```c
/*
 * dma.c
 *
 *  Created on: 30-Nov-2017
 *      Author: Gunj Manseta
 */


#include "dma.h"
#include "MKL25Z4.h"
#include "time_profiler.h"
#include "logger.h"

#define DMA_COUNT (4)

volatile DMA_state_t DMA_CurrentState[4] = {0};
volatile tickTime tickStart = 0;
volatile tickTime tickEnd = 0;

void dma_clockEnable()
{
	SIM->SCGC7 |= SIM_SCGC7_DMA(1);
}

void dma_clockDisable()
{
	SIM->SCGC7 &= ~(SIM_SCGC7_DMA_MASK);
	DMA_CurrentState[DMA_0] = DMA_Disabled;
	DMA_CurrentState[DMA_1] = DMA_Disabled;
	DMA_CurrentState[DMA_2] = DMA_Disabled;
	DMA_CurrentState[DMA_3] = DMA_Disabled;
}

int8_t dma_configure(DMA_t dma_n, DMA_Configure_t *DMA_config_data)
{
	if(dma_n < DMA_COUNT)
	{
		dma_clockEnable();
		DMA_DSR_BCR(dma_n) |= DMA_DSR_BCR_DONE(1);

		DMA_DCR(dma_n) = 0;
		DMA_DCR(dma_n) |= DMA_DCR_AA(DMA_config_data->AutoAlign);
		DMA_DCR(dma_n) |= DMA_DCR_EINT(DMA_config_data-
>EnableInterrupt);
		DMA_DCR(dma_n) |= DMA_DCR_CS(DMA_config_data->CycleSteal);
		DMA_DCR(dma_n) |= DMA_DCR_ERQ(DMA_config_data-
>EnablePeripheralReq);
		DMA_DCR(dma_n) |= DMA_DCR_D_REQ(DMA_config_data->D_REQ);

		NVIC_ClearPendingIRQ(dma_n);
		NVIC_EnableIRQ(dma_n);
		DMA_CurrentState[dma_n] = DMA_Ready;
		return 0;
	}
	else
	{
		return -1;
	}
```

```
}

int32_t dma_initiate(DMA_t dma_n, DMA_Addresses_t *DMA_addresses_data)
{
	if(dma_n < DMA_COUNT && ((DMA_CurrentState[dma_n] == DMA_Ready) ||
(DMA_CurrentState[dma_n] == DMA_Complete)) )
	{
		DMA_SAR(dma_n)	=	DMA_addresses_data->Src_Add;
		DMA_DAR(dma_n)	=	DMA_addresses_data->Dest_Add;
		DMA_DSR_BCR(dma_n) |= DMA_DSR_BCR_BCR(DMA_addresses_data-
>NumberOfBytes);


		DMA_DCR(dma_n) &= ~DMA_DCR_SINC_MASK;
		DMA_DCR(dma_n) &= ~DMA_DCR_DINC_MASK;
		DMA_DCR(dma_n) &= ~DMA_DCR_SSIZE_MASK;
		DMA_DCR(dma_n) &= ~DMA_DCR_DSIZE_MASK;

		DMA_DCR(dma_n) |= (DMA_addresses_data->Src_Size < 3)?
DMA_DCR_SSIZE(DMA_addresses_data->Src_Size) : DMA_DCR_SSIZE(DMA_8Bits);
		DMA_DCR(dma_n) |= (DMA_addresses_data->Dest_Size <
3)?DMA_DCR_DSIZE(DMA_addresses_data->Dest_Size):
DMA_DCR_DSIZE(DMA_8Bits);

		DMA_DCR(dma_n) |= DMA_DCR_SINC(DMA_addresses_data-
>SrcAddr_Inc);
		DMA_DCR(dma_n) |= DMA_DCR_DINC(DMA_addresses_data-
>DestAddr_Inc);

		DMA_CurrentState[dma_n] = DMA_Busy;
		DMA_DCR_REG(DMA0,dma_n) |= DMA_DCR_START(DMA_addresses_data-
>Start);
		return 0;
	}
	else
	{
		return -1;
	}
}

int8_t dma_startTransfer(DMA_t dma_n)
{
	if(dma_n < DMA_COUNT && ((DMA_CurrentState[dma_n] == DMA_Ready) ||
(DMA_CurrentState[dma_n] == DMA_Complete)))
	{
		DMA_CurrentState[dma_n] = DMA_Busy;
		DMA_DCR_REG(DMA0,dma_n) |= DMA_DCR_START(1);
		return 0;
	}
	else
	{
		return -1;
	}
}

void DMA0_IRQHandler()
{
```

```
	tickEnd = profiler_getCurrentTick(tickEnd);
	__disable_irq();
	NVIC_ClearPendingIRQ(DMA0_IRQn);
	if( DMA_DSR_BCR(DMA_0) & ( DMA_DSR_BCR_CE_MASK |
DMA_DSR_BCR_BES_MASK | DMA_DSR_BCR_BED_MASK ) )
	{
		DMA_CurrentState[DMA_0] = DMA_Error;
		//logger_log(ERROR,"ERROR DMA%d. DCR:
0x%x\r\n",DMA_0,DMA_DSR_BCR(DMA_0) & (DMA_DSR_BCR_CE_MASK |
DMA_DSR_BCR_BES_MASK | DMA_DSR_BCR_BED_MASK));
	}
	else
		DMA_CurrentState[DMA_0] = DMA_Complete;

	DMA_DSR_BCR(DMA_0) |= DMA_DSR_BCR_DONE(1);
	__enable_irq();
}

void DMA1_IRQHandler()
{
	__disable_irq();
	NVIC_ClearPendingIRQ(DMA1_IRQn);
	if( DMA_DSR_BCR(DMA_1) & ( DMA_DSR_BCR_CE_MASK |
DMA_DSR_BCR_BES_MASK | DMA_DSR_BCR_BED_MASK ) )
	{
		DMA_CurrentState[DMA_1] = DMA_Error;
		logger_log(ERROR,"ERROR DMA%d. DCR:
0x%x\r\n",DMA_1,DMA_DSR_BCR(DMA_1) & (DMA_DSR_BCR_CE_MASK |
DMA_DSR_BCR_BES_MASK | DMA_DSR_BCR_BED_MASK));
	}
	else
		DMA_CurrentState[DMA_1] = DMA_Complete;

	DMA_DSR_BCR(DMA_1) |= DMA_DSR_BCR_DONE(1);
	__enable_irq();
}
void DMA2_IRQHandler()
{
	__disable_irq();
	NVIC_ClearPendingIRQ(DMA2_IRQn);
	DMA_DSR_BCR(DMA_2) |= DMA_DSR_BCR_DONE(1);
	DMA_CurrentState[DMA_2] = DMA_Complete;
	__enable_irq();
}
void DMA3_IRQHandler()
{
	__disable_irq();
	NVIC_ClearPendingIRQ(DMA3_IRQn);
	DMA_DSR_BCR(DMA_3) |= DMA_DSR_BCR_DONE(1);
	DMA_CurrentState[DMA_3] = DMA_Complete;
	__enable_irq();
}
/**
* @file - memory.c
* @brief - Implementation file for the memory functions
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 02/10/2017
```

```
**/

#include "memory.h"
#include <malloc.h>
#include "debug.h"
#ifdef PLATFORM_KL25Z
#include "dma.h"

uint8_t dma_setValue = 0;
#endif

uint8_t* my_memmove(uint8_t * src, uint8_t * dst, size_t length)
{
	uint8_t *p_ret = NULL;
	if (NULL != dst && NULL != src)
	{
//		uint8_t *p_tempMem =
(uint8_t*)malloc(sizeof(uint8_t)*length);
//		uint8_t *p_src = src;
		uint8_t *p_dst = dst;
		uint8_t *p_tempMem = src;

//		//Deep copy of the Src memory to a temp memory to handle any
memory overlap issue
		size_t tempLength = length;
//		while (tempLength--)
//		{
//			*p_tempMem = *p_src++;
//			++p_tempMem;
//		}
//
//		p_tempMem -= length;	//bringing back the pointer to point to
the start of the allocated mem
		tempLength = length;
		while (tempLength--)
		{
			*p_dst = *p_tempMem++;
			++p_dst;
		}

		//p_tempMem -= length; //bringing back the pointer to point to
the start of the allocated mem
		//free(p_tempMem);
		p_tempMem = NULL;

		p_ret = dst;
	}

	return p_ret;
}
//{
//    uint8_t *p_ret = NULL;
//    if (NULL != dst && NULL != src)
//    {
//        uint8_t *p_src = src;
//        uint8_t *p_dst = dst;
//        size_t length1=0;
//        while ((p_src!=p_dst) || (length!=length1)) {
```

```
//            ++p_src;
//            ++length1;
//        }
//        if (length1<length) {
//            p_dst+=length1;
//            while(length!=length1) {
//                *p_dst=*p_src;
//                length--;
//                p_dst++;
//                p_src++;
//            }
//        }
//        p_src=src;
//        p_dst=dst;
//        //p_tempMem -= length;  //bringing back the pointer to point to
the start of the allocated mem
//        //tempLength = length;
//        while (length1--)
//        {
//            *p_dst = *p_src;
//            ++p_dst;
//            ++p_src;
//        }
//
//        //p_tempMem -= length;  //bringing back the pointer to point to
the start of the allocated mem
//        //free(p_tempMem);
//        //p_tempMem = NULL;
//        print_memory(dst,length);
//        p_ret = dst;
//    }
//
//    return p_ret;
//}

uint8_t* my_memcpy(uint8_t * src, uint8_t * dst, size_t length)
{
      uint8_t *p_ret = NULL;
      if (NULL != dst && NULL != src)
      {
            uint8_t *p_src = src;
            uint8_t *p_dst = dst;

            //Deep copy of the Src memory to Dst memory
            size_t tempLength = length;
            while (tempLength--)
            {
                  *p_dst = *p_src++;
                  ++p_dst;
            }

            p_ret = dst;
      }

      return p_ret;
}

int8_t* my_memset(uint8_t * src, size_t length, uint8_t value)
```

```
{
	if (NULL != src)
	{
		uint8_t *p_src = src;
		size_t tempLength = length;
		while (tempLength--)
		{
			*p_src = value;
			++p_src;
		}
		p_src = NULL;
	}
	return (int8_t*)src;
}

uint8_t* my_memzero(uint8_t * src, size_t length)
{
	if (NULL != src)
	{
		uint8_t *p_src = src;
		size_t tempLength = length;
		while (tempLength--)
		{
			*p_src = 0;
			++p_src;
		}
		p_src = NULL;
	}
	return src;
}

uint8_t* my_reverse(uint8_t *src, size_t length)
{
	if (NULL != src && 0 < length)
	{
		uint8_t *forwardItr = src;
		uint8_t *backwardItr = src+length-1;

		//divide the length by two to get the midpoint value to use
it in the loop
		int itr = length >> 1;
		while (itr && forwardItr && backwardItr)
		{
			//swapping routine
			uint8_t temp = *forwardItr;
			*forwardItr = *backwardItr;
			*backwardItr = temp;
			++forwardItr; //incrementing the forward pointer
			--backwardItr; //decrementing the reverse pointer
			--itr;
		}
		forwardItr = NULL;
		backwardItr = NULL;
	}
	return src;
}

int32_t* reserve_words(size_t length)
```

```
{
      int32_t *reservedMem = (int32_t*)malloc(sizeof(int32_t)*length);
      return reservedMem;
}

void free_words(int32_t *src)
{
      free(src);
}

#ifdef PLATFORM_KL25Z

int8_t memmove_dma(uint8_t *src, uint8_t *dst, size_t length)
{
      if(src && dst )
      {
//          uint8_t *p_tempMem= (uint8_t*)malloc(sizeof(uint8_t)*length);
//          if(p_tempMem == NULL)
//                return -1;
//          //Deep copy of the Src memory to a temp memory to handle any
memory overlap issue
//          size_t itr = 0;
//          while (itr < length)
//          {
//                *(p_tempMem+itr) = *(src+itr);
//                ++itr;
//          }

            DMA_Addresses_t addresses;
            addresses.Src_Add = (uint32_t)src;
            addresses.Dest_Add = (uint32_t)dst;
            addresses.NumberOfBytes = length;
//          if((length > 3) && ((length % 4) == 0))
            if(((length+4) % 4) == 0)
            {
                  addresses.Dest_Size = DMA_32Bits;
                  addresses.Src_Size = DMA_32Bits;
            }
//          else if((length > 1) && ((length %2) == 0))
            else if(((length+2) % 2) == 0)
            {
                  addresses.Dest_Size = DMA_16Bits;
                  addresses.Src_Size = DMA_16Bits;
            }
            else
            {
                  addresses.Dest_Size = DMA_8Bits;
                  addresses.Src_Size = DMA_8Bits;
            }
            addresses.SrcAddr_Inc = 1;
            addresses.DestAddr_Inc = 1;
            addresses.Start = 1;

            return dma_initiate(DMA_0, &addresses);
      }
      return -1;

}
```

```
int8_t memset_dma(uint8_t *src, size_t length, uint8_t value)
{
	if(src)
	{
		DMA_Addresses_t addresses;
		dma_setValue = value;
		addresses.Src_Add = (uint32_t)&dma_setValue;
		addresses.Dest_Add = (uint32_t)src;
		addresses.NumberOfBytes = length;
		addresses.Dest_Size = DMA_8Bits;
		addresses.Src_Size = DMA_8Bits;
		addresses.SrcAddr_Inc = 0;
		addresses.DestAddr_Inc = 1;
		addresses.Start = 1;
		return dma_initiate(DMA_0, &addresses);
	}
	return -1;
}
#else

int8_t memmove_dma(uint8_t *src, uint8_t *dst, size_t length)
{
	my_memmove(src, dst, length);
	return 0;
}
int8_t memset_dma(uint8_t *src, size_t length, uint8_t value)
{
	my_memset(src, length, value);
	return 0;
}

#endif
/**
* @file - gpio.h
* @brief - Gives the HAL implementation for GPIO ports/pins
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 27/10/2017
**/

#include "gpio.h"

//Stores all the GPIO ports' base address
GPIO_Type * const g_GPIO_PORT[5] = {GPIOA, GPIOB, GPIOC, GPIOD, GPIOE};

PORT_Type * const g_PORT[5] = {PORTA, PORTB, PORTC, PORTD, PORTE};

