Application Report
SPMA050A–April 2013–Revised May 2013

Migrating Software Projects from StellarisWare® to TivaWare™

David Wilson *Stellaris® Microcontrollers*

ABSTRACT

As TI launches the Tiva™ family of 32-bit ARM® Cortex™-M4 microcontrollers (MCUs), the StellarisWare® suite of comprehensive software tools is also migrating to TivaWare™. This new software platform offers the same features found in earlier versions, but a few changes have been made in the migration process that require simple modifications to be made in existing StellarisWare applications in order for them to build and run within the new *TivaWare for C Series* software tree. This document details the differences between StellarisWare for LM3S and LM4F MCUs and TivaWare for C Series MCUs, and provides guidance on the changes that are needed in existing customer code to migrate from the older platform to the new one.

Unless otherwise indicated, the terms *TivaWare* and *TivaWare for C Series* are synonymous.

Contents

List of Tables

1 Background

The StellarisWare software development platform has served TI's Stellaris ARM Cortex-M3 and Cortex-M4F microcontrollers well since its introduction in 2006. With the expansion of TI's microcontroller offerings to now include multiple series of ARM Cortex-based parts that support different market segments and target applications, however, StellarisWare is becoming TivaWare. This migration requires that some simple changes be made in the platform architecture and implementation. These minor changes have been made for several reasons:

- To allow support for multiple, different MCU series under the same TivaWare umbrella;
- To enhance portability of the source code across different processor platforms; and
- To correct known architectural deficiencies and inconsistencies in the StellarisWare code.

All existing StellarisWare features and functionality are maintained in TivaWare for C Series, and all APIs in the Peripheral Driver Library, Graphics Library, and USB Library that are applicable to Tiva parts are maintained in the new platform. The required changes are not completely backwards-compatible, so some source code and project file modifications are necessary in order to port applications that were previously built with StellarisWare to be able to build correctly with TivaWare.

Programmers value backwards-compatibility. TI has tried to balance the need for—and long-term benefits of—these changes with the inconvenience caused by some application source modifications in the migration process. We believe that the migration process is not a difficult challenge, and that most of your applications can work properly with TivaWare without a great deal of effort or costly software re-development.

Each series of Tiva parts has its own software installation with contents tailored to the features of that specific series. This document primarily addresses migrating a StellarisWare application to the TivaWare for C Series environment. One of the goals of the new approach to Tiva series software is to provide consistent APIs and environments across all Tiva series; therefore, the information presented here is broadly applicable to other TivaWare series versions as they become available.

2 Quick Start Guide

Application source code changes are required because of these differences between StellarisWare and TivaWare:

- Directory structures and library naming have changed.
- C99 data types are now used throughout TivaWare.
- Hungarian prefixes have changed, resulting in name changes for many structure fields.
- Existing Stellaris LM4F device part numbers have changed.
- USB device initialization has been simplified.
- Functions and labels that were deprecated in StellarisWare have been removed from TivaWare.

To allow you to get started on changes to your project, the following procedures provide an overview of the work required to build and link a typical StellarisWare project in the new TivaWare tree. Users with applications that do not use the USB Library may find that the changes described in this list are sufficient to allow them to build and run in the new TivaWare tree. For USB-enabled applications, there are a handful of other small changes required; these changes are described in a later section.

1. Copy your existing project directory to an equivalent location in the new TivaWare tree. For example, if your application is at *C:\StellarisWare\boards\<your board>\<your app>*, copy it to *C:\ti\TivaWare_C_Series_1.0\boards\<your board>\<your app>*.

   NOTE: Make sure you copy any board-dependent driver code that you developed, typically found in C:\StellarisWare\boards\<your board>\drivers, and understand that you must create the *boards* directory in the TivaWare tree. **Pre-built board examples have moved to a new location.**
2. Add #includes for <stdint.h> and <stdbool.h> at the top of all source files that include StellarisWare or TivaWare headers. These headers define the C99 standard data types now used in TivaWare.
3. If your source file includes *utils/ustdlib.h*, add a #include for <time.h> at the top of the file. This addition is required to pick up the standard definition of *struct tm*, that is now used in place of the proprietary *tTime* type
4. Globally replace *tBoolean* in your source code with the standard *bool* type.

