Азбука халтурщика-ARMатурщика
основы CMSIS

учебный курс по микроконтроллерам Cortex-Mx:
Миландр 1986BE, STM32F, LPC21xx

(copypasta) Понятов Д.А. <**dponyatov@gmail.com**>, ИКП СГАУ

20 июля 2014 г.

Оглавление

0.1 © . . . 2
0.2 Introduction . . . 3
0.3 Структура CMSIS . . . 5
0.3.1 File Structure . . . 8
0.3.2 Tool Independence . . . 9
0.3.3 MISRA-C . . . 11
0.3.4 CPAL Functions . . . 11
0.3.5 Interrupt Service Routines . . . 11
0.3.6 Other Coding Conventions . . . 11
Identifiers . . . 11
Comments . . . 11
Data Types . . . 11
0.3.7 Debugging . . . 11
0.3.8 Future Updates . . . 11
0.4 Tutorial 1 – A First Example . . . 11
0.4.1 Example 1 Starting point . . . 11
0.4.2 Example 2 Converting to CMSIS . . . 11
0.5 Tutorial 2 – ITM Debug . . . 11

0.6 Summary 11

Литература **12**

0.1 ©

[1]

This tutorial material is part of a series to be published progressively by Doulos.

You can find the full set of currently published Tutorials and register for notification of future additional at www.doulos.com/knowhow

You can also download the full source code of the examples used within the Tutorial at the same URL.

Also check out the Doulos ARM Training and service options at www.doulos.com/arm

Or email <**info@doulos.com**> for further information

First published by Doulos March 2009

[1] копипаста: двойной лось http://www.doulos.com/knowhow/arm/CMSIS/CMSIS_Doulos_Tutorial.pdf

0.2 Introduction

The Cortex Microcontroller Software Interface Standard (CMSIS) supports developers and vendors in creating reusable software components for ARM Cortex-M based systems.

Cortex Microcontroller Software Interface Standard (CMSIS)[2] обеспечивает разработчикам и производителям МК создание повторно используемых программных компонентов для систем на основе микроконтроллеров Cortex-M.

The ARM Cortex-M3 processor is the first core from ARM specifically designed for the Microcontroller market. This core includes many common features (NVIC, Timer, Debug-hardware) needed for this market. This will enable developers to port and reuse software (e.g. a real time kernel) with much less effort to Cortex-M3 based MCUs.

Процессор ARM Cortex-M3 первое ядро от компании ARM специально разработанное для рынка микроконтроллеров. Это ядро включает множество типовых блоков (NVIC, таймеры, отладочный интрефейс) необходимых на этом рынке. Это позволяет разработчикам с минимальными усилиями портировать и повторно использовать уже написанное ПО[3] для МК семейства Cortex-M3 любых производителей.

With a significant amount of hardware components being identical, a large portion of the Hardware Abstraction Layer (HAL) can be identical. However, reality has shown that lacking a common standard we find a variety of HAL/driver libraries for different devices, which, as far as the Cortex-M3 part is concerned essentially do the same thing — just differently.

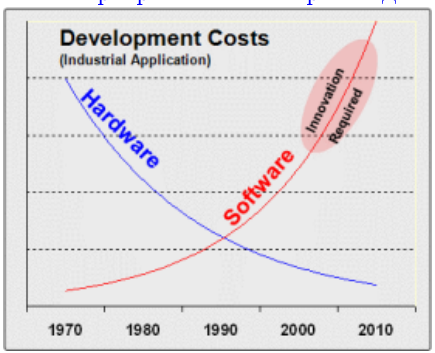
Благодаря идентичности большого колиства аппаратных компонентов, также идентичным оказывается и Hardware Abstraction Layer (HAL)[4]. Тем не менее, реальность показывает что отсутствие общего стандарта приводит к множеству несовместимых версий библотек HAL и драйверов для различных МК, что не соответствует идее полной переносимости ПО в серии Cortex-M3.

The latest study of the development for the embedded market shows that software complexity and cost will increase over time, see figure left. Reusing Software and having a common standard to govern how to write and

[2] стандарт программного интерфейса микроконтроллеров Cortex
[3] например ядро ОС реального времени
[4] программный слой аппаратной абстракции

debug the software will be essential to minimising costs for future developments.

Последние исследования разработок для рынка встраиваемого ПО показывают, что сложность программного обеспечения и его стоимость постоянно увеличиваются. Повторное использование кода и наличие общего стандарта управляющего способами написания и отладки ПО необходимы для минимизации стоимости разработки и сопровождения.



