HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

Chapter 3: Development tools and software support

Stellaris® Cortex™-M3 - Microcontroller Family

Texas Instruments

Texas Instruments - University Program

Heilbronn University, Campus Künzelsau

Prof. Dr.-Ing. Ralf Gessler **Rev. 1.0**

Content

- **Topics:** Stellaris® Peripheral Driver Library, EK-LM3S1968 Firmware Development Package, API, Direct Register Access, Software Driver Model

Learning Objectives

- The chapter describes the basics (theory) of StellarisWare®
- Embedded C applications introduced
  - "Blinky": simple blinking LED
  - Display Driver API
- Main focus:
  - Stellaris® Peripheral Driver Library
  - EK-LM3S1968 Firmware Development Package
  - Code Samples
- Structure and questions:
  - What is StellarisWare®?
  - What is a Peripheral Driver Library and a Firmware Development Package?
  - What is a Direct Register Access / Software Driver Model?

Overview

- StellarisWare® software is an extensive suite of software designed to simplify and speed development of Stellaris® -based microcontroller applications.
- All StellarisWare® software has a free license and royalty-free use to allow the creation of full-function, easy-to-maintain code.

Main Features

- Possible to use 100% C/C++ programming, even ISR and initialization
- Includes source code and royalty-free libraries for applications support
- Stellaris® Software compiles on:
  - Code Composer Studio 4
  - ARM/Keil Microcontroller Development Toolkit for ARM
  - IAR Embedded Workbench
  - Code Red Technologies' RedSuite
  - Code Sourcery SourceryG++
  - Generic GNU development tools
- Key functional areas (Free license and royalty-free source code):
  - Stellaris Peripheral Driver Library
  - EK-LM3S1968 Firmware Development Package
  - Code Samples: general, board
  - Stellaris® Graphics Library
  - Stellaris® USB Library
  - Stellaris® IQMath Library
  - Stellaris® IEC 60730 Library
  - Stellaris® In-System Programming Support
  - ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS)

Main Features (cont.)

- Stellaris® Peripheral Driver Library
  - The Stellaris® Peripheral Driver Library is a royalty-free set of functions for controlling the peripherals found on the Stellaris® family of ARM Cortex™ -M3 microcontrollers.
  - Note: for use "driverlib.lib" must be included in Linker search path
  - Vastly superior to a GUI peripheral configuration tool, the Stellaris® Peripheral Driver Library performs both peripheral initialization and peripheral control functions with a choice of polled or interrupt-driven peripheral support.
  - Some Stellaris® microcontrollers provide the Stellaris® Peripheral Driver Library on-chip in ROM (read-only memory), leaving on-chip flash for the end application.

- Firmware Development Package
  - Including board-specific drivers and example applications

Main Features (cont.)

- Code Examples
  - Discover the ease of working with the code-efficient easy to use ARM® Cortex™ -M3 platform with its extensive set of sample applications.
  - TI provides these applications royalty-free, so that you can condense your development time and quickly bring your projects to market.
- Stellaris® Graphics Library
  - The Stellaris® Graphics Library is a royalty-free set of graphics primitives and a widget set for creating graphical user interfaces on Stellaris® microcontroller-based boards that have a graphical display.
  - The sample applications and detailed documentation make it easy to integrate rich graphics into projects.
- Stellaris® USB Library
  - Comprehensive sub-set of USB functions simplify embedded USB control
  - Royalty-free sample applications are provided to quickly enable efficient USB Host, USB Device, and USB On-The-Go operations.

Main Features (cont.)

- Stellaris® IQMath Library
  - Texas Instruments' IQmath Library is a collection of highly optimized and high precision mathematical functions for C/C++ programmers to seamlessly port the floating-point algorithm into fixed point code. The IQmath Library also addresses the limitations of fixed point math by defining a programmable dynamic range and resolution.
  - By using these routines you can achieve execution speeds considerably faster than equivalent code written in standard ANSI C language.
- Stellaris® IEC 60730 Support
  - In an effort to help household appliance customers take steps to ensure safe and reliable operation of their Stellaris® -based products, StellarisWare® includes an IEC 60730 Class B Test Library to support its customers in the Class B certification process.
  - In addition, Stellaris® microcontrollers are designed specifically for safety critical industrial and consumer applications, offering several integrated safety features for precision control, connectivity, and monitoring.
  - These libraries are also very useful for testing applications and in the production test environment for end application builds.

