 迁移至 TivaWare 过程中，I^2S，EPI 和以太网模块已经从 DriverLib 中删除。 这些外设在任一现有的 Tiva 部件上已不可用，因此也就没有必要保留它们了。 当支持这些接口的 Tiva 系列器件被发布时，相关模块或它们的替代产品将被添加到 DriverLib 中。

3.5.2 被删除的具有 API 或标签的模块

在移动至库的 TivaWare 版本的过程中，已经将 DriverLib 内几个模块的函数和标签删除。 在大多数情况下，已经在 StellarisWare 将这些函数或标签标记为过时不用，并且从 TivaWare 头文件中将它们删除也仅仅是完成这个不再使用的过程。 在所有情况下，可直接替换函数或标签，并且使用过时不再使用值的代码可轻松更新为使用替代值。

在少数情况下，在 StellarisWare 中还有一些函数，这些函数支持任何 Tiva 部件不再需要的功能；这些函数已经从 TivaWare 中删除。 可从现有代码中安全删除这些调用。

下面的子部分检查了受影响的函数和文本属性，并且提供了与函数相关的对应替换代码或注释。

3.5.2.1 模数转换器 (ADC)

函数名称	替代产品	注释
ADCResolutionSet	无	所有 Tiva 部件包含 12 位 ADC，所以这些函数在 TivaWare 中是多余的
ADCResolutionGet		

3.5.2.2 CAN

函数名称	替代产品	注释
CANSetBitTiming	CANBitTimingSet	之前已不再使用
CANGetBitTiming	CANBitTimingGet	之前已不再使用

3.5.2.3 比较器

函数名称	替代产品	注释
COMP_OUTPUT_NONE	COMP_OUTPUT_NORMAL	之前已不再使用

3.5.2.4 闪存

函数名称	替代产品	注释
FlashIntGetStatus	FlashIntStatus	之前已不再使用
FlashUsecGet	无	任何 Tiva C 系列器件上都不需要这个函数。
FlashUsecSet	无	任何 Tiva C 系列器件上都不需要这个函数。

3.5.2.5 通用输出输出 (GPIO)

如果将要使用函数**GPIOPinConfigure()**的话，那么除了 gpio.h 之外，现在必须包含 Pin_map.h。 这个头文件之前被包含在 gpio.h 内。

函数名称	替代产品	注释
GPIOPinIntEnable	GPIOIntEnable	为了与其它外设保持一致，函数被重命名。
GPIOPinIntDisable	GPIOIntDisable	
GPIOPinIntStatus	GPIOIntStatus	
GPIOPinIntClear	GPIOIntClear	
GPIOPortIntRegister	GPIOIntRegister	
GPIOPortIntUnregister	GPIOIntUnregister	

3.5.2.6 休眠

函数名称	替代产品	注释
HibernateClockSelect	无	在任何 Tiva C 系列器件上不需要此函数。
HibernateEnable	HibernateEnableExpClk	之前已不再使用。
HibernateRTCMatch0Set	HibernateRTCMatchSet	替代函数将匹配寄存器索引作为一个参数。 这个方法可在具有不同数量匹配寄存器的器件上实现一个更加简单清洁、更加灵活的执行。
HibernateRTCMatch0Get	HibernateRTCMatchGet	
HibernateRTCMatch1Set	HibernateRTCMatchSet	
HibernateRTCMatch1Get	HibernateRTCMatchGet	
HibernateRTCSSMatch0Set	HibernateRTCSSMatchSet	
HibernateRTCSSMatch0Get	HibernateRTCSSMatchGet	

3.5.2.7 I^2C

函数名称	替代产品	注释
I2CMasterInit	I2CMasterInitExpClk	之前已不再使用。

函数名称	替代产品	注释
I2Cn_MASTER_BASE	I2Cn_BASE	I^2C 是唯一一个使用 2 个单独基地址标签定义的 StellarisWare 外设。 现在每个 I^2C 事件由一个单个基地址标签识别以确保与所有其它外设的一致性。 仍然定义了 hw_i2c.h 中的所有寄存器偏移标签，但是针对从属寄存器的值已经被修改，现在它们与单个外设基地址相对应，而非从属寄存器块的基地址相对应。
I2Cn_SLAVE_BASE		