5. Globally replace *tTime* with *struct tm*. If your application uses *tTime*, you must rework the code to use the equivalent fields from the C standard *struct tm*. This modification is a straightforward substitution of names such as **ucMon** or **ucYear** with **tm_mon** or **tm_year**.
6. If the application uses any StellarisWare API that requires a structure from one of the StellarisWare headers, check the field names used in your source code because many of these names have changed to comply with our new Hungarian prefix convention and use of C99 data types.
7. If using the **GPIOPinConfigure()** function, modify your project settings or Makefile to replace the existing **PART_LM4Fxxx** label with the replacement part number (refer to Table 5).
8. If the application uses any part-specific header file of the form *inc/lm4f*.h*, replace this name with the equivalent *tm4c* part header as described in Section 3.4.
9. Modify your project settings or Makefile to link the Driver Library, Graphics Library, and/or USB Library from the new locations within the TivaWare tree. The TivaWare libraries are placed as they were in the StellarisWare tree but have had the *-cm3* and *-cm4* suffixes removed from both the toolchain directory names and the library names. See the Library Naming section for more information.

3 Specific API and Source Code Changes

3.1 Directory Structure Changes

The TivaWare directory structure closely mirrors the structure used in StellarisWare releases with two exceptions:

- The default installation directory has changed from *C:\StellarisWare* to *C:\ti\TivaWare_C_Series_<version>*
- Board-specific example applications have been moved from the *boards* subdirectory to the *examples\boards* subdirectory.

The first change allows for a cleaner overall directory structure when using multiple TivaWare installations, or installations for different series of Tiva MCUs. This new default directory also ensures that different versions can be installed side-by-side without overwriting one another if the default directory is chosen during installation. Of course, the installation directory can be overwritten very easily during the installation process; users are free to install the TivaWare tree in any directory that suits their specific application needs. No software or directory node within TivaWare makes any assumption about the name or position of the root directory.

The second change reduces confusion over the location of code examples. In StellarisWare, code examples could be found under two different subdirectories: **boards** for board-specific example applications, and **examples** for peripheral-specific examples. All example source code can now be found under the single **examples** directory with lower-level subdirectories for boards and peripherals.

StellarisWare users have been free to create their own application projects anywhere in their respective file systems. This flexibility is also present within TivaWare. For example, if your project resided inside the *C:\StellarisWare* directory, moving it to the same location under *C:\TI\TivaWare-C-Series-<version>* allows it to compile without including any path-related modifications because the relative paths to all header files remain unchanged. A previous project stored in *C:\StellarisWare\boards\<your board>\<your app>* can be moved into *C:\TI\TivaWare-C-Series\boards\<your board>\<your app>* (even though the base TivaWare release does not contain the *boards* directory at this level) and remains able to be compiled without any path changes required in the code.

For projects stored outside the *C:\StellarisWare* subtree, or projects stored within the tree but that contain references to items outside the tree, some modifications to the toolchain project or makefile are necessary to account for the fact that the TivaWare libraries and headers are now in a different location as a result of the change in the default installation directory.

3.2 Library Naming

The Stellaris family included MCUs based on both the Cortex-M3 and Cortex-M4F architectures; all Tiva devices are M4F-based. As a result, the names of libraries within TivaWare have been simplified to remove the core-specific suffix used in recent StellarisWare builds. Table 1 lists the names for both TivaWare and StellarisWare libraries.

Table 1. Library Naming

Description	Toolchain	StellarisWare File	TivaWare File
Peripheral Driver Library	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 and Code Bench	driverlib/gcc-cm3/libdriver-cm3.a	driverlib/gcc/libdriver.a
		driverlib/gcc-cm4f/libdriver-cm4f.a	
Graphics Library	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 and Code Bench	grlib /gcc-cm3/libgr-cm3.a	grlib/gcc/libgr.a
		grlib /gcc-cm4f/libgr-cm4f.a	
USB Library	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 and Code Bench	usblib/gcc-cm3/libusb-cm3.a	usblib/gcc/libusb.a
		usblib/gcc-cm4f/libusb-cm4f.a	

3.3 C99 Types and Hungarian Prefix Changes

The move from StellarisWare to TivaWare provided an opportunity to correct one aspect of StellarisWare architectural decision that we have wanted to address for several years. StellarisWare APIs use simple C data types such as *unsigned long*; however, these types typically suffer from a significant problem: the size of each type differs depending on the CPU on which they are used. While this distinction may not be an issue when using StellarisWare with a single microprocessor family, it created several portability problems and left the API open to further issues as the underlying microprocessor architecture evolved over time.

