main.c

```
////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////

// Standard C Included Files
#include <string.h>
#include <stdio.h>
// SDK Included Files

#include "board.h"
#include "pin_mux.h"
#include "fsl_clock_manager.h"

#include "fsl_debug_console.h"
//#include "adc_hw_trigger.h"
//#include "fsl_adc16_driver.h"

#include "Custom_Main.h"
#include "Custom_Circular_Buffer.h"
#include "Custom_UART.h"
#include "Custom_ASCII_Counter.h"

UART0_Operation_Type State;

char UART_print[50];

DWord Fib_n, Fib_1 = 0, Fib_2 = 1;
Byte led = 0;

#ifdef   FRDM
void hardware_init(void);

//FGETS using custom UART function
void FGETS(char *array_to_write, Byte bytes, FILE *stream)
{
        //Cleanup and assign FGETS buffer (of fixed length)
        CBuffer_Assign(FGETS_Buffer_ID);

        //Set the proper state for ISR
        State = FGETS_Operation;

        //Don't proceed until enter is pressed (or length is reached)
        while(State == FGETS_Operation);

        if(CBuffer_Instance[FGETS_Buffer_ID].Status == Full)
        Output_String("\n\rOverwriting");
```

```
	//Form a proper string/array
	char *tmp;
	tmp = array_to_write;
	do{
		if(CBuffer_Byte_Read(FGETS_Buffer_ID, array_to_write))	break;
		if(*array_to_write == Enter_Detected)  break;
		array_to_write += 1;
	}while(array_to_write < (tmp + bytes));
}
#endif

int main (void)
{
	//Init hardware
	hardware_init();

	//Fun
	PORTB->PCR[18] = PORT_PCR_MUX(0x01);
	PTB->PDDR |= (1 << 18);
	PTB->PDOR |= (1 << 18);

	PORTB->PCR[19] = PORT_PCR_MUX(0x01);
	PTB->PDDR |= (1 << 19);
	PTB->PDOR |= (1 << 19);

	PORTD->PCR[1] = PORT_PCR_MUX(0x01);
	PTD->PDDR |= (1 << 1);
	PTD->PDOR |= (1 << 1);

	//UART and ASCII counter (application) Init
	Custom_UART0_Init();
	ASCII_Counter_Init();

	//If polling mode then just echo the received characters
#ifdef	POLLING_MODE
	Output_String("\n\rPolling Mode\n\r");

	while(1)
	{
		Custom_UART0_Rx_Byte(&test);
		Custom_UART0_Tx_Byte(test);
	}
#else
	Output_String("\n\rInterrupt Mode\n\r");
#if APPLICATION
	Output_String("\n\rApplication Running\n\r");
	if(CBuffer_Init())	Output_String("\n\rError in Buffer Init\n\r");
	else	Output_String("\n\rBuffer Init Success\n\r\n\r");
	while(1)
	{
```

```
		switch(led)
		{
			case 0:
				PTD->PDOR |= (1 << 1);
				PTB->PDOR |= (1 << 18);
				PTB->PDOR |= (1 << 19);
				break;
			case 1:
				PTD->PDOR &= ~(1 << 1);
				PTB->PDOR |= (1 << 18);
				PTB->PDOR |= (1 << 19);
				break;
			case 2:
				PTD->PDOR |= (1 << 1);
				PTB->PDOR &= ~(1 << 18);
				PTB->PDOR |= (1 << 19);
				break;
			case 3:
				PTD->PDOR |= (1 << 1);
				PTB->PDOR |= (1 << 18);
				PTB->PDOR &= ~(1 << 19);
				break;
			case 4:
				PTD->PDOR &= ~(1 << 1);
				PTB->PDOR &= ~(1 << 18);
				PTB->PDOR |= (1 << 19);
				break;
			case 5:
				PTD->PDOR &= ~(1 << 1);
				PTB->PDOR |= (1 << 18);
				PTB->PDOR &= ~(1 << 19);
				break;
			case 6:
				PTD->PDOR |= (1 << 1);
				PTB->PDOR &= ~(1 << 18);
				PTB->PDOR &= ~(1 << 19);
				break;
			case 7:
				PTD->PDOR &= ~(1 << 1);
				PTB->PDOR &= ~(1 << 18);
				PTB->PDOR &= ~(1 << 19);
				break;
		}
		Fib_n = Fib_1 + Fib_2;
		while(CBuffer_Instance[UART0_Rx_Buffer_ID].Status != Empty)
	ASCII_Counter();
		if(Fib_n > 3900000000)
		{
			Fib_2 = 0;
			Fib_n = 1;
```

```
                }
                Fib_1 = Fib_2;
                Fib_2 = Fib_n;
        }

#else
        Output_String("\n\rToggling LEDs and Echo\n\r");
        while(1)
        {
                PTD->PTOR |= (1 << 1);
                PTB->PTOR |= (1 << 18);
                PTB->PTOR |= (1 << 19);
        }

#endif

#endif
}
```

lptmr_trigger.c

```
////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////

// SDK Included Files
#include "adc_hw_trigger.h"
#include "fsl_lptmr_driver.h"
#include "fsl_sim_hal.h"
#if defined(KM34Z7_SERIES)
#include "fsl_xbar_driver.h"
#endif
////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////

extern SIM_Type * gSimBase[];
static lptmr_state_t gLPTMRState;

////////////////////////////////////////////////////////////////////////////////
// Code
////////////////////////////////////////////////////////////////////////////////

/*!