3.5.2.8 脉宽调制 (PWM)

函数名称	替代产品	注释
PWM_INT_FAULT	PWM_INT_FAULTn (0 ≤ n ≤ 3)	之前已不再使用。 现在 PWM 支持高达 4 个故障，所以针对每个故障定义单个标签。

3.5.2.9 同步串行接口 (SSI)

函数名称	替代产品	注释
SSIConfig	SSIConfigSetExpClk	之前已不再使用
SSIDataNonBlockingGet	SSIDataGetNonBlocking	之前已不再使用
SSIDataNonBlockingPut	SSIDataPutNonBlocking	之前已不再使用

3.5.2.10 SYSCTL

函数名称	替代产品	注释
SysCtlPinPresent	无	全部 Tiva 器件支持复用，所以这个函数不再相关。
SysCtlI2SMClkSet	无	当前的 Tiva 器件全都不包含 I^2C，所以不再需要这个函数。
SysCtlLDOSet	无	不与任何当前的 Tiva 器件相关。
SysCtlLDOGet	无	不与任何当前的 Tiva 器件相关。

标签名称	替代产品	注释
SYSCTL_PERIPH_WDOG	SYSCTL_PERIPH_WDOG0	之前已不再使用
SYSCTL_PERIPH_ADC	SYSCTL_PERIPH_ADC0	之前已不再使用
SYSCTL_PERIPH_PWM	SYSCTL_PERIPH_PWM0	之前已不再使用
SYSCTL_PERIPH_SSI	SYSCTL_PERIPH_SSI0	之前已不再使用
SYSCTL_PERIPH_QEI	SYSCTL_PERIPH_QEI0	之前已不再使用
SYSCTL_PERIPH_I2C	SYSCTL_PERIPH_I2C0	之前已不再使用
SYSCTL_PERIPH_IEEE1588	无	当前的 Tiva 器件不支持以太网，所以不需要这个特性。
SYSCTL_PERIPH_PLL	无	这个标签用于 SysCtlPeripheralPresent。 然而，锁相环 (PLL) 出现在所有 Tiva 器件上，所以此标签是多余的。
SYSCTL_PERIPH_TEMP	无	这个标签用于 SysCtlPeripheralPresent。 然而，基于 ADC 的内部温度传感器出现在所有 Tiva 部件内，所以此标签是多余的。
SYSCTL_PERIPH_MPU	无	这个标签用于 SysCtlPeripheralPresent。 然而，MPU 出现在所有 Tiva 器件内，所以此标签是多余的。
SYSCTL_PERIPH2_<外设>	SYSCTL_PERIPH_<外设>	Tiva 器件不再支持被用来启用和禁用外设的遗留 LM3S SysCtl 寄存器，所以就不再需要之前被用来区分 SysCtl 寄存器集的传统外设标签了。

3.5.2.11 定时器

标签名称	替代产品	注释
TIMER_CFG_32_BIT_OS	TIMER_CFG_ONE_SHOT	之前已不再使用
TIMER_CFG_32_BIT_OS_UP	TIMER_CFG_ONE_SHOT_UP	之前已不再使用
TIMER_CFG_32_BIT_PER	TIMER_CFG_PERIODIC	之前已不再使用
TIMER_CFG_32_BIT_PER_UP	TIMER_CFG_A_PERIODIC_UP	之前已不再使用
TIMER_CFG_32_RTC	TIMER_CFG_RTC	之前已不再使用
TIMER_CFG_16_BIT_PAIR	TIMER_CFG_SPLIT_PAIR	之前已不再使用

