Component-based Software Development for Cortex-M Microcontrollers

Reinhard Keil
Director of MCU Tools
Software Complexity – The Challenge

- Well-known issues that drive software development costs
  - Increasing product requirements that are implemented by software
  - Hardware problems tend to become compensated by software

- Hardware uses defined interfaces that simplify re-use
  - Software components are hard to integrate

Software Standards and Software Components are key for productivity!
An Ecosystem is considered helpful...

What’s most important when choosing a microprocessor?

<table>
<thead>
<tr>
<th></th>
<th>2012 (N = 1,662)</th>
<th>2011 (N = 1,859)</th>
<th>2010 (N = 1,501)</th>
<th>2009 (N = 1,530)</th>
<th>2008 (N = 1,056)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The chip itself</td>
<td>30%</td>
<td>43%</td>
<td>45%</td>
<td>48%</td>
<td>36%</td>
</tr>
<tr>
<td>The ecosystem surrounding the chip (software, tools, support, etc.)</td>
<td>61%</td>
<td>46%</td>
<td>43%</td>
<td>41%</td>
<td>57%</td>
</tr>
<tr>
<td>The chip's supplier/vendor</td>
<td>9%</td>
<td>11%</td>
<td>13%</td>
<td>10%</td>
<td>7%</td>
</tr>
</tbody>
</table>

Source: UBM Electronics – 2012 Embedded Market Study
...but Software Reuse is Limited

Does your current project reuse code from a previous embedded project?

A very slight change in usage of RTOS, kernels, execs, schedulers over past 5 years

In 2012, 85% reused code.
In 2011, 87% reused code.
In 2010, 86% reused code.
In 2009, 89% reused code.
In 2008, 89% reused code.

Source: UBM Electronics – 2012 Embedded Market Study
Questions when using external source code

- Usage of files (source code, header, library) is unclear
  - Will my Compiler and Tool Environment work?
  - Where is the documentation to the software component?
  - What other requirements (i.e. libraries or RTOS) does this software have?
  - Will the source code run on my target hardware?
  - Does the source code need adaptations or configuration?

- Project Maintenance after files are integrated
  - Where did I get the source code from; who to contact for support?
  - What is the version of that source code?
  - What do I need to change when I update the files?

- What are the License conditions of the code
  - Can I use this files in my project?
Packing Software is a Solution

- All software components are delivered in one Software Pack that is easy to install (like an IC)

- A package description file (PDSC) contains:
  - Supplier information
  - Download URL
  - License
  - Release version

- Usage of source code and libraries files for:
  - Specific processors
  - Specific microcontroller families and devices
  - Tool Chains

- Other components that are required or related
Pack Description File Example

```xml
<package schemaVersion="1.0" xmlns:xs="..." xs:noNamespaceSchemaLocation="PACK.xsd">
    <vendor>Keil</vendor>
    <name>CMSIS_RTX</name>
    <description>RTX is a CMSIS-RTOS compliant RTOS for Cortex-M based devices</description>
    <license>License.txt</license>
    <url>http://www.keil.com/demo/eval/rtx.htm</url>

    <releases>
        <release version="4.70.0">
            Updates:
            - `osTimerCreate` can be called prior to `osKernelStart` (but after `osKernelInitialize`)
            - Initialization of external timer corrected for Cortex-M0/M0+/M1
            - Message/Mail Queue behaviour corrected when timeout expires
        </release>
    </releases>

    <conditions>
        <condition id="CMSIS_Core">
            <description>This component requires the CMSIS CORE component</description>
            <require Cclass="CMSIS" Cgroup="CORE"/>
        </condition>
    </conditions>

    <!-- ARMCC -->
    <condition id="CM0_LE_ARMCC">
        <description>Cortex-M0 or Cortex-M0+ or SC000 processor based device in little endian mode for the ARM Compiler</description>
        <accept Dcore="Cortex-M0"/>
        <accept Dcore="Cortex-M0+"/>
        <accept Dcore="SC000"/>
        <require Dendian="Little-endian"/>
        <require Tcompiler="ARMCC"/>
    </condition>
</package>
```
Pack Description File Example

```xml
<components>
  <component Cclass="CMSIS" Cgroup="RTOS" Csub="Keil RTX" condition="CMSIS_Core">
    <description>RTX is a CMSIS RTOS implementation for Cortex-M, processor based devices.</description>
    <files>
      <!-- CPU and Compiler independent -->
      <file category="doc" name="Doc\index.html"/>
      <file category="header" name="INC\cmsis_os.h"/>
      <file category="source" name="Templates\RTX_Conf_CM.c" copy="true"/>
      <!-- CPU and Compiler dependent -->
      <!-- ARMCC -->
      <file category="library" condition="CM0_LE_ARMCC" name="Lib\ARM\RTX_CM0.lib"/>
      <file category="library" condition="CM0_BE_ARMCC" name="Lib\ARM\RTX_CM0_B.lib"/>
      <file category="library" condition="CM3_LE_ARMCC" name="Lib\ARM\RTX_CM3.lib"/>
      <file category="library" condition="CM3_BE_ARMCC" name="Lib\ARM\RTX_CM3_B.lib"/>
      <file category="library" condition="CM4F_LE_ARMCC" name="Lib\ARM\RTX_CM4.lib"/>
      <file category="library" condition="CM4F_BE_ARMCC" name="Lib\ARM\RTX_CM4_B.lib"/>
      <!-- GCC -->
      <file category="library" condition="CM0_LE_GCC" name="Lib\GCC\libRTX_CM0.a"/>
      <file category="library" condition="CM0_BE_GCC" name="Lib\GCC\libRTX_CM0_B.a"/>
      <!-- IAR -->
      <file category="library" condition="CM4F_LE_IAR" name="Lib\IAR\RTX_CM4.a"/>
      <file category="library" condition="CM4F_BE_IAR" name="Lib\IAR\RTX_CM4_B.a"/>
    </files>
  </component>
</components>
</package>
```
Pack Installation and Updates