стоимости разработки

With more Cortex-M3 based MCUs about to come onto the market, ARM has recognized that after solving the diversity issue on the hardware side, there is still a need to create a standard to access these hardware components.

Анализируя ситуацию с взрывным ростом количества моделей МК Cortex-M3 на рынке, компания ARM обнаружила что полная идентичность аппаратной части недостаточна для обеспечения совместимости, и необходимо создание стандарта доступа к аппаратным компонентам.

The result of that effort is CMSIS; a framework to be extended by vendors, while taking advantage of a common API (Application Programming Interface) for core specific components and conventions that define how the device specific portions should be implemented to make developers feel right at home when they reuse code or develop new code for ARM Cortex-M based devices.

Результатом этих исследований является CMSIS: фреймворк, расширямый поставщиками МК, с со-

хранением полезных свойств общего API (Application Programming Interface)[5] для ядреных компонентов и соглашениями о том, как должны быть реализованы части зависимые от железа, чтобы разработчики чувствовали себя как дома при повторном использовании ил разработки нового кода для семейства Cortex-M.

0.3 Структура CMSIS

CMSIS can be divided into three basic function layers:
CMSIS может быть поделен на три основных слоя:

- Core Peripheral Access Layer (CPAL)

  The lowest level defines addresses, and access methods for common components and functionality that exists in every Cortex-M system. Access to core registers, NVIC, debug subsystem is provided by this layer. Tool specific access to special purpose registers (e.g. CONTROL, xPSR), will be provided in the form of inline functions or compiler intrinsics. This layer will be provided by ARM.

  Самый нижний уровень определяет адреса, и методы доступа к общим компонентам и функциям, существующим в каждой Cortex-M-системе. Этим уровнем описывается доступ к регистрам ядра, NVIC[6], подсистеме отладки. Инструментальный доступ к спецрегистрам (**CONTROL**,**xPSR**) предоставляется в форме inline-функций или интринсик компилятора. Этот уровень обеспечивается лицензиатом архитектуры — компанией ARM.

- Middleware Access Layer (MWAL)

  This layer is also defined by ARM, but will be adapted by silicon vendors for their respective devices. The Middleware Access Layer defines a common API for accessing peripherals. The Middleware Access Layer is still under development and no further information is available at this point.

[5] прикладной программный интерфейс
[6] Nested Vector Interrupt Controller, контроллер вложенных прерываний

Этот слой также специфицируется ARM, но адаптируется производителем кристаллов для их конкретных изделий. Слой MWAL определяет общий API для доступа к периферии. Этот слой все еще находится на стадии доработки, и на текущий момент более подробная информация неступна.

- Device Peripheral Access Layer (DPAL)

  Hardware register addresses and other definitions, as well as device specific access functions will be defined in this layer. The Device Peripheral Access Layer is very similar to the Core Peripheral Access Layer and will be provided by the silicon vendor. Access methods provided by CPAL may be referenced and the vector table will be adapted to include device specific exception handler address.

  Слой содержит адреса аппаратных регистров и другие определения, в том числе функции доступа к специфичным особенностям чипов. DPAL сильно похож на CPAL, но предоставляется поставщиком кристаллов. В CPAL могут быть описаны методы доступа и адаптированная таблица векторов, содержащая обработчики исключений, специфичные для конкретного МК.

  While DPAL is intended to be extended by the silicon vendor, let's not forget about Cortex-M based FPGA products, which effectively put developers into the position of a silicon vendor.

  DPAL предназначен для расширения вендором, но не стоит забывать о FPGA-продуктах с применением Cortex-M-ядер, которые ставят разработчиков в положение вендора.

The basic structure and the functional flow is illustrated in the Figure 2. below.