void GPIO_PORTA_ENABLE()
{
	SIM_SCGC5 |= SIM_SCGC5_PORTA(1);
}

void GPIO_PORTB_ENABLE()
{
	SIM_SCGC5 |= SIM_SCGC5_PORTB(1);
}

void GPIO_PORTC_ENABLE()
```

```
{
	SIM_SCGC5 |= SIM_SCGC5_PORTC(1);
}

void GPIO_PORTD_ENABLE()
{
	SIM_SCGC5 |= SIM_SCGC5_PORTD(1);
}

void GPIO_PORTE_ENABLE()
{
	SIM_SCGC5 |= SIM_SCGC5_PORTE(1);
}

void GPIO_PORT_ENABLE(GPIO_PORT_t gpio)
{
	switch(gpio)
	{
	case 0:
		GPIO_PORTA_ENABLE();
		break;
	case 1:
		GPIO_PORTB_ENABLE();
		break;
	case 2:
		GPIO_PORTC_ENABLE();
		break;
	case 3:
		GPIO_PORTD_ENABLE();
		break;
	case 4:
		GPIO_PORTE_ENABLE();
		break;
	default:
		break;
	}
}

void GPIO_PinDir(GPIO_PORT_t gpioPort, uint8_t pin, GPIO_PORT_DIR_t dir)
{
	if(dir == gpio_output)
		g_GPIO_PORT[gpioPort]->PDDR |= (1<<pin);
	else
		g_GPIO_PORT[gpioPort]->PDDR &= ~(1<<pin);
}

void GPIO_PinOutClear(GPIO_PORT_t gpioPort, uint8_t pin)
{
	g_GPIO_PORT[gpioPort]->PCOR |= (1 << pin);
}

void GPIO_PinOutSet(GPIO_PORT_t gpioPort, uint8_t pin)
{
	g_GPIO_PORT[gpioPort]->PSOR |= (1 << pin);
}

void GPIO_PinOutToggle(GPIO_PORT_t gpioPort, uint8_t pin)
{
```

```
	g_GPIO_PORT[gpioPort]->PTOR |= (1 << pin);
}

uint8_t GPIO_PinOutGet(GPIO_PORT_t gpioPort, uint8_t pin)
{
	return (((g_GPIO_PORT[gpioPort]->PDOR) >> pin) & 1);
}

uint8_t GPIO_PinInGet(GPIO_PORT_t gpioPort, uint8_t pin)
{
	return (((g_GPIO_PORT[gpioPort]->PDIR) >> pin) & 1);
}

void GPIO_PinAltFuncSel(GPIO_PORT_t gpioPort, uint8_t pin,
GPIO_ALT_FUNC_t altFunctionSel)
{
	g_PORT[gpioPort]->PCR[pin] |= (altFunctionSel << 8);
}

void GPIO_Red_Led_En()
{
	GPIO_PORTB_ENABLE();
	GPIO_PinDir(gpioPortB,18,gpio_output);
	GPIO_PinAltFuncSel(gpioPortB,18,gpioAlt1_GPIO);
}

void GPIO_Green_Led_En()
{
	GPIO_PORTB_ENABLE();
	GPIO_PinDir(gpioPortB,19,gpio_output);
	GPIO_PinAltFuncSel(gpioPortB,19,gpioAlt1_GPIO);
}

void GPIO_Blue_Led_En()
{
	GPIO_PORTD_ENABLE();
	GPIO_PinDir(gpioPortD,1,gpio_output);
	GPIO_PinAltFuncSel(gpioPortD,1,gpioAlt1_GPIO);
}

void GPIO_Red_On()
{
	GPIO_PinOutClear(gpioPortB,18);
}
void GPIO_Red_Off()
{
	GPIO_PinOutSet(gpioPortB,18);
}
void GPIO_Red_Toggle()
{
	GPIO_PinOutToggle(gpioPortB,18);
}

void GPIO_Green_On()
{
	GPIO_PinOutClear(gpioPortB,19);
}
void GPIO_Green_Off()
```

```
{
      GPIO_PinOutSet(gpioPortB,19);
}
void GPIO_Green_Toggle()
{
      GPIO_PinOutToggle(gpioPortB,19);
}

void GPIO_Blue_On()
{
      GPIO_PinOutClear(gpioPortD,1);
}
void GPIO_Blue_Off()
{
      GPIO_PinOutSet(gpioPortD,1);
}
void GPIO_Blue_Toggle()
{
      GPIO_PinOutToggle(gpioPortD,1);
}

/**
* @file - debug.c
* @brief - Implementation file for the memory dump on stdio in DEBUG mode
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 02/10/2017
**/

#include "debug.h"
#ifdef VERBOSE
#include <stdio.h>
#endif

void print_memory(uint8_t *start, uint32_t length)
{
#ifdef VERBOSE
           uint32_t tempLength = length;
           printf("0x");
           while (tempLength--)
           {
                 printf("%x", *start++);
           }
           printf("/n");
#endif
}
/**
* @file - conversion.c
* @brief - Implementation file for the data conversion functions
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 02/10/2017
**/

#include "conversion.h"
#include "memory.h"
#include <malloc.h>
```

```
#define SIGN_MASK (0x80000000)
#define ASCII_0 (0x30)
#define ASCII_7 (0x37)
#define ASCII_9 (0x39)

/**
* @brief - (Internal function) Raises 'base' to power 'pow'
* Takes unsigned base and power to return an unsigned integer
* @param - base uint32_t
* @param - pow   uint8_t
* @return uint32_t
**/
uint32_t power(uint32_t base, uint8_t pow) {
      uint32_t op=1;
      while (pow>0) {
            op*=base;
            --pow;
      }
      return op;
}

uint8_t my_itoa(int32_t data, uint8_t * ptr, uint32_t base)
{
      uint8_t length=0;
        uint8_t * temp= (uint8_t *) malloc (sizeof(uint8_t)*33);
      uint8_t * temp_copy= temp;
        //Start with a '-', if the sign bit is set
      if (data & SIGN_MASK) {
                 *ptr='-';
                 ++ptr;
            //Perform 2's complement for the negative integer
            data=(~data)+1;
            ++length;
        }
        do {    //Repeatedly divide by base and collect the remainders in
temp
                 *temp= data%base;
                 data= data/base;
                 ++length;
                 ++temp;
        } while(data);
        --temp;
      //Reverse the order of characters in temp to obtain the actual
ASCII string
        for (uint8_t i=0;i<length;i++) {
            if (*temp>9) {
                  //ASCII conversion for letters A through F
                  *ptr=(*temp)+ASCII_7;
                  ++ptr;
            }
            else if (*temp>=0 && *temp<9) {
                  //ASCII conversion for digits 0 through 9
                  *ptr=(*temp)+ASCII_0;
                  ++ptr;
            }
                 --temp;
        }
      free(temp_copy);
```

```
	//Terminate with a null character
	  *ptr='\0';
	  return length;
}

int32_t my_atoi(uint8_t * ptr, uint8_t digits, uint32_t base)
{
	int32_t result=0;
	uint8_t neg_flag=0;
	//Set the negative flag if the ASCII string starts with '-'
	if (*ptr=='-') {
		neg_flag=1;
		--digits;
		++ptr;
	}
	while (digits>0) {
		//Reverse conversion from ASCII to Hex for A through F
		if (*ptr>ASCII_9)
			(*ptr)-=ASCII_7;
		//Reverse conversion from ASCII to Hex for 0 through 9
		else
			(*ptr)-=ASCII_0;
		result+= (int32_t)(*ptr) * (power(base,digits-1));
		--digits;
		++ptr;
	}
	//Perform 2's complement on the result if the negative flag is set
	if (neg_flag) {
		result=(~result)+1;
	}
	return result;
}

int8_t big_to_little32(uint32_t * data, uint32_t length)
{
	int8_t *p_reverse= NULL;
	int8_t p_ret = 1;
	for (uint8_t i=0;i<length;i++) {
		p_reverse = (int8_t*)my_reverse((uint8_t*)data, 4);
	}
	if(p_reverse)
		p_ret = 0;
	return p_ret;
}

int8_t little_to_big32(uint32_t * data, uint32_t length)
{
	int8_t *p_reverse= NULL;
	int8_t p_ret = 1;
	for (uint8_t i=0;i<length;i++) {
		p_reverse = (int8_t*)my_reverse((uint8_t*)data, 4);
	}
	if(p_reverse)
		p_ret = 0;
	return p_ret;
}
/*
** ###################################################################
```

```
**     Processors:          MKL25Z128FM4
**                          MKL25Z128FT4
**                          MKL25Z128LH4
**                          MKL25Z128VLK4
**
**     Compilers:           Keil ARM C/C++ Compiler
**                          Freescale C/C++ for Embedded ARM
**                          GNU C Compiler
**                          GNU C Compiler - CodeSourcery Sourcery G++
**                          IAR ANSI C/C++ Compiler for ARM
**
**     Reference manual:    KL25P80M48SF0RM, Rev.3, Sep 2012
**     Version:             rev. 2.5, 2015-02-19
**     Build:               b150220
**
**     Abstract:
**         Provides a system configuration function and a global variable
that
**         contains the system frequency. It configures the device and
initializes
**         the oscillator (PLL) that is part of the microcontroller
device.
**
**     Copyright (c) 2015 Freescale Semiconductor, Inc.
**     All rights reserved.
**
**     Redistribution and use in source and binary forms, with or without
modification,
**     are permitted provided that the following conditions are met:
**
**     o Redistributions of source code must retain the above copyright
notice, this list
**       of conditions and the following disclaimer.
**
**     o Redistributions in binary form must reproduce the above
copyright notice, this
**       list of conditions and the following disclaimer in the
documentation and/or
**       other materials provided with the distribution.
**
**     o Neither the name of Freescale Semiconductor, Inc. nor the names
of its
**       contributors may be used to endorse or promote products derived
from this
**       software without specific prior written permission.
**
**     THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS "AS IS" AND
**     ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED
**     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
**     DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
BE LIABLE FOR
**     ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES
**     (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
```

```
**     LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON
**     ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT
**     (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE OF THIS
**     SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**
**     http:                 www.freescale.com
**     mail:                 support@freescale.com
**
**     Revisions:
**     - rev. 1.0 (2012-06-13)
**         Initial version.
**     - rev. 1.1 (2012-06-21)
**         Update according to reference manual rev. 1.
**     - rev. 1.2 (2012-08-01)
**         Device type UARTLP changed to UART0.
**     - rev. 1.3 (2012-10-04)
**         Update according to reference manual rev. 3.
**     - rev. 1.4 (2012-11-22)
**         MCG module - bit LOLS in MCG_S register renamed to LOLS0.
**         NV registers - bit EZPORT_DIS in NV_FOPT register removed.
**     - rev. 1.5 (2013-04-05)
**         Changed start of doxygen comment.
**     - rev. 2.0 (2013-10-29)
**         Register accessor macros added to the memory map.
**         Symbols for Processor Expert memory map compatibility added to
the memory map.
**         Startup file for gcc has been updated according to CMSIS 3.2.
**         System initialization updated.
**     - rev. 2.1 (2014-07-16)
**         Module access macro module_BASES replaced by module_BASE_PTRS.
**         System initialization and startup updated.
**     - rev. 2.2 (2014-08-22)
**         System initialization updated - default clock config changed.
**     - rev. 2.3 (2014-08-28)
**         Update of startup files - possibility to override DefaultISR
added.
**     - rev. 2.4 (2014-10-14)
**         Interrupt INT_LPTimer renamed to INT_LPTMR0.
**     - rev. 2.5 (2015-02-19)
**         Renamed interrupt vector LLW to LLWU.
**
** ###################################################################
*/

/*!
 * @file MKL25Z4
 * @version 2.5
 * @date 2015-02-19
 * @brief Device specific configuration file for MKL25Z4 (implementation
file)
 *
 * Provides a system configuration function and a global variable that
contains
 * the system frequency. It configures the device and initializes the
oscillator
```

```
 * (PLL) that is part of the microcontroller device.