 * @Brief enable the trigger source of LPTimer
 */
void init_trigger_source(uint32_t adcInstance)
{
```

```
    uint32_t freqUs;

    lptmr_user_config_t lptmrUserConfig =
    {
        .timerMode = kLptmrTimerModeTimeCounter,
        .freeRunningEnable = false,
        .prescalerEnable = false, // bypass perscaler
#if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
        // use MCGIRCCLK, 4M or 32KHz
        .prescalerClockSource = kClockLptmrSrcMcgIrClk,
#else
        // Use LPO clock 1KHz
        .prescalerClockSource = kClockLptmrSrcLpoClk,
#endif
        .isInterruptEnabled = false
    };

    // Init LPTimer driver
    LPTMR_DRV_Init(0, &gLPTMRState, &lptmrUserConfig);

    // Set the LPTimer period
    freqUs = 1000000U/(INPUT_SIGNAL_FREQ*NR_SAMPLES)*2;
    LPTMR_DRV_SetTimerPeriodUs(0, freqUs);

    // Start the LPTimer
    LPTMR_DRV_Start(0);

    // Configure SIM for ADC hw trigger source selection
#if defined(KM34Z7_SERIES)
    SIM_HAL_EnableClock(gSimBase[0], kSimClockGateXbar0);
    SIM_HAL_SetAdcTrgSelMode(gSimBase[0], kSimAdcTrgSelXbar);
    XBAR_DRV_ConfigSignalConnection(kXbaraInputLPTMR0_Output, kXbaraOutputADC_TRGA);
#else
    SIM_HAL_SetAdcAlternativeTriggerCmd(gSimBase[0], adcInstance, true);
    SIM_HAL_SetAdcPreTriggerMode(gSimBase[0], adcInstance, kSimAdcPretrgselA);
    SIM_HAL_SetAdcTriggerMode(gSimBase[0], adcInstance, kSimAdcTrgSelLptimer);
#endif
}

/*!
 * @Brief disable the trigger source
 */
void deinit_trigger_source(uint32_t adcInstance)
{
    LPTMR_DRV_Stop(0);
    LPTMR_DRV_Deinit(0);
}
```

fsl_adc_irq.c

```
////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////

// Standard C Included Files
#include <stdint.h>
#include <stdbool.h>
// SDK Included Files
#include "fsl_adc16_driver.h"

////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////

// Define array to keep run-time callback set by application.
void (* volatile g_AdcTestCallback[ADC_INSTANCE_COUNT][ADC_SC1_COUNT])(void);
volatile uint16_t g_AdcValueInt[ADC_INSTANCE_COUNT][ADC_SC1_COUNT];

////////////////////////////////////////////////////////////////////////////////
// Code
////////////////////////////////////////////////////////////////////////////////

/* User-defined function to install callback. */
void ADC_TEST_InstallCallback(uint32_t instance, uint32_t chnGroup, void (*callbackFunc)(void)
)
{
   g_AdcTestCallback[instance][chnGroup] = callbackFunc;
}

/* User-defined function to read conversion value in ADC ISR. */
uint16_t ADC_TEST_GetConvValueRAWInt(uint32_t instance, uint32_t chnGroup)
{
   return g_AdcValueInt[instance][chnGroup];
}

/* User-defined ADC ISR. */
static void ADC_TEST_IRQHandler(uint32_t instance)
{
   uint32_t chnGroup;
   for (chnGroup = 0U; chnGroup < ADC_SC1_COUNT; chnGroup++)
   {
      if (   ADC16_DRV_GetChnFlag(instance, chnGroup, kAdcChnConvCompleteFlag) )
      {
         g_AdcValueInt[instance][chnGroup] = ADC16_DRV_GetConvValueRAW(instance,
chnGroup);
         if ( g_AdcTestCallback[instance][chnGroup] )
         {
```

```
            (void)(*(g_AdcTestCallback[instance][chnGroup]))();
        }
      }
    }
}

////////////////////////////////////////////////////////////////////////////////
// IRQ Handlers
////////////////////////////////////////////////////////////////////////////////

/* ADC IRQ handler that would cover the same name's APIs in startup code */
void ADC0_IRQHandler(void)
{
    // Add user-defined ISR for ADC0
    ADC_TEST_IRQHandler(0U);
}

#if (ADC_INSTANCE_COUNT > 1U)
void ADC1_IRQHandler(void)
{
    // Add user-defined ISR for ADC1
    ADC_TEST_IRQHandler(1U);
}
#endif

#if (ADC_INSTANCE_COUNT > 2U)
void ADC2_IRQHandler(void)
{
    // Add user-defined ISR for ADC2
    ADC_TEST_IRQHandler(2U);
}
#endif

#if (ADC_INSTANCE_COUNT > 3U)
void ADC3_IRQHandler(void)
{
    // Add user-defined ISR for ADC3
    ADC_TEST_IRQHandler(3U);
}
#endif
```

---

startup.c

```
#include "startup.h"
#include "fsl_device_registers.h"

#if (defined(__ICCARM__))
  #pragma section = ".data"
```

```
    #pragma section = ".data_init"
    #pragma section = ".bss"
#endif

/*****************************************************************************
 * Code
 *****************************************************************************/

/*FUNCTION*****************************************************************
*
 *
 * Function Name : init_data_bss
 * Description   : Make necessary initializations for RAM.
