Freescale Semiconductor, Inc.
Application Note

Document Number: AN5141
Rev. 0, 06/2015

Relevant and Non-Relevant Code Separation with Kinetis M

Overview and tips for using Kinetis M series MCU with reliable measuring instruments

by: Martin Sebest

1. Introduction

This application note describes the capabilities of the Kinetis M microcontroller devices which are to be used in applications where reliability of the measuring instrument must be ensured. These applications include power meters, water meters, gas meters, heat meters, weighing instruments, taximeters, and many other electronic measuring applications. These devices mostly include a microcontroller that handles billing information and parameters that are subject to legal control. There are several organizations which provide advisory guidelines for writing applications for software controlled measuring instruments. For example, the International Organization of Legal Metrology (OIML) is a worldwide, intergovernmental organization. The European Cooperation in Legal Metrology (WELMEC) is an organization that includes the legal metrology authorities of the Member States of the European Union which provide the rules and recommendation for legally relevant and non-relevant software separation.

Contents

2. Basics of software separation

2.1. Reason to separate the software

Only legally relevant application code of the measuring instrument firmware is subject to legal control. After the relevant application has been approved the manufacturer cannot modify it without re-approval. If a software separation methodology is not implemented then the entire firmware of the device is considered to be a legally relevant application and any modification requires a costly and time-consuming re-approval. However, if a software separation methodology is implemented according to the OIML and WELMEC advisory guidelines then manufacturers can modify the legally non-relevant application without re-approval, gaining flexibility and significant cost savings.

The measuring instrument is controlled by legally relevant and legally non-relevant software applications, as shown in this image.

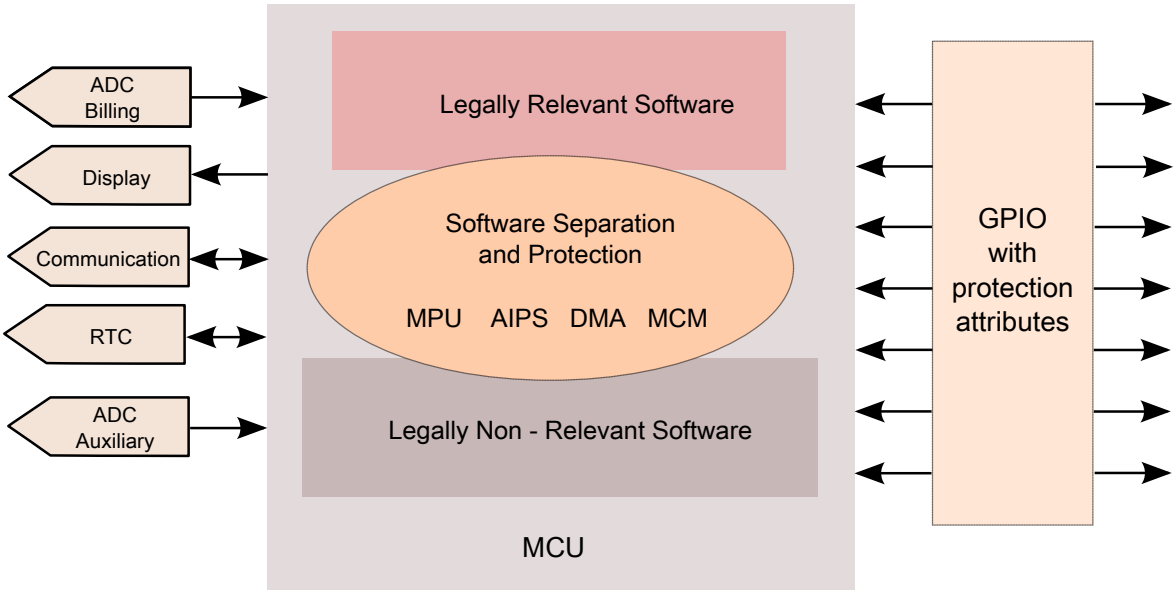

Figure 1. Measuring instrument software structure

The system platform of the Kinetis M device is designed specifically to provide hardware support for software separation within a single chip.

2.2. Legally relevant application code

The legally relevant application code ensures that billing quantities are measured by analog-to-digital converters (ADCs), post-processed, displayed, printed, and transformed into encrypted data packets. This application also maintains billing information, log files, and load profiles in a Non-Volatile Memory (NVM). Certain information must be stored at predefined times so operation of the Real-Time Clock (RTC) module is controlled by the legally relevant application.

2.3. Legally non-relevant application code

Legally non-relevant applications perform all remaining software tasks including communicating digitally-signed packets to the utilities and providing data to equipment attached to a Home Area Network (HAN). For example, a washing machine equipped with a HAN communication interface can be programmed to start washing automatically during non-peak hours so the consumer can take advantage of lower electricity rates. Other smart appliances such as electric heaters can be set to turn on oroff automatically at specified times and thus manage the peak load efficiently. The amount of legally non-relevant code is increasing for metering instruments. The capability of a measuring instrument to share informative data using various protocols and formats with smart appliances is becoming crucial. If the required functionality or protocol is not supported then the manufacturer of the metering instrument must produce it quickly and inexpensively.

3. Kinetis M microcontroller series

The Freescale Kinetis M MCU devices provide the necessary on-chip peripherals, computation performance and power capabilities to enable development of low-cost and highly integrated metering instruments. This is shown in the family block diagram figure below. Kinetis M MCU devices are based on the 32-bit ARM® Cortex® M0+ core with CPU clock rates up to 75 MHz. The Measurement Front-End is integrated on all devices. They include a highly accurate 24-bit Sigma Delta ADC, Programmable Gain Amplifier (PGA), high precision internal 1.2 V voltage reference (VRef), Phase Shift Compensation block, 16-bit SAR ADC and Peripheral Crossbar (XBAR). The XBAR module acts as a programmable switch matrix allowing multiple simultaneous connections of internal and external signals. An Accurate Independent Real Time Clock (IRTC) with passive and active tamper detection capability is also available on all devices. The special feature of the latest chip in the Kinetis M MCU series (KM34Z75) is the Memory Mapped Arithmetic Unit (MMAU). The MMAU provides acceleration to a set of math operations, including signed or unsigned fractional or integer multiplication, accumulation, division, and square-root.