Main Features (cont.)

- Stellaris® In-System Programming Support-- Serial Boot Loader
  - For applications requiring in-field programmability, Texas Instruments also provides royalty-free Stellaris® boot loader source code that can be added to your application at the beginning of the flash memory. This small piece of code can act as an application loader and stays resident to support in-field programmability for your end application.
  - With flexible interface options including a UART, I²C, SSI, CAN, USB DFU, or Ethernet and selectable methods for signaling an in-field update, the Stellaris® Boot Loader provides users with maximum flexibility in boot loader requirements.
  - The Stellaris® Peripheral Driver Library includes source code and information about the Stellaris® boot loader, including example applications that utilize the boot loader for in-field updates.
  - Some Stellaris® microcontrollers provide the Stellaris® Boot Loader in read-only memory (ROM) integrated on the device, resulting in a savings of application flash space.
  - Even if the Stellaris® Boot Loader is in ROM, for maximum flexibility in applications, the ROM-based boot loader can be overridden with a customized version in flash.

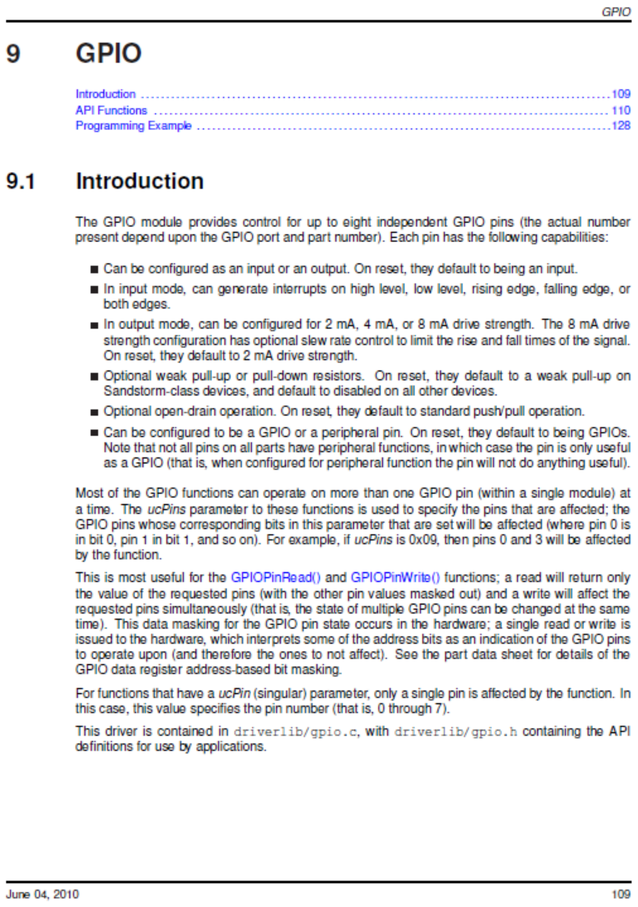
Main Features (cont.)

- Stellaris® In-System Programming Support - Serial Flash Loader
  - All Stellaris® microcontrollers that do not have a ROM-based boot loader ship with a royalty-free serial flash loader application pre-programmed into flash.
  - The serial flash loader is a small application that allows programming of the flash without the need for a debugger interface or production programmer. With easy interface options including UART or SSI, the serial flash loader provides users with maximum flexibility in their production programming options.

Peripheral Driver Lib

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING | BUSINESS | INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- High-level API interface to complete peripheral set
- Free license and royalty-free use
- Simplifies and speeds development of applications
- Can be used for application development or as programming example
- Available as object library and as source code
- Compiles on ARM/Keil, IAR, Code Red, and GNU tools
- Includes Stellaris® Graphics Library and Stellaris® USB Library



GPIO

9 GPIO

9.1 Introduction

The GPIO module provides control for up to eight independent GPIO pins (the actual number present depend upon the GPIO port and part number). Each pin has the following capabilities:

- Can be configured as an input or an output. On reset, they default to being an input.
- In input mode, can generate interrupts on high level, low level, rising edge, falling edge, or both edges.
- In output mode, can be configured for 2 mA, 4 mA, or 8 mA drive strength. The 8 mA drive strength configuration has optional slew rate control to limit the rise and fall times of the signal. On reset, they default to 2 mA drive strength.
- Optional weak pull-up or pull-down resistors. On reset, they default to a weak pull-up on Sandstorm-class devices, and default to disabled on all other devices.
- Optional open-drain operation. On reset, they default to standard push/pull operation.
- Can be configured to be a GPIO or a peripheral pin. On reset, they default to being GPIOs. Note that not all pins on all parts have peripheral functions, in which case the pin is only useful as a GPIO (that is, when configured for peripheral function the pin will not do anything useful).