 * - Copy initialized data from ROM to RAM.
 * - Clear the zero-initialized data section.
 * - Copy the vector table from ROM to RAM. This could be an option.
 *
 * Tool Chians:
 *   __GNUC__   : GCC
 *   __CC_ARM   : KEIL
 *   __ICCARM__ : IAR
 *
 *END**********************************************************************/
void init_data_bss(void)
{
    uint32_t n;

    /* Addresses for VECTOR_TABLE and VECTOR_RAM come from the linker file */
#if defined(__CC_ARM)
    extern uint32_t Image$$VECTOR_ROM$$Base[];
    extern uint32_t Image$$VECTOR_RAM$$Base[];
    extern uint32_t Image$$RW_m_data$$Base[];

    #define __VECTOR_TABLE Image$$VECTOR_ROM$$Base
    #define __VECTOR_RAM Image$$VECTOR_RAM$$Base
    #define __RAM_VECTOR_TABLE_SIZE (((uint32_t)Image$$RW_m_data$$Base -
(uint32_t)Image$$VECTOR_RAM$$Base))
#elif defined(__ICCARM__)
    extern uint32_t __RAM_VECTOR_TABLE_SIZE[];
    extern uint32_t __VECTOR_TABLE[];
    extern uint32_t __VECTOR_RAM[];
#elif defined(__GNUC__)
    extern uint32_t __VECTOR_TABLE[];
    extern uint32_t __VECTOR_RAM[];
    extern uint32_t __RAM_VECTOR_TABLE_SIZE_BYTES[];
    uint32_t __RAM_VECTOR_TABLE_SIZE = (uint32_t)(__RAM_VECTOR_TABLE_SIZE_BYTES);
#endif

    if (__VECTOR_RAM != __VECTOR_TABLE)
    {
```

```
        /* Copy the vector table from ROM to RAM */
        for (n = 0; n < ((uint32_t)__RAM_VECTOR_TABLE_SIZE)/sizeof(uint32_t); n++)
        {
            __VECTOR_RAM[n] = __VECTOR_TABLE[n];
        }
        /* Point the VTOR to the position of vector table */
        SCB->VTOR = (uint32_t)__VECTOR_RAM;
    }
    else
    {
        /* Point the VTOR to the position of vector table */
        SCB->VTOR = (uint32_t)__VECTOR_TABLE;
    }

#if !defined(__CC_ARM) && !defined(__ICCARM__)

    /* Declare pointers for various data sections. These pointers
     * are initialized using values pulled in from the linker file */
    uint8_t * data_ram, * data_rom, * data_rom_end;
    uint8_t * bss_start, * bss_end;

    /* Get the addresses for the .data section (initialized data section) */
#if defined(__GNUC__)
    extern uint32_t __DATA_ROM[];
    extern uint32_t __DATA_RAM[];
    extern char __DATA_END[];
    data_ram = (uint8_t *)__DATA_RAM;
    data_rom = (uint8_t *)__DATA_ROM;
    data_rom_end  = (uint8_t *)__DATA_END;
    n = data_rom_end - data_rom;
#endif

    /* Copy initialized data from ROM to RAM */
    while (n--)
    {
        *data_ram++ = *data_rom++;
    }

    /* Get the addresses for the .bss section (zero-initialized data) */
#if defined(__GNUC__)
    extern char __START_BSS[];
    extern char __END_BSS[];
    bss_start = (uint8_t *)__START_BSS;
    bss_end = (uint8_t *)__END_BSS;
#endif

    /* Clear the zero-initialized data section */
    n = bss_end - bss_start;
    while(n--)
    {
```

```
    *bss_start++ = 0;
  }
#endif /* !__CC_ARM && !__ICCARM__*/
}

/*******************************************************************************
 * EOF
 ******************************************************************************/
```

startup.h

```
#ifndef _STARTUP_H_
#define _STARTUP_H_

/*******************************************************************************
 * API
 ******************************************************************************/

/*!
 * @brief Make necessary initializations for RAM.
 *
 * - Copy initialized data from ROM to RAM.
 * - Clear the zero-initialized data section.
 * - Copy the vector table from ROM to RAM. This could be an option.