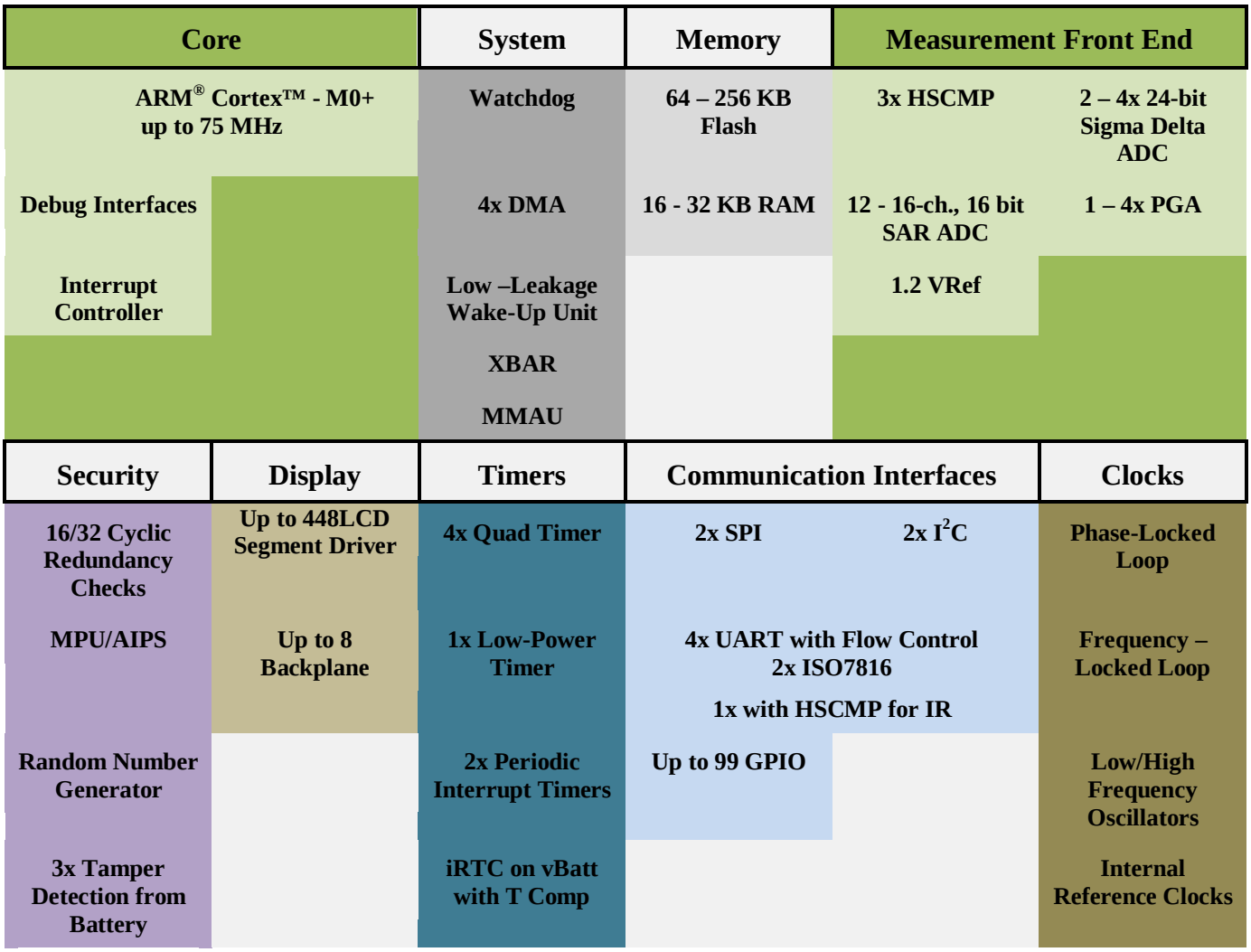

Figure 2. Kinetis M MCU family block diagram

In addition to high performance analog and digital blocks, Kinetis M MCU devices are designed with an emphasis on achieving the required software separation. They integrate hardware blocks supporting distinct separation of the legally relevant software from other software functions. The hardware blocks controlling or checking the access attributes include:

- ARM Cortex-M0+ Core
- DMA Controller Module
- Miscellaneous Control Module
- Memory Protection Unit
- Peripheral Bridge
- General Purpose Input-Output Module

The Kinetis M system platform supports two bus masters — these are the ARM Cortex-M0+ core and DMA controller module. The masters can be optionally enabled or forced by the miscellaneous control module (MCM) to generate either User or Privileged access modes [3].

The next figure shows the hardware blocks that control access to on-chip memories and on-chip peripherals for the ARM Cortex-M0+ core and DMA controller bus masters using Privileged Secure, User Secure, or User Nonsecure attributes.

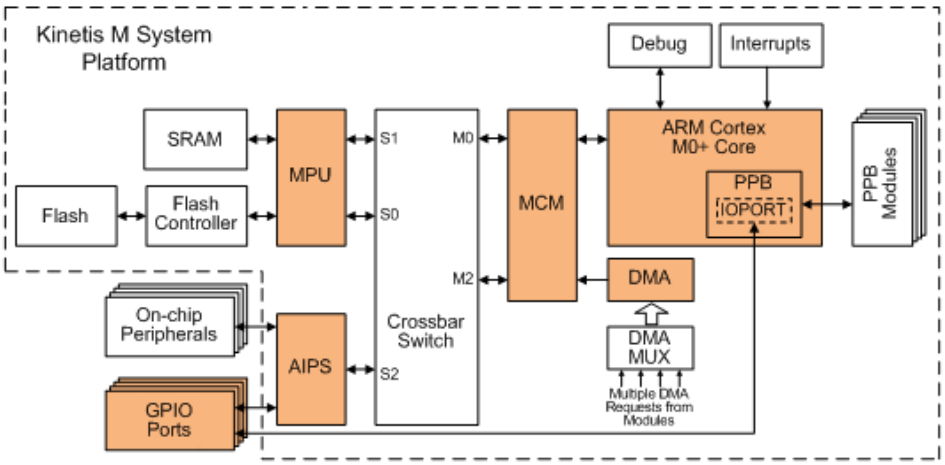

Figure 3. Hardware blocks with controlled access attributes

3.1. ARM Cortex-M0+ core

The ARM Cortex M0+ core provides optional privilege levels for software execution:

- Privileged: The software can use all the instructions and has access to all resources.
- Unprivileged: The software has limited access to system registers
  - Cannot access the system timer, the NVIC, or the system control block.
  - May have restricted access to memory or peripherals.
  - Is commonly named as the User mode of execution.

Only privileged software can write to the control register to change the privilege level for software execution in thread mode. Unprivileged software can use the SVC instruction to make a Supervisor Call to transfer control to privileged software.

In addition to execution levels the ARM Cortex M0+ implements two stacks, the main stack and the process stack, with independent copies of the stack pointer.

3.2. DMA Controller Module

The Kinetis M DMA Controller Module enables fast transfer of data and provides an efficient way to move blocks of data with minimal processor interaction. The DMA Controller module has four independent DMA channels, each with a programmable Transfer Channel Descriptor to operate in Privileged Secure, User Secure, or User Non-secure mode. Attempts at access that are not allowed terminate the bus cycle with an error.

3.3. Miscellaneous Control Module

Besides traditional User or Privileged access modes, the device-specific Miscellaneous Control Module adds an access attribute indicating a Secure or Nonsecure state based on a software-controlled process identifier. Software or a DMA channel that executes in Privileged Secure access mode has no restrictions and gains immediate access to the device resources. Conversely, software or a DMA access that executes in User Secure or Nonsecure access mode has lower priority than those executing in Privileged Secure mode. Software or DMA accesses that execute in User Secure or Nonsecure mode cannot access the System Control Block within the core, the Nested Vectored Interrupt Controller, and the System Timer. These basic User Secure or Nonsecure access mode restrictions are further extended by the platform to limit access to all on-chip peripherals that are critical to chip configuration, reset control, and power management.

3.4. Memory Protection Unit

The Memory Protection Unit provides hardware access control to on-chip flash and SRAM memories. It features eight programmable 128-bit region descriptors. Each descriptor defines start and end addresses and supports read, write, and execute protection attributes for bus masters and supported access modes. This block detects access protection errors if a memory reference does not register in any memory region, or if the reference is illegal in all of the hit memory regions. Accesses that are not allowed generate an error termination (Hard Fault). The MPU is programmable only in Privileged access mode and therefore is the most important unit to develop the application with software separation.

3.5. Peripheral Bridge

The Peripheral Bridge (AIPS) converts the crossbar switch interface to an interface that can access most of the slave peripherals on this chip. It manages all bus master transactions *bus cycles* destined for the attached slave devices and allows programmable unique access rights for each attached slave device. Each peripheral slot defines read and write protection attributes for bus masters and access modes supported by the module. Accesses that are not allowed generate a Hard Fault. The AIPS is programmable only in Privileged access mode.