Most of the GPIO functions can operate on more than one GPIO pin (within a single module) at a time. The *ucPins* parameter to these functions is used to specify the pins that are affected; the GPIO pins whose corresponding bits in this parameter that are set will be affected (where pin 0 is in bit 0, pin 1 in bit 1, and so on). For example, if *ucPins* is 0x09, then pins 0 and 3 will be affected by the function.

This is most useful for the GPIOPinRead() and GPIOPinWrite() functions; a read will return only the value of the requested pins (with the other pin values masked out) and a write will affect the requested pins simultaneously (that is, the state of multiple GPIO pins can be changed at the same time). This data masking for the GPIO pin state occurs in the hardware; a single read or write is issued to the hardware, which interprets some of the address bits as an indication of the GPIO pins to operate upon (and therefore the ones to not affect). See the part data sheet for details of the GPIO data register address-based bit masking.

For functions that have a *ucPin* (singular) parameter, only a single pin is affected by the function. In this case, this value specifies the pin number (that is, 0 through 7).

This driver is contained in `driverlib/gpio.c`, with `driverlib/gpio.h` containing the API definitions for use by applications.

June 04, 2010 109

- DriverLib functions are preprogrammed in ROM on select Stellaris® MCUs

Peripheral Driver Lib (cont.)

- Collection of ‘c’ source and header files
- Base root directory structure with individual compiler specific library output directory.
- “.\StellarisWare” contains Hardware specific header files
  - Include peripheral specific definition
  - Required ‘Type’ definitions
  - Macros
- ‘.\StellarisWare\driverlib’ contains
  - ‘c’ source and header files peripheral specific functionality
  - Compiler specific project files for building the driver library ‘libraries’
  - ‘Compiler Specific’ output directories and files, i.e. the actual ‘library’ file used by each compiler.

• C:\StellarisWare\driverlib\ewarm	- IAR
• C:\StellarisWare\driverlib\gcc	- CodeRed
• C:\StellarisWare\driverlib\rvmdk	- Keil
• C:\StellarisWare\driverlib\sourcerygxx	- CodeSourcery

Peripheral Driver Lib (cont.)

- The peripheral driver library provides support for two Programming models
  - direct register access model
  - software driver model.
- Each model can be used independently, or combined,
  based on the needs of the application or the programming environment desired by the developer.
- Pros and Cons
  - Each programming model has advantages and disadvantages.
  - Use of the direct register access model will generally result in smaller and more efficient code than using the software driver model.
    Direct register access model does require detailed knowledge of the operation of each register, bit field, their interactions, and any sequencing required for proper operation of the peripheral; the developer is insulated from these details by the software driver model, generally requiring less time to develop applications.

Peripheral Driver Lib (cont.)

- Direct register access model
  - In the direct register access model, the peripherals are programmed by the application by writing values directly into the peripherals' registers.
  - A set of macros is provided that simplifies this process. These macros are stored in part-specific header files contained in the inc directory; the name of the header file matches the part number (for example, the header file for the LM3S1968 microcontroller is inc/lm3s1968.h).
  - Example: "Blinky" example for each board utilizes the direct register access model to blink the on-board LED.
- Software Driver Model
  - In the software driver model, the API provided by the peripheral driver library is used by applications to control the peripherals.
  - Since these drivers provide complete control of the peripherals in their normal mode of operation, it is possible to write an entire application without direct access to the hardware. This provides for rapid development of the application without requiring detailed knowledge of how to program the peripherals.

Peripheral Driver Lib (cont.)

- Source code overview
  - Driverlib/
    This directory contains the source code for the drivers.
  - Hw_*.h
    Header files, one per peripheral, that describe all the registers and the bitfields within those registers for each peripheral. These header files are used by the drivers to directly access a peripheral, and can be used by application code to bypass the peripheral driver library API.
  - Inc/
    This directory holds the part specific header files used for the direct register access programming model.