 */
void init_data_bss(void);

#endif /* _STARTUP_H_*/
/*******************************************************************************
 * EOF
 ******************************************************************************/
```

fsl_debug_console.c

```
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#if defined(UART_INSTANCE_COUNT)
#include "fsl_uart_hal.h"
#endif
#if defined(LPUART_INSTANCE_COUNT)
#include "fsl_lpuart_hal.h"
#endif
#if defined(UART0_INSTANCE_COUNT)
#include "fsl_lpsci_hal.h"
```

```
#endif
#include "fsl_clock_manager.h"
#include "fsl_os_abstraction.h"
#include "print_scan.h"

#if (defined(USB_INSTANCE_COUNT) &&  (defined(BOARD_USE_VIRTUALCOM)))
  #include "usb_device_config.h"
  #include "usb.h"
  #include "usb_device_stack_interface.h"
  #include "usb_descriptor.h"
  #include "virtual_com.h"
#endif

extern uint32_t g_app_handle;
#if __ICCARM__
#include <yfuns.h>
#endif

static int debug_putc(int ch, void* stream);

/*******************************************************************************
 * Definitions
 ******************************************************************************/

/*! @brief Operation functions definiations for debug console. */
typedef struct DebugConsoleOperationFunctions {
   union {
      void (* Send)(void *base, const uint8_t *buf, uint32_t count);
#if defined(UART_INSTANCE_COUNT)
      void (* UART_Send)(UART_Type *base, const uint8_t *buf, uint32_t count);
#endif
#if defined(LPUART_INSTANCE_COUNT)
      void (* LPUART_Send)(LPUART_Type* base, const uint8_t *buf, uint32_t count);
#endif
#if defined(UART0_INSTANCE_COUNT)
      void (* UART0_Send)(UART0_Type* base, const uint8_t *buf, uint32_t count);
#endif
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
      void (* USB_Send)(uint32_t base, const uint8_t *buf, uint32_t count);
#endif
   } tx_union;
   union{
      void (* Receive)(void *base, uint8_t *buf, uint32_t count);
#if defined(UART_INSTANCE_COUNT)
      uart_status_t (* UART_Receive)(UART_Type *base, uint8_t *buf, uint32_t count);
#endif
#if defined(LPUART_INSTANCE_COUNT)
      lpuart_status_t (* LPUART_Receive)(LPUART_Type* base, uint8_t *buf, uint32_t count);
#endif
#if defined(UART0_INSTANCE_COUNT)
```

```
        lpsci_status_t (* UART0_Receive)(UART0_Type* base, uint8_t *buf, uint32_t count);
#endif
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
        usb_status_t (* USB_Receive)(uint32_t base, uint8_t *buf, uint32_t count);
#endif

    } rx_union;
} debug_console_ops_t;

/*! @brief State structure storing debug console. */
typedef struct DebugConsoleState {
    debug_console_device_type_t type;/*<! Indicator telling whether the debug console is inited.
*/
    uint8_t instance;             /*<! Instance number indicator. */
    void*   base;                 /*<! Base of the IP register. */
    debug_console_ops_t ops;      /*<! Operation function pointers for debug uart operations. */
} debug_console_state_t;

/*******************************************************************************
 * Variables
 ******************************************************************************/
/*! @brief Debug UART state information.*/
static debug_console_state_t s_debugConsole;

/*******************************************************************************
 * Code
 ******************************************************************************/
/* See fsl_debug_console.h for documentation of this function.*/
debug_console_status_t DbgConsole_Init(
        uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t device)
{
    if (s_debugConsole.type != kDebugConsoleNone)
    {
        return kStatus_DEBUGCONSOLE_Failed;
    }

    /* Set debug console to initialized to avoid duplicated init operation.*/
    s_debugConsole.type = device;
    s_debugConsole.instance = uartInstance;

    /* Switch between different device. */
    switch (device)
    {
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))  /*&&
defined()*/
        case kDebugConsoleUSBCDC:
         {
              VirtualCom_Init();
              s_debugConsole.base = (void*)g_app_handle;
              s_debugConsole.ops.tx_union.USB_Send = VirtualCom_SendDataBlocking;
```

```
            s_debugConsole.ops.rx_union.USB_Receive = VirtualCom_ReceiveDataBlocking;
        }
        break;
#endif
#if defined(UART_INSTANCE_COUNT)
    case kDebugConsoleUART:
      {
            UART_Type * g_Base[UART_INSTANCE_COUNT] = UART_BASE_PTRS;
            UART_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableUartClock(uartInstance);

            /* UART clock source is either system or bus clock depending on instance */
            uartSourceClock = CLOCK_SYS_GetUartFreq(uartInstance);

            /* Initialize UART baud rate, bit count, parity and stop bit. */
            UART_HAL_SetBaudRate(base, uartSourceClock, baudRate);
            UART_HAL_SetBitCountPerChar(base, kUart8BitsPerChar);
            UART_HAL_SetParityMode(base, kUartParityDisabled);
#if FSL_FEATURE_UART_HAS_STOP_BIT_CONFIG_SUPPORT
            UART_HAL_SetStopBitCount(base, kUartOneStopBit);
#endif

            /* Finally, enable the UART transmitter and receiver*/
            UART_HAL_EnableTransmitter(base);
            UART_HAL_EnableReceiver(base);