 */

#include <stdint.h>
#include "MKL25Z4.h"



/* ----------------------------------------------------------------------------
------
   -- Core clock
   ----------------------------------------------------------------------------
------ */

uint32_t SystemCoreClock = DEFAULT_SYSTEM_CLOCK;

/* ----------------------------------------------------------------------------
------
   -- SystemInit()
   ----------------------------------------------------------------------------
------ */

void SystemInit (void) {
#if (DISABLE_WDOG)
  /* SIM_COPC: COPT=0,COPCLKS=0,COPW=0 */
  SIM->COPC = (uint32_t)0x00u;
#endif /* (DISABLE_WDOG) */
#ifdef CLOCK_SETUP
  if((RCM->SRS0 & RCM_SRS0_WAKEUP_MASK) != 0x00U)
  {
    if((PMC->REGSC & PMC_REGSC_ACKISO_MASK) != 0x00U)
    {
       PMC->REGSC |= PMC_REGSC_ACKISO_MASK; /* Release hold with ACKISO:
Only has an effect if recovering from VLLSx.*/
    }
  }

  /* Power mode protection initialization */
#ifdef SYSTEM_SMC_PMPROT_VALUE
  SMC->PMPROT = SYSTEM_SMC_PMPROT_VALUE;
#endif

  /* System clock initialization */
  /* Internal reference clock trim initialization */
#if defined(SLOW_TRIM_ADDRESS)
  if ( *((uint8_t*)SLOW_TRIM_ADDRESS) != 0xFFU) {
/* Skip if non-volatile flash memory is erased */
    MCG->C3 = *((uint8_t*)SLOW_TRIM_ADDRESS);
  #endif /* defined(SLOW_TRIM_ADDRESS) */
  #if defined(SLOW_FINE_TRIM_ADDRESS)
    MCG->C4 = (MCG->C4 & ~(MCG_C4_SCFTRIM_MASK)) | ((*((uint8_t*)
SLOW_FINE_TRIM_ADDRESS)) & MCG_C4_SCFTRIM_MASK);
  #endif
  #if defined(FAST_TRIM_ADDRESS)
    MCG->C4 = (MCG->C4 & ~(MCG_C4_FCTRIM_MASK)) |((*((uint8_t*)
FAST_TRIM_ADDRESS)) & MCG_C4_FCTRIM_MASK);
  #endif
#if defined(SLOW_TRIM_ADDRESS)
```

```
  }
  #endif /* defined(SLOW_TRIM_ADDRESS) */

  /* Set system prescalers and clock sources */
  SIM->CLKDIV1 = SYSTEM_SIM_CLKDIV1_VALUE; /* Set system prescalers */
  SIM->SOPT1 = ((SIM->SOPT1) & (uint32_t)(~(SIM_SOPT1_OSC32KSEL_MASK))) |
((SYSTEM_SIM_SOPT1_VALUE) & (SIM_SOPT1_OSC32KSEL_MASK)); /* Set 32 kHz
clock source (ERCLK32K) */
  SIM->SOPT2 = ((SIM->SOPT2) & (uint32_t)(~(SIM_SOPT2_PLLFLLSEL_MASK))) |
((SYSTEM_SIM_SOPT2_VALUE) & (SIM_SOPT2_PLLFLLSEL_MASK)); /* Selects the
high frequency clock for various peripheral clocking options. */
  SIM->SOPT2 = ((SIM->SOPT2) & (uint32_t)(~(SIM_SOPT2_TPMSRC_MASK))) |
((SYSTEM_SIM_SOPT2_VALUE) & (SIM_SOPT2_TPMSRC_MASK)); /* Selects the
clock source for the TPM counter clock. */
#if ((MCG_MODE == MCG_MODE_FEI) || (MCG_MODE == MCG_MODE_FBI) ||
(MCG_MODE == MCG_MODE_BLPI))
  /* Set MCG and OSC */
#if  ((((SYSTEM_OSC0_CR_VALUE) & OSC_CR_ERCLKEN_MASK) != 0x00U) ||
(((SYSTEM_MCG_C5_VALUE) & MCG_C5_PLLCLKEN0_MASK) != 0x00U))
  /* SIM_SCGC5: PORTA=1 */
  SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK;
  /* PORTA_PCR18: ISF=0,MUX=0 */
  PORTA_PCR18 &= (uint32_t)~(uint32_t)((PORT_PCR_ISF_MASK |
PORT_PCR_MUX(0x07)));
  if (((SYSTEM_MCG_C2_VALUE) & MCG_C2_EREFS0_MASK) != 0x00U) {
  /* PORTA_PCR19: ISF=0,MUX=0 */
  PORTA_PCR19 &= (uint32_t)~(uint32_t)((PORT_PCR_ISF_MASK |
PORT_PCR_MUX(0x07)));
  }
#endif
  MCG->SC = SYSTEM_MCG_SC_VALUE;       /* Set SC (fast clock internal
reference divider) */
  MCG->C1 = SYSTEM_MCG_C1_VALUE;       /* Set C1 (clock source selection,
FLL ext. reference divider, int. reference enable etc.) */
  /* Check that the source of the FLL reference clock is the requested
one. */
  if (((SYSTEM_MCG_C1_VALUE) & MCG_C1_IREFS_MASK) != 0x00U) {
    while((MCG->S & MCG_S_IREFST_MASK) == 0x00U) {
    }
  } else {
    while((MCG->S & MCG_S_IREFST_MASK) != 0x00U) {
    }
  }
  MCG->C2 = (SYSTEM_MCG_C2_VALUE) & (uint8_t)(~(MCG_C2_LP_MASK)); /* Set
C2 (freq. range, ext. and int. reference selection etc.; low power bit is
set later) */
  MCG->C4 = ((SYSTEM_MCG_C4_VALUE) & (uint8_t)(~(MCG_C4_FCTRIM_MASK |
MCG_C4_SCFTRIM_MASK))) | (MCG->C4 & (MCG_C4_FCTRIM_MASK |
MCG_C4_SCFTRIM_MASK)); /* Set C4 (FLL output; trim values not changed) */
  OSC0->CR = SYSTEM_OSC0_CR_VALUE;     /* Set OSC_CR (OSCERCLK enable,
oscillator capacitor load) */
  #if (MCG_MODE == MCG_MODE_BLPI)
  /* BLPI specific */
  MCG->C2 |= (MCG_C2_LP_MASK);         /* Disable FLL and PLL in bypass
mode */
  #endif

#else /* MCG_MODE */
```

```
  /* Set MCG and OSC */
  /* SIM_SCGC5: PORTA=1 */
  SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK;
  /* PORTA_PCR18: ISF=0,MUX=0 */
  PORTA_PCR18 &= (uint32_t)~(uint32_t)((PORT_PCR_ISF_MASK |
PORT_PCR_MUX(0x07)));
  if (((SYSTEM_MCG_C2_VALUE) & MCG_C2_EREFS0_MASK) != 0x00U) {
  /* PORTA_PCR19: ISF=0,MUX=0 */
  PORTA_PCR19 &= (uint32_t)~(uint32_t)((PORT_PCR_ISF_MASK |
PORT_PCR_MUX(0x07)));
  }
  MCG->SC = SYSTEM_MCG_SC_VALUE;       /* Set SC (fast clock internal
reference divider) */
  MCG->C2 = (SYSTEM_MCG_C2_VALUE) & (uint8_t)(~(MCG_C2_LP_MASK)); /* Set
C2 (freq. range, ext. and int. reference selection etc.; low power bit is
set later) */
  OSC0->CR = SYSTEM_OSC0_CR_VALUE;     /* Set OSC_CR (OSCERCLK enable,
oscillator capacitor load) */
  #if (MCG_MODE == MCG_MODE_PEE)
  MCG->C1 = (SYSTEM_MCG_C1_VALUE) | MCG_C1_CLKS(0x02); /* Set C1 (clock
source selection, FLL ext. reference divider, int. reference enable etc.)
- PBE mode*/
  #else
  MCG->C1 = SYSTEM_MCG_C1_VALUE;       /* Set C1 (clock source selection,
FLL ext. reference divider, int. reference enable etc.) */
  #endif
  if (((SYSTEM_MCG_C2_VALUE) & MCG_C2_EREFS0_MASK) != 0x00U) {
    while((MCG->S & MCG_S_OSCINIT0_MASK) == 0x00U) { /* Check that the
oscillator is running */
    }
  }
  /* Check that the source of the FLL reference clock is the requested
one. */
  if (((SYSTEM_MCG_C1_VALUE) & MCG_C1_IREFS_MASK) != 0x00U) {
    while((MCG->S & MCG_S_IREFST_MASK) == 0x00U) {
    }
  } else {
    while((MCG->S & MCG_S_IREFST_MASK) != 0x00U) {
    }
  }
  MCG->C4 = ((SYSTEM_MCG_C4_VALUE)  & (uint8_t)(~(MCG_C4_FCTRIM_MASK |
MCG_C4_SCFTRIM_MASK))) | (MCG->C4 & (MCG_C4_FCTRIM_MASK |
MCG_C4_SCFTRIM_MASK)); /* Set C4 (FLL output; trim values not changed) */
#endif /* MCG_MODE */

  /* Common for all MCG modes */

  /* PLL clock can be used to generate clock for some devices regardless
of clock generator (MCGOUTCLK) mode. */
  MCG->C5 = (SYSTEM_MCG_C5_VALUE) & (uint8_t)(~(MCG_C5_PLLCLKEN0_MASK));
/* Set C5 (PLL settings, PLL reference divider etc.) */
  MCG->C6 = (SYSTEM_MCG_C6_VALUE) & (uint8_t)~(MCG_C6_PLLS_MASK); /* Set
C6 (PLL select, VCO divider etc.) */
  if ((SYSTEM_MCG_C5_VALUE) & MCG_C5_PLLCLKEN0_MASK) {
    MCG->C5 |= MCG_C5_PLLCLKEN0_MASK;  /* PLL clock enable in mode other
than PEE or PBE */
  }
  /* BLPE, PEE and PBE MCG mode specific */
```

```
#if (MCG_MODE == MCG_MODE_BLPE)
  MCG->C2 |= (MCG_C2_LP_MASK);          /* Disable FLL and PLL in bypass
mode */
#elif ((MCG_MODE == MCG_MODE_PBE) || (MCG_MODE == MCG_MODE_PEE))
  MCG->C6 |= (MCG_C6_PLLS_MASK);        /* Set C6 (PLL select, VCO divider
etc.) */
  while((MCG->S & MCG_S_LOCK0_MASK) == 0x00U) { /* Wait until PLL is
locked*/
  }
  #if (MCG_MODE == MCG_MODE_PEE)
  MCG->C1 &= (uint8_t)~(MCG_C1_CLKS_MASK);
  #endif
#endif
#if ((MCG_MODE == MCG_MODE_FEI) || (MCG_MODE == MCG_MODE_FEE))
  while((MCG->S & MCG_S_CLKST_MASK) != 0x00U) { /* Wait until output of
the FLL is selected */
  }
  /* Use LPTMR to wait for 1ms for FLL clock stabilization */
  SIM_SCGC5 |= SIM_SCGC5_LPTMR_MASK;   /* Alow software control of LPMTR
*/
  LPTMR0->CMR = LPTMR_CMR_COMPARE(0);  /* Default 1 LPO tick */
  LPTMR0->CSR = (LPTMR_CSR_TCF_MASK | LPTMR_CSR_TPS(0x00));
  LPTMR0->PSR = (LPTMR_PSR_PCS(0x01) | LPTMR_PSR_PBYP_MASK); /* Clock
source: LPO, Prescaler bypass enable */
  LPTMR0->CSR = LPTMR_CSR_TEN_MASK;    /* LPMTR enable */
  while((LPTMR0_CSR & LPTMR_CSR_TCF_MASK) == 0u) {
  }
  LPTMR0_CSR = 0x00;                   /* Disable LPTMR */
  SIM_SCGC5 &= (uint32_t)~(uint32_t)SIM_SCGC5_LPTMR_MASK;
#elif ((MCG_MODE == MCG_MODE_FBI) || (MCG_MODE == MCG_MODE_BLPI))
  while((MCG->S & MCG_S_CLKST_MASK) != 0x04U) { /* Wait until internal
reference clock is selected as MCG output */
  }
#elif ((MCG_MODE == MCG_MODE_FBE) || (MCG_MODE == MCG_MODE_PBE) ||
(MCG_MODE == MCG_MODE_BLPE))
  while((MCG->S & MCG_S_CLKST_MASK) != 0x08U) { /* Wait until external
reference clock is selected as MCG output */
  }
#elif (MCG_MODE == MCG_MODE_PEE)
  while((MCG->S & MCG_S_CLKST_MASK) != 0x0CU) { /* Wait until output of
the PLL is selected */
  }
#endif
#if (((SYSTEM_SMC_PMCTRL_VALUE) & SMC_PMCTRL_RUNM_MASK) == (0x02U <<
SMC_PMCTRL_RUNM_SHIFT))
  SMC->PMCTRL = (uint8_t)((SYSTEM_SMC_PMCTRL_VALUE) &
(SMC_PMCTRL_RUNM_MASK)); /* Enable VLPR mode */
  while(SMC->PMSTAT != 0x04U) {        /* Wait until the system is in
VLPR mode */
  }
#endif

  /* PLL loss of lock interrupt request initialization */
  if (((SYSTEM_MCG_C6_VALUE) & MCG_C6_LOLIE0_MASK) != 0U) {
    NVIC_EnableIRQ(MCG_IRQn);          /* Enable PLL loss of lock
interrupt request */
  }
```

```
#endif
}

/* ----------------------------------------------------------------------------
------
   -- SystemCoreClockUpdate()
   ----------------------------------------------------------------------------
------ */

void SystemCoreClockUpdate (void) {
  uint32_t MCGOUTClock;                /* Variable to store output clock
frequency of the MCG module */
  uint16_t Divider;

  if ((MCG->C1 & MCG_C1_CLKS_MASK) == 0x00U) {
    /* Output of FLL or PLL is selected */
    if ((MCG->C6 & MCG_C6_PLLS_MASK) == 0x00U) {
      /* FLL is selected */
      if ((MCG->C1 & MCG_C1_IREFS_MASK) == 0x00U) {
        /* External reference clock is selected */
        MCGOUTClock = CPU_XTAL_CLK_HZ; /* System oscillator drives MCG
clock */
        if ((MCG->C2 & MCG_C2_RANGE0_MASK) != 0x00U) {
          switch (MCG->C1 & MCG_C1_FRDIV_MASK) {
          case 0x38U:
            Divider = 1536U;
            break;
          case 0x30U:
            Divider = 1280U;
            break;
          default:
            Divider = (uint16_t)(32LU << ((MCG->C1 & MCG_C1_FRDIV_MASK)
>> MCG_C1_FRDIV_SHIFT));
            break;
          }
        } else {/* ((MCG->C2 & MCG_C2_RANGE_MASK) != 0x00U) */
          Divider = (uint16_t)(1LU << ((MCG->C1 & MCG_C1_FRDIV_MASK) >>
MCG_C1_FRDIV_SHIFT));
        }
        MCGOUTClock = (MCGOUTClock / Divider); /* Calculate the divided
FLL reference clock */
      } else { /* (!((MCG->C1 & MCG_C1_IREFS_MASK) == 0x00U)) */
        MCGOUTClock = CPU_INT_SLOW_CLK_HZ; /* The slow internal reference
clock is selected */
      } /* (!((MCG->C1 & MCG_C1_IREFS_MASK) == 0x00U)) */
      /* Select correct multiplier to calculate the MCG output clock  */
      switch (MCG->C4 & (MCG_C4_DMX32_MASK | MCG_C4_DRST_DRS_MASK)) {
        case 0x00U:
          MCGOUTClock *= 640U;
          break;
        case 0x20U:
          MCGOUTClock *= 1280U;
          break;
        case 0x40U:
          MCGOUTClock *= 1920U;
          break;
        case 0x60U:
          MCGOUTClock *= 2560U;
```

```
          break;
        case 0x80U:
          MCGOUTClock *= 732U;
          break;
        case 0xA0U:
          MCGOUTClock *= 1464U;
          break;
        case 0xC0U:
          MCGOUTClock *= 2197U;
          break;
        case 0xE0U:
          MCGOUTClock *= 2929U;