函数名称	替代产品	注释
TimerQuiesce	无	SysCtlPeripheralReset 可被用来执行同样的函数。

3.5.2.12 通用异步收发器 (UART)

函数名称	替代产品	注释
UARTConfigSet	UARTConfigSetExpClk	之前已不再使用
UARTConfigGet	UARTConfigGetExpClk	之前已不再使用
UARTCharNonBlockingGet	UARTCharGetNonBlocking	之前已不再使用
UARTCharNonBlockingPut	UARTCharPutNonBlocking	之前已不再使用

3.5.2.13 USB

函数名称	替代产品	注释
USBIntStatus	USBIntStatusControl 或 USBIntStatusEndpoint 由使用的端点而定。	之前已不再使用。 对于端点 0（控制端点）和所有其它端点，现在使用不同的 API 来处理中断。
USBIntDisable	USBIntDisableControl 或 USBIntDisableEndpoint 由使用的端点而定。	之前已不再使用。 对于端点 0（控制端点）和所有其它端点，现在使用不同的 API 来处理中断。
USBIntEnable	USBIntEnableControl 或 USBIntEnableEndpoint 由使用的端点而定。	之前已不再使用。 对于端点 0（控制端点）和所有其它端点，现在使用不同的 API 来处理中断。
USBDevEndpointConfig	USBDevEndpointConfigSet	之前已不再使用
USBHostPwrFaultConfig	USBHostPwrConfig	之前已不再使用

标签名称	替代产品	注释
USB_HOST_PWREN_LOW	USB_HOST_PWREN_AUTOLOW	之前已不再使用
USB_HOST_PWREN_HIGH	USB_HOST_PWREN_AUTOHIGH	之前已不再使用
USB_HOST_PWREN_VBLOW	USB_HOST_PWREN_AUTOLOW	之前已不再使用
USB_HOST_PWREN_VBHIGH	USB_HOST_PWREN_AUTOHIGH	之前已不再使用
USB_INT_*	USB_INTCTRL_* 或 USB_INTEP_* 取决于中断所涉及的端点。	之前已不再使用。 对于控制端点和其它端点，中断已经被分成两组。

3.5.2.14 uartstdio

函数名称	替代产品	注释
UARTStdioInit	UARTStdioConfig	UARTStdio 之前提供 3 个可对模块进行配置的 API。UARTStdioInit 不支持波特率选择。而这个功能已经被添加到 UARTStdioInitExpClk 中。对于 Tiva 器件，对时钟配置的更改要求一个全新函数也将 UART 模块频率作为一个参数；因此，我们判定用一个单个新函数来替代头两个函数，而不是使用第三个功能重叠的函数，是一个更加简单清洁的解决方案。 因此，现在提供了 UARTStdioConfig；这个函数将所需的波特率和 UART 模块时钟速率作为参数。
UARTStdioInitExpClk	UARTStdioConfig	

3.6 图形库变更

结构在 StellarisWare 图形库 API 中发挥了重大作用。 因此，大多数将图形代码移动至 TivaWare 中时所需的源代码变更与字段名称的变化相关，这是由新的数据类型和相关的不同匈牙利前缀表示法所引起的。 然而，除了数据类型，在从 StellarisWare 转换为 TivaWare 的过程中，无需改变 GrLib API 函数。

一个值得注意的数据类型变化是另外一个与字体相关的匈牙利前缀表示法更正示例。 在 StellarisWare 中，由图形库输出的字体指针被命名为*g_pFontSomething*。 为了符合新的、更加严格的匈牙利符号规则，这些指针已经被更改为*g_psFontSomething*。 之前包含在图形库中的所有字体仍然存在，但是采用这个全新的命名规则。

如果您的应用使用其自身的定制字体、字符串表、或图像，请注意*ftrasterize*，*mkstringtable*和*pnmtoc*工具已经被更新以生成与这些数据类型和 TivaWare 中命名规则变化兼容的输出。