            /* Set the funciton pointer for send and receive for this kind of device. */
            s_debugConsole.ops.tx_union.UART_Send = UART_HAL_SendDataPolling;
            s_debugConsole.ops.rx_union.UART_Receive = UART_HAL_ReceiveDataPolling;
        }
        break;
#endif
#if defined(UART0_INSTANCE_COUNT)
    case kDebugConsoleLPSCI:
      {
            /* Declare config sturcuture to initialize a uart instance. */
            UART0_Type * g_Base[UART0_INSTANCE_COUNT] = UART0_BASE_PTRS;
            UART0_Type * base = g_Base[uartInstance];
            uint32_t uartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableLpsciClock(uartInstance);

            uartSourceClock = CLOCK_SYS_GetLpsciFreq(uartInstance);

            /* Initialize LPSCI baud rate, bit count, parity and stop bit. */
            LPSCI_HAL_SetBaudRate(base, uartSourceClock, baudRate);
```

```
            LPSCI_HAL_SetBitCountPerChar(base, kLpsci8BitsPerChar);
            LPSCI_HAL_SetParityMode(base, kLpsciParityDisabled);
#if FSL_FEATURE_LPSCI_HAS_STOP_BIT_CONFIG_SUPPORT
            LPSCI_HAL_SetStopBitCount(base, kLpsciOneStopBit);
#endif

            /* Finally, enable the LPSCI transmitter and receiver*/
            LPSCI_HAL_EnableTransmitter(base);
            LPSCI_HAL_EnableReceiver(base);

            /* Set the funciton pointer for send and receive for this kind of device. */
            s_debugConsole.ops.tx_union.UART0_Send = LPSCI_HAL_SendDataPolling;
            s_debugConsole.ops.rx_union.UART0_Receive = LPSCI_HAL_ReceiveDataPolling;
        }
        break;
#endif
#if defined(LPUART_INSTANCE_COUNT)
    case kDebugConsoleLPUART:
        {
            LPUART_Type* g_Base[LPUART_INSTANCE_COUNT] = LPUART_BASE_PTRS;
            LPUART_Type* base = g_Base[uartInstance];
            uint32_t lpuartSourceClock;

            s_debugConsole.base = base;
            CLOCK_SYS_EnableLpuartClock(uartInstance);

            /* LPUART clock source is either system or bus clock depending on instance */
            lpuartSourceClock = CLOCK_SYS_GetLpuartFreq(uartInstance);

            /* initialize the parameters of the LPUART config structure with desired data */
            LPUART_HAL_SetBaudRate(base, lpuartSourceClock, baudRate);
            LPUART_HAL_SetBitCountPerChar(base, kLpuart8BitsPerChar);
            LPUART_HAL_SetParityMode(base, kLpuartParityDisabled);
            LPUART_HAL_SetStopBitCount(base, kLpuartOneStopBit);

            /* finally, enable the LPUART transmitter and receiver */
            LPUART_HAL_SetTransmitterCmd(base, true);
            LPUART_HAL_SetReceiverCmd(base, true);

            /* Set the funciton pointer for send and receive for this kind of device. */
            s_debugConsole.ops.tx_union.LPUART_Send = LPUART_HAL_SendDataPolling;
            s_debugConsole.ops.rx_union.LPUART_Receive = LPUART_HAL_ReceiveDataPolling;

        }
        break;
#endif
    /* If new device is requried as the low level device for debug console,
     * Add the case branch and add the preprocessor macro to judge whether
     * this kind of device exist in this SOC. */
    default:
```

```
        /* Device identified is invalid, return invalid device error code. */
        return kStatus_DEBUGCONSOLE_InvalidDevice;
    }

    /* Configure the s_debugConsole structure only when the inti operation is successful. */
    s_debugConsole.instance = uartInstance;

    return kStatus_DEBUGCONSOLE_Success;
}

/* See fsl_debug_console.h for documentation of this function.*/
debug_console_status_t DbgConsole_DeInit(void)
{
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return kStatus_DEBUGCONSOLE_Success;
    }

    switch(s_debugConsole.type)
    {
#if defined(UART_INSTANCE_COUNT)
        case kDebugConsoleUART:
            CLOCK_SYS_DisableUartClock(s_debugConsole.instance);
            break;
#endif
#if defined(UART0_INSTANCE_COUNT)
        case kDebugConsoleLPSCI:
            CLOCK_SYS_DisableLpsciClock(s_debugConsole.instance);
            break;
#endif
#if defined(LPUART_INSTANCE_COUNT)
        case kDebugConsoleLPUART:
             CLOCK_SYS_DisableLpuartClock(s_debugConsole.instance);
            break;
#endif
#if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
        case kDebugConsoleUSBCDC:
             VirtualCom_Deinit();
             CLOCK_SYS_DisableUsbfsClock(0);
            break;
#endif
        default:
            return kStatus_DEBUGCONSOLE_InvalidDevice;
    }

    s_debugConsole.type = kDebugConsoleNone;
    return kStatus_DEBUGCONSOLE_Success;
}

#if (defined(__KSDK_STDLIB__))
```

```
int _WRITE(int fd, const void *buf, size_t nbytes)
{
    if (buf == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
        /* handles should be flushed.)*/
        return 0;
    }


    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buf, nbytes);
    return nbytes;

}

int _READ(int fd, void *buf, size_t nbytes)
{

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buf, nbytes);
    return nbytes;
}
#elif __ICCARM__

#pragma weak __write
size_t __write(int handle, const unsigned char * buffer, size_t size)
{
    if (buffer == 0)
    {
        /* This means that we should flush internal buffers.  Since we*/
        /* don't we just return.  (Remember, "handle" == -1 means that all*/