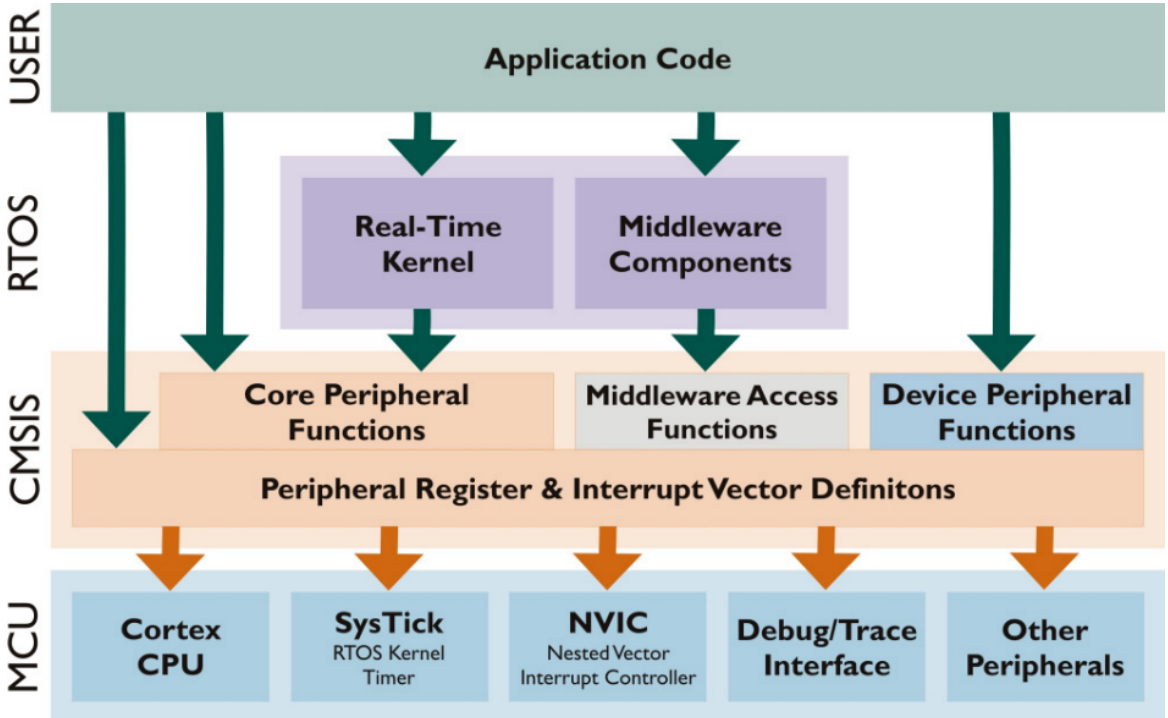


Figure 2 CMSIS Structure functional flow
Рис.2 Функциональная структура CMSIS

As far as MCU based systems are concerned it might make sense for developers to treat the entire PCB system as monolithic block. There is no reason to differentiate between a memory mapped register inside the MCU and a memory mapped register external to the MCU, connected via external memory interface. The

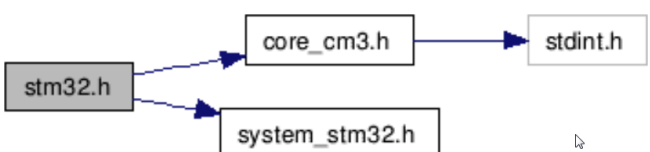
benefit of applying a standard like CMSIS is that existing guidelines on how to access these devices set a clear goal on how to implement and integrate critical parts of the software. Other team members will find a familiar environment.

0.3.1 File Structure

File names in CMSIS are standardized as follows:

core_cm3.h	Cortex-M3 global declarations and definitions, static function definitions
core_cm3.c	Cortex-M3 global definitions
<device>.h	Top-level header file (device specific). To be included by application code. Includes **core_cm3.h** and **system_<device>.h**
system_<device>.h	Device specific declarations
system_<device>.c	Device specific definitions, e.g. `SystemInit()`

Application code will only include the top-level header file which implicitly pulls in all other essential



header files. The illustration below shows the flow and dependencies of the header files **stm32.h**, **core_cm3.h** and **system_stm32.h**, which are part of CMSIS release V1P0.

```
1 /* Configuration of the Cortex-M3 Processor and Core Peripherals */
2 #define __MPU_PRESENT 0          /*!< STM32 does not provide a MPU present or not*/
3 #define __NVIC_PRIO_BITS 4       /*!< STM32 uses 4 Bits for the Priority Levels */
```

```
#define __Vendor_SysTickConfig 0 /*!< Set to 1 if different SysTick Config isused *
#include "core_cm3.h"            /* Cortex-M3 processor and core peripherals */
#include "system_stm32.h"        /* STM32 System */
```

The **<device>.h** file is the central include file and provided by the silicon vendor. The application programmer is using that as the main include file in his C source code. Note that the ARM Cortex-M3 has some optional hardware features (e.g. the MPU, number of Interrupts and the number of the NVIC priority bits) the silicon vendors may have implemented differently. The listing above shows that STM32 implements four out of eight possible priority bits. The macro `__NVIC_PRIO_BITS` is set here to 4. STM32 does not offer a Memory Protection Unit (MPU). Accordingly, the macro `__MPU_PRESENT` has the value 0.