To address this potential issue, and to allow the software to be ported to other architectures without the need for significant source code rework, TivaWare now uses standard, unambiguously-sized C99 data types. By using these types, for example, a 32-bit entity is guaranteed to remain at 32 bits regardless of the processor on which the code is run.

Although the StellarisWare API has migrated to TivaWare, this data type change has several consequences that require existing applications to be reworked if they previously used StellarisWare APIs and are now to be used with TivaWare. First, C99 types are not compiler-intrinsic; instead, these types are defined via standard C runtime headers: **stdint.h** for basic types and **stdbool.h** for the *bool* Boolean type. Both StellarisWare and TivaWare headers have a policy of not nesting other headers, so these two standard headers must now be added to all source files that include any TivaWare header to ensure that the required basic data types are available.

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

While making this change, we also decided to remove the use of a proprietary data type, *tTime*, to represent date and time values. Instead, functions in the utils/ustdlib files have been reworked to use the standard *struct tm* structure instead of *tTime*. Consequently, source files that include utils/ustdlib.h must also include the standard *time.h* header file.

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

Although source code modifications are required as a result of the use of *struct tm*, the field names in *struct tm* closely mirror those in *tTime*, making the modifications straightforward. The field names in *struct tm* and *tTime* are mapped as Table 2 shows.

Table 2. Field Name Mappings for *struct tm* and *tTime*

tTime Field Name	struct tm Field Name	Notes
usYear	tm_year	*tm_year* is defined as “Years since 1900” whereas *usYear* contained the actual year. This difference must be taken into account when reworking code that uses this field.
ucMon	tm_mon	
ucMday	tm_mday	
ucWdat	tm_wday	
ucHour	tm_hour	
ucMin	tm_min	
ucSec	tm_sec	

The second consequence of the C99 type change—and the one that more likely requires source code updates—is the associated change of Hungarian prefixes. While this change poses no problems for parameter naming, it does change the names of most structure fields, and therefore impacts users of USBLib and GrLib. In addition to changing the prefix used as a result of the new data types, TI has made every effort to ensure that Hungarian prefixes are used consistently throughout the codebase. This change has resulted in some additional structure field name changes where the previous name did not employ the correct prefix.

The list of data types used in StellarisWare along with their Hungarian prefixes and the new TivaWare replacements is given in Table 3.

Table 3. Data Types Used in StellarisWare and TivaWare

StellarisWare Type	Old Prefix	TivaWare Type	New Prefix	Example
tBoolean	b	bool	b	bFoo
char(1)	c	char	c	cFoo
char(2)	c	int8_t	i8	i8Foo
short	s	int16_t	i16	i16Foo
long	l	int32_t	i32	i32Foo
long long	ll	int64_t	i64	i64Foo

(1) When used to represent a text character encoding.
(2) When used to represent an 8-bit signed number.

Table 3. Data Types Used in StellarisWare and TivaWare (continued)

StellarisWare Type	Old Prefix	TivaWare Type	New Prefix	Example
unsigned char	uc	uint8_t	ui8	ui8Foo
unsigned short	us	uint16_t	ui16	ui16Foo
unsigned long	ul	uint32_t	ui32	ui32Foo
unsigned long long	ull	uint64_t	ui64	ui64Foo

Table 4 lists the Hungarian prefixes that were often used inconsistently in StellarisWare. Affected variables and field names have been changed in TivaWare:

Table 4. Hungarian Prefixes Changed in TivaWare

Type	Prefix	Example
pointer	p<prefix>	pcFoo, pui32Foo
typedef	t	tFoo
enumeration values	e	eFoo
function pointer	pfn	pfnFoo
structure variable	s	sFoo
union variable	u	uFoo
enumeration variable	i	iFoo
array	p<prefix>	pcFoo[], pui32Foo[]
Pointer to pointer or two-dimensional array	pp<prefix>	ppcFoo, ppui32Foo

The effect of passing parameters using the old data types to TivaWare functions varies depending on the toolchain and switches in use. When using GCC 4.3.6 with *–Wall –pedantic*, for example, passing *unsigned long* variables into functions and expecting *uint32_t* parameters produces no warning. Doing the same action when using Keil RVMDK with *All warnings* enabled but *Strict ANSI C* disabled, however, yields a warning but generates output that links and runs correctly.