        /* handles should be flushed.)*/
        return 0;
    }

    /* This function only writes to "standard out" and "standard err",*/
```

```
    /* for all other file handles it returns failure.*/
    if ((handle != _LLIO_STDOUT) && (handle != _LLIO_STDERR))
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buffer, size);
    return size;
}

#pragma weak __read
size_t __read(int handle, unsigned char * buffer, size_t size)
{
    /* This function only reads from "standard in", for all other file*/
    /* handles it returns failure.*/
    if (handle != _LLIO_STDIN)
    {
        return _LLIO_ERROR;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return _LLIO_ERROR;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buffer, size);

    return size;
}

#elif (defined(__GNUC__))
#pragma weak _write
int _write (int handle, char *buffer, int size)
{
    if (buffer == 0)
    {
        /* return -1 if error */
        return -1;
    }

    /* This function only writes to "standard out" and "standard err",*/
```

```
    /* for all other file handles it returns failure.*/
    if ((handle != 1) && (handle != 2))
    {
        return -1;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Send data.*/
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t *)buffer, size);
    return size;
}

#pragma weak _read
int _read(int handle, char *buffer, int size)
{
    /* This function only reads from "standard in", for all other file*/
    /* handles it returns failure.*/
    if (handle != 0)
    {
        return -1;
    }

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }

    /* Receive data.*/
    s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, (uint8_t *)buffer, size);
    return size;
}
#elif defined(__CC_ARM) && !defined(MQX_STDIO)
struct __FILE
{
    int handle;
    /* Whatever you require here. If the only file you are using is */
    /* standard output using printf() for debugging, no file handling */
    /* is required. */
};

/* FILE is typedef in stdio.h. */
#pragma weak __stdout
FILE __stdout;
FILE __stdin;
```

```
#pragma weak fputc
int fputc(int ch, FILE *f)
{
  /* Do nothing if the debug uart is not initialized.*/
  if (s_debugConsole.type == kDebugConsoleNone)
  {
    return -1;
  }

  /* Send data.*/
  s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (const uint8_t*)&ch, 1);
  return 1;
}

#pragma weak fgetc
int fgetc(FILE *f)
{
  uint8_t temp;
  /* Do nothing if the debug uart is not initialized.*/
  if (s_debugConsole.type == kDebugConsoleNone)
  {
    return -1;
  }

  /* Receive data.*/
  s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &temp, 1);
  return temp;
}

#endif

/*************Code for debug_printf/scanf/assert*****************************/
int debug_printf(const char  *fmt_s, ...)
{
  va_list  ap;
  int  result;
  /* Do nothing if the debug uart is not initialized.*/
  if (s_debugConsole.type == kDebugConsoleNone)
  {
    return -1;
  }
  va_start(ap, fmt_s);
  result = _doprint(NULL, debug_putc, -1, (char *)fmt_s, ap);
  va_end(ap);

  return result;
}

static int debug_putc(int ch, void* stream)
```

```
{
    const unsigned char c = (unsigned char) ch;
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    s_debugConsole.ops.tx_union.Send(s_debugConsole.base, &c, 1);

    return 0;

}

int debug_putchar(int ch)
{
    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    debug_putc(ch, NULL);

    return 1;
}

int debug_scanf(const char  *fmt_ptr, ...)
{
    char    temp_buf[IO_MAXLINE];
    va_list ap;
    uint32_t i;
    char result;

    /* Do nothing if the debug uart is not initialized.*/
    if (s_debugConsole.type == kDebugConsoleNone)
    {
        return -1;
    }
    va_start(ap, fmt_ptr);
    temp_buf[0] = '\0';

    for (i = 0; i < IO_MAXLINE; i++)
    {
        temp_buf[i] = result = debug_getchar();

        if ((result == '\r') || (result == '\n'))
        {
            /* End of Line */
            if (i == 0)
            {
                i = (uint32_t)-1;
```

```
        }
        else
        {
            break;
        }
      }

      temp_buf[i + 1] = '\0';
   }

   result = scan_prv(temp_buf, (char *)fmt_ptr, ap);
   va_end(ap);

  return result;
}

int debug_getchar(void)
{
   unsigned char c;

   /* Do nothing if the debug uart is not initialized.*/
   if (s_debugConsole.type == kDebugConsoleNone)
   {
      return -1;
   }
   s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &c, 1);

   return c;
}
/*******************************************************************************
 * EOF
 ******************************************************************************/
```

Fsl_debug_console.h

```
#if !defined(__FSL_DEBUG_CONSOLE_H__)
#define __FSL_DEBUG_CONSOLE_H__

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

/*
 * @addtogroup debug_console
 * @{
 */

/*******************************************************************************
 * Definitions
```

```
 ******************************************************************************/
#define IO_MAXLINE  20

#if (defined (FSL_RTOS_MQX) && (MQX_COMMON_CONFIG != MQX_LITE_CONFIG))
#define PRINTF        printf
#define SCANF         scanf
#define PUTCHAR       putchar
#define GETCHAR       getchar
#include <stdio.h>