3.6. General Purpose Input Output Module

Particular emphasis has been given to access control support for the General Purpose Input/Output Module (GPIO). Kinetis M MCU devices have GPIO pins grouped into eight pin ports. Each eight-pin port supports read and write protection attributes for all bus masters and access modes supported by the port. The GPIOs are accessible through the Peripheral Bridge or IOPORT, a special single-cycle interface with the ARM Cortex-M0+ core. Attempts at illegal access through IOPORT are treated as RAZ/WI (Read as Zero/Write Ignored), while access attempts through the Peripheral Bridge generate errors.

3.7. Bare Metal Drivers for Kinetis M MCUs

The Freescale metering engineering group has developed the unique bare metal software drivers for the Kinetis M series of microcontrollers. These software drivers support an optimal application development that is provided in the source code. The bare metal drivers come with a variety of software examples. The software examples demonstrate the correct use of the bare metal software drivers in an application. Each peripheral driver is complemented by one or more software examples depending on the driver complexity and number of features to be demonstrated, as shown in the next figure. For example, the highlighted examples demonstrate the common practices of controlling access to on-chip memories and on-chip peripherals.

The bare metal drivers for Kinetis M MCUs are available for download on the Freescale webpage freescale.com.

Peripheral	Example 1	Example 2	Example 3	Example 4	Example 5
ADC	ADC Example	ADC Operation from Internal 1.0 V PMC Bandgap VREF			
AFE	AFE Software Trigger Polling Mode	AFE Software Trigger Interrupt Mode	AFE Low Power Operation	AFE Clocked by 8.0 MHz OSC, FEE Mode	AFE Clocked by 8.0 MHz OSC,
AIPS	AIPS Example				
CMP	CMP Example				
CRC	Calculating Standardized CRCs	DMA Memory to Peripheral Transfer			
DMA	DMA Memory to Memory Transfer	DMA Peripheral to Memory Transfer	DMA Memory to Peripheral Transfer	DMA Applying Access Protection	
EWM	EWM Example				
FLL	FLL Engaged Controlled by 32.0 kHz IRC	FLL Engaged Controlled by 32.768 kHz RTCOSC	FLL Engaged Controlled by 8.0 MHz OSC	FLL Bypassed Controlled by 4.0 MHz IRC	FLL Bypassed Controlled by 32.76
FTFA	FTFA Example				
GPIO	GPIO Pin Control	GPIO Port's Access Control	GPIO Pin Control with FreeRTOS Tasks with Queue	GPIO Pin Control with FreeRTOS Tasks	
iRTC	IRTC R/W Calendar	IRTC R/W Battery Back-up RAM	IRTC External LDO Regulator Control	IRTC Fine Compensation Logic	
I2C	I2C Polling Mode	I2C Interrupt Mode			
LLWU	LLWU Pin Falling Edge Wakeup	LLWU Periodical Wakeup Using LPTMR			
LPTMR	LPTMR Example	LPTMR Periodic Wakeup			
LPUART	LPUART Polling Mode	LPUART Interrupt Mode			
MCM	MCM Example				
MMAU	MMAU Sine Computing	MMAU IIR-4ord Filter Computing	MMAU Register Save/Restore		
MMCAU	MMCAU AES128 Cipher Calculation	MMCAU DES Cipher Calculation	MMCAU MD5 Hash Calculation	MMCAU SHA1 Hash Calculation	MMCAU SHA256 Hash Calcula
MPU	MPU Example				
OSC	OSC Example				
OSC32K	OSC32K Example				
PDB	PDB Example	PDB Pulse Out Generation	PDB AFE Trigger	PDB Back-to-Back Trigger	
PIT	PIT Example	PIT Triggered FreeRTOS Handler Task			
PLL	PLL Engaged Controlled by 32.0 kHz IRC	PLL Engaged Controlled by 32.768 kHz RTCOSC	PLL Engaged Controlled by 8.0 MHz OSC	PLL & FLL Engaged Controlled by 32.768 kHz RTCOSC	PLL Controlled by 32.768 kHz RT Engaged Controlled by 32.0
PMC	PMC Example				
RCM	RCM Example				
RNGA	RNGA Example				
SIM	SIM Example				
SMC	SMC Example				
SPI	SPI Polling Mode	SPI Interrupt Mode	SPI FIFO Slave Interrupt Mode	SPI FIFO Master Interrupt Mode	
SWISR	SWISR Example				

Figure 4. Kinetis M bare metal drivers software examples

4. Application development

This section will show how the application with relevant and non-relevant software separation can be developed on Kinetis M MCU devices using bare metal software drivers and IAR Embedded Workbench IDE.

4.1. Prepare the concept

There is no strict rule regards how the relevant and the non-relevant software should look. There are specific requirements and recommendations referenced in documents [1] [2]. The software example discussed in this section shows a typical approach for relevant and non-relevant software separation in a

simple application. It is assumed that some adjustments must be made to the application to use it in a real application.

4.1.1. Application concept

The final application must consist of the legally relevant and the legally non-relevant parts of the software. This means that the flash memory and RAM memory of the device must be divided into two sections. The first section is for relevant software code and the second section is for non-relevant software code. The size requirements of legally non-relevant software code are continuously increasing. Two requirement must remain in effect during the entire life of the application—that is, legally relevant code must remain and the legally relevant data must only be affected by the legally relevant code. These requirements are fulfilled through the use of, and correct setting of, the Memory Protection Unit (MPU) that defines eight memory regions with different protection properties.

The placement of the relevant and non-relevant parts of software code in the specific location in the memory and creating of the memory region is defined in the linker file.

A peripheral can be setup by using the Peripheral Bridge (AIPS) when peripheral memory space can or cannot be accessed from the relevant or the non-relevant part of the software.

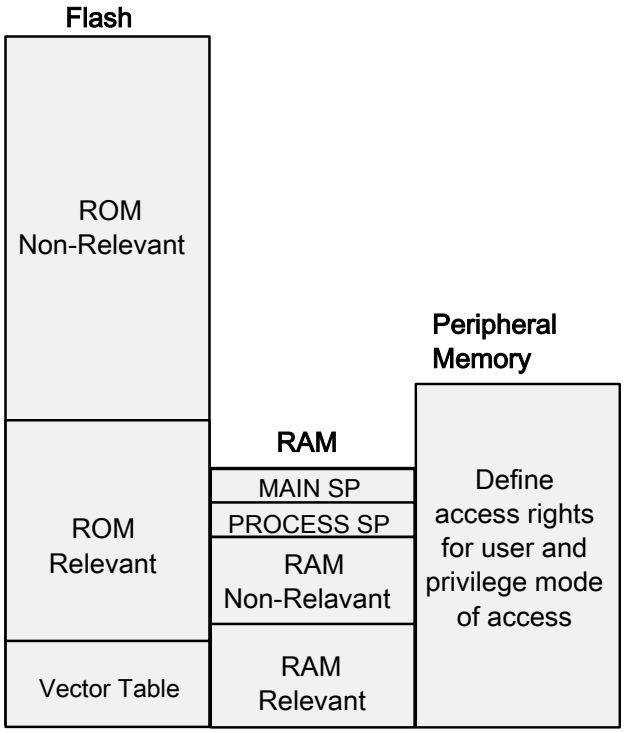

Figure 5. Distribution of memory for specific parts of application

The main content of the legally relevant part of the application software is the measuring of billing quantities by the analog-to-digital converter. It is good for this measuring to be performed in interrupts because all interrupts are handled in a supervisor mode of execution. The remainder of the application, the non-relevant part, is executed in the user mode of execution. The time schedule of the application is shown in this figure.