Peripheral Driver Lib (cont.)

- Example: UART

```
Int
main(void)
{
    // Set the clocking to run directly from the crystal.
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC |
                            SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);

    // Enable the peripherals used by this example.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Enable processor interrupts.
    IntMasterEnable();

    // Set GPIO A0 and A1 as UART pins.
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Configure the UART for 115,200, 8-N-1 operation.
    UARTConfigSet(UART0_BASE, 115200, (UART_CONFIG_WLEN_8 |
                            UART_CONFIG_STOP_ONE |
                            UART_CONFIG_PAR_NONE));

    // Enable the UART interrupt.
    IntEnable(INT_UART0);
    UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

    // Loop forever echoing data through the UART.
    while(1)
    {
    }
}
```

```
void
UARTIntHandler(void)
{
    unsigned long ulStatus;

    // Get the interrrupt status.
    ulStatus = ROM_UARTIntStatus(UART0_BASE, true);

    // Clear the asserted interrupts.
    ROM_UARTIntClear(UART0_BASE, ulStatus);

    // Loop while there are characters in the receive FIFO.
    while(ROM_UARTCharsAvail(UART0_BASE))
    {
        // Read the next character from the UART and write it back to the
UART.
        ROM_UARTCharNonBlockingPut(UART0_BASE,
                        ROM_UARTCharNonBlockingGet(UART0_BASE));
    }
}
```

Firmware Development Package

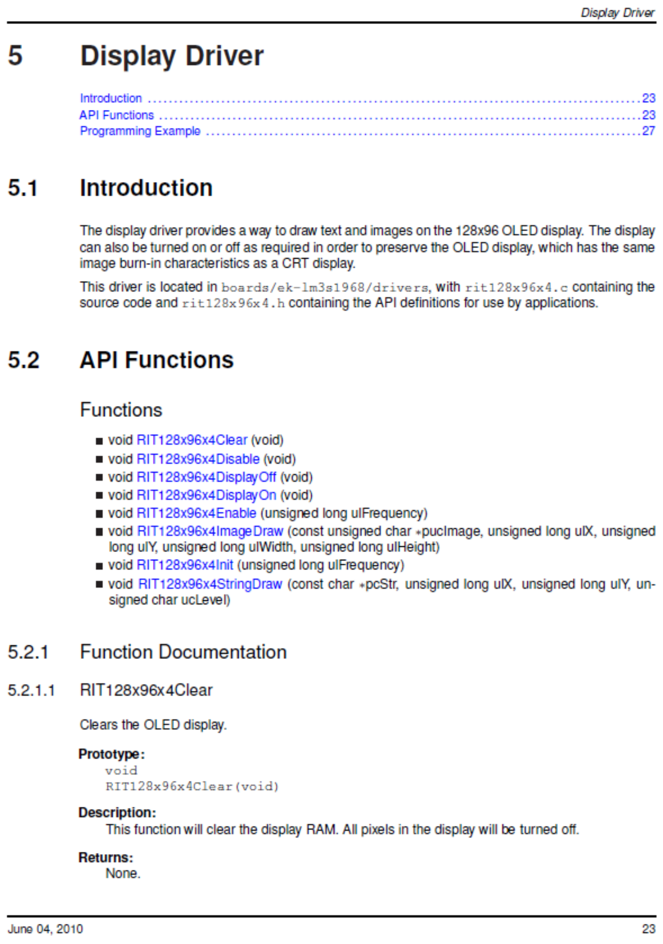
- The Stellaris® EK-LM3S1968 evaluation board is a platform that can be used for software development and to prototype a hardware design.
- This package delivers
  - The board-specific drivers and
  - Example applications

  that are provided for this development board.

Firmware Development Package

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING | BUSINESS | INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- Example: Display Driver
  - The display driver provides a way to draw text and images on the 128x96 OLED display.
  - The display can also be turned on or off as required in order to preserve the OLED display, which has the same image burn-in characteristics as a CRT display.
  - This driver is located in boards/ek-lm3s1968/drivers, with rit128x96x4.c containing the source code and rit128x96x4.h containing the API definitions for use by applications.
- Link: see Chapter 4.1 “General Purpose IO” application “Hello World”



5 Display Driver

5.1 Introduction

The display driver provides a way to draw text and images on the 128x96 OLED display. The display can also be turned on or off as required in order to preserve the OLED display, which has the same image burn-in characteristics as a CRT display.