- Pack Installer copies relevant files of a PACK to tool/project environment
- *.PDSC file contains web URL for initial download & update
**MDK-ARM Version 5 – Overview**

- MDK-ARM Core: contains development tools

<table>
<thead>
<tr>
<th>MDK Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>μVision IDE with Editor</td>
</tr>
<tr>
<td>Pack Installer</td>
</tr>
<tr>
<td>ARM C/C++ Compiler</td>
</tr>
<tr>
<td>μVision Debugger with Trace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Packs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
</tr>
<tr>
<td>Startup / System</td>
</tr>
<tr>
<td>Driver 1: SPI</td>
</tr>
<tr>
<td>Driver 2: Ethernet</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td>Driver n: USB</td>
</tr>
<tr>
<td>CMSIS</td>
</tr>
<tr>
<td>CMSIS-CORE</td>
</tr>
<tr>
<td>CMSIS-DSP</td>
</tr>
<tr>
<td>CMSIS-RTOS</td>
</tr>
<tr>
<td>MDK Professional Middleware</td>
</tr>
<tr>
<td>TCP/IP Networking</td>
</tr>
<tr>
<td>File System</td>
</tr>
<tr>
<td>USB Host Stack</td>
</tr>
<tr>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>USB Device Stack</td>
</tr>
<tr>
<td>CAN Interface</td>
</tr>
</tbody>
</table>

- Software Packs: contain device support & middleware
  - Installed & updated on demand using the Pack Installer
**Component Example: TCP/IP Networking**

- **Network Component**
  - Socket: TCP, UDP, BSD
  - Interface: Ethernet, PPP (Serial), SLIP (Serial)
  - Driver: Ethernet MAC, Ethernet PHY, UART

- **Variants:**
  - Compact Web Server
  - Full Web Server using File System
  - FTP Server
  - TFTP Server
  - Telnet Server
  - SNMP Agent
  - DNS Client
  - SNTP Client
  - FTP Client
  - TFTP Client
  - SMTP Client

- **TCP/IP Networking Components**
  - Easy customization of application requirements
Component View in MDK-ARM

The Run-Time Environment is created by selecting available software components.

Dependencies and conflicts between components can be automatically resolved.
Design with Software Components

1. Explore available Software Components from the installed Software Packs

2. Choose components and create a configurable Run-Time Environment (RTE)

3. Develop application in C/C++ using the APIs of RTE components
Using external source code needs Standards

What is required to use standard Software Components on Microcontrollers?

- **Standard Processor Architecture**
  - ARM Cortex™-M series ensure a compatible target processor and provide common core peripherals (i.e. NVIC interrupt system, SysTick timer)

- **Programming Standards on ARM Cortex-M series already provide**
  - Consistent Code Generation and Parameter Passing (ARM EABI)
  - ABI for Special Processor Instructions and Core Peripherals (CMSIS-CORE)
  - Guidelines for definition of Peripheral-Registers & Interrupts (CMSIS-CORE)
  - A rich set of optimized DSP Functions (CMSIS-DSP Library)
  - A standardized RTOS Kernel API (CMSIS-RTOS API)

- **APIs and Drivers for Peripherals are provided by Silicon vendors**
  - But have problems: RTOS interface, DMA, Interrupt, Low-power modes ...
  - Inconsistent APIs across silicon vendors
CMSIS-Drivers: Structure with Drivers