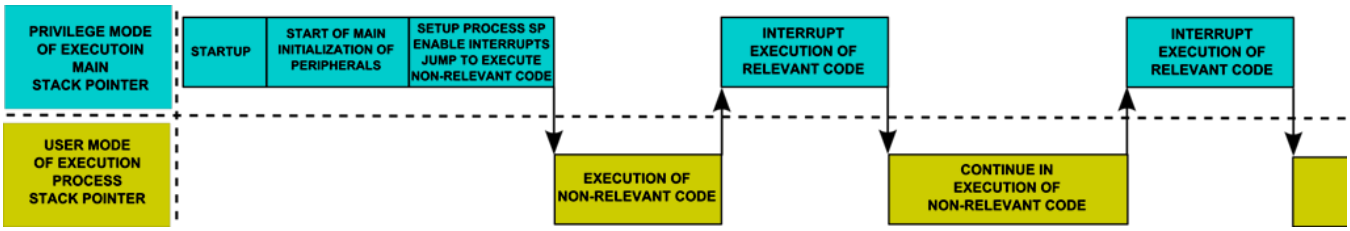

Figure 6. Time schedule of application with relevant and non-relevant software

After any reset condition, including Power on Reset (POR), the ARM Cortex-M0+ Core starts executing software in Privileged Secure mode. It is necessary to initialize all discussed hardware blocks and to program their associated access attributes. All configuration attributes can be locked by the software until the next POR. After programming all access attributes, the measuring instrument firmware can initiate the legally relevant and the legally non-relevant software.

4.2. Developing the application

To demonstrate the capabilities and usability of Kinetis M MCUs in applications where software separation is necessary the following example will be used.

Table 1. Definition of example

Software part	Definition	Notes
Relevant	Sensing voltage signal from potentiometer placed on KM Tower System board using AD converter.	This part of the application must remain in the same location during the entire life of the application.
Non-Relevant	Blinking with random LEDs. The frequency of blinking LEDs changes according to the value of sensed voltage signal.	This part of the application may be changed. For example a different combination of LEDs may be used.
Example notes: Take into account that the legally non-relevant part of the application has the possibility to be updated later. The legally relevant part of the application must remain unaffected.		

The legally relevant part of the application is in most cases the fixed part of the application. There is usually no need to change this part of the software if everything in this part of the application functions well. The legally non-relevant software is the part of the application that may be requested to be changed or updated frequently during the lifetime of the application. Therefore it is good to have a standalone project for the relevant part of the software and a standalone project for the non-relevant software development. Projects for the relevant part of the application and for the non-relevant part of the application based on Kinetis M bare metal drivers are attached in the associated package with the Application Note.

4.2.1. Project for the relevant part of software development

The legally relevant part of the application code is written in the standard project file which is created with the project generator that is part of the Kinetis M bare metal drivers which was mentioned in *Chapter 3.7*. The developer must focus only on the project linker file where the legally relevant code and data (flash and RAM memories) must be managed with respect to the legally non-relevant code and data capacity requirement.

The project linker file must be updated in the following way:

- Define the memory regions for the legally relevant and the legally non-relevant code and data.
- Define the memory block for the process stack pointer.
- Place the process stack pointer memory block into the legally non-relevant RAM data region.
- Place the non-relevant code and the non-relevant data into specific memory blocks.

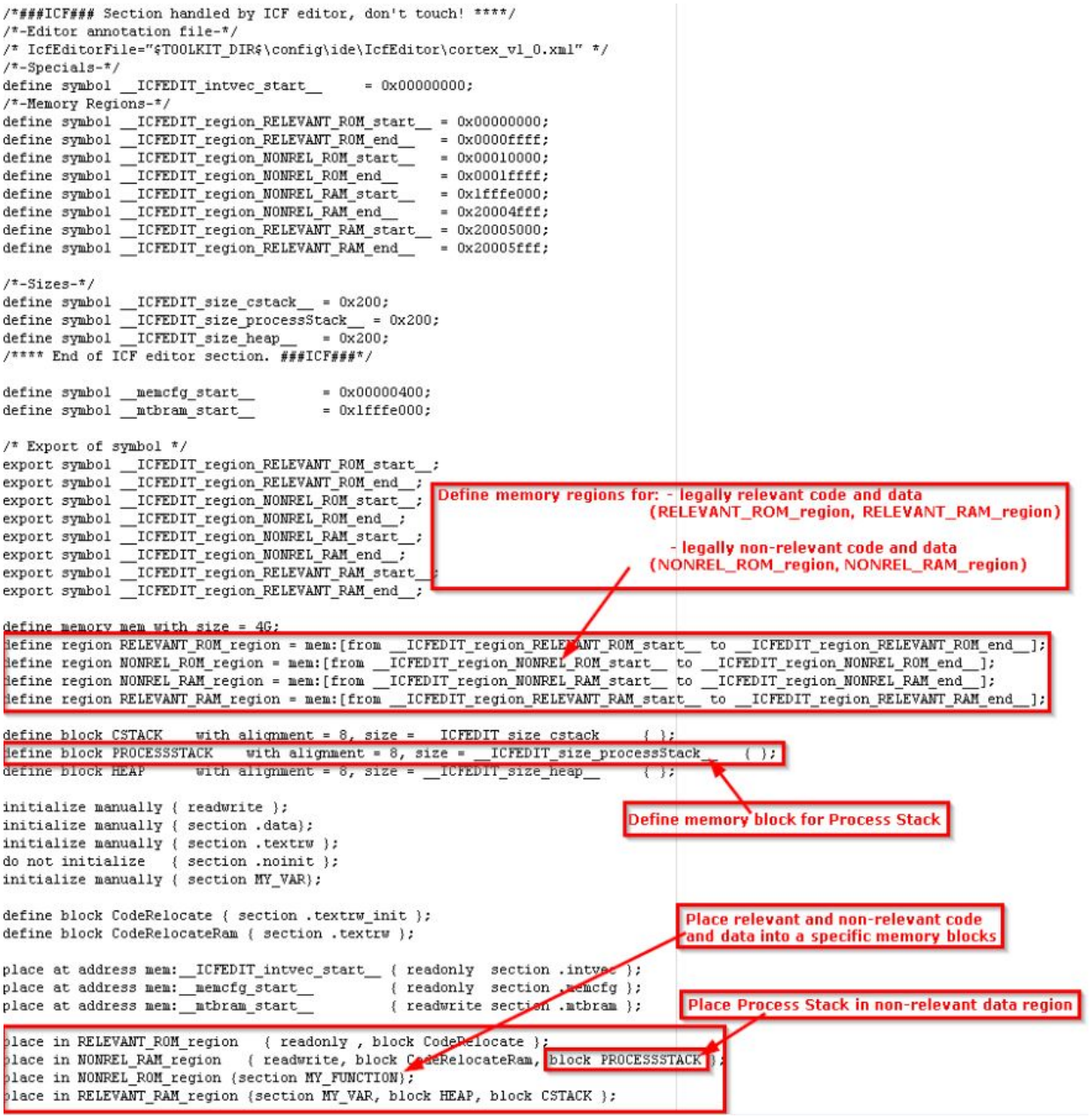

Figure 7. Linker file for main application

The legally relevant part of the application code executes through interrupts as was mentioned in *Chapter 4.1*. All interrupts are executed in privileged mode of access using the main stack pointer. The whole non-relevant part of the application is executed in the User mode of access using the process stack

pointer. The application flowchart is in the next figure and illustrates how the legally relevant part of the application is developed.