#else
/*Configuration for toolchain's printf/scanf or KSDK version printf/scanf */
#define PRINTF        debug_printf
//#define PRINTF        printf
#define SCANF         debug_scanf
//#define SCANF         scanf
#define PUTCHAR       debug_putchar
//#define PUTCHAR       putchar
#define GETCHAR       debug_getchar
//#define GETCHAR       getchar
#endif

/*! @brief Error code for the debug console driver. */
typedef enum _debug_console_status {
   kStatus_DEBUGCONSOLE_Success = 0U,
   kStatus_DEBUGCONSOLE_InvalidDevice,
   kStatus_DEBUGCONSOLE_AllocateMemoryFailed,
   kStatus_DEBUGCONSOLE_Failed
} debug_console_status_t;

/*! @brief Supported debug console hardware device type. */
typedef enum _debug_console_device_type {
   kDebugConsoleNone   = 0U,
   kDebugConsoleLPSCI  = 15U,    /*<! Use strange start number to avoid treating 0
                        as correct device type. Sometimes user forget
                        to specify the device type but only use the
                        default value '0' as the device type.  */
   kDebugConsoleUART   = 16U,
   kDebugConsoleLPUART = 17U,
   kDebugConsoleUSBCDC = 18U
} debug_console_device_type_t;

/*******************************************************************************
 * API
 ******************************************************************************/

#if defined(__cplusplus)
extern "C" {
#endif

/*! @name Initialization*/
```

```
/*@{*/

/*!
 * @brief Init the UART/LPUART used for debug messages.
 *
 * Call this function to enable debug log messages to be output via the specified UART/LPUART
 * base address and at the specified baud rate. Just initializes the UART/LPUART to the given baud
 * rate and 8N1. After this function has returned, stdout and stdin will be connected to the
 * selected UART/LPUART. The debug_printf() function also uses this UART/LPUART.
 *
 * @param uartInstance Which UART/LPUART instance is used to send debug messages.
 * @param baudRate The desired baud rate in bits per second.
 * @param device Low level device type for the debug console.
 * @return Whether initialization was successful or not.
 */
debug_console_status_t DbgConsole_Init(
      uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t device);

/*!
 * @brief Deinit the UART/LPUART used for debug messages.
 *
 * Call this function to disable debug log messages to be output via the specified UART/LPUART
 * base address and at the specified baud rate.
 * @return Whether de-initialization was successful or not.
 */
debug_console_status_t DbgConsole_DeInit(void);

/*!
 * @brief   Prints formatted output to the standard output stream.
 *
 * Call this function to print formatted output to the standard output stream.
 *
 * @param   fmt_s   Format control string.
 * @return  Returns the number of characters printed, or a negative value if an error occurs.
 */
int debug_printf(const char  *fmt_s, ...);

/*!
 * @brief   Writes a character to stdout.
 *
 * Call this function to write a character to stdout.
 *
 * @param   ch  Character to be written.
 * @return  Returns the character written.
 */
int debug_putchar(int ch);

/*!
 * @brief   Reads formatted data from the standard input stream.
```

```
 *
 * Call this function to read formatted data from the standard input stream.
 *
 * @param   fmt_ptr Format control string.
 * @return  Returns the number of fields successfully converted and assigned.
 */
int debug_scanf(const char  *fmt_ptr, ...);

/*!
 * @brief   Reads a character from standard input.
 *
 * Call this function to read a character from standard input.
 *
 * @return  Returns the character read.
 */
int debug_getchar(void);

/*@}*/

#if defined(__cplusplus)
}
#endif

/*! @}*/

#endif /* __FSL_DEBUG_CONSOLE_H__*/
/*******************************************************************************
 * EOF
 ******************************************************************************/
```

fsl_misc_utilities.c

```
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include "fsl_misc_utilities.h"
#if defined(__GNUC__)
#include <errno.h>
#endif
#include "fsl_debug_console.h"

#if (defined(__CC_ARM))

/*FUNCTION**********************************************************************
*
 *
 * Function Name : __aeabi_assert
 * Description   : called by assert in KEIL
```

```
 * This function is called by the assert function in KEIL.
 *
 *END*********************************************************************/
void __aeabi_assert(const char *expr, const char *file, int line)
{
   printf("assert failed:%s, file %s:%d\r\n",expr,file,line);
}

#endif

#if defined(__GNUC__)
caddr_t
_sbrk (int incr)
{
  extern char   end __asm ("end");
  extern char   heap_limit __asm ("__HeapLimit");
  static char * heap_end;
  char *        prev_heap_end;

  if (heap_end == NULL)
    heap_end = & end;

  prev_heap_end = heap_end;

  if (heap_end + incr > &heap_limit)
    {
#ifdef NIO_ENOMEM   //TODO: Update NIO error code for MQX
       errno = NIO_ENOMEM;
#else
       errno = ENOMEM;
#endif
     return (caddr_t) -1;
    }

  heap_end += incr;

  return (caddr_t) prev_heap_end;
}
#endif

/*FUNCTION*********************************************************************
*
 *
 * Function Name : assert_func
 * Description   : Print out failure messages.