          break;
        default:
          break;
      }
    } else { /* (!((MCG->C6 & MCG_C6_PLLS_MASK) == 0x00U)) */
      /* PLL is selected */
      Divider = (((uint16_t)MCG->C5 & MCG_C5_PRDIV0_MASK) + 0x01U);
      MCGOUTClock = (uint32_t)(CPU_XTAL_CLK_HZ / Divider); /* Calculate
the PLL reference clock */
      Divider = (((uint16_t)MCG->C6 & MCG_C6_VDIV0_MASK) + 24U);
      MCGOUTClock *= Divider;          /* Calculate the MCG output clock
*/
    } /* (!((MCG->C6 & MCG_C6_PLLS_MASK) == 0x00U)) */
  } else if ((MCG->C1 & MCG_C1_CLKS_MASK) == 0x40U) {
    /* Internal reference clock is selected */
    if ((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U) {
      MCGOUTClock = CPU_INT_SLOW_CLK_HZ; /* Slow internal reference clock
selected */
    } else { /* (!((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U)) */
      Divider = (uint16_t)(0x01LU << ((MCG->SC & MCG_SC_FCRDIV_MASK) >>
MCG_SC_FCRDIV_SHIFT));
      MCGOUTClock = (uint32_t) (CPU_INT_FAST_CLK_HZ / Divider); /* Fast
internal reference clock selected */
    } /* (!((MCG->C2 & MCG_C2_IRCS_MASK) == 0x00U)) */
  } else if ((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U) {
    /* External reference clock is selected */
    MCGOUTClock = CPU_XTAL_CLK_HZ;     /* System oscillator drives MCG
clock */
  } else { /* (!((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U)) */
    /* Reserved value */
    return;
  } /* (!((MCG->C1 & MCG_C1_CLKS_MASK) == 0x80U)) */
  SystemCoreClock = (MCGOUTClock / (0x01U + ((SIM->CLKDIV1 &
SIM_CLKDIV1_OUTDIV1_MASK) >> SIM_CLKDIV1_OUTDIV1_SHIFT)));
}
/**
* @file - circular_buffer.c
* @brief - Implementation file for the circular buffer functionalities
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 27/10/2017
**/

#include "circular_buffer.h"
#include "memory.h"
#include <malloc.h>
```

```
#define CB_STATUS_CODE_COUNT 10

const uint8_t * const CB_Status_Strings[CB_STATUS_CODE_COUNT] = {
(uint8_t *)"Operation Successful",
(uint8_t *)"Buffer is Empty",
(uint8_t *)"Buffer has some data",
(uint8_t *)"Buffer is Full",
(uint8_t *)"Buffer is not Full",
(uint8_t *)"Some NULL pointer Error",
(uint8_t *)"Buffer is not allocated memory",
(uint8_t *)"Buffer allocation failure",
};

CB_Status_t CB_init(CB_t *cbuffer, uint16_t length)
{
	CB_Status_t returnStatus = CB_SUCCESS;

	if (NULL == cbuffer)
		returnStatus = CB_NULL_POINTER_ERROR;
	else
	{
		cbuffer->buffer =
(bufferElement_t*)malloc(sizeof(bufferElement_t)*length);
		if (NULL == cbuffer->buffer)
		{
			cbuffer->size = 0;
			cbuffer->head = NULL;
			cbuffer->tail = NULL;
			cbuffer->count = 0;
			returnStatus = CB_BUFFER_ALLOCATION_FAILURE;
		}
		else
		{
			cbuffer->size = length;
			cbuffer->head = cbuffer->buffer;
			cbuffer->tail = cbuffer->buffer;
			cbuffer->count = 0;
			my_memzero(cbuffer->buffer, length);
		}
	}

	return returnStatus;
}


CB_Status_t CB_buffer_add_item(CB_t *cbuffer, bufferElement_t dataToAdd)
{
	CB_Status_t returnStatus = CB_is_full(cbuffer);

	if (CB_BUFFER_FULL == returnStatus || CB_BUFFER_NOT_ALLOCATED ==
returnStatus || CB_NULL_POINTER_ERROR == returnStatus)
		return returnStatus;
	else
	{
		*(cbuffer->head) = dataToAdd;
		cbuffer->count++;
```

```
		if ((cbuffer->head - cbuffer->buffer) < (cbuffer->size - 1))
	//there is still buffer location empty in the buffer, so we can
move the head to the next buffer location
			cbuffer->head++;
		else
			cbuffer->head = cbuffer->buffer;  //rolling back the
head to the front of the buffer

		returnStatus = CB_SUCCESS;
	}
	return returnStatus;
}

CB_Status_t CB_buffer_remove_item(CB_t *cbuffer, bufferElement_t
*outData)
{
	CB_Status_t returnStatus = CB_is_empty(cbuffer);

	if (CB_BUFFER_EMPTY == returnStatus || CB_BUFFER_NOT_ALLOCATED ==
returnStatus || CB_NULL_POINTER_ERROR == returnStatus)
		return returnStatus;
	else
	{
		*outData = *(cbuffer->tail);
		*(cbuffer->tail) = 0;
		(cbuffer->count)--;
		if ((cbuffer->tail - cbuffer->buffer) < (cbuffer->size - 1))
	//there is still buffer location empty in the buffer, so we can
move the tail to the next buffer location
			cbuffer->tail++;
		else
			cbuffer->tail = cbuffer->buffer;  //rolling back the
tail to the front of the buffer

		returnStatus = CB_SUCCESS;
	}
	return returnStatus;
}

CB_Status_t CB_peek(CB_t * cbuffer, uint16_t position, bufferElement_t
*outPeekData)
{
	CB_Status_t returnStatus = CB_is_empty(cbuffer);

	if (CB_BUFFER_EMPTY == returnStatus || CB_BUFFER_NOT_ALLOCATED ==
returnStatus || CB_NULL_POINTER_ERROR == returnStatus)
		return returnStatus;
	else
	{	/*Since the position right next to head is supposed to be
empty, this function implementation will navigate 'position'
		  no. of items before head to do a peek*/
		if (cbuffer->count < position)	//Can't move through more
than 'count' positions
			returnStatus = CB_NULL_POINTER_ERROR;
		else if ((cbuffer->head - cbuffer->buffer)>(cbuffer->count))
{	//No need to wrap around
			*outPeekData=*(cbuffer->head - position);
			returnStatus = CB_SUCCESS;
```

```
            }
            else {      //Need to wrap around to move through 'position'
no. of items
                  *outPeekData= *(cbuffer->head + (cbuffer->size -
position +1));
                  returnStatus = CB_SUCCESS;
            }
      }
      return returnStatus;
}

CB_Status_t CB_is_full(CB_t *cbuffer)
{
      CB_Status_t returnStatus = CB_BUFFER_NOT_FULL;

      if (NULL == cbuffer)
            returnStatus = CB_NULL_POINTER_ERROR;
      else if (NULL == cbuffer->buffer)
            returnStatus = CB_BUFFER_NOT_ALLOCATED;
      else
      {
            if (cbuffer->size == cbuffer->count)
                  returnStatus = CB_BUFFER_FULL;
      }

      return returnStatus;
}

CB_Status_t CB_is_empty(CB_t *cbuffer)
{
      CB_Status_t returnStatus = CB_BUFFER_NOT_EMPTY;

      if (NULL == cbuffer)
            returnStatus = CB_NULL_POINTER_ERROR;
      else if (NULL == cbuffer->buffer)
            returnStatus = CB_BUFFER_NOT_ALLOCATED;
      else
      {
            if (0 == cbuffer->count)
                  returnStatus = CB_BUFFER_EMPTY;
      }

      return returnStatus;
}

CB_Status_t CB_destroy(CB_t *cbuffer)
{
      CB_Status_t returnStatus = CB_SUCCESS;

      if (NULL == cbuffer)
            returnStatus = CB_NULL_POINTER_ERROR;
      else
      {
            if (NULL != cbuffer->buffer)
            {
                  free(cbuffer->buffer);
                  cbuffer->buffer = NULL;
            }
```

```
		cbuffer->size = 0;
		cbuffer->head = NULL;
		cbuffer->tail = NULL;
		cbuffer->count = 0;
	}

	return returnStatus;
}


const uint8_t* get_CB_error_String(CB_Status_t cbStatusEnum)
{
	return CB_Status_Strings[cbStatusEnum];
}
#include "MKL25Z4.h"
#include "timer0.h"
#include "uart0.h"
#include "gpio.h"
#include "conversion.h"
#include "memory.h"

#define TPM0_CLKSRC_DISABLE			(0)
#define TPM0_CLKSRC_MCGFLL_PLLBY2	(1)
#define TPM0_CLKSRC_OSCERCLK		(2)
#define TPM0_CLKSRC_MCGIRCCLK		(3)

#define TPM0_CLK_GATE_EN			(1)
#define TPM0_CLK_GATE_DIS			(0)

/*
000 Divide by 1
001 Divide by 2
010 Divide by 4
011 Divide by 8
100 Divide by 16
101 Divide by 32
110 Divide by 64
111 Divide by 128
*/
#define TPM0_CLK_PRES_1					(0)
#define TPM0_CLK_PRES_2					(1)
#define TPM0_CLK_PRES_4					(2)
#define TPM0_CLK_PRES_64			(6)
#define TPM0_CLK_PRES_128			(7)




//volatile uint32_t systick1 = 0;
//volatile uint32_t systick2 = 0;
//volatile uint8_t flag = 0;

void timer0_clockInit()
{
	//selecting clk source
	SIM->SOPT2 |= SIM_SOPT2_TPMSRC(TPM0_CLKSRC_MCGFLL_PLLBY2);

	//enabling TPM0 clock gate
```

```
	SIM->SCGC6 |= SIM_SCGC6_TPM0(TPM0_CLK_GATE_EN);
}

void timer0_configure()
{
	timer0_clockInit();

	TPM0->SC |= TPM_SC_TOIE(1);

	//count up mode
	TPM0->SC |= TPM_SC_CPWMS(0);

	//Prescaler of 64
	TPM0->SC |= TPM_SC_PS(TPM0_CLK_PRES_128);

	//Clock Freq - 47939584 Hz, Timer0 Pres - 64, Time wanted = 50ms
	//Count = (47939584 / 128) * (174/1000);
	TPM0->MOD = 65535;

	//software compare
	TPM0->CONTROLS->CnSC |= TPM_CnSC_MSA(1) | TPM_CnSC_MSB(1);

	//counter inc every LPTPM counter clock
	TPM0->SC |= TPM_SC_CMOD(1);

	NVIC_EnableIRQ(TPM0_IRQn);

}

void TPM0_IRQHandler()
{
	__disable_irq();
	if((TPM0->SC & TPM_SC_TOF_MASK))
	{
		TPM0->SC |= TPM_SC_TOF(1);
		GPIO_Red_Toggle();
	}
	__enable_irq();
}
/*
 * data_processing.c
 *
 *  Created on: 05-Dec-2017
 *      Author: Gunj Manseta
 */

#include "logger.h"
#include <malloc.h>
#include "conversion.h"
#include "data_processing.h"

#ifndef __STATIC_INLINE
#define __STATIC_INLINE static inline
#endif

#define ALPHA_UPPER_START    (0x40)
#define ALPHA_UPPER_END      (0x5B)
#define ALPHA_LOWER_START    (0X60)
```

```
#define ALPHA_LOWER_END      (0X7B)
#define NUM_START                 (0X2F)
#define NUM_END              (0X3A)

#define MAX_STATISTICS_DATA       (64)
#define TYPES_OF_STATISTICS_DATA  (4)

__STATIC_INLINE uint8_t is_alphabet(uint8_t val)
{
      if ((val>ALPHA_UPPER_START && val<ALPHA_UPPER_END) ||
(val>ALPHA_LOWER_START && val<ALPHA_LOWER_END))
            return 1;
      else
            return 0;
}
__STATIC_INLINE uint8_t is_numeric(uint8_t val)
{
      if (val>NUM_START && val<NUM_END)
            return 1;
      else
            return 0;
}
__STATIC_INLINE uint8_t is_punctuation(uint8_t val)
{
      switch ((unsigned char)val)
      {
          case '.':
          case '\'':
          case '\"':
          case ':':
          case ';':
          case ',':
          case '?':
          case '!': return 1;
          default: return 0;
      }
}

void processData(CB_t *RXBuffer)
{
      logger_log(DATA_ANALYSIS_STARTED,NULL);

      CB_Status_t status = CB_is_empty(RXBuffer);
      if(CB_BUFFER_EMPTY == status || CB_NULL_POINTER_ERROR == status)
      {
            return;
      }

      uint8_t alphabets_count           = 0;
      uint8_t numerics_count                 = 0;
      uint8_t punctuations_count        = 0;
      uint8_t miscChar_count            = 0;
      uint8_t itr                            = 0;
      uint8_t currentChar                    = 0;

      while(CB_SUCCESS == CB_buffer_remove_item(RXBuffer,&currentChar) &&
(itr < MAX_STATISTICS_DATA))
      {
```

```
		if(is_alphabet(currentChar))
		{
			alphabets_count++;
		}
		else if(is_numeric(currentChar))
		{
			numerics_count++;
		}
		else if(is_punctuation(currentChar))
		{
			punctuations_count++;
		}
		else
		{
			miscChar_count++;
		}

	}

	logger_log(DATA_ALPHA_COUNT,"%d",alphabets_count);
	logger_log(DATA_NUMERIC_COUNT,"%d",numerics_count);
	logger_log(DATA_PUNCTUATION_COUNT,"%d",punctuations_count);
	logger_log(DATA_MISC_COUNT,"%d",miscChar_count);

	logger_log(DATA_ANALYSIS_COMPLETED,"");
}
/*
 * logger.c
 *
 *  Created on: 04-Dec-2017
 *      Author: Gunj Manseta
 */

#include "logger.h"
#include "logger_helper.h"
#include <stdarg.h>
#include <stdint.h>
#include <malloc.h>
#include <strings.h>
#include <stdio.h>

#ifdef VERBOSE
uint8_t verbose_flag = 1;
#else
uint8_t verbose_flag = 0;
#endif

volatile uint8_t logging = 1;
volatile LOG_FORMAT_t log_format = ASCII_LOGGER;

void logger_log(LOG_ID_t log_id, char *fmt, ...)