3.4 Part Number Changes

Pre-production **LM4F** Stellaris part numbers are changing to **TM4C** (Tiva C Series) part numbers as the parts are fully qualified for production status. The old and new parts are functionally identical, but the part number change affects some software that uses the part-specific header files found in the *inc* directory of a StellarisWare or TivaWare release. These headers are named for the specific part number in use and contain all register definitions relevant to that specific part. If your project code uses one of these headers, use Table 5 to determine the new part number for your MCU, and replace the *lm4f* header with the equivalent *tm4c* version. Label definitions carry forward from the old headers to the new ones, so no source code modification other than the filename change is required as a result of this change.

Software that uses the DriverLib API does not generally use part-specific header files, with the exception of the **GPIOPinConfigure()** function that makes used of label definitions found in pin_map.h. This header contains label definitions tailored for each specific part. These definitions are controlled by a preprocessor definition of the form PART_X where ***X*** is the target part number. Because pin_map.h now uses Tiva part numbers, the label defined in the application makefile or project must be modified to ensure that the correct set of part-specific pin muxing labels is defined.

As an example, if your application was previously targeting an LM4F232H5QD part, and your project or makefile defines the label *PART_LM4F232H5QD*, you must replace this label with *PART_TM4C123GH6PGE* to ensure that the correct pin muxing labels are included for the target device.

Table 5. Replacement Tiva C Series Part Numbers for LM4F Part Numbers

LM4F Part No	TM4C Part No		LM4F Part No	TM4C Part No
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 Changes

All peripherals and functions that were on Stellaris devices and are on Tiva devices are supported by the DriverLib API in TivaWare. Function names and operations are unchanged; in general, the only source code changes required to use the TivaWare DriverLib relate to C99 data types (see Section 3.3). Depending on your toolchain, these changes may also prove unnecessary. As noted earlier, some toolchains allow intermixing of C99 types and equivalently-sized objects defined that use the previous types without generating warnings.

3.5.1 Peripherals Removed

The I^2S, EPI, and Ethernet modules have been removed from DriverLib in the migration from StellarisWare to TivaWare. These peripherals are not available on any current Tiva parts, and are therefore unnecessary. When Tiva series devices are released that support these interfaces, the relevant modules or their replacements will be added to DriverLib.

3.5.2 Modules with APIs or Labels Removed

Several modules within DriverLib have had functions or labels removed in the process of moving to the TivaWare version of the library. In most cases, these functions or labels were already marked as deprecated in StellarisWare, and the removal from the TivaWare header files is merely the completion of this deprecation process. In all cases, a straightforward replacement function or label exists, and code that uses the deprecated value can be easily updated to use the replacement value.

In a handful of cases, functions within StellarisWare existed to support features that are no longer required on any Tiva part; these functions have been removed from TivaWare. These calls can be safely deleted from existing code.

The following subsections review the affected functions and text attributes, and provide the corresponding replacement code or comments about the function.

3.5.2.1 ADC

Function Name	Replacement	Notes
ADCResolutionSet	None	All Tiva parts contain 12-bit ADCs so these functions are redundant in TivaWare
ADCResolutionGet		

3.5.2.2 CAN

Function Name	Replacement	Notes
CANSetBitTiming	CANBitTimingSet	Previously deprecated
CANGetBitTiming	CANBitTimingGet	Previously deprecated

3.5.2.3 Comparator

Function Name	Replacement	Notes
COMP_OUTPUT_NONE	COMP_OUTPUT_NORMAL	Previously deprecated

3.5.2.4 Flash

Function Name	Replacement	Notes
FlashIntGetStatus	FlashIntStatus	Previously deprecated
FlashUsecGet	None	This function is not required on any Tiva C Series devices.
FlashUsecSet	None	This function is not required on any Tiva C Series devices.

