CMSIS Standard and TivaWare Library

Prof. Hugo Vieira Neto

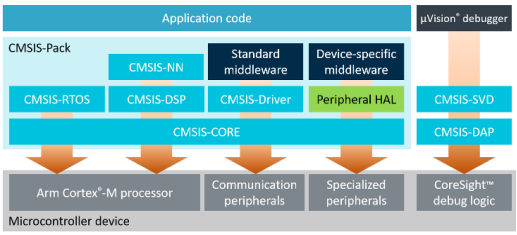
2020/1

Objective

- To study the main features of the CMSIS standard and TivaWare library:
  - CMSIS-Core
  - Project file structure
  - Standard system exception names
  - Hardware abstraction layers

CMSIS

Cortex Microcontroller Software Interface Standard



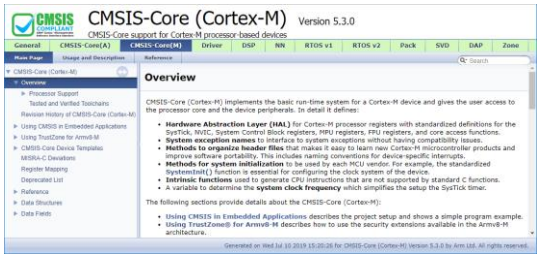
Main Components

- CMSIS-Core
  - API for Cortex-M or Cortex-A cores
- CMSIS-Driver
  - Generic drivers for communication systems, file systems, graphic user interfaces, etc.
- CMSIS-DSP
  - Digital signal processing library
- CMSIS-NN
  - Neural network implementation library
- CMSIS-RTOS
  - API for real-time operating systems

CMSIS-Core Component

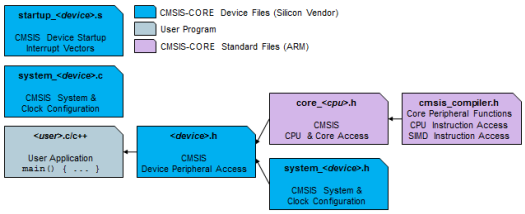
- Basic system that provides access to the Cortex-M processor core and device peripherals, defining:
  - Hardware abstraction layer
  - Names for system exceptions
  - Intrinsic functions [core instructions]
  - **`SystemInit`** function [system initialization]
  - **`SystemCoreClock`** variable [clock frequency]

Documentation



http://www.keil.com/pack/doc/cmsis/Core/html/index.html

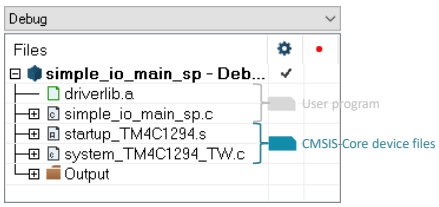
File Structure



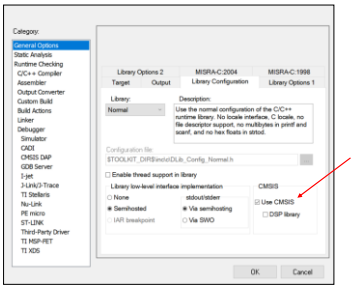
CMSIS-Core Component

- Files (Cortex-M):
  - **`startup_<device>.c`** or **`startup_<device>.s`**
    - Exception vector table
  - **`system_<device>.c`** and **`system_<device>.h`**
    - Initialization (clock, FPU, WDT, bus wait states, etc.)
  - **`<device>.h`**
    - Hardware abstraction layer (HAL) for core registers and device peripheral registers

File Structure in IAR



Project Options in IAR



System Exception Names

- **`Reset_Handler`**
- `NMI_Handler`
- `HardFault_Handler`
- `MemManage_Handler`
- `BusFault_Handler`
- `UsageFault_Handler`
- **`SVC_Handler`**
- `DebugMon_Handler`
- **`PendSV_Handler`**
- **`SysTick_Handler`**

Hardware Abstraction Layer

- Data structures with pointers to registers in each device block
- Single header file (`TM4C1294NCPDT.h`)
- Example of use:
  - `GPIOK->DIR` accesses the `DIR` register in block `GPIOK` (address `0x40061400`)
  - See Section 10.5 of the TM4C1294NCPDT device datasheet

Datasheet Information

GPIO Base Adresses

- GPIO Port A (AHB): 0x4005.8000
- GPIO Port B (AHB): 0x4005.9000
- GPIO Port C (AHB): 0x4005.A000
- GPIO Port D (AHB): 0x4005.B000
- GPIO Port E (AHB): 0x4005.C000
- GPIO Port F (AHB): 0x4005.D000
- GPIO Port G (AHB): 0x4005.E000
- GPIO Port H (AHB): 0x4005.F000
- GPIO Port J (AHB): 0x4006.0000
- GPIO Port K (AHB): 0x4006.1000
- GPIO Port L (AHB): 0x4006.2000
- GPIO Port M (AHB): 0x4006.3000
- GPIO Port N (AHB): 0x4006.4000
- GPIO Port P (AHB): 0x4006.5000
- GPIO Port Q (AHB): 0x4006.6000