{
	if(logging && verbose_flag)
	{
		va_list args;
		va_start(args, fmt);
		char *payload =(char*)malloc(100);
```

```
		uint32_t len = vsnprintf(payload,100,fmt, args);
		va_end(args);
		log_t *log_struct = log_vector[log_id](payload,len+1);
	//len+1 because the payload is null terminated string
		log_item(log_struct, log_format);
		free(payload);
		free(log_struct->payload);
		free(log_struct);
	}
}
/*
 * timestamp.c
 *
 *  Created on: 05-Dec-2017
 *      Author: Gunj Manseta
 */

#include "timestamp.h"
#include "logger.h"
#include "time.h"

#ifdef PLATFORM_KL25Z

#include "MKL25Z4.h"
#include "gpio.h"

#define G_Current_Time RTC_TSR

void rtc_init()
{
	__disable_irq();
	NVIC_DisableIRQ(RTC_IRQn);
	NVIC_DisableIRQ(RTC_Seconds_IRQn);

	SIM->SOPT1 &= ~(SIM_SOPT1_OSC32KSEL(3));
	SIM->SOPT1 |= SIM_SOPT1_OSC32KSEL(3);

	//Enable RTC Access control and interrupts
	SIM->SCGC6|= SIM_SCGC6_RTC(1);

	RTC_CR = RTC_CR_SWR_MASK;
	RTC_CR&= ~RTC_CR_SWR_MASK;

	//Clear RTC interrupts
	//RTC->IER = 0x00;

	//Remove locks on Control, Status and Lock register
	RTC->LR|=RTC_LR_LRL(1) | RTC_LR_CRL(1) | RTC_LR_SRL(1);

	//Enable writing to registers in non-supervisor mode
	RTC->CR|=RTC_CR_SUP(1);

	//32.768 kHz oscillator is enabled
	//RTC->CR &= ~RTC_CR_OSCE(1);
	//RTC->CR|=RTC_CR_OSCE(1);

	//Disable counter, load Seconds and Prescalar registers and enable
the counter again
```

```
	RTC_SR &= ~RTC_SR_TCE(1);
	RTC_TSR= BUILD_EPOCH_TIME;
	RTC_TPR|= RTC_TPR_TPR(0x7BFF);
	RTC_SR |=RTC_SR_TCE(1);

	RTC->IER |= RTC_IER_TSIE(1) | RTC_IER_TOIE(1);

//	NVIC_ClearPendingIRQ(RTC_IRQn);
	NVIC_ClearPendingIRQ(RTC_Seconds_IRQn);
//	NVIC_EnableIRQ(RTC_IRQn);
	NVIC_EnableIRQ(RTC_Seconds_IRQn);
	__enable_irq();

}

void RTC_IRQHandler()
{
	logger_log(INFO,"RTC_IRQHandler");
}

void RTC_Seconds_IRQHandler()
{
	__disable_irq();
	NVIC_ClearPendingIRQ(RTC_IRQn);
	logger_log(HEARTBEAT,"");
	GPIO_Red_Toggle();
	RTC_SR &= ~RTC_SR_TCE(1);
	RTC->TPR |= RTC_TPR_TPR(0x7BFF);
	RTC_SR |= RTC_SR_TCE(1);
	__enable_irq();
}

#else

#define G_Current_Time time(NULL)

#endif

char* getcurrentTimeStampString()
{
	time_t t= G_Current_Time;
	if(t > 0)
	{
		char *timeStamp_string = ctime(&t);
		return timeStamp_string;
	}
	else
		return NULL;
}

char* getString_of_TimeStamp(time_t epochTime)
{
	if(epochTime > 0)
	{
		char *timeStamp_string = ctime(&epochTime);
		return timeStamp_string;
	}
	else
```

```
            return NULL;
}

uint32_t getTimeStamp()
{
      return G_Current_Time;

}
/*
 * spi.c
 *
 *  Created on: Dec 1, 2017
 *      Author: ashis
 */

#include "spi.h"
#include "uart0.h"

SPI_Type *SPI[2] = {SPI0, SPI1};

void SPI_GPIO_init(SPI_t spi) {
      if(spi==SPI_0) {
            GPIO_PORTD_ENABLE();
            //Set SCK, MOSI and MISO pins for SPI functionality
            GPIO_PinAltFuncSel(gpioPortD,1,gpioAlt2);
            GPIO_PinAltFuncSel(gpioPortD,2,gpioAlt2);
            GPIO_PinAltFuncSel(gpioPortD,3,gpioAlt2);
            //Set the pin for Chip Selection logic as GPIO
            GPIO_PinAltFuncSel(gpioPortD,0,gpioAlt1_GPIO);
            GPIO_PinDir(gpioPortD,0,gpio_output);
      }
}

void SPI_disable()
{

      GPIO_PinAltFuncSel(gpioPortD,1,gpioAlt1_GPIO);
      GPIO_PinOutClear(gpioPortD,1);
      GPIO_PinAltFuncSel(gpioPortD,1,gpioAlt0_Disabled);
      GPIO_PinAltFuncSel(gpioPortD,2,gpioAlt0_Disabled);
      GPIO_PinAltFuncSel(gpioPortD,3,gpioAlt0_Disabled);
      GPIO_PinAltFuncSel(gpioPortD,0,gpioAlt0_Disabled);

}

void SPI_clock_init(SPI_t spi) {
      if(spi==SPI_0)
            SIM->SCGC4|= SIM_SCGC4_SPI0(1);
      else if (spi==SPI_1)
            SIM->SCGC4|= SIM_SCGC4_SPI1(1);
}

void SPI_init(SPI_t spi) {

      SPI_clock_init(spi);

      SPI_GPIO_init(spi);
```

```
	//Enable SPI Interrupt and Transmit interrupt
	//SPI[spi]->C1|= SPI_C1_SPIE(1) | SPI_C1_SPTIE(1);

	//Configure the device as Master
	SPI[spi]->C1|= SPI_C1_MSTR(1);

	//Idle low and be active on the rising edge
	SPI[spi]->C1|= SPI_C1_CPOL(0);

	//Configure for MSB first
	SPI[spi]->C1|= SPI_C1_LSBFE(0);

	SPI[spi]->C1 &= ~(SPI_C1_CPHA(1));
	//Slave Select Output Enable
	//SPI[spi]->C1|= SPI_C1_SSOE(1);

	//Set the Baud Rate Prescalar as 1 and the Baud Rate Divisor as 4
	//For Bus clock at 20 MHz, the frequency would be 5MHz
	SPI[spi]->BR|= SPI_BR_SPPR(0) | SPI_BR_SPR(2);

	//Master mode-fault function enable: To make the SS pin act as
Slave Select output
	//SPI[spi]->C2|= SPI_C2_MODFEN(1);

	//SPI System Enable
	SPI[spi]->C1|= SPI_C1_SPE(1);

}

uint8_t SPI_read_byte(SPI_t spi) {

	while ((SPI[spi]->S & SPI_S_SPRF_MASK) == 0);
	return SPI[spi]->D;
}

void SPI_write_byte(SPI_t spi, uint8_t byte) {

	SPI_flush(spi);
	SPI[spi]->D = byte;
	SPI_flush(spi);
}

void SPI_write_packet(SPI_t spi, uint8_t* p, size_t length) {
	uint8_t i=0;
	while (i<length) {
		SPI_write_byte(spi, *(p+i));
		++i;
	}
}

void SPI_read_packet(SPI_t spi, uint8_t* p, size_t length) {
	uint8_t i=0;
	while (i<length) {
		*(p+i) = SPI_read_byte(spi);
		++i;
	}
}
```

```
void SPI0_IRQHandler() {
//    if (SPI0_S & SPI_S_SPRF_MASK) {
//          recd= SPI0_D;
//    }
      __disable_irq();



      uint8_t c = 'I';
      UART0_send(&c);
      __enable_irq();
}
/*
 * logger_helper.c
 *
 *  Created on: 05-Dec-2017
 *      Author: Gunj Manseta
 */


#include "logger_helper.h"
#include <stdarg.h>
#include <stdint.h>
#include <malloc.h>
#include <stdio.h>
#include <string.h>
#include "timestamp.h"

const char* const LOG_ID_Strings[LOG_TYPE_NUM] = {
(const char *)"<HEARTBEAT>",
(const char *)"<LOGGER_INITIALZED>",
(const char *)"<GPIO_INITIALZED>",
(const char *)"<SYSTEM_INITIALIZED>",
(const char *)"<SYSTEM_HALTED>",
(const char *)"<INFO>",
(const char *)"<WARNING>",
(const char *)"<ERROR>",
(const char *)"<PROFILING_STARTED>",
(const char *)"<PROFILING_RESULT>",
(const char *)"<PROFILING_COMPLETED>",
(const char *)"<DATA_RECEIVED>",
(const char *)"<DATA_ANALYSIS_STARTED>",
(const char *)"<DATA_ALPHA_COUNT>",
(const char *)"<DATA_NUMERIC_COUNT>",
(const char *)"<DATA_PUNCTUATION_COUNT>",
(const char *)"<DATA_MISC_COUNT>",
(const char *)"<DATA_ANALYSIS_COMPLETED>",
};

log_t* log_hearbeat(char* payload, uint32_t len);
log_t* log_logger_initialized(char* payload, uint32_t len);
log_t* log_gpio_initialized(char* payload, uint32_t len);
log_t* log_system_initialized(char* payload, uint32_t len);
log_t* log_system_halted(char* payload, uint32_t len);
log_t* log_info(char* payload, uint32_t len);
log_t* log_warning(char* payload, uint32_t len);
log_t* log_error(char* payload, uint32_t len);
log_t* log_profiling_started(char* payload, uint32_t len);
```

```
log_t* log_profiling_result(char* payload, uint32_t len);
log_t* log_profiling_completed(char* payload, uint32_t len);
log_t* log_data_received(char* payload, uint32_t len);
log_t* log_data_analysis_started(char* payload, uint32_t len);
log_t* log_data_alpha_count(char* payload, uint32_t len);
log_t* log_data_numeric_count(char* payload, uint32_t len);
log_t* log_data_punctuation_count(char* payload, uint32_t len);
log_t* log_data_misc_count(char* payload, uint32_t len);
log_t* log_data_analysis_completed(char* payload, uint32_t len);

//function pointer tables to handle specific log events
log_t* (*const log_vector[LOG_TYPE_NUM])(char*,uint32_t) = {

		log_hearbeat,
		log_logger_initialized,
		log_gpio_initialized,
		log_system_initialized,
		log_system_halted,
		log_info,
		log_warning,
		log_error,
		log_profiling_started,
		log_profiling_result,
		log_profiling_completed,
		log_data_received,
		log_data_analysis_started,
		log_data_alpha_count,
		log_data_numeric_count,
		log_data_punctuation_count,
		log_data_misc_count,
		log_data_analysis_completed
};


uint8_t getChecksum(log_t* log_item, size_t log_item_size)
{
	//(in bytes)log_item_size - payload_pointer_size - checksum_size
	size_t data_size = log_item_size - 4 - 4;
	size_t payload_size = log_item->payloadSize;
	uint8_t checksum = 0;
	size_t itr = 0;
	while(itr < data_size)
	{
		checksum ^= ((uint8_t*)log_item)[itr++];
	}
	itr = 0;
	while(itr < payload_size)
	{
		checksum ^= *(log_item->payload+itr);
		itr++;
	}
	//srand(checksum);
	//return rand();
	return checksum;
}

log_t* log_hearbeat(char* payload, uint32_t len)
{
```

```
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = HEARTBEAT;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_logger_initialized(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = LOGGER_INITIALZED;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_gpio_initialized(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = GPIO_INITIALZED;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_system_initialized(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = SYSTEM_INITIALIZED;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_system_halted(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = SYSTEM_HALTED;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_info(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = INFO;
	log_item->payload = (uint8_t*)malloc(len);
```

```
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}
log_t* log_warning(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = WARNING;
	log_item->payload = (uint8_t*)malloc(sizeof(uint8_t)*len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_error(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = ERROR;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_profiling_started(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = PROFILING_STARTED;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}
log_t* log_profiling_result(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = PROFILING_RESULT;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}
log_t* log_profiling_completed(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = PROFILING_COMPLETED;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
```

```
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_data_received(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_RECEIVED;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}

log_t* log_data_analysis_started(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_ANALYSIS_STARTED;
	log_item->payload = NULL;
	log_item->payloadSize = 0;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;

}
log_t* log_data_alpha_count(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_ALPHA_COUNT;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;
}
log_t* log_data_numeric_count(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_NUMERIC_COUNT;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;

}
log_t* log_data_punctuation_count(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_PUNCTUATION_COUNT;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
```

```
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;

}
log_t* log_data_misc_count(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_MISC_COUNT;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;

}
log_t* log_data_analysis_completed(char* payload, uint32_t len)
{
	log_t *log_item = (log_t*)malloc(sizeof(log_t));
	log_item->logId = DATA_ANALYSIS_COMPLETED;
	log_item->payload = (uint8_t*)malloc(len);
	log_item->payload = memmove(log_item->payload,payload,len);
	log_item->payloadSize = len;
	log_item->timeStamp = getTimeStamp();
	log_item->checksum = getChecksum(log_item, sizeof(*log_item));
	return log_item;

}
const char* get_LOG_ID_String(LOG_ID_t log_id)
{
	return LOG_ID_Strings[log_id];
}

void log_binary(log_t *log_item)
{
	if(log_item)
	{
		LOG_RAW_DATA((uint8_t*)log_item,4);     //sending LOG_ID
		LOG_RAW_DATA(((uint8_t*)log_item)+4,4); //sending TIMESTAMP
		LOG_RAW_DATA(((uint8_t*)log_item)+8,4); //sending PAYLOAD LEN

	LOG_RAW_DATA((uint8_t*)*(uint32_t*)(((uint8_t*)log_item)+12),log_it
em->payloadSize);//sending PAYLOAD
		LOG_RAW_DATA(((uint8_t*)log_item)+16,1);      //sending
CHECKSUM
	}
	else
		LOG_RAW_STRING("<INTERNAL ERROR> Log_item not
found.\r\n\r\n");
}

void log_ascii(log_t *log_item)
{
	if(log_item)