3.5.2.5 GPIO

Pin_map.h must now be included in addition to gpio.h if the function **GPIOPinConfigure()** is to be used. Previously, this header was included within gpio.h.

Function Name	Replacement	Notes
GPIOPinIntEnable	GPIOIntEnable	Functions renamed for consistency with other peripherals.
GPIOPinIntDisable	GPIOIntDisable	
GPIOPinIntStatus	GPIOIntStatus	
GPIOPinIntClear	GPIOIntClear	
GPIOPortIntRegister	GPIOIntRegister	
GPIOPortIntUnregister	GPIOIntUnregister	

3.5.2.6 Hibernate

Function Name	Replacement	Notes
HibernateClockSelect	None	Function is not required on any Tiva C Series device.
HibernateEnable	HibernateEnableExpClk	Previously deprecated
HibernateRTCMatch0Set	HibernateRTCMatchSet	The replacement functions take the match register index as a parameter. This method allows a cleaner, more flexible implementation across devices with different numbers of match registers.
HibernateRTCMatch0Get	HibernateRTCMatchGet	
HibernateRTCMatch1Set	HibernateRTCMatchSet	
HibernateRTCMatch1Get	HibernateRTCMatchGet	
HibernateRTCSSMatch0Set	HibernateRTCSSMatchSet	
HibernateRTCSSMatch0Get	HibernateRTCSSMatchGet	

3.5.2.7 I^2C

Function Name	Replacement	Notes
I2CMasterInit	I2CMasterInitExpClk	Previously deprecated

Function Name	Replacement	Notes
I2Cn_MASTER_BASE	I2Cn_BASE	I^2C was the only StellarisWare peripheral to be defined using two independent base address labels. Each I^2C instance is now identified using a single base address label to ensure consistency with all other peripherals. All register offset labels in hw_i2c.h are still defined, but the values for the slave registers have been modified such that they are now relative to the single peripheral base address rather than the base address of the slave register block.
I2Cn_SLAVE_BASE		

3.5.2.8 PWM

Function Name	Replacement	Notes
PWM_INT_FAULT	PWM_INT_FAULTn (0 ≤ n ≤ 3)	Previously deprecated. PWM now supports up to four faults, so individual labels are defined for each fault.

3.5.2.9 SSI

Function Name	Replacement	Notes
SSIConfig	SSIConfigSetExpClk	Previously deprecated
SSIDataNonBlockingGet	SSIDataGetNonBlocking	Previously deprecated
SSIDataNonBlockingPut	SSIDataPutNonBlocking	Previously deprecated

3.5.2.10 SYSCTL

Function Name	Replacement	Notes
SysCtlPinPresent	None	Tiva devices all support pin muxing, so this function is no longer relevant.
SysCtlI2SMClkSet	None	No current Tiva device includes I^2S so this function is not required.
SysCtlLDOSet	None	Not relevant to any current Tiva device.
SysCtlLDOGet	None	Not relevant to any current Tiva device.

Label Name	Replacement	Notes
SYSCTL_PERIPH_WDOG	SYSCTL_PERIPH_WDOG0	Previously deprecated
SYSCTL_PERIPH_ADC	SYSCTL_PERIPH_ADC0	Previously deprecated
SYSCTL_PERIPH_PWM	SYSCTL_PERIPH_PWM0	Previously deprecated
SYSCTL_PERIPH_SSI	SYSCTL_PERIPH_SSI0	Previously deprecated
SYSCTL_PERIPH_QEI	SYSCTL_PERIPH_QEI0	Previously deprecated
SYSCTL_PERIPH_I2C	SYSCTL_PERIPH_I2C0	Previously deprecated
SYSCTL_PERIPH_IEEE1588	None	No current Tiva device supports Ethernet, so this feature is not required.
SYSCTL_PERIPH_PLL	None	This label was intended for use with SysCtlPeripheralPresent. The PLL is present in all Tiva devices, however, so the label is redundant.
SYSCTL_PERIPH_TEMP	None	This label was intended for use with SysCtlPeripheralPresent. The ADC-based internal temperature sensor is present in all Tiva parts, however, so the label is redundant.
SYSCTL_PERIPH_MPU	None	This label was intended for use with SysCtlPeripheralPresent. The MPU is present in all Tiva devices, however, so the label is redundant.
SYSCTL_PERIPH2_<peripheral>	SYSCTL_PERIPH_<peripheral>	Tiva devices no longer support the legacy LM3S SysCtl registers used to enable and disable peripherals, so the additional peripheral labels that were used to differentiate between the SysCtl register sets are no longer required.