GPIO Register Map

Offset	Name	Type
~~0x000~~	GPIODATA	RW
0x400	GPIODIR	RW
0x404	GPIOIS	RW
0x408	GPIOIBE	RW
0x40C	GPIOIEV	RW
0x410	GPIOIM	RW
0x414	GPIORIS	RO
0x418	GPIOMIS	RO
0x41C	GPIOICR	W1C

TivaWare Library

- Also provides an abstraction layer
- Macros for register access
- Multiple header files with the base addresses for each block and the offsets for each register (`inc/hw_*.h`)
- Important: the target device for compilation must be defined (e.g. `PART_TM4C1294NCPDT`)
  - Options → C/C++ Compiler → Preprocessor

TivaWare Library

- Example of access to a register:
  - `HWREG(GPIO_PORTK_BASE + GPIO_O_DIR)`
- The `HWREG` macro is defined in `hw_types.h`
- `GPIO_PORTK_BASE = 0x40061000` defined in `hw_memmap.h`
- `GPIO_O_DIR = 0x00000400` defined in `hw_gpio.h`

TivaWare Peripheral Driver Library

- Device driver API for device peripherals (`driverlib/*.h`)
- Definitions of constants for configuration of peripherals
- Examples (`driverlib/gpio.h`):
  - `GPIO_DIR_MODE_IN = 0x00000000`
  - `GPIO_DIR_MODE_OUT = 0x00000001`
  - `GPIO_DIR_MODE_HW = 0x00000002`

Code Legibility

- Which of the equivalent code versions presented below is more legible?
  - `GPIOK->DIR |= GPIO_PIN_1`
  - `GPIOK->DIR |= 0x00000002`
- See “TivaWare Peripheral Driver Library User’s Guide”
  - `GPIOPinTypeGPIOInput(GPIO_PORTK_BASE, GPIO_PIN_1)`
- See `inc/hw_memmap.h` e `driverlib/gpio.h`

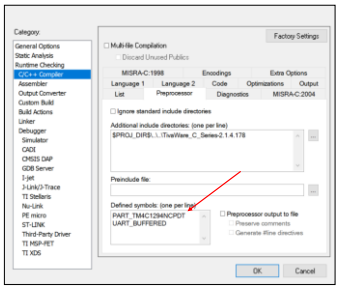
Exercise

- Select the “simple_uart” project from the “EK-TM4C1294XL_IAR” workspace.
- Use a terminal emulator application such as Tera Term with a serial connection to “COMn: Stellaris Virtual Serial Port” to communicate with the EK-TM4C1294XL kit.

Exercise

- Notice that the project makes use of the "uartstdio.c" file in "TivaWare_C_Series-2.1.4.178\utils"
  - Macro `PART_TM4C1294NCPDT` must be defined for proper configuration of RX and TX pins
  - Macro `UART_BUFFERED` must be defined if interrupt-controlled buffering is to be used

Exercise



Exercise

- A quick look at the files "uartstdio.h" and "uartstdio.c" provides important information:
  - `UART_RX_BUFFER_SIZE` defines the receive buffer size, which can be overriden
  - `UART_TX_BUFFER_SIZE` defines the transmit buffer size , which can be overriden
  - `UARTStdioIntHandler` is the UART interrupt handler name, which is **different** from the default in "startup_TM4C1294.s"

Exercise

- Inspect the "simple_uart.c" file:
  - What does the `UARTInit` function do?
  - What happens if one removes the definition of `PART_TM4C1294NCPDT` from the C/C++ Compiler Preprocessor options in IAR?
  - How was the problem with the name of the UART interrupt handler having a name other than the default in "startup_TM4C1294.s" handled? Are there alternative solutions? What are the potential problems with them?

Exercise

- Make changes to the code so that the SysTick handler besides alternating the state of LED D1 also sends "SysTick_Handler\n" via UART
- Observe what happens in Tera Term – are the messages received synchronized with changes in the state of LED D1?
- Note: baud rate should be set to 9600 bps

Exercise

- Pause the program in IAR and inspect these variables:
  - `g_ui32UARTTxWriteIndex`
  - `g_ui32UARTTxReadIndex`
  - `g_pcUARTTxBuffer`
- What do these variables tell you? See comments for "output ring buffer" in "uartstdio.c"
- What happens if the baud rate is lowered to 600 bps? And to 300 bps? Do not forget to reset both sides of the connection (kit **and** Tera Term)