The next example shows the corresponding definitions for a NXP LPC17xx device. In this Cortex-M3 implementation five priority bits have been implemented and an MPU is available.

```
/* Configuration of the Cortex-M3 Processor and Core Peripherals */
#define __MPU_PRESENT 1           /*!< MPU present or not */
#define __NVIC_PRIO_BITS 5        /*!< Number of Bits used for Priority Levels */
#define __Vendor_SysTickConfig 0 /*!< Set to 1 if different SysTick Config is used
#include "..\core_cm3.h"     /* Cortex-M3 processor and core peripherals */
#include "system_LPC17xx.h" /* System Header */
```

The `__Vendor_SysTickConfig` defined is showing in both cases the default setting. When this macro is set to 1, the `SysTickConfig()` function in the **cm3_core.h** is excluded. In this case the file **<device>.h** must contain a vendor specific implementation of this function.

0.3.2 Tool Independence

CMSIS exists in a three-dimensional space of the form vendor÷device÷tool chain. In order to remove one dimension (tool chain), the common files **core_cm3.c** and **core_cm3.h** contain all essential tool specific

declarations and definitions.

Example:

```
/* define compiler specific symbols */
#if defined ( __CC_ARM )
    #define __ASM __asm                 /*!< asm keyword for armcc */
    #define __INLINE __inline           /*!< inline keyword for armcc */
#elif defined ( __ICCARM__ )
    #define __ASM __asm                 /*!< asm keyword for iarcc */
    #define __INLINE inline             /*!< inline keyword for iarcc.
                                         Only avaiable in High optimization mode! */
    #define __nop __no_operation        /*!< no operation intrinsic in iarcc */
#elif defined ( __GNUC__ )
    #define __ASM asm                   /*!< asm keyword for gcc */
    #define __INLINE inline             /*!< inline keyword for gcc */
#endif
```

The remaining parts of CMSIS can now simply use the macro `__INLINE` to define an inline function.

Остальная часть CMSIS теперь может просто использовать макрос `__INLINE` для определения инлайн-функций.

Currently three of the most important C-compilers are supported: ARM RealView (armcc), IAR EWARM (iccarm), and GNU Compiler Collection (gcc). This is expected to cover the majority of tool chains.

На настоящий момент поддерживаются три наиболее применяемых Си-компилятора: ARM RealView (armcc), IAR EWARM (iccarm), и GNU Compiler Collection (gcc).

0.3.3 MISRA-C

0.3.4 CPAL Functions

0.3.5 Interrupt Service Routines

0.3.6 Other Coding Conventions

Identifiers

Comments

Data Types

0.3.7 Debugging

0.3.8 Future Updates

0.4 Tutorial 1 – A First Example

0.4.1 Example 1 Starting point

0.4.2 Example 2 Converting to CMSIS

0.5 Tutorial 2 – ITM Debug

0.6 Summary

Литература

[1] https://github.com/ponyatov/CortexMx Азбука халтурщика-ARMатурщика

[2] Getting started with CMSIS http://www.doulos.com/knowhow/arm/CMSIS/CMSIS_Doulos_Tutorial.pdf

[3] Ю.С. Магда Программирование и отладка C/C++ приложений для микроконтроллеров ARM. — М.: ДМК Пресс, 2012. — 168 с.: ил.

[4] © Quantum®L^eaPs

[5] http://www.state-machine.com/arm/Building_bare-metal_ARM_with_GNU.pdf Quantum®L^eaPs Building Bare-Metal ARM Systems with GNU

[6] http://milandr.ru/ ЗАО «ПКК Миландр»

[7] http://git-scm.com/book/ru перевод: Scott Chacon **Pro Git**

[8] http://habrahabr.ru/post/114239/ хабра: Quantum®L^eaPs QP и диаграммы состояний в UML

[9] http://www.state-machine.com/ Quantum®L^eaPs State Machines & Tools

[10] http://makesystem.net/?p=988 Изучаем ARM. Собираем свою IDE для ARM

[11] http://makesystem.net/?p=2146 Изучаем ARM. Отладка ARM приложений в Eclipse IDE

[12] Львовский С.М. Набор и вёрстка в пакете LaTeX