	{
		LOG_RAW_INT(log_item->timeStamp);
```

```
//          LOG_RAW_STRING(getString_of_TimeStamp(log_item->timeStamp));
            LOG_RAW_STRING("\t");
            LOG_RAW_INT(log_item->logId);
            LOG_RAW_STRING(":");
            LOG_RAW_STRING(get_LOG_ID_String(log_item->logId));
            LOG_RAW_STRING(" ");
            LOG_RAW_DATA(log_item->payload,log_item->payloadSize);
//          LOG_RAW_STRING(" CS: ");
//          LOG_RAW_INT(log_item->checksum);
            LOG_RAW_STRING("\r\n\r\n");
      }
      else
            LOG_RAW_STRING("<INTERNAL ERROR> Log_item not
found.\r\n\r\n");
}

void log_item(log_t *log_item, LOG_FORMAT_t format)
{
      if(format == BINARY_LOGGER)
            log_binary(log_item);
      else
            log_ascii(log_item);
}
///*
// * logger_queue.c
// *
// *  Created on: 04-Dec-2017
// *      Author: Gunj Manseta
// */
//
//
//#include "logger_queue.h"
//#include "memory.h"
//#include <malloc.h>
//
//#define LOGGERQ_STATUS_CODE_COUNT 10
//
//const uint8_t * LOGGERQ_Status_Strings[LOGGERQ_STATUS_CODE_COUNT] = {
//(uint8_t *)"Operation Successful",
//(uint8_t *)"Logger Queue is Empty",
//(uint8_t *)"Logger Queue has some data",
//(uint8_t *)"Logger Queue is Full",
//(uint8_t *)"Logger Queue is not Full",
//(uint8_t *)"Some NULL pointer Error",
//(uint8_t *)"Logger Queue is not allocated memory",
//(uint8_t *)"Logger Queue allocation failure",
//};
//
//LOGGERQ_Status_t LOGGERQ_init(Logger_Queue_t *loggerQ, uint16_t length)
//{
//    LOGGERQ_Status_t returnStatus = LOGGERQ_SUCCESS;
//
//    if (NULL == loggerQ)
//          returnStatus = LOGGERQ_NULL_POINTER_ERROR;
//    else
//    {
//          loggerQ->buffer =
(loggerQElement_t*)malloc(sizeof(loggerQElement_t)*length);
```

```
//          if (NULL == loggerQ->buffer)
//          {
//                loggerQ->size = 0;
//                loggerQ->head = NULL;
//                loggerQ->tail = NULL;
//                loggerQ->count = 0;
//                returnStatus = LOGGERQ_BUFFER_ALLOCATION_FAILURE;
//          }
//          else
//          {
//                loggerQ->size = length;
//                loggerQ->head = loggerQ->buffer;
//                loggerQ->tail = loggerQ->buffer;
//                loggerQ->count = 0;
//                my_memzero((uint8_t*)loggerQ->buffer, sizeof(*loggerQ-
>buffer));
//          }
//    }
//
//    return returnStatus;
//}
//
//
//LOGGERQ_Status_t LOGGERQ_buffer_add_item(Logger_Queue_t *loggerQ,
loggerQElement_t dataToAdd)
//{
//    LOGGERQ_Status_t returnStatus = LOGGERQ_is_full(loggerQ);
//
//    if (LOGGERQ_BUFFER_FULL == returnStatus ||
LOGGERQ_BUFFER_NOT_ALLOCATED == returnStatus ||
LOGGERQ_NULL_POINTER_ERROR == returnStatus)
//          return returnStatus;
//    else
//    {
//          *(loggerQ->head) = dataToAdd;
//          loggerQ->count++;
//          if ((loggerQ->head - loggerQ->buffer) < (loggerQ->size - 1))
      //there is still buffer location empty in the buffer, so we can
move the head to the next buffer location
//                loggerQ->head++;
//          else
//                loggerQ->head = loggerQ->buffer;  //rolling back the
head to the front of the buffer
//
//          returnStatus = LOGGERQ_SUCCESS;
//    }
//    return returnStatus;
//}
//
//LOGGERQ_Status_t LOGGERQ_buffer_remove_item(Logger_Queue_t *loggerQ,
loggerQElement_t *outData)
//{
//    LOGGERQ_Status_t returnStatus = LOGGERQ_is_empty(loggerQ);
//
//    if (LOGGERQ_BUFFER_EMPTY == returnStatus ||
LOGGERQ_BUFFER_NOT_ALLOCATED == returnStatus ||
LOGGERQ_NULL_POINTER_ERROR == returnStatus)
//          return returnStatus;
```

```
//	else
//	{
//		*outData = *(loggerQ->tail);
//		my_memset((uint8_t*)loggerQ->tail,sizeof(*(loggerQ-
>tail)),0);
//		(loggerQ->count)--;
//		if ((loggerQ->tail - loggerQ->buffer) < (loggerQ->size - 1))
	//there is still buffer location empty in the buffer, so we can
move the tail to the next buffer location
//			loggerQ->tail++;
//		else
//			loggerQ->tail = loggerQ->buffer;	//rolling back the
tail to the front of the buffer
//
//		returnStatus = LOGGERQ_SUCCESS;
//	}
//	return returnStatus;
//}
//
//LOGGERQ_Status_t LOGGERQ_peek(Logger_Queue_t * loggerQ, uint16_t
position, loggerQElement_t *outPeekData)
//{
//	LOGGERQ_Status_t returnStatus = LOGGERQ_is_empty(loggerQ);
//
//	if (LOGGERQ_BUFFER_EMPTY == returnStatus ||
LOGGERQ_BUFFER_NOT_ALLOCATED == returnStatus ||
LOGGERQ_NULL_POINTER_ERROR == returnStatus)
//		return returnStatus;
//	else
//	{	/*Since the position right next to head is supposed to be
empty, this function implementation will navigate 'position'
//		  no. of items before head to do a peek*/
//		if (loggerQ->count < position)	//Can't move through more
than 'count' positions
//			returnStatus = LOGGERQ_NULL_POINTER_ERROR;
//		else if ((loggerQ->head - loggerQ->buffer)>(loggerQ->count))
{	//No need to wrap around
//			*outPeekData=*(loggerQ->head - position);
//			returnStatus = LOGGERQ_SUCCESS;
//		}
//		else {	//Need to wrap around to move through 'position'
no. of items
//			*outPeekData= *(loggerQ->head + (loggerQ->size -
position +1));
//			returnStatus = LOGGERQ_SUCCESS;
//		}
//	}
//	return returnStatus;
//}
//
//LOGGERQ_Status_t LOGGERQ_is_full(Logger_Queue_t *loggerQ)
//{
//	LOGGERQ_Status_t returnStatus = LOGGERQ_BUFFER_NOT_FULL;
//
//	if (NULL == loggerQ)
//		returnStatus = LOGGERQ_NULL_POINTER_ERROR;
//	else if (NULL == loggerQ->buffer)
//		returnStatus = LOGGERQ_BUFFER_NOT_ALLOCATED;
```

```
//	else
//	{
//		if (loggerQ->size == loggerQ->count)
//			returnStatus = LOGGERQ_BUFFER_FULL;
//	}
//
//	return returnStatus;
//}
//
//LOGGERQ_Status_t LOGGERQ_is_empty(Logger_Queue_t *loggerQ)
//{
//	LOGGERQ_Status_t returnStatus = LOGGERQ_BUFFER_NOT_EMPTY;
//
//	if (NULL == loggerQ)
//		returnStatus = LOGGERQ_NULL_POINTER_ERROR;
//	else if (NULL == loggerQ->buffer)
//		returnStatus = LOGGERQ_BUFFER_NOT_ALLOCATED;
//	else
//	{
//		if (0 == loggerQ->count)
//			returnStatus = LOGGERQ_BUFFER_EMPTY;
//	}
//
//	return returnStatus;
//}
//
//LOGGERQ_Status_t LOGGERQ_destroy(Logger_Queue_t *loggerQ)
//{
//	LOGGERQ_Status_t returnStatus = LOGGERQ_SUCCESS;
//
//	if (NULL == loggerQ)
//		returnStatus = LOGGERQ_NULL_POINTER_ERROR;
//	else
//	{
//		if (NULL != loggerQ->buffer)
//		{
//			free(loggerQ->buffer);
//			loggerQ->buffer = NULL;
//		}
//		loggerQ->size = 0;
//		loggerQ->head = NULL;
//		loggerQ->tail = NULL;
//		loggerQ->count = 0;
//	}
//
//	return returnStatus;
//}
//
//
//const uint8_t* get_LOGGERQ_error_String(LOGGERQ_Status_t cbStatusEnum)
//{
//	return LOGGERQ_Status_Strings[cbStatusEnum];
//}
/*
 * mcg.c
 *
 *  Created on: 27-Oct-2017
 *      Author: Gunj Manseta
```

```
 */

#include "MKL25Z4.h"
#include "mcg.h"
#include <time.h>

void mcg_Init()
{
      /*    Using the on reset default FEI mode having Internal clock
reference on
      *     and sourcing the FLL. The MCGOUTCLK will be from FLL.
      *     Setting up clock to 48MHz using the MCG reg
      *     IRefClock - 32768Hz
      *     FLL Factor - 1463
      *     So, MCGFLLCLOCK - 32768*1463 = 47939584 ~48MhZ
      */
      MCG->C4 |= MCG_C4_DRST_DRS(MCG_C4_DRST_DRS_48MHZ) |
MCG_C4_DMX32(MCG_C4_DMX32_48MHZ);
}
/**
* @file - uart0.c
* @brief - Implementation file for the UART0 functions
*
* @author Gunj/Ashish University of Colorado Boulder
* @date 27/10/2017
**/

#include "MKL25Z4.h"
#include "mcg.h"
#include "uart0.h"
#include "gpio.h"
#include <malloc.h>
#include "conversion.h"
#include "memory.h"
#include "stdarg.h"
#include "stdio.h"
#include "platform.h"
#include "logger.h"

#define UART0ODE              (0)   //Open drain disable
#define UART0RXSRC                  (0)   //UART0_Rx pin
#define UART0TXSRC                  (0)   //UART0_Tx pin
#define UART0_CLKSEL_FLL      (0) //Select FLL as UART0 clk to MCGCLKFLL
#define UART0CLKSRC_FLLPLL    (1)   //MCGFLLCLK or MCGPLLCLK/2
#define UART0CLK_GATE_EN      (1)   //UART0 clock gate enable

#define _OSR_16_REG           (15)
#define OSR_16                      (16)
#define _OSR_32_REG           (31)
#define OSR_32                      (32)
#define _OSR                        (OSR_16)
#define _OSR_REG              (_OSR_16_REG)

#define BUFFER_TX_LEN         (4096)
#define BUFFER_RX_LEN         (64)

const uint16_t BUFFER_COUNT_THRESHOLD = ((BUFFER_TX_LEN*2)/3);
```

```
CB_t *UART0_TX_buffer = NULL;
CB_t *UART0_RX_buffer = NULL;

volatile uint8_t RX_bufferDataCount = 0;
volatile uint8_t processDataNow =0 ;
uint8_t bufferSet = 0;

void UART0_getBuffer(CB_t *outTXBuffer, CB_t *outRXBuffer)
{
	if(bufferSet)
	{
		outTXBuffer = UART0_TX_buffer;
		outRXBuffer = UART0_RX_buffer;
	}
}

CB_Status_t UART0_setBuffer(CB_t *TXBuffer, CB_t *RXBuffer)
{
	if(bufferSet)
		return CB_SUCCESS;

	if(NULL == TXBuffer)
	{
		UART0_TX_buffer = (CB_t*)malloc(sizeof(CB_t));
		if(NULL == UART0_TX_buffer)
			return CB_BUFFER_ALLOCATION_FAILURE;
		if(CB_SUCCESS == CB_init(UART0_TX_buffer,BUFFER_TX_LEN))
			TXBuffer = UART0_TX_buffer;
		else
			return CB_BUFFER_ALLOCATION_FAILURE;

	}
	else if(NULL == TXBuffer->buffer)
	{
		if(CB_BUFFER_ALLOCATION_FAILURE ==
CB_init(UART0_TX_buffer,BUFFER_TX_LEN))
			return CB_BUFFER_ALLOCATION_FAILURE;
		TXBuffer = UART0_TX_buffer;
	}
	else
		UART0_TX_buffer = TXBuffer;


	if(NULL == RXBuffer)
	{
		UART0_RX_buffer = (CB_t*)malloc(sizeof(CB_t));
		if(NULL == UART0_RX_buffer)
		{
			free(UART0_TX_buffer->buffer);	//we need to free the
UART0_TX_buffer->buffer allocated above.
			free((void*)UART0_TX_buffer);			//we also
need to free the UART0_TX_buffer
			UART0_TX_buffer = NULL;
			TXBuffer = NULL;
			return CB_BUFFER_ALLOCATION_FAILURE;
		}
		if(CB_SUCCESS == CB_init(UART0_RX_buffer,BUFFER_RX_LEN))
			RXBuffer = UART0_RX_buffer;
```

```
            else
                  return CB_BUFFER_ALLOCATION_FAILURE;
      }
      else if(NULL == RXBuffer->buffer)
      {
            if(CB_BUFFER_ALLOCATION_FAILURE ==
CB_init(UART0_RX_buffer,BUFFER_RX_LEN))
            {
                  free(UART0_TX_buffer->buffer);    //we need to free the
UART0_TX_buffer->buffer allocated above.
                  free((void*)UART0_TX_buffer);                 //we also
need to free the UART0_TX_buffer
                  UART0_TX_buffer = NULL;
                  TXBuffer = NULL;
                  return CB_BUFFER_ALLOCATION_FAILURE;
            }
            RXBuffer = UART0_RX_buffer;
      }
      else
            UART0_RX_buffer = RXBuffer;

      bufferSet = 1;
      return CB_SUCCESS;
}

int8_t UART0_configure(BAUD_RATE_ENUM baudRateSel)
{
      if(baudRateSel != BAUD_115200 && baudRateSel != BAUD_38400 &&
baudRateSel != BAUD_57200 && baudRateSel != BAUD_9600)
            return -1;

      //selecting the FLL clock source for UART0
      SIM->SOPT2 |= SIM_SOPT2_UART0SRC(UART0CLKSRC_FLLPLL);
      SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL(UART0_CLKSEL_FLL);

      //UART0 clock gate enable
      SIM->SCGC4 |= SIM_SCGC4_UART0(UART0CLK_GATE_EN);

      //Selecting the UART0 RX/TX pin behavior and source
      SIM->SOPT5 |= SIM_SOPT5_UART0ODE(UART0ODE);
      SIM->SOPT5 |= SIM_SOPT5_UART0RXSRC(UART0RXSRC);
      SIM->SOPT5 |= SIM_SOPT5_UART0TXSRC(UART0TXSRC);

      //Disabling the RX TX before configuring UART0
      UART0->C2 = 0;

      //Selecting 8 bit data, 1 stop bit, No parity
      UART0->C1 |= UART_C1_M(0) | UART_C1_PE(0);
      UART0->BDH |= UART_BDH_SBNS(0);

      //Setting OSR bits to 0b01111 = 15, which gives OSR to 16
      UART0->C4 |= UART0_C4_OSR(_OSR_REG);

      //SBR(BR) 1-8191 i.e. 13 bit value
      //Calculating the correct SBR(BR) for the selected BAUDRATE keeping
Clock of 48MHz, and OSR of 15+1
      //Formula used to calculate -> BaudRate = BaudClock/((OSR+1)*BR)
      //where BaudClock is the clock freq used for UART, BR=SBR
```

```
	uint16_t SBR = (uint16_t)(((uint32_t)FLL_CLK/(baudRateSel*_OSR)) & 
0x1FFF);
	UART0->BDL = UART_BDL_SBR(SBR);
	UART0->BDH |= UART_BDH_SBR(SBR>>8);

	//Enabling RIE Interrupt and the TCIE interrupt now.