3.5.2.11 Timers

Label Name	Replacement	Notes
TIMER_CFG_32_BIT_OS	TIMER_CFG_ONE_SHOT	Previously deprecated
TIMER_CFG_32_BIT_OS_UP	TIMER_CFG_ONE_SHOT_UP	Previously deprecated
TIMER_CFG_32_BIT_PER	TIMER_CFG_PERIODIC	Previously deprecated
TIMER_CFG_32_BIT_PER_UP	TIMER_CFG_A_PERIODIC_UP	Previously deprecated
TIMER_CFG_32_RTC	TIMER_CFG_RTC	Previously deprecated
TIMER_CFG_16_BIT_PAIR	TIMER_CFG_SPLIT_PAIR	Previously deprecated

Function Name	Replacement	Notes
TimerQuiesce	None	SysCtlPeripheralReset may be used to perform the same function.

3.5.2.12 UART

Function Name	Replacement	Notes
UARTConfigSet	UARTConfigSetExpClk	Previously deprecated
UARTConfigGet	UARTConfigGetExpClk	Previously deprecated
UARTCharNonBlockingGet	UARTCharGetNonBlocking	Previously deprecated
UARTCharNonBlockingPut	UARTCharPutNonBlocking	Previously deprecated

3.5.2.13 USB

Function Name	Replacement	Notes
USBIntStatus	USBIntStatusControl or USBIntStatusEndpoint depending on the endpoint that is in use.	Previously deprecated. Interrupts are now handled using different APIs for endpoint 0 (the control endpoint) and all other endpoints.
USBIntDisable	USBIntDisableControl or USBIntDisableEndpoint depending on the endpoint that is in use.	Previously deprecated. Interrupts are now handled using different APIs for endpoint 0 (the control endpoint) and all other endpoints.
USBIntEnable	USBIntEnableControl or USBIntEnableEndpoint depending on the endpoint that is in use.	Previously deprecated. Interrupts are now handled using different APIs for endpoint 0 (the control endpoint) and all other endpoints.
USBDevEndpointConfig	USBDevEndpointConfigSet	Previously deprecated
USBHostPwrFaultConfig	USBHostPwrConfig	Previously deprecated

Label Name	Replacement	Notes
USB_HOST_PWREN_LOW	USB_HOST_PWREN_AUTOLOW	Previously deprecated
USB_HOST_PWREN_HIGH	USB_HOST_PWREN_AUTOHIGH	Previously deprecated
USB_HOST_PWREN_VBLOW	USB_HOST_PWREN_AUTOLOW	Previously deprecated
USB_HOST_PWREN_VBHIGH	USB_HOST_PWREN_AUTOHIGH	Previously deprecated
USB_INT_*	USB_INTCTRL_* or USB_INTEP_* depending on the endpoint referred to by the interrupt.	Previously deprecated. Interrupts have been split into two groups for the control endpoint and other endpoints.

3.5.2.14 uartstdio

Function Name	Replacement	Notes
UARTStdioInit	UARTStdioConfig	Previously, UARTStdio offered three different APIs that allowed the module to be configured. UARTStdioInit did not allow selection of the baud rate. and this capability was added with UARTStdioInitExpClk. Changes to clock configuration for Tiva devices require that a new function also takes the UART module clock frequency as a parameter; therefore, rather than have a third, overlapping function, we determined that replacing the first two with a single new function was a cleaner solution. As a result, UARTStdioConfig is now provided; this function takes both the desired baud rate and UART module clock rate as parameters.
UARTStdioInitExpClk	UARTStdioConfig	

3.6 Graphics Library Changes

Structures play a significant role in the StellarisWare Graphics Library API. As a result, the majority of source changes required to move graphics code to TivaWare are related to changes in the field names because of new data types and associated differing Hungarian prefixes. Except for data types, however, no GrLib API function has been changed in the transition from StellarisWare to TivaWare.