This driver is located in `boards/ek-lm3s1968/drivers`, with `rit128x96x4.c` containing the source code and `rit128x96x4.h` containing the API definitions for use by applications.

5.2 API Functions

Functions

- void RIT128x96x4Clear (void)
- void RIT128x96x4Disable (void)
- void RIT128x96x4DisplayOff (void)
- void RIT128x96x4DisplayOn (void)
- void RIT128x96x4Enable (unsigned long ulFrequency)
- void RIT128x96x4ImageDraw (const unsigned char *pucImage, unsigned long ulX, unsigned long ulY, unsigned long ulWidth, unsigned long ulHeight)
- void RIT128x96x4Init (unsigned long ulFrequency)
- void RIT128x96x4StringDraw (const char *pcStr, unsigned long ulX, unsigned long ulY, unsigned char ucLevel)

5.2.1 Function Documentation

5.2.1.1 RIT128x96x4Clear

Clears the OLED display.

Prototype:
```
void
RIT128x96x4Clear(void)
```

Description:
This function will clear the display RAM. All pixels in the display will be turned off.

Returns:
None.

Example: Algorithm “Square Root”

- The following example shows how to compute the integer square root (“isqrt”) of a number.

```
// EK-LM3S1968 Firmware Development Package
// Integer Square Root
// utils/isqrt.c
//
#include "utils/isqrt.h“
…
//
// Integer Square Root
//
unsigned long ulValue;
…
//
// Get the square root of 81.
// The result returned will be 9, which is the square root of 81.
//
ulValue = isqrt(81);
```

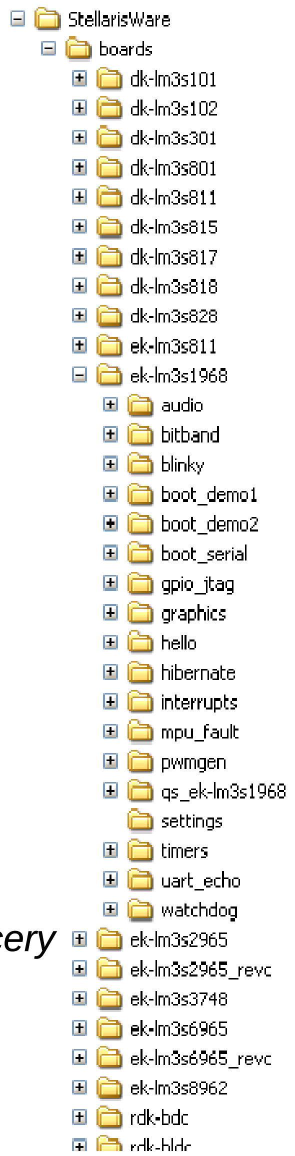
- Link: “lab34a.zip”

Code examples

- StellarisWare® software provides code examples in two different locations.
  - The first type of code example is specific to a particular board and is found in the boards directory.
    - The examples in this directory can be recompiled, downloaded and run on the specified board without modification. For more information on these examples, refer to the specific Board Firmware Development Package User's Guide.
  - The second type of example applies to all Stellaris® microcontrollers with a particular peripheral and can be found in the examples directory.
    - These examples are small, single-purpose code segments that are meant to clearly and simply demonstrate a specific feature and must be customized to run on a particular board.
    - Not every example can run on every Stellaris® device; consult the device data sheet to determine if a particular feature is present.
    - Furthermore please note: these examples are not ready to run projects.
- For ready-to-run projects please see the boards directory.

Code examples (cont.)

- '.\StellarisWare\boards' contains all kits projects
- Each kit has it's own directory
- There are multi-project workspace files
  for all available projects and supported compilers
- Each individual project directory contains
  - 'Readme' giving details of the project
  - Project files supported compilers
  - Project source code
  - Compiler specific 'library' output directories
    - *C:\StellarisWare\boards\kit\project\ccs* - *CCS*
    - *C:\StellarisWare\boards\kit\project\ewarm* - *IAR*
    - *C:\StellarisWare\kit\project\gcc* - *CodeRed*
    - *C:\StellarisWare\kit\project\\rvmdk* - *Keil*
    - *C:\StellarisWare\kit\project\\sourcerygxx* - *CodeSourcery*