	UART0->C2 |= UART_C2_RIE(1) | UART_C2_TCIE(1);

	//Enabling Rx and TX
	UART0->C2 |= UART_C2_RE(1) | UART_C2_TE(1);

	//Allocates TX/RX buffer. If fails, gives allocation failure.
	if(CB_BUFFER_ALLOCATION_FAILURE == UART0_setBuffer(NULL,NULL))
		return -2;

	//Enabling the NVIC Interrupt for UART0
	NVIC_EnableIRQ(UART0_IRQn);

	//Enabling the portA as the UART0 TX/RX pins are on portA
	GPIO_PORTA_ENABLE();
	//Setting the GPIO PA1 and PA2 to alt function 2 for UART0 Rx/Tx
	GPIO_PinAltFuncSel(gpioPortA, 1, gpioAlt2);
	GPIO_PinAltFuncSel(gpioPortA, 2, gpioAlt2);

//	GPIO_Green_Led_En();
//	GPIO_Red_Led_En();

	return 0;

}


void UART0_send(uint8_t *sendData)
{
	__disable_irq();
	if(NULL != sendData)
	{
		while(!(UART0->S1 & UART_S1_TDRE_MASK)); //Waiting for the 
buffer to get empty
		UART0->D = *sendData;
		while(!(UART0->S1 & UART_S1_TC_MASK)); //Waiting for 
transmission to get complete
	}
	__enable_irq();

}

void UART0_sendN(uint8_t *sendDataN, size_t len)
{
	__disable_irq();
	if(NULL != sendDataN)
	{
		size_t i = 0;
		while(i < len)
		{
			UART0_send(sendDataN + i);
			i++;
		}
```

```
	}
	__enable_irq();
}

void UART0_puts(uint8_t *sendDataN)
{
	if(NULL != sendDataN)
	{
		size_t i = 0;
		while(*(sendDataN+i))
		{
			UART0_CBsend(sendDataN+i);
			i++;
		}
	}
}

void UART0_putstr(const char *sendDataN)
{
	if(NULL != sendDataN)
	{
		size_t i = 0;
		while(*(sendDataN+i))
		{
			if(UART0_TX_buffer->count < BUFFER_COUNT_THRESHOLD)
			{
				UART0_CBsend((uint8_t*)sendDataN+i);
				i++;
			}
			else
				for(uint8_t a = 0; a < 200; a++)   //allowing the
CB buffer to get empty
				{
					int b = 0;
					b++;
				}

		}

	}
}

char* convert(unsigned int num, int base)
{
	static char buf[33];
	char *ptr;

	ptr=&buf[sizeof(buf)-1];
	*ptr='\0';
	do
	{
	*--ptr="0123456789abcdef"[num%base];
	num/=base;
	}while(num!=0);
	return(ptr);
}

void UART0_printf(char *fmt, ...)
```

```
{
	char *p;
	int i;
	unsigned int u;
	char *s;
	double d;
	char str[6];
	va_list argp;

	va_start(argp, fmt);

	p=fmt;
	for(p=fmt; *p!='\0';p++)
	{
		if(*p != '%')
		{
			UART0_CBsend((uint8_t*)p);
			continue;
		}

		p++;

		switch(*p)
		{
		case 'f' :
			d=va_arg(argp,double);
			if(d<0)
			{
				d=-d;
				UART0_CBsend((uint8_t*)'-');;
			}
			sprintf(str,"%f",d);
			UART0_putstr(str);
			break;
		case 'c' :
			i=va_arg(argp,int);
			UART0_CBsend((uint8_t*)&i);
			break;
		case 'd' :
			i=va_arg(argp,int);
			if(i<0)
			{
				i=-i;
				UART0_CBsend((uint8_t*)'-');
			}
			UART0_putstr(convert(i,10));
			break;
		case 'o':
			i=va_arg(argp,unsigned int);
			UART0_putstr(convert(i,8));
			break;
		case 's':
			s=va_arg(argp,char *);
			UART0_putstr(s);
			break;
		case 'u':
			u=va_arg(argp,unsigned int);
			UART0_putstr(convert(u,10));
```

```
				break;
			case 'x':
				u=va_arg(argp,unsigned int);
				UART0_putstr(convert(u,16));
				break;
			case '%':
				UART0_CBsend((uint8_t*)'%');
				break;
			}
	}

	va_end(argp);
}



void UART0_CBsend(uint8_t *sendData)
{
//	__disable_irq();
	//if(CB_BUFFER_NOT_FULL == CB_is_full(UART0_TX_buffer))
	//{
//	CB_Status_t status = 1;
	__disable_irq();
	CB_buffer_add_item(UART0_TX_buffer,*sendData);
	UART0_TX_INT_ENABLE;
	__enable_irq();
	//}
	//else		//TX buffer is full.
	//{
	//}
//	UART0_TX_INT_ENABLE;
//	__enable_irq();
}

void UART0_receive(uint8_t *recvData)
{
	__disable_irq();
	if(NULL != recvData)
	{
		while((UART0->S1 & UART_S1_RDRF_MASK) == 0); //Waiting for
the data to recv
		*recvData = UART0->D;
	}
	__enable_irq();
}

void UART0_receiveN(uint8_t *recvDataN, size_t len)
{
//	__disable_irq();
	if(NULL != recvDataN)
	{
		size_t i = 0;
		while(i < len)
		{
			__disable_irq();
			UART0_receive(recvDataN+i);
			i++;
			__enable_irq();
```

```
			}
		}
//		__enable_irq();
}


void UART0_CBsendN(uint8_t *sendDataN, size_t len)
{

	size_t itr = 0;
	CB_Status_t status = CB_SUCCESS;
	while((CB_SUCCESS == status ) && (itr < len))
	{
		__disable_irq();
		status =
CB_buffer_add_item(UART0_TX_buffer,*(sendDataN+itr));
		itr++;
		UART0_TX_INT_ENABLE;
		__enable_irq();
	}
}

void UART0_CBreceive(uint8_t *recvData)
{
//	__disable_irq();
//	if(CB_BUFFER_NOT_EMPTY == CB_is_empty(UART0_RX_buffer))
//	{
		__disable_irq();
		CB_Status_t status =
CB_buffer_remove_item(UART0_RX_buffer,recvData);
		if(status == CB_SUCCESS)
			UART0_RX_INT_ENABLE;
		else
			*recvData = 0xFF;
		__enable_irq();
//	}
//	else //RX Buffer is empty.
//	{
//		*recvData = 0xFF;
//	}
//	__enable_irq();
}

void UART0_CBreceiveN(uint8_t *recvDataN, size_t len)
{
	size_t itr = 0;
	CB_Status_t status = CB_SUCCESS;
	while((CB_SUCCESS == status) && (itr < len))
	{
		__disable_irq();
		status =
CB_buffer_remove_item(UART0_RX_buffer,(recvDataN+itr));
		itr++;
		UART0_RX_INT_ENABLE;
		__enable_irq();
	}
}
```

```
void UART0_logFlush()
{
	while(UART0_TX_buffer->count != 0);
}


void UART0_IRQHandler(void)
{
	__disable_irq();

    unsigned char UART0_bufferData;

    if((UART0->S1 & UART_S1_TC_MASK) && (UART0->C2 & UART_C2_TCIE_MASK))
	//TX TC flag is set and We are ready to transfer data
    {
//		GPIO_Green_On();

	UART0->S1 &= ~UART_S1_TC_MASK;
	CB_Status_t status =
CB_buffer_remove_item(UART0_TX_buffer,&UART0_bufferData);
	if(CB_SUCCESS == status)
	{
		UART0->D = UART0_bufferData;
		UART0_TX_INT_ENABLE;
	}
	else if(CB_BUFFER_EMPTY == status) //TX buffer empty
	{
		UART0_TX_INT_DISABLE;  //We dont have anything to transmit, so
disable the TX Int. Tx int will be enabled by UART_Send functions.
	}
	else  //Some other Buffer error conditions
	{
	}

		//GPIO_Green_Off();
    }
    else if((UART0->S1 & UART_S1_RDRF_MASK) && (UART0->C2 &
UART_C2_RIE_MASK))
	{
		//GPIO_Red_On();

		UART0_bufferData = UART0->D;
		CB_Status_t status = CB_buffer_add_item(UART0_RX_buffer,
UART0_bufferData);
		if(CB_SUCCESS == status)
		{
			logger_log(DATA_RECEIVED,"Data: %c",UART0_bufferData);
			(UART0_bufferData == '~') ? logging = 0 : 0;
			(UART0_bufferData == '!') ? logging = 1 : 0;
			(UART0_bufferData == '|') ? log_format = BINARY_LOGGER :
0;
			(UART0_bufferData == '/') ? log_format = ASCII_LOGGER :
0;
			(UART0_bufferData == '\r') ? processDataNow = 1: 0;
		}
		else if(CB_BUFFER_FULL == status)  //RX Buffer is full
		{
```

```
                 UART0_RX_INT_DISABLE;  //We can't handle more data, so
disable the RX Int. Rx int will be enabled by UART_Recv functions which
will remove some items from the buffer
                 //UART0_putstr("\r\nRX BUFFER IS FULL");
            }
            //GPIO_Red_Off();
      }
      __enable_irq();
}

/*
 * nordic_driver.c
 *
 *  Created on: 02-Dec-2017
 *      Author: Gunj Manseta
 */


#include <stdint.h>

#include "nordic_driver.h"
#include "spi.h"
#include "gpio.h"

//Commands Byte
#define NORDIC_TXFIFO_FLUSH_CMD    (0xE1)
#define NORDIC_RXFIFO_FLUSH_CMD    (0xE2)
#define NORDIC_W_TXPAYLD_CMD (0xA0)
#define NORDIC_R_RXPAYLD_CMD (0x61)
#define NORDIC_ACTIVATE_CMD        (0x50)
#define NORDIC_ACTIVATE_DATA (0x73)
#define NORDIC_RXPAYLD_W_CMD (0x60)

//Register Addresses
#define NORDIC_CONFIG_REG          (0x00)
#define NORDIC_STATUS_REG          (0x07)
#define NORDIC_RF_SETUP_REG        (0x06)
#define NORDIC_RF_CH_REG           (0x05)
#define NORDIC_TX_ADDR_REG         (0x10)
#define NORDIC_TX_ADDR_LEN         (5)
#define NORDIC_FIFO_STATUS_REG     (0x17)


void NRF_gpioInit()
{

      //Enabling the GPIO PTD5 for Nordic CE pin
      GPIO_PORT_ENABLE(NORDIC_CE_PORT);
      GPIO_PinDir(NORDIC_CE_PORT,NORDIC_CE_PIN,gpio_output);
      GPIO_PinAltFuncSel(NORDIC_CE_PORT,NORDIC_CE_PIN,gpioAlt1_GPIO);
      GPIO_PinOutClear(NORDIC_CE_PORT,NORDIC_CE_PIN);

      //Enabling the GPIO PTA13 for Nordic IRQ pin
      GPIO_PORT_ENABLE(NORDIC_IRQ_PORT);
      GPIO_PinDir(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN,gpio_input);
      GPIO_PinAltFuncSel(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN,gpioAlt1_GPIO);
}
```

```
void NRF_moduleInit()
{
	NRF_gpioInit();

	SPI_init(SPI_0);
}

void NRF_moduleDisable()
{
	GPIO_PinAltFuncSel(NORDIC_CE_PORT,NORDIC_CE_PIN,gpioAlt0_Disabled);
	GPIO_PinAltFuncSel(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN,gpioAlt0_Disabled
);
	SPI_disable();
}

uint8_t NRF_read_register(uint8_t regAdd)
{
	//SPI_clear_RXbuffer(SPI_0); //used to clear the previously value in
the RX FIFO
	uint8_t readValue = 0;

	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_0,regAdd);
	SPI_read_byte(SPI_0);  //used to clear the previously value in the
RX FIFO
	SPI_write_byte(SPI_0,0xFF);
	readValue = SPI_read_byte(SPI_0);

	//Marking the end of transaction by CSN high
	NRF_chip_disable();

	return readValue;
}

void NRF_write_command(uint8_t command)
{
	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_0,command);
	//SPI_clear_RXbuffer(SPI_0); //used to clear the previously value in
the RX FIFO
	SPI_read_byte(SPI_0);

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

void NRF_write_register(uint8_t regAdd, uint8_t value)
{
	//SPI_clear_RXbuffer(SPI_0); //used to clear the previously value in
the RX FIFO
```

```
	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_0,regAdd | 0x20);
	SPI_read_byte(SPI_0);  //used to clear the previously value in the
RX FIFO
	SPI_write_byte(SPI_0,value);
	SPI_read_byte(SPI_0);  //used to clear the previously value in the
RX FIFO

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

uint8_t NRF_read_status()
{
	return  NRF_read_register(NORDIC_STATUS_REG);
}

void NRF_write_config(uint8_t configValue)
{
	NRF_write_register(NORDIC_CONFIG_REG, configValue);
}

uint8_t NRF_read_config()
{
	return NRF_read_register(NORDIC_CONFIG_REG);
}

uint8_t NRF_read_rf_setup()
{
	return NRF_read_register(NORDIC_RF_SETUP_REG);
}

void NRF_write_rf_setup(uint8_t rfSetupValue)
{
	NRF_write_register(NORDIC_RF_SETUP_REG, rfSetupValue);
}

uint8_t NRF_read_rf_ch()
{
	return NRF_read_register(NORDIC_RF_CH_REG);
}

void NRF_write_rf_ch(uint8_t channel)
{
	NRF_write_register(NORDIC_RF_CH_REG, channel);
}

void NRF_read_TX_ADDR(uint8_t *address)
{
	uint8_t i = 0;

	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_0,NORDIC_TX_ADDR_REG);
```

```
	SPI_read_byte(SPI_0);  //used to clear the previously value in the
RX FIFO
	//SPI_read_byte(SPI_0);      //used to clear the previously value in
the RX FIFO
	while(i < NORDIC_TX_ADDR_LEN)
	{
		SPI_write_byte(SPI_0, 0xFF); //Dummy to get the data
		*(address+i) = SPI_read_byte(SPI_0);
		i++;
	}

	NRF_chip_disable();
}

void NRF_write_TX_ADDR(uint8_t * address)
{
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_0,NORDIC_TX_ADDR_REG | 0x20);
	SPI_read_byte(SPI_0);  //used to clear the previously value in the
RX FIFO
	SPI_write_packet(SPI_0,address,NORDIC_TX_ADDR_LEN);
	SPI_read_byte(SPI_0); //used to clear the previously value in the
RX FIFO
	SPI_read_byte(SPI_0); //used to clear the previously value in the
RX FIFO

	NRF_chip_disable();
}

uint8_t NRF_read_fifo_status()
{
	return NRF_read_register(NORDIC_FIFO_STATUS_REG);
}

void NRF_flush_tx_fifo()
{
	NRF_write_command(NORDIC_TXFIFO_FLUSH_CMD);
}

void NRF_flush_rx_fifo()
{
	NRF_write_command(NORDIC_RXFIFO_FLUSH_CMD);
}

void NRF_activate_cmd()
{
	NRF_write_register(NORDIC_ACTIVATE_CMD, NORDIC_ACTIVATE_DATA);
}

/*
 * platform.c
 *
 *  Created on: 30-Oct-2017
 *      Author: Gunj Manseta
 */
```

```c
#include "platform.h"

#ifdef PLATFORM_KL25Z

#include "uart0.h"
#include "stdarg.h"

int printf(const char *fmt,...)