In this project the legally relevant and legally non relevant parts of the application may be developed at the same time. However, if the requirement for changing the non-relevant part of the application occurs, it is better to change it only in the project dedicated for non-relevant application (see *Chapter 4.2.2*).

The source code of the main application project for the solved example is in *Appendix.1*.

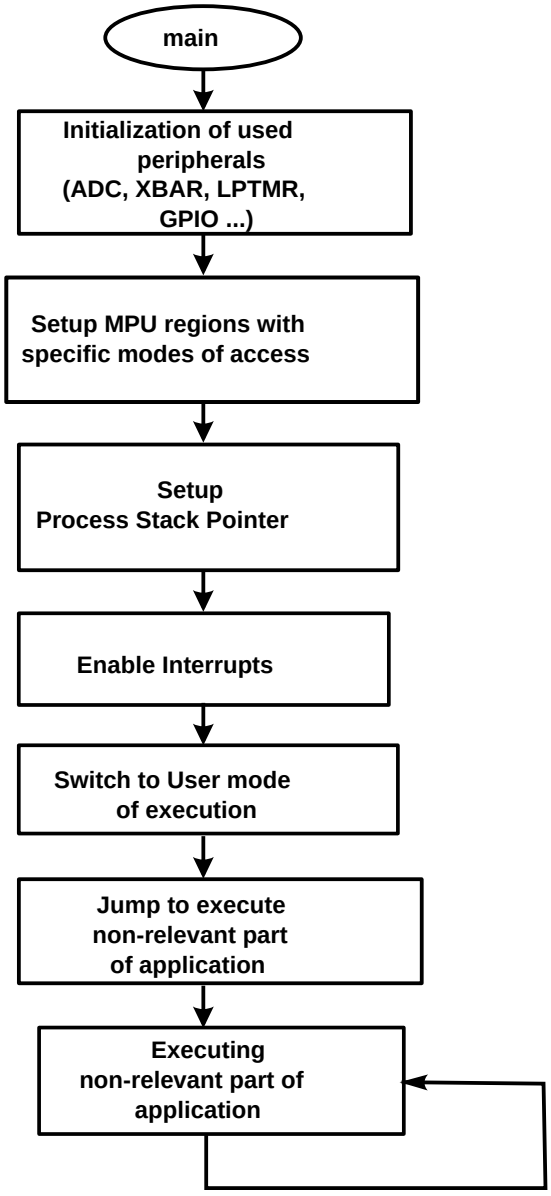

Figure 8. Application flowchart

4.2.2. Project for the non-relevant part of software development

The legally non-relevant part of the application is written in the project and is derived from the standard project. The linker file for the “dummy” project contains only necessary commands such as:

- Define the memory regions for legally non-relevant code and data
- Tell the linker to copy non-relevant data at start up time (instruction – *initialize by copy*)
- Place the non-relevant code and the non-relevant data into specific memory blocks

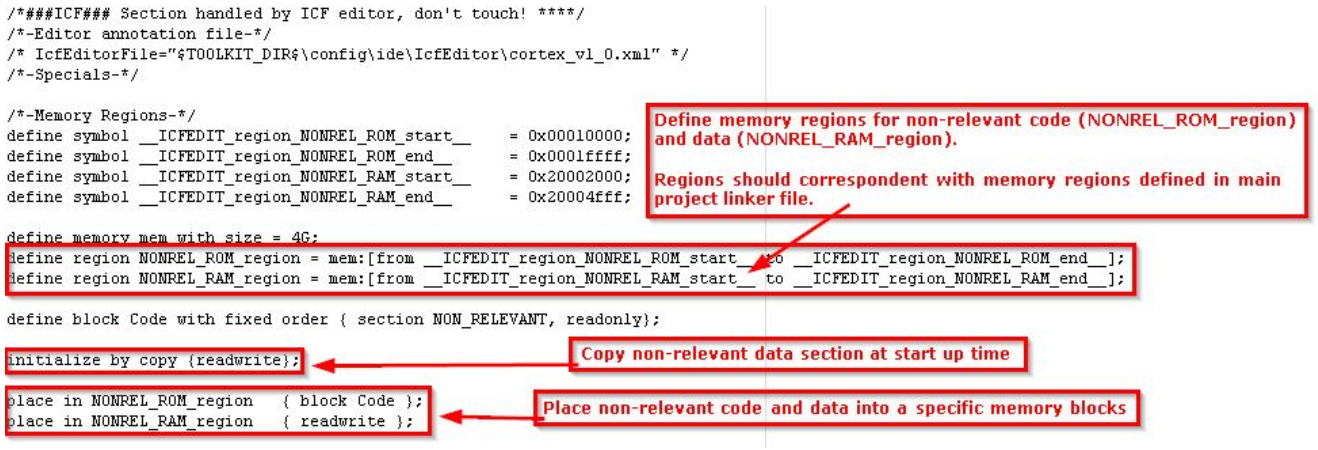

Figure 9. Linker file for “dummy” project where the legally non-relevant part of the application may be updated

The developer of the non-relevant part of the application may use all microcontroller peripherals that a non-relevant application can handle (according to settings in the relevant application project) otherwise it will generate a Hard Fault after downloading to the main application. The developer must be aware of the memory size of the non-relevant part of the application.

The flowchart in the figure below outlines the process. At startup the global variables (if any exist) are initialized and at the end of startup the main function is called and executed. The source code for the non-relevant part of the application for the solved example is found in *Appendix 2*.

After the new non-relevant part of the application is prepared the binary or s-record (s19) file must be generated.

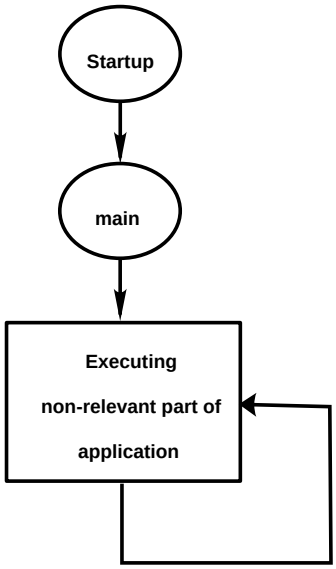

Figure 10. Dummy project flowchart

4.3. Reflashing the non-relevant part of the application

One of the main requirements in the application code separation topic is the ability to change the non-relevant part of the application during the runtime. There are many ways to achieve this. The UART periphery is used as a frontend for changing the non-relevant part of the application in the solved example. Any other communication peripherals available on the Kinetis M MCU may be used for reflashing the non-relevant part of the application.

In this example the new non-relevant part of the application is sent through the UART communication interface in the form of a binary file previously generated from a “dummy” project.

The reflashing process takes place in port callback and is performed with the following steps:

1. Disable all interrupts.
2. Erase the flash memory sector which contains the non-relevant part of the application.
3. Send and receive the new, non-relevant part of the application through the MCU communication interface (for example UART) in the form of a binary file.
4. Write the new non-relevant part of the application into the intended flash memory region.
5. Reset the process stack pointer and set the correct values for xPSR, Program Counter and Link register stacked in process stack pointer.
6. Enable interrupts.
7. The application runs with the updated non-relevant part and the legally relevant part of the application remains untouched.