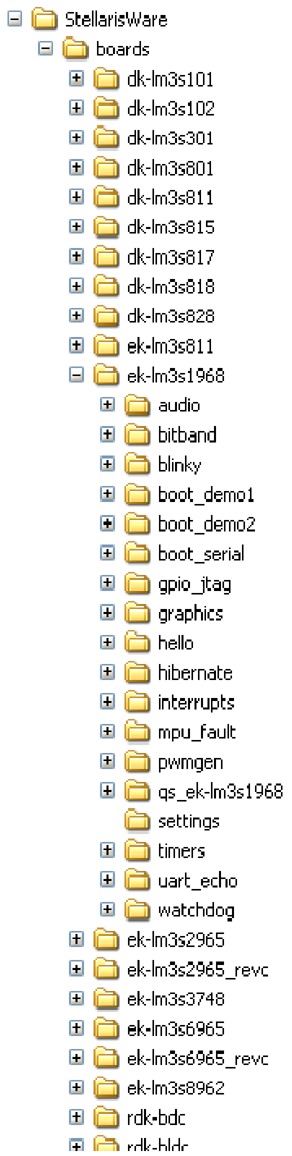


```
StellarisWare
  boards
    dk-lm3s101
    dk-lm3s102
    dk-lm3s301
    dk-lm3s801
    dk-lm3s811
    dk-lm3s815
    dk-lm3s817
    dk-lm3s818
    dk-lm3s828
    ek-lm3s811
    ek-lm3s1968
      audio
      bitband
      blinky
      boot_demo1
      boot_demo2
      boot_serial
      gpio_jtag
      graphics
      hello
      hibernate
      interrupts
      mpu_fault
      pwmgen
      qs_ek-lm3s1968
      settings
      timers
      uart_echo
      watchdog
    ek-lm3s2965
    ek-lm3s2965_revc
    ek-lm3s3748
    ek-lm3s6965
    ek-lm3s6965_revc
    ek-lm3s8962
    rdk-bdc
    rdk-bldc
```

HOCHSCHULE HEILBRONN
HEILBRONN UNIVERSITY
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

Code examples (cont.)

- All project that use the driver library reference
  - Driver library ‘library’ file from ‘C:\StellarisWare\driverlib\complier’
  - Driver Library header ‘*.h’ file from ‘C:\StellarisWare\driverlib’
- Other related files
  - ‘c:\StellarisWare\inc’ - Device specific header ‘lm3sxxxx.h’ files
  - ‘c:\StellarisWare’ - Peripheral hardware header ‘hw_xxxx.h’ files

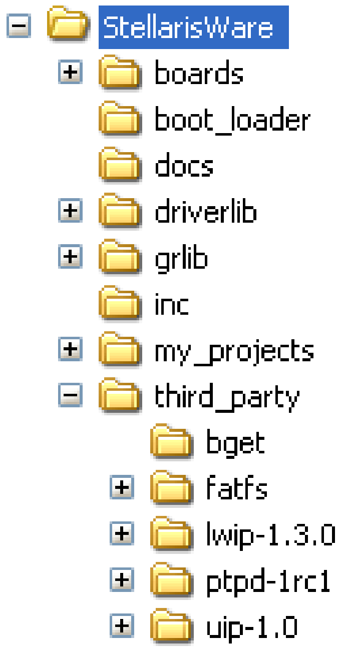


```
StellarisWare
  boards
    dk-lm3s101
    dk-lm3s102
    dk-lm3s301
    dk-lm3s801
    dk-lm3s811
    dk-lm3s815
    dk-lm3s817
    dk-lm3s818
    dk-lm3s828
    ek-lm3s811
    ek-lm3s1968
      audio
      bitband
      blinky
      boot_demo1
      boot_demo2
      boot_serial
      gpio_jtag
      graphics
      hello
      hibernate
      interrupts
      mpu_fault
      pwmgen
      qs_ek-lm3s1968
      settings
      timers
      uart_echo
      watchdog
    ek-lm3s2965
    ek-lm3s2965_revc
    ek-lm3s3748
    ek-lm3s6965
    ek-lm3s6965_revc
    ek-lm3s8962
    rdk-bdc
    rdk-bldc
```