{
	va_list args;
	va_start(args, fmt);
	UART0_printf((char*)fmt, args);
	va_end(args);
	return 0;
}
#endif
/*
 * time_profiler.c
 *
 *  Created on: 01-Dec-2017
 *      Author: Gunj Manseta
 */

#include "time_profiler.h"

volatile uint8_t tick_overflow = 1;

#ifdef PLATFORM_KL25Z

#include "MKL25Z4.h"

#define G_SYSTICKS (SysTick->VAL)

void profiler_setup()
{
	SysTick->CTRL &= ~SysTick_CTRL_CLKSOURCE_Msk;
	SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
	SysTick->LOAD = SysTick_LOAD_RELOAD_Msk;
	SysTick->VAL = 0;
	SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;
	SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}

void SysTick_IRQHandler()
{
	__disable_irq();
	tick_overflow++;
	__enable_irq();
}

tickTime profiler_getTickDiff(tickTime since)
{
	tickTime now = G_SYSTICKS;
	return (((since - now) >= 0) ? (since - now) : (since + ((1+
TICK_MAX)*tick_overflow) - now));
}

tickTime profiler_getCurrentTick()
```

```
{
	//tick_overflow = 1;
	return G_SYSTICKS;
}

#else

//#define G_SYSTICKS clock()
#define G_SYSTICKS timevalue.tv_usec

tickTime profiler_getTickDiff(tickTime since)
{
	gettimeofday(&timevalue,NULL);
	tickTime now = G_SYSTICKS;
	return (((now - since) >= 0) ? (now - since) : (now + ((1+
TICK_MAX)*tick_overflow) - since));
}

tickTime profiler_getCurrentTick()
{
	//tick_overflow = 1;
	gettimeofday(&timevalue,NULL);
	return G_SYSTICKS;
}

#endif



float profiler_getTime_us(tickTime ticks)
{
	return ((((float)ticks)*(1000000.0))/CLK_PER_SEC);
}


/*
 * project3.c
 *
 *  Created on: 30-Nov-2017
 *      Author: Gunj Manseta
 */

#include "project3.h"
#include "memory.h"
#include "time_profiler.h"
#include "conversion.h"
#include <string.h>
#include "platform.h"
#include "logger.h"
#include "data_processing.h"
#include "timestamp.h"

#ifdef PLATFORM_KL25Z
#include "MKL25Z4.h"
#include "uart0.h"
#include "timer0.h"
#include "dma.h"
#include "gpio.h"
```

```
#include "mcg.h"
#include "spi.h"
#include "nordic_driver.h"

extern uint32_t __HeapLimit;
extern uint32_t STACK_SIZE;
extern uint32_t HEAP_SIZE;

void Nordic_Test();

#else
CB_t processingBuffer;
tickTime tickStart = 0;
tickTime tickEnd = 0;
#endif


void profiler_Test();
void profile_memoryFunctions(uint32_t Data_Size);

void project3()
{

#ifdef PLATFORM_KL25Z
      __disable_irq();
      mcg_Init();
      UART0_configure(BAUD_115200);
      rtc_init();
      logger_log(INFO,"CLOCK INIT");
      logger_log(INFO,"UART0 INIT");
      logger_log(INFO,"BUILD EPOCH TIME: %u",BUILD_EPOCH_TIME);
      logger_log(LOGGER_INITIALZED,"");

      GPIO_Red_Led_En();
      GPIO_Red_Off();

      logger_log(GPIO_INITIALZED,"");

      //timer0_configure();
      logger_log(INFO,"MEMORY - HEAP SIZE: 0x%x",&HEAP_SIZE);
      logger_log(INFO,"MEMORY - STACK SIZE: 0x%x",&STACK_SIZE);
      logger_log(INFO,"MEMORY - HEAP END: 0x%x",&__HeapLimit);
      logger_log(SYSTEM_BOOTED,"PES PROJECT 3 - KL25Z");

      __enable_irq();

#else
      CB_init(&processingBuffer,64);
      uint8_t data;
      uint8_t processDataNow = 0;
#endif

      logger_log(SYSTEM_INITIALIZED,"");
#ifdef PLATFORM_KL25Z
      Nordic_Test();
#endif
```

```
	profiler_Test();

	while(1)
	{
		if(processDataNow)
		{
			processDataNow = 0;
#ifdef PLATFORM_KL25Z
			processData(UART0_RX_buffer);
		}
#else
			processData(&processingBuffer);
		}
		scanf("%c",&data);
		CB_buffer_add_item(&processingBuffer,data);
		logger_log(DATA_RECEIVED,"%c",data);
		(data == '~') ? logging = 0 : 0;
		(data == '!') ? logging = 1 : 0;
		(data == '|') ? log_format = BINARY_LOGGER : 0;
		(data == '/') ? log_format = ASCII_LOGGER : 0;
		(data == '\n') ? processDataNow = 1: 0;

#endif
	}

}


void profile_memoryFunctions(uint32_t Data_Size)
{
	logger_log(PROFILING_STARTED,"To test %d of transfer",Data_Size);
	uint8_t samples_i = 0;

	uint8_t *src = (uint8_t*)malloc(Data_Size);
	uint8_t *dst = (uint8_t*)malloc(Data_Size);
//	uint8_t src[Data_Size];
//	uint8_t dst[Data_Size];
#ifdef PLATFORM_KL25Z
	if(((uint32_t)&__HeapLimit < (uint32_t)(src+Data_Size)) ||
((uint32_t)&__HeapLimit < (uint32_t)(dst+Data_Size)))
		logger_log(WARNING,"HEAP AND STACK MIGHT INTERSECT");
#endif
	if(src == NULL)
	{
		logger_log(ERROR,"MALLOC FAILED: SRC");
		return;
	}
	if(dst == NULL)
	{
		free(src);
		logger_log(ERROR,"MALLOC FAILED: DEST");
		return;
	}

	while( samples_i < 1)
	{
		memset(src,0,Data_Size);
		memset(dst,0,Data_Size);
```

```
            tickStart = 0;
            tickEnd = 0;
            profiler_setup();
            tickStart = profiler_getCurrentTick();
            my_memset(src,Data_Size,'S');
            tickTime diff = profiler_getTickDiff(tickStart);
            //UART0_putstr("my_memset Ticks: ");
            //uint8_t len  = sprintf(str,"%u\tTime: %f
us",diff,profiler_getTime_us(diff));
            //UART0_printf("%u\tTime: %f
us",diff,profiler_getTime_us(diff));
            //UART0_putstr(str);
            //UART0_NEWLINE;
            logger_log(PROFILING_RESULT,"my_memset Time: %f
us",profiler_getTime_us(diff));

            tickStart = 0;
            tickEnd = 0;
            //profiler_setup();
            tickStart = profiler_getCurrentTick();
            my_memmove(src,dst,Data_Size);
            diff = profiler_getTickDiff(tickStart);
            logger_log(PROFILING_RESULT,"my_memmove Time: %f
us",profiler_getTime_us(diff));

            tickStart = 0;
            tickEnd = 0;
            //profiler_setup();
            tickStart = profiler_getCurrentTick();
            if(memset_dma(src,Data_Size,'X') == -1)
                  logger_log(ERROR,"memset_dma failed");

#ifdef PLATFORM_KL25Z
            //while(DMA_CurrentState[DMA_0] != DMA_Complete &&
DMA_CurrentState[DMA_0] != DMA_Error);
            while(DMA_CurrentState[DMA_0] == DMA_Busy);
            diff = tickStart - tickEnd;
            if(DMA_CurrentState[DMA_0] == DMA_Error)
            {
                  logger_log(ERROR,"memset_dma : Error in transfer on
DMA0");
                  DMA_CurrentState[DMA_0] = DMA_Ready;
            }
            else
            {
                  logger_log(PROFILING_RESULT,"memset_dma Time: %f
us",profiler_getTime_us(diff));
            }
#else
            diff = profiler_getTickDiff(tickStart);
            logger_log(PROFILING_RESULT,"my_memset Time: %f
us",profiler_getTime_us(diff));
#endif

            tickStart = 0;
            tickEnd = 0;
            //profiler_setup();
            tickStart = profiler_getCurrentTick();
```

```
			if(memmove_dma(dst,src,Data_Size) == -1)
				logger_log(ERROR,"memmovet_dma failed");
#ifdef PLATFORM_KL25Z
			//while(DMA_CurrentState[DMA_0] != DMA_Complete &&
DMA_CurrentState[DMA_0] != DMA_Error);
			while(DMA_CurrentState[DMA_0] == DMA_Busy);
			diff = tickStart - tickEnd;
			if(DMA_CurrentState[DMA_0] == DMA_Error)
			{
				logger_log(ERROR,"memmove_dma : Error in transfer on
DMA0");
				DMA_CurrentState[DMA_0] = DMA_Ready;
			}
			else
			{
				logger_log(PROFILING_RESULT,"memmove_dma Time: %f
us",profiler_getTime_us(diff));
			}
#else
			diff = profiler_getTickDiff(tickStart);
			logger_log(PROFILING_RESULT,"my_memmove Time: %f
us",profiler_getTime_us(diff));
#endif

			tickStart = 0;
			tickEnd = 0;
			//profiler_setup();
			tickStart = profiler_getCurrentTick();
			memset(src,'Z',Data_Size);
			diff = profiler_getTickDiff(tickStart);
			logger_log(PROFILING_RESULT,"memset Time: %f
us",profiler_getTime_us(diff));

			tickStart = 0;
			tickEnd = 0;
			//profiler_setup();
			tickStart = profiler_getCurrentTick();
			memmove(src,dst,Data_Size);
			diff = profiler_getTickDiff(tickStart);
			logger_log(PROFILING_RESULT,"memmove Time: %f
us",profiler_getTime_us(diff));

			tickStart = 0;
			tickEnd = 0;
			samples_i++;

		}

		free(src);
		free(dst);

		logger_log(PROFILING_COMPLETED,"");
}


void profiler_Test()
{
#ifdef PLATFORM_KL25Z
```

```
	DMA_Configure_t config;
	config.AutoAlign=1;
	config.CycleSteal=0;
	config.D_REQ=0;
	config.EnableInterrupt=1;
	config.EnablePeripheralReq = 0;
	if(dma_configure(DMA_0,&config) == -1)
		logger_log(ERROR,"DMA_0 Configuration.");
#endif

//	profile_memoryFunctions(1);
//	profile_memoryFunctions(2);
//	profile_memoryFunctions(3);
//	profile_memoryFunctions(4);
	profile_memoryFunctions(10);
	profile_memoryFunctions(100);
	profile_memoryFunctions(500);
	profile_memoryFunctions(5000);

}

#ifdef PLATFORM_KL25Z
void Nordic_Test()
{
	NRF_moduleInit();

	logger_log(INFO, "SPI Initialized");
	logger_log(INFO,"Nordic Initialized");
	logger_log(INFO,"Nordic Test");
	uint8_t sendValue = 0x48;
	uint8_t readValue = 0;
	NRF_write_config(sendValue);
	readValue = NRF_read_config();
	if(readValue == sendValue)
	{
		logger_log(INFO,"Value Matched");
		logger_log(INFO,"Sent: 0x%x",sendValue);
		logger_log(INFO,"Recv: 0x%x",readValue);
	}

	uint8_t sendAddr[5] = {0xBA,0x56,0xBA,0x56,0xBA};
	logger_log(INFO,"TX ADDRESSES SET:
0x%x%x%x%x",sendAddr[0],sendAddr[1],sendAddr[2],sendAddr[3],sendAddr[4]);
	NRF_write_TX_ADDR(sendAddr);
	uint8_t *readAddr = (uint8_t*)malloc(5);
	NRF_read_TX_ADDR(readAddr);
	logger_log(INFO,"TX ADDRESSES GET:
0x%x%x%x%x\r\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);
	free(readAddr);


	logger_log(INFO,"Nordic Test End");

	NRF_moduleDisable();
}
#endif
/**
```

```
* @file - main.c
* @brief - Contains the entry point of the program which tests various
functionalities
*
* @author Gunj/Ashish University of Colorado Boulder
* @date    02/10/2017
**/

#ifdef PROJECT1
#include "project1.h"
#endif // PROJECT1

#ifdef PROJECT2
#include "project2.h"
#endif // PROJECT2

#ifdef HW5
#include "hw5.h"
#endif // HW5

#ifdef PROJECT3
#include "project3.h"
#endif // PROJECT3

int main()
{
#ifdef PROJECT1
      project1();
#endif // PROJECT1

#ifdef PROJECT2
      project2();
#endif // PROJECT2

#ifdef HW5
      hw5();
#endif // PROJECT2

#ifdef PROJECT3
      project3();
#endif

      return 0;
}
```