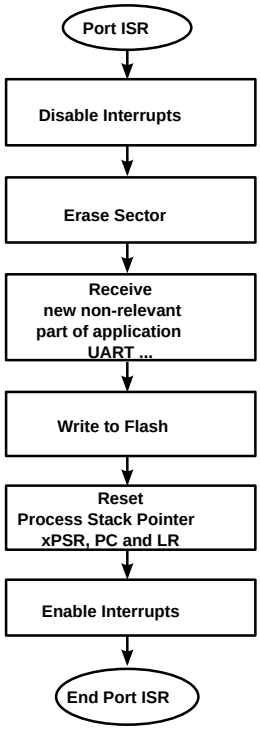

Figure 11. Reflashing process flowchart

4.3.1. Set up the process stack pointer after reflash

When a new non-relevant part of the application is loaded the process stack pointer needs to be correctly set up. If the process stack pointer remains unchanged the move from the interrupt handler back to the new non-relevant application will not be successful and will generate a Hard Fault.

According to the ARM Cortex-M0+ core documentation at the exception entry the processor pushes eight data registers onto the current stack. In this example it is the process stack. This is outlined in the next figure.

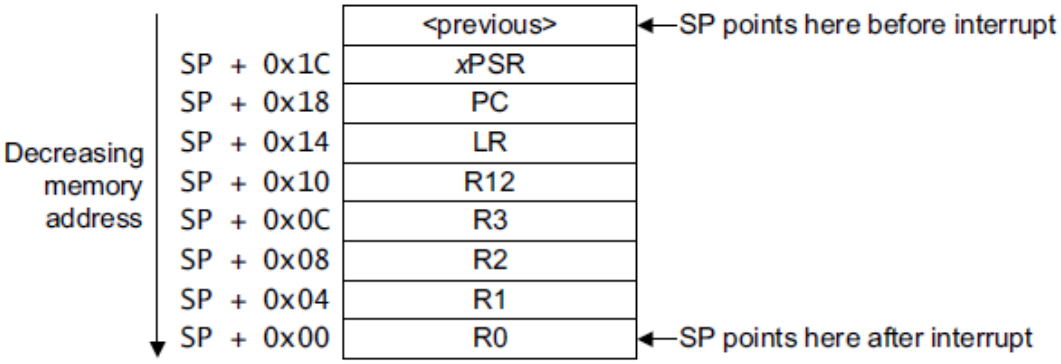

Figure 12. Stacking frame when exception (interrupt) occurs

For the successful return from the reflashing process back to the start of the new non-relevant application, the process stack address must be reinitialized and the following stacked registers of the process stack must be correctly adjusted:

- xPSR register – Execution Program Status Register – set to 0x01000000 value.
- PC – Program Counter – set to the first address of the non-relevant application (0x00010000 in our example).
- LR – Link Register – LR stores the return information for the subroutines, the function calls and the interrupts (exceptions) (0x00010001 in our example).

Other stacked registers (R0 — R3, R12) do not need to be adjusted for specific values, they may remain as they are. To handle the process stack a C-code inline macro called PSP_HANDLE (psp_add, ret_add) was created and is used in this example.

4.4. Final application

The most important steps in the development process of the application where the legally relevant and the legally non-relevant parts must be separate have been described in the example. The legally relevant part of the application remains the same and cannot be modified. In case of an attempt to modify the legally relevant software a Hard Fault will be generated.

The legally non-relevant part of the application can be modified. The new version of the non-relevant part of the application is developed in a separate project with the specific linker file. After the modified non-relevant part is prepared then it can be loaded into the application through one of the microcontroller's communication interfaces (UART, SPI, or I2C).

A special procedure must be carried out with the process stack after the reflashing process. The stacked program status register, program counter register, and link register must be correctly adjusted before the start of executing a new non-relevant part of the application.

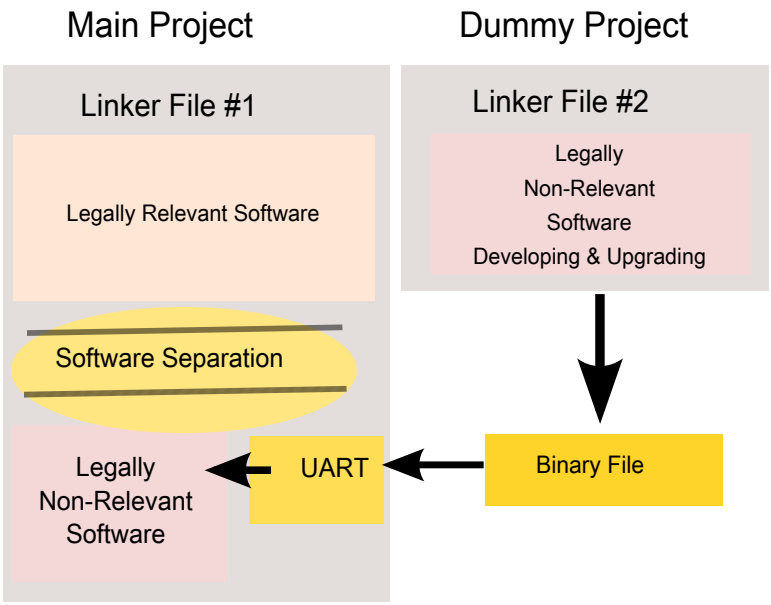

Figure 13. Block diagram of solved example

In the example presented in this chapter, the relevant part of the application ensures the voltage sensing from the potentiometer placed on the development KM34Z75M Tower System development board. The function of the non-relevant part of the application is to control the blinking frequency and blinking arrangement of the LEDs that are assembled on the Tower System development board according to the actual level of sensed voltage. The non-relevant part of the application can be modified. The modification is performed as a standalone project for non-relevant project developing. The UART communication interface is used for loading the new non-relevant part of the application into the microcontroller. The following figure shows the existing application being modified during its runtime. After the non-relevant part of the application has been reflashed the application continues without the need to perform a reset and the relevant part remains unaffected.

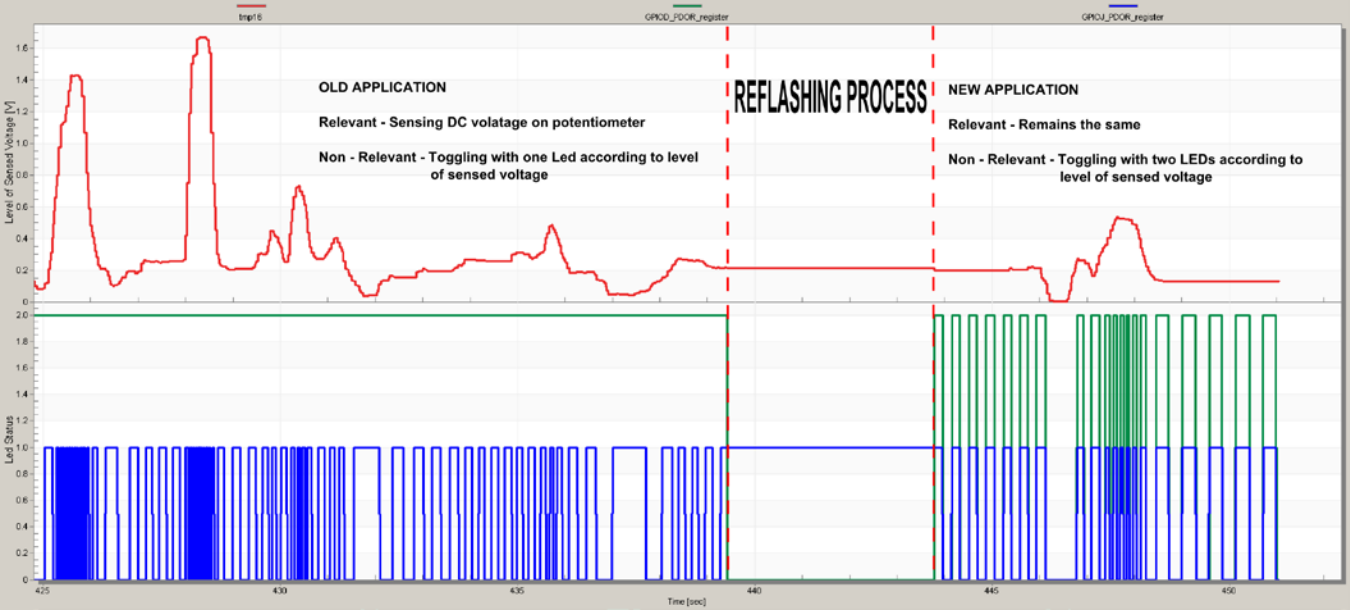

Figure 14. The non-relevant part is updated, the relevant part is unaffected

5. Summary

In the previous chapters the essential characteristics of separation of legally relevant and legally non-relevant software (or parts of the application) were explained.