在图形库显示驱动程序接口中也进行了一个更改。 在过去，**PixelDrawMultiple**函数的 lBPP 参数（现在被重命名为 i32BPP）包含值 1，4 或 8 以表示源图像颜色分辨率。 这个条件仍然为真，但是此参数的最高有效 24 位现在作为可选标志，这些标志有助于优化某些驱动器的性能。 由于这个变更，现有显示驱动程序必须可靠以清除此参数顶部三个字节来提取每像素比特位数量信息。

此外，已经执行了一个也许有助于显示驱动程序的优化标志。 在首次调用**PixelDrawMultiple**获得一个新图像时，i32BPP 的`GRLIB_DRIVER_FLAG_NEW_IMAGE`（位 30）被置位。 这个操作指示驱动程序，告诉它应该重新建立任一查询表，查询表用来根据已传送的调色板来提交图像。 如果这个标志被清零，此驱动程序可假定图像调色板自最后一次调用后未发生改变，并且避免了重建其查询表的开销。

3.7 USB 库变更

除了基本数据类型和匈牙利前缀表示法变化，已经在 USB 库中做出了几个要求源代码变更的额外小改动。

注: 与 USB 技术规范中定义的结构相映射的现有 USBLib 结构已经被修改以使用全新的匈牙利类型前缀。 这些结构成员的名称仍然与此字段的 USB 技术规范名称向匹配。 此类结构包括：

- *tUSBRequest*
- *tDescriptorHeader*
- *tDeviceDescriptor*
- *tDeviceQualifierDescriptor*
- *tConfigDescriptor*
- *tBOSDescriptor*
- *tInterfaceDescriptor*
- *tEndpointDescriptor*
- *tString0Descriptor*
- *tStringDescriptor*

这个描述枚举数据类型成员的新规范已经被应用于*tUSBMode*，从而需要替换（例如）现在提及*USB_MODE_OTG*的地方为*eUSBModeOTG*。

3.7.1 VID

现有经 TI/Stellaris 供应商 ID**0x1CBE**再许可的应用将会发现，标签*USB_VID_STELLARIS*不再包含在 usb-ide.h 头文件内。 这个标签已经被替换为*USB_VID_TI_1CBE*。

3.7.2 器件类别私有实例数据

USB 器件私有实例数据的处理已经变化为简化应用。 在 StellarisWare 中，诸如*tUSBDBulkDevice*，*tUSBDHIDDevice*和*tUSBDCompositeDevice*的 USB 器件定义结构包含指向私有实例结构的指针，此器件类别用这个实例结构在应用运行期间存储器件状态信息。 这个间接性已经被删除，现在私有实例数据结构已经被嵌入到器件定义结构本身内部。 因此，应用不再需要声明独立于器件定义之外的器件工作区。

当在 TivaWare 中初始化器件定义结构时，此应用能够忽略私有实例结构，这是因为此器件类别初始化函数执行全部所需的结构初始化。

以散装器件为例。 StellarisWare 源代码也许以这种方式出现。

```
tBulkInstance g_sBulkInstance;

const tUSBDBulkDevice g_sBulkDevice =
{
    USB_VID_STELLARIS,
    USB_PID_BULK,
    500,
    USB_CONF_ATTR_SELF_PWR,
    USBBufferEventCallback,
    (void *)&g_sRxBuffer,
    USBBufferEventCallback,
    (void *)&g_sTxBuffer,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_sBulkInstance
};
```

现在应该按照以下方式声明这个代码结构。 请注意，由于此结构的某些内部字段由 USBLib 写入，此结构一定不能再被标记为*const*。 因此，它必须保存在 RAM 中，而非闪存中。

```
tUSBDBulkDevice g_sBulkDevice =
{
    USB_VID_TI_1CBE,
    USB_PID_BULK,
    500,
    USB_CONF_ATTR_SELF_PWR,
    USBBufferEventCallback,
    (void *)&g_sRxBuffer,
    USBBufferEventCallback,
    (void *)&g_sTxBuffer,
    g_ppui8StringDescriptors,
    NUM_STRING_DESCRIPTORS
    //
    // No initializer necessary for the instance data.
    //
};
```