TEXAS INSTRUMENTS

Stellaris® 3rd Party Organization

- ‘.\StellarisWare\third_party ’contains any required third party source code files and information
- Each ‘third_party’ directory contains all required information
  - Documentation, source code files, etc
- Example projects using any ‘third_party’ directly links the required source code ‘*.c’ and header ‘*.h’ from the relevant directory
- Available in the Stellaris® Peripheral Driver Library (DriverLib) Download

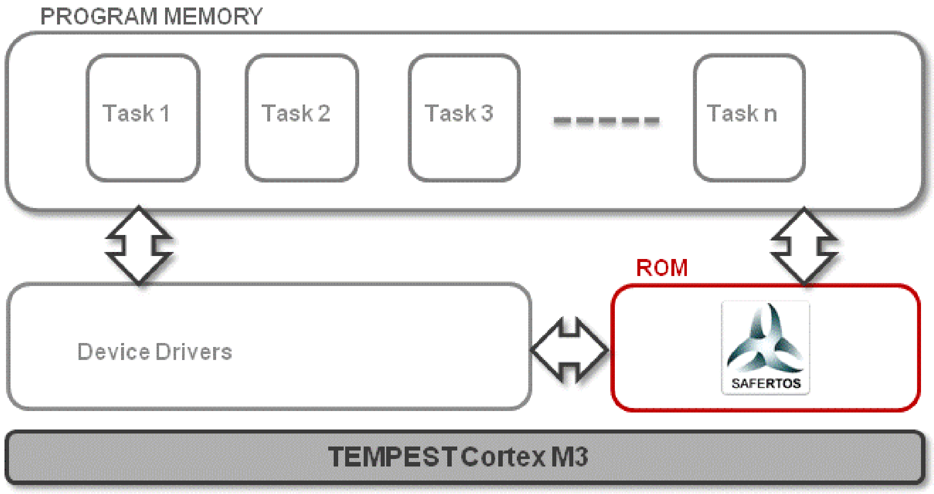


On-chip SW Enhancements (ROM)

- StellarisWare® DriverLib
  - High-level API interface to complete peripheral set.
  - Simplifies and speeds development of applications.
  - Saves user flash by storing peripheral setup and configuration code
  - Allows programmer focus to be on the application - not setup
- StellarisWare® Bootloader
  - Download code to flash memory for firmware updates
  - Interface options include UART (default), I2C, SSI, Ethernet
- Other flash memory-saving options
  - Advanced Encryption Standard (AES) tables – for cryptography
    - Supported by the current AES example application
    - Covers all three sizes: 128, 192, 256
  - Cyclic Redundancy Check (CRC) functionality – for error detection

On-chip SW Enhancements (ROM)

- SAFERTOS for Tempest
  - High-integrity RTOS in ROM
  - RTOS value $65k free with Tempest LM3S9B96
  - Can be used as a standard operating system OR as part of a high integrity application which requires certification to IEC61508 or FDA510(k)
  - Integrated hardware/software solution shortens the time to market and significantly reduces cost for Industrial and Medical Applications
  - Innovative Design Assurance Pack available separately from WITTENSTEIN provides complete turnkey evidence and process documentation



Questions and Exercises

1. Implement “Blinky” with the GPIO API
2. Use the Display Driver API
3. Explain advantages and disadvantages of the two programming models Direct Register Access and Software Driver?
4. What are the advantages of using StellarisWare®.

Summary and Outlook

- Summary
  - From Main Features to Peripheral Driver Library, EK-LM3S1968 Firmware Development Package and Code Samples
- Outlook
  - Chapter 4: “Peripheral programming in C” shows the practical use of StellarisWare® in different C-Code examples

References

- [1] Henri, G.; Texas Instruments: *MCU Training Module Collaterals and Graphics*. EMEA, Oct. 7, 2010; page 14 ff..
- [2] Texas Instruments Homepage: *StellarisWare®*.
  Stellaris® ARM® Cortex™ -M3 MCU - StellarisWare® - TI.com.
- [3] Texas Instruments: *User's Guide - EK-LM3S1968 Firmware Development Package*. SW-EK-LM3S1968-UG-6075.pdf, 2010.
- [4] Texas Instruments: *User's Guide - StellarisWare Examples*. SW-EXAMPLES-UG-6075.pdf, 2010.
- [5] Texas Instruments: *User's Guide - Stellaris Peripheral Driver Library*. SW-DRL-UG-6075.pdf, 2010.