CMSIS-DSP
- DSP-Library

CMSIS-RTOS
- API

Drivers
- (unified)

Real Time Kernel
- (3rd Party)

Device HAL
- (Silicon Vendor)

Middleware
- (3rd Party)

Core Access Functions, Peripheral & Interrupt Definitions

SIMD Cortex-M4

Cortex CPU

SysTick
- RTOS Kernel Timer

NVIC
- Nested Vectored Interrupt Controller

Debug + Trace

Other Peripherals

CoreSight
Consistent Driver APIs for Peripherals

- Each driver module provides one or more drivers for a specific peripheral type
  - Devices may offer several peripherals of same type
  - Driver is configurable to use interrupt and/or DMA
  - Driver uses CMSIS-RTOS functionality to synchronise interrupt and DMA events

- Each driver is accessed by a control structure containing control and data functions
  - Each driver provides the functions: Initialize, Uninitialize, and PowerControl
  - Other functions are specific to the peripheral type

- The RTE_Device.h file centralizes the configuration for all drivers
  - RTE_Device.h could be controlled by a configuration utility.
Drivers: Potential Design Flow

1. Define Peripheral Pin Assignment and Configure Peripherals

Configuration creates control structures

Device Pack
- RTE_Device.h Configuration File
- Startup / System
- UART Driver
- SPI Driver
- Ethernet Driver
- MCI: Memory Card Interface Driver
- NAND Flash Driver
- NOR Flash Driver
- USB Device Driver
- USBH1
- USBD2
Drivers: Potential Design Flow

1. Define Peripheral Pin Assignment and Configure Peripherals

Possible vendor Config Tool integration
Drivers: Potential Design Flow

2. Assign Middleware to Peripherals by selecting the right control structure.
Drivers: Implementation & Optimization

- Driver functions are accessed via a struct
  - Pre-configured drivers (or middleware) may reside in on-chip ROM

```c
typedef struct DRIVER_SPI {
    DRV_VERSION (*GetVersion) (void);
    SPI_CAPABILITIES (*GetCapabilities)(void);
    SPI_STATUS (*Initialize) (SPI_SignalEvent_t cb_event);
    SPI_STATUS (*Uninitialize) (uint32_t slave);
    SPI_STATUS (*PowerControl) (POWER_STATE state);
    SPI_STATUS (*Configure) (uint32_t slave, SPI_FRAME_FORMAT frame_format,
                              SPI_BIT_ORDER bit_order);
    uint32_t (*BusSpeed) (uint32_t slave, uint32_t bps);
    SPI_STATUS (*SlaveSelect) (uint32_t slave, SPI_SS_SIGNAL ss);
    uint8_t (*TransferByte) (uint8_t out);
    SPI_STATUS (*SendData) (const uint8_t *buf, uint32_t len);
    SPI_STATUS (*ReceiveData) (uint8_t *buf, uint32_t len, uint8_t out);
    SPI_STATUS (*AbortTransfer) (void);
} const DRIVER_SPI;
```

- Driver implementation uses CMSIS-RTOS
  - Synchronisation via Semaphore or Mail/Message passing
- Optimal implementations use interrupts and DMA
  - Configuration options are stored in RTE_Drivers.h
Drivers: Function Call Sequence

- Drivers provide common functions, even for power control
  - After initialization, drivers are in power-off mode (minimal power consumption)
  - Before using “transfer” functions, power-on mode must be activated
  - “transfer” functions may cause thread switches

- GetVersion (optional): middleware may use this to insure a certain driver
- GetCapabilities (optional): implemented capabilities may differ; middleware can adjust to implemented functionality
- Initialize: allocate memory, reserve RTOS & peripheral resources, register (optional) call back events
- PowerControl: put the peripheral into low-power (to detect events) or power-up (to transfer data) mode
- Transfer: receive or transmit information; buffers are available after the call to hold new data
- PowerControl: put the peripheral into low-power (to detect events) or power-off (to stop operation) mode
- Uninitialize: release resources consumed by driver
User Benefits: Component-based Approach

- Faster development of complex projects
  - Software components are selected with a mouse click from a library
  - Clean overview of available software components relevant to a device
  - RTE Manager identifies component requirements and connects to device drivers

- Components are separate and give a clean view to application code
  - Libraries: not copied to the project folder; separate of the project
  - Configuration files: accessed as part of a component; stored separate from application code
  - Version control: Select a specific version of a component

- Help System integrated provides painless reading of user manuals
  - Easy access to the API documentation of a component

- Easy adaption to new project requirements
  - Software components can be added or removed any time; project is kept up-to-date
  - Devices can be changed quickly; RTE Manager selects device-relevant software components
Thank you!