Kinetis M microcontrollers are based on the ARM Cortex M0+ core which supports supervisor (privilege) and user (non-privilege) modes of execution. A broad range of modules and extended peripherals with controlled access attributes intended for software separation and protection like MCM, AIPS, MPU, DMA, and GPIO are standard features of Kinetis M microcontrollers.

An example for legally relevant and legally non-relevant software separation with the possibility to change the content of the non-relevant part of the application is shown in this application note. The significant benefit for the application approval process is that the relevant part of the application remains unaffected.

Kinetis M MCU devices are supported by unique bare metal drivers with an interactive reference manual to make development easier. Several pre-compiled examples are included in the Kinetis M bare metal drivers installation package including examples for software separation development.

6. References

1. *General Requirements for Software Controlled Measuring Instruments*, (OIML, 2008 http://workgroups.oiml.org/tcsc/tc-07/tc-07-sc-04/reference-documentation/D031-e08.pdf)
2. *WELMEC 7.2, Software Guide (Measuring Instruments Directive 2004/22/EC)*, (www.welmec.org/fileadmin/user_files/publications/WELMEC_07.02_Issue5_SW_2012-03-19.pdf)
3. *Kinetis M Sub-Family Reference Manual*, available on freescale.com
4. *Cortex-M0+ Devices - Generic User Guide*, (2012 ARM, http://infocenter.arm.com/help/topic/com.arm.doc.dui0662b/DUI0662B_cortex_m0p_r0p1_dgug.pdf)
5. *Kinetis M Support for Distinct Separation of Legally Relevant Software*, Joe Circello and Martin Mienkina, (document KINETISMWP)
6. *IAR C/C++ Development Guide, Compiling and Linking*, (http://supp.iar.com/FilesPublic/UPDINFO/005832/arm/doc/EWARM_DevelopmentGuide.ENU.pdf)
7. *The Definitive Guide to the ARM Cortex* - M0, Joseph Yiu, (http://www.sciencedirect.com/science/book/9780123854773)