3.7.3 复合器件变更

除了上面描述的实例数据变化（此变化也会影响*tUSBDCompositeDevice*），初始化一个复合 USB 器件的方法已经被简化。 这个变化使得应用无需在每个包含此复合器件的器件类别实例上初始化包含信息的*tCompositeEntry*数组的需要，并且使所有私有实例数据对应用不可见。 此*tCompositeEntry*数组现在由**USBC<class>CompositeInit()**函数内的单独类初始化；私有数据在每个器件结构内部直接例示，而不是作为一个独立缓冲器由应用声明。

执行一个复合器件的代码支持 2 个 CDC 串行器件，此代码也许已经被写入，所采用的方法与以下摘录中使用的方法相类似：

```
//*****************************************************************************
//
// The array of devices supported by this composite device.
//
//*****************************************************************************
tCompositeEntry g_psCompDevices[2]=
{
    //
    // Serial port 0 device instance.
    //
    {
        &g_sCDCDeviceInfo,
        0
    },

    //
    // Serial port 1 device instance.
    //
    {
        &g_sCDCDeviceInfo,
        0
    }
};

//
// …other structure definitions removed for clarity.
//
```

```
//*****************************************************************************
//
// The memory allocated to hold the composite descriptor that is created by
// the call to USBDCompositeInit().
//
//*****************************************************************************
#define DESCRIPTOR_DATA_SIZE (COMPOSITE_DCDC_SIZE * 2)
uint8_t g_pui8DescriptorData[DESCRIPTOR_DATA_SIZE];

//
// Initialize each of the CDC instances and add them to the composite
// device interface array.
//
g_sCompDevice.psDevices[0].pvInstance =
USBDCDCCompositeInit(0, &g_psCDCDevice[0]);
g_sCompDevice.psDevices[1].pvInstance =
USBDCDCCompositeInit(0, &g_psCDCDevice[1]);

//
// Pass the device information to the USB library and place the device
// on the bus.
//
USBDCompositeInit(0, &g_sCompDevice, DESCRIPTOR_DATA_SIZE,
g_pui8DescriptorData);
```

在 TivaWare，这个执行应该重写如下：

```
//*****************************************************************************
//
// The array of devices supported by this composite device. We no longer need
// to initialize this array.
//
//*****************************************************************************
tCompositeEntry g_psCompDevices[2];

//
// …other structure definitions removed for clarity.
//

//*****************************************************************************
//
// The memory allocated to hold the composite descriptor that is created by
// the call to USBDCompositeInit().
//
//*****************************************************************************
#define DESCRIPTOR_DATA_SIZE (COMPOSITE_DCDC_SIZE * 2)
uint8_t g_pui8DescriptorData[DESCRIPTOR_DATA_SIZE];

//
// Initialize each of the CDC instances and add them to the composite
// device interface array.
//
USBDCDCCompositeInit(0, &g_psCDCDevice[0], &g_psCompDevices[0]);
USBDCDCCompositeInit(0, &g_psCDCDevice[1], &g_psCompDevices[1]);

//
// Pass the device information to the USB library and place the device
// on the bus.
//
USBDCompositeInit(0, &g_sCompDevice, DESCRIPTOR_DATA_SIZE, g_pui8DescriptorData);
```

4 结论

TivaWare 提供与 StellarisWare Cortex-M4F MCU 一样的 API，所具有的改动是为了在单个软件架构和目录结构内支持德州仪器 (TI) 的多个系列全新 Tiva MCU 产品。 在这个全新的 TivaWare 软件环境中建立您现有的 StellarisWare 应用会需要一些轻微的源代码和项目文件改动，其中的大多数已经在您选择的编辑器中通过简单地使用搜索和替换操作完成。

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122