One noteworthy data type change is another example of Hungarian prefix correction that relates to fonts. In StellarisWare, font pointers exported by the graphics library were named *g_pFontSomething*. To comply with the new, stricter Hungarian notation, these pointers have been changed to *g_psFontSomething*. All fonts that were previously included in the Graphics Library are still present, but now employ this new naming convention.

If your application uses its own custom fonts, string tables, or images, note that the *ftrasterize*, *mkstringtable*, and *pnmtoc* tools have been updated to generate output compatible with these data type and naming convention changes in TivaWare.

One change has also been made in the Graphics Library display driver interface. In the past, the lBPP parameter (now renamed i32BPP) to the **PixelDrawMultiple** function contained the value 1, 4, or 8 to indicate the source image color resolution. This condition is still true, but the most significant 24 bits of the parameter are now reserved for optional flags that may help to optimize performance for some drivers. As a result of this change, existing display drivers must be sure to clear the top three bytes of the parameter to extract the number of bits per pixel information.

Additionally, one optimization flag has been implemented that may be helpful for display drivers. On the first call to **PixelDrawMultiple** for a new image, `GRLIB_DRIVER_FLAG_NEW_IMAGE` (bit 30) of i32BPP is set. This action indicates to the driver that it should rebuild any lookup tables it requires to render the image based on the palette passed. If this flag is clear, the driver can assume that the image palette has not changed since the last call, and avoid the overhead of reconstructing its lookup table.

3.7 USB Library Changes

In addition to the basic data type and Hungarian prefix changes, several additional minor changes have been made in the USB Library that require source code changes.

NOTE: Existing USBLib structures that mirror structures defined in the USB specification have not been modified to use the new Hungarian type prefixes. The names of these structure members continue to match the USB specification name for the field. Structures in this category include:

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

The new convention for describing members of an enumerated type has been applied to *tUSBMode*, resulting in the need to replace (for example) existing mentions of *USB_MODE_OTG* with *eUSBModeOTG*.

3.7.1 VID

Existing applications that may have sublicensed the TI/Stellaris vendor ID **0x1CBE** will find that the label *USB_VID_STELLARIS* is no longer included in the usb-ids.h header file. This label has been replaced by *USB_VID_TI_1CBE*.

3.7.2 Device Class Private Instance Data

The handling of USB device private instance data has changed to simplify applications. In StellarisWare, USB device definition structures such as *tUSBDBulkDevice*, *tUSBDHIDDevice*, and *tUSBDCompositeDevice* contained pointers to a private instance structure that the device class used to store device state information during application operation. This indirection has now been removed, and the private instance data structure is now embedded within the device definition structure itself. As a result, applications no longer need to declare device workspace independently of the device definition.

When initializing the device definition structure in TivaWare, the application can ignore the private instance structure because the device class initialization function performs all needed initialization of the structure.

As an example, take a bulk device. The StellarisWare source code may have appeared in this manner:

```
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
};
```

This code structure should now be declared as shown below. Note that the structure must no longer be marked *const* because some of its internal fields are written by USBLib. Therefore, it must be stored in RAM rather than flash.

```
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 Composite device changes

In addition to the instance data change described above (which also affects *tUSBDCompositeDevice*), the method of initializing a composite USB device has been simplified. This change removes the requirement for an application to initialize the *tCompositeEntry* array containing information on each device class instance that comprises the composite device, and also hides all private instance data from the application. The *tCompositeEntry* array is now initialized by the individual classes in the **USBC<class>CompositeInit()** function; private data are instantiated directly within each device structure, rather than as an independent buffer declared by the application.

Code to implement a composite device that supports two CDC serial device instances may have been written in a manner similar to the following snippet:

```
//*****************************************************************************
//
// 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);
```

In TivaWare, this implementation should be rewritten as shown:

```
//*****************************************************************************
//
// 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 Conclusion

TivaWare offers the same API set as provided for Stellaris Cortex-M4F MCUs, with changes intended to support multiple series of TI's new Tiva MCU products within a single software architecture and directory structure. Building your existing StellarisWare application in this new TivaWare software environment requires some minor source code and project file changes, most of which can be accomplished by simply using search-and-replace operations in your editor of choice.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265