7. Revision History

Table 2. Revision history

Revision Number	Date	Substantive changes
0	6/2015	Initial release

Appendix A. Main project source code

```
#include "drivers.h"

/* LEDs definition */
#define LED1 PIN3 /* PTJ3 */
#define LED2 PIN4 /* PTJ4 */
#define LED3 PIN0 /* PTD0 */
#define BTN2 PIN1 /* PTD1 */
#define JMPADDR 0x00010001
#define TOP_PSP 0x20005000

/*****************************************************************************//*!
 * @brief   Handles Process stack pointer after reflashing process
 * @details This macro setup process stack to smoothly start new
          non-relevant part of application.
 * @param   psp_add adress of top of process stack
 * @param   ret_add adress of non-relevant part of application start
 * @note    Implemented as an inline macro.
 *****************************************************************************/
#define PSP_HANDLE(psp_add,ret_add) {                           \
                uint32 *p_psp = (uint32 *)(psp_add);            \
                SetPSP((psp_add)-32);                           \
                *(p_psp-1)=0x01000000;                          \
                *(p_psp-2)=((ret_add));                         \
                *(p_psp-3)=((ret_add));                         \
                 }


/* Imported symbols from linker                                 */
  extern char __ICFEDIT_region_RELEVANT_ROM_start__[];
  extern char __ICFEDIT_region_RELEVANT_ROM_end__[];
  extern char __ICFEDIT_region_NONREL_RAM_start__[];
  extern char __ICFEDIT_region_NONREL_RAM_end__[];
  extern char __ICFEDIT_region_NONREL_ROM_start__[];
  extern char __ICFEDIT_region_NONREL_ROM_end__[];
  extern char __ICFEDIT_region_RELEVANT_RAM_start__[];
  extern char __ICFEDIT_region_RELEVANT_RAM_end__[];
  #define RELEVANT_ROM_START_ADDR     __ICFEDIT_region_RELEVANT_ROM_start__
  #define RELEVANT_ROM_END_ADDR       __ICFEDIT_region_RELEVANT_ROM_end__
  #define NONREL_RAM_START_ADDR     __ICFEDIT_region_NONREL_RAM_start__
  #define NONREL_RAM_END_ADDR       __ICFEDIT_region_NONREL_RAM_end__
  #define NONREL_ROM_START_ADDR  __ICFEDIT_region_NONREL_ROM_start__
  #define NONREL_ROM_END_ADDR     __ICFEDIT_region_NONREL_ROM_end__
  #define RELEVANT_RAM_START_ADDR  __ICFEDIT_region_RELEVANT_RAM_start__
  #define RELEVANT_RAM_END_ADDR    __ICFEDIT_region_RELEVANT_RAM_end__

/* tmp16 --> adc result register ... stored at specific address */
volatile uint16 tmp16 @ "MY_VAR" ;

uint32 counter=0;

/* Callback functions declaration                               */
/* Callback functions are considered as relevant code           */
static void port_callback(PORT_CALLBACK_SRC src, uint8 pin);

void ADC_ISR(ADC_CALLBACK_TYPE type, register int16 result);

/* Non relevant functions declaration                           */
#pragma location="MY_FUNCTION"
void NonRelevant (void);
#pragma location="MY_FUNCTION"
void blink (uint16 input);

void main (void)
{
  /* enable clocks to all on chip peripherals                   */
```

```
SIM_Init (SIM_MODULE_ALL_PERIPH_ON_CONFIG);

/* route core clock to PTF7 for monitoring                          */
SIM_SelClkout (CLKOUT_SRC1);
PORT_Init (PORTF,PORT_MODULE_ALT3_MODE,PIN7);

/* clock mode 1:1:1, 24MHz */
SIM_SetClkMode (SYSCLK_MODE0);
SIM_SetClkDiv (SYSCLK_DIV1);
FLL_Init (FLL_MODULE_FEE_24MHZ_CONFIG);

/* Port and GPIO Init */
PORT_Init (PORTJ, PORT_MODULE_LED_MODE, LED1);
PORT_Init (PORTJ, PORT_MODULE_LED_MODE, LED2);
PORT_Init (PORTD, PORT_MODULE_LED_MODE, LED3);
PORT_Init (PORTD, PORT_MODULE_BUTTON_IRQ_MODE, BTN2);
GPIO_Init (GPIOJ, GPIO_OUT_LOGIC0_MODE, LED1);
GPIO_Init (GPIOJ, GPIO_OUT_LOGIC1_MODE, LED2);
GPIO_Init (GPIOD, GPIO_OUT_LOGIC1_MODE, LED3);
GPIO_Init (GPIOD, GPIO_INP_MODE, BTN2);

PORT_InstallCallback (PRI_LVL0,port_callback);

/* Uart initialization                                              */
PORT_Init (PORTI, PORT_MODULE_ALT2_MODE, PIN6|PIN7);
UART_Init (UART2, UART_MODULE_POLLMODE_CONFIG(9600,24e6));

XBAR_Init (XBAR_MODULE_ANYEDGE_DETECT_CONFIG,
        XBAR_MODULE_NO_EDGE_DETECT_CONFIG,
        XBAR_MODULE_NO_EDGE_DETECT_CONFIG,
        XBAR_MODULE_NO_EDGE_DETECT_CONFIG,
        PRI_LVL0, NULL);
/* PIT0 overflow triggers ADC CHA                                   */
XBAR_Path (XBAR_PIT0TIF1, XBAR_ADCTRGCHA);
/* PIT1 overflow triggers ADC CHB                                   */
XBAR_Path (XBAR_PIT1TIF1, XBAR_ADCTRGCHB)
/*PIT initialization                                                */
PIT_Init (PIT0, CH1, PIT_CH_TMR_EN_CONFIG, 1000000);
PIT_Init (PIT1, CH1, PIT_CH_TMR_EN_CONFIG, 2400000);

/* ADC initialization, HW triggered, interrupt enable               */
ADC_Init (ADC_MODULE_16B_HWTRG_XREF_CONFIG,
       HWAVG_16,
       ADC_CH_SE_IRQ_CONFIG(ADC_SE8),
       ADC_CH_SE_IRQ_CONFIG(ADC_SE9),
       ADC_CH_DISABLE_CONFIG,
       ADC_CH_DISABLE_CONFIG,
       PRI_LVL1, ADC_ISR);

 /* Set bus masters attribute to be controlled internally by the core       */
MCM_SetMasterAttr (MCM_CM0_MASTER|MCM_DMA_MASTER,
            MCM_MASTER_EN_PRIV_OR_USER_SECURE_OR_NONSEC,
            TRUE);

/* Initialize RGD1 = Relevant code; RWX supervisor mode, RX user mode      */
MPU_RgdInit(RGD1,
        MPU_RGD_EN_CM0_PID_OFF_DMA_PID_OFF_CONFIG(MPU_SPVR_RWX, /* CM0+ */
                                    MPU_USER_RX,
                                    MPU_SPVR_RWX, /* DMA  */
                                    MPU_USER_RX,
                                    RELEVANT_ROM_START_ADDR,
                                    RELEVANT_ROM_END_ADDR));
/* Initialize RGD2 = Non relevant code; RWX supervisor mode, RWX user mode  */
MPU_RgdInit(RGD2,
        MPU_RGD_EN_CM0_PID_OFF_DMA_PID_OFF_CONFIG(MPU_SPVR_RWX, /* CM0+ */
                                    MPU_USER_RWX,
                                    MPU_SPVR_RWX, /* DMA  */
                                    MPU_USER_RWX,
                                    NONREL_ROM_START_ADDR,
                                    NONREL_ROM_END_ADDR));
```

```
 /* Initialize RGD3 = Generally used RAM; RWX supervisor mode, RWX user mode  */
  MPU_RgdInit(RGD3,
          MPU_RGD_EN_CM0_PID_OFF_DMA_PID_OFF_CONFIG(MPU_SPVR_RWX,/* CM0+ */
                                    MPU_USER_RWX,
                                    MPU_SPVR_RWX, /* DMA */
                                    MPU_USER_RWX,
                                    NONREL_RAM_START_ADDR,
                                    NONREL_RAM_END_ADDR));
  /* Initialize RGD4 = RAM ; RW supervisor mode, R user mode              */
  /* In this part of RAM relevant variable is placed, user can read only     */
  MPU_RgdInit(RGD4,
          MPU_RGD_EN_CM0_PID_OFF_DMA_PID_OFF_CONFIG(MPU_SPVR_RW, /* CM0+ */
                                    MPU_USER_R,
                                    MPU_SPVR_RW, /* DMA */
                                    MPU_USER_R,
                                    RELEVANT_RAM_START_ADDR,
                                    RELEVANT_RAM_END_ADDR));
  /* Initialize RGD5 = MSP and PSP STACK; RW supervisor mode, RW user mode   */

  /* Invalidate overlapping RGD0                                  */
 MPU_DisableRGD0();


 SetPSP(TOP_PSP);    /* Set Process Stack Pointer                        */
 SelPSP();          /* Select Process Stack Pointer                     */
 EnableInterrupts();
 UserMode();         /* Switch into User mode of execution with using PSP    */

 NonRelevant();      /*Jump to execute Non relevant code                  */

}

void ADC_ISR(ADC_CALLBACK_TYPE type, register int16 result)
{
 if(type == CHA_CALLBACK)
 {
   tmp16 = ADC_Read(CHA);
 }
}


 static void port_callback(PORT_CALLBACK_SRC src, uint8 pin)
 {
    uint8 array[2048];
    uint8 bin ;

    DisableInterrupts();

    FTFA_EraseSector (NONREL_ROM_START_ADDR); /* Erase 2KB flash sector     */

    if (UART_RxFull(UART2))
    {
     bin = UART2_D;
     }
    UART_Rd(UART2,array,324); /* Read 324B of data from UART2              */

    FTFA_WriteArray(NONREL_ROM_START_ADDR,array,sizeof(array));/*write into flash */

    PSP_HANDLE(TOP_PSP,JMPADDR);

    EnableInterrupts();
 }

/* Definition of non relevant functions                                 */
#pragma location="MY_FUNCTION"
void NonRelevant(void)
{
 while(1)
 {
 asm("nop");
```

```
 blink(tmp16);
  }
}

#pragma location="MY_FUNCTION"
void blink(uint16 input)
{
 counter += input;
 if(counter > 500000000)
   {

    GPIO_Tgl(GPIOJ,LED1);
    counter=0;
    GPIO_Tgl(GPIOJ,LED2);
    GPIO_Tgl(GPIOD,LED3);
   }
}
```

Appendix B. Dummy project source code

```
#include "drivers.h"
#define LED1 PIN3 /* PTJ3 */
#define LED2 PIN4 /* PTJ4 */
#define LED3 PIN0 /* PTD0 */

void entry(void);
void __iar_program_start(void);

/*****************************************************************************/
/*Here is the place to declare your non-relevant functions                   */
void blink(uint16 *p_input);

/*****************************************************************************/
/* Define your global variables here :                                       */
int mydata = 5;
int zerodata = 0;
uint32 counter=0;

/*****************************************************************************/
/* Function entry is placed at specific place in the flash memory to ensure  */
/* correct start of non-relevant part of application. The purpose of */
/*             !!! Do not touch this part of code !!!                        */
#pragma location="NON_RELEVANT"
void entry(void)
{
   // call iar c-startup
   __iar_program_start();
}

/*****************************************************************************/
/* Here you can write your non-relevant application:                         */
/* main is called by iar startup and globals are ready to run c-code         */
void main (void)
{
  volatile int unused = 0x35;
  mydata = 2;
  zerodata = 7;
  counter=1;

    unused +=  mydata;

  while(1)
  {
```

```
  asm("nop");
   blink((uint16 *)0x20005000);
  }
}
/*****************************************************************************/
/* Here is the place for yor non-relevant functions definition:              */
void blink(uint16 *p_input)
{
   counter += *p_input;
  if(counter > 500000000)
  {
   GPIO_Tgl(GPIOJ,PIN3);
   counter=0;
   GPIO_Tgl(GPIOJ,PIN4);
   GPIO_Tgl(GPIOD,PIN0);
  }
}
```

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5141
Rev. 0
06/2015