 * This function is used to print out failure messages.
 *
 *END*********************************************************************/
void assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
```

```
    PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n",
failedExpr, file , line, func);

    for (;;)
    {}

}
/*****************************************************************************
 * EOF
 *****************************************************************************/
```

---

print_scan.c

```
#include "print_scan.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdint.h>
#include <stdbool.h>
// Keil: suppress ellipsis warning in va_arg usage below
#if defined(__CC_ARM)
#pragma diag_suppress 1256
#endif

#define FLAGS_MINUS     (0x01)
#define FLAGS_PLUS      (0x02)
#define FLAGS_SPACE     (0x04)
#define FLAGS_ZERO      (0x08)
#define FLAGS_POUND     (0x10)

#define IS_FLAG_MINUS(a)    (a & FLAGS_MINUS)
#define IS_FLAG_PLUS(a)     (a & FLAGS_PLUS)
#define IS_FLAG_SPACE(a)    (a & FLAGS_SPACE)
#define IS_FLAG_ZERO(a)     (a & FLAGS_ZERO)
#define IS_FLAG_POUND(a)    (a & FLAGS_POUND)

#define LENMOD_h        (0x01)
#define LENMOD_l        (0x02)
#define LENMOD_L        (0x04)
#define LENMOD_hh       (0x08)
#define LENMOD_ll       (0x10)

#define IS_LENMOD_h(a)  (a & LENMOD_h)
#define IS_LENMOD_hh(a) (a & LENMOD_hh)
#define IS_LENMOD_l(a)  (a & LENMOD_l)
#define IS_LENMOD_ll(a) (a & LENMOD_ll)
#define IS_LENMOD_L(a)  (a & LENMOD_L)
```

```
#define SCAN_SUPPRESS            0x2

#define SCAN_DEST_MASK           0x7c
#define SCAN_DEST_CHAR           0x4
#define SCAN_DEST_STRING         0x8
#define SCAN_DEST_SET            0x10
#define SCAN_DEST_INT            0x20
#define SCAN_DEST_FLOAT          0x30

#define SCAN_LENGTH_MASK         0x1f00
#define SCAN_LENGTH_CHAR         0x100
#define SCAN_LENGTH_SHORT_INT    0x200
#define SCAN_LENGTH_LONG_INT     0x400
#define SCAN_LENGTH_LONG_LONG_INT 0x800
#define SCAN_LENGTH_LONG_DOUBLE   0x1000

#define SCAN_TYPE_SIGNED         0x2000

/*!
 * @brief Scanline function which ignores white spaces.
 *
 * @param[in] s The address of the string pointer to update.
 *
 * @return String without white spaces.
 */
static uint32_t scan_ignore_white_space(const char **s);

#if defined(SCANF_FLOAT_ENABLE)
static double fnum = 0.0;
#endif

/*!
 * @brief Converts a radix number to a string and return its length.
 *
 * @param[in] numstr    Converted string of the number.
 * @param[in] nump      Pointer to the number.
 * @param[in] neg       Polarity of the number.
 * @param[in] radix     The radix to be converted to.
 * @param[in] use_caps  Used to identify %x/X output format.

 * @return Length of the converted string.
 */
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps);

#if defined(PRINTF_FLOAT_ENABLE)
/*!
 * @brief Converts a floating radix number to a string and return its length.
 *
 * @param[in] numstr          Converted string of the number.
 * @param[in] nump            Pointer to the number.
```

```
 * @param[in] radix           The radix to be converted to.
 * @param[in] precision_width   Specify the precision width.

 * @return Length of the converted string.
 */
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t
precision_width);
#endif


static void fput_pad(int32_t c, int32_t curlen, int32_t field_width, int32_t *count,
PUTCHAR_FUNC func_ptr, void *farg, int *max_count);

double modf(double input_dbl, double  *intpart_ptr);

#if !defined(PRINT_MAX_COUNT)
#define n_putchar(func, chacter, p, count)      func(chacter, p)
#else
static int n_putchar(PUTCHAR_FUNC func_ptr, int chacter, void *p, int *max_count)
{
   int result = 0;
   if (*max_count)
   {
      result = func_ptr(chacter, p);
      (*max_count)--;
   }
   return result;
}
#endif

/*FUNCTION*****************************************************************
*
 *
 * Function Name : _doprint
 * Description   : This function outputs its parameters according to a
 * formatted string. I/O is performed by calling given function pointer
 * using following (*func_ptr)(c,farg);
 *
 *END*********************************************************************/
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char *fmt, va_list ap)
{
   /* va_list ap; */
   char *p;
   int32_t c;

   char vstr[33];
   char *vstrp;
   int32_t vlen;

   int32_t done;
```

```
    int32_t count = 0;
    int temp_count = max_count;


    uint32_t flags_used;
    uint32_t field_width;

    int32_t ival;
    int32_t schar, dschar;
    int32_t *ivalp;
    char *sval;
    int32_t cval;
    uint32_t uval;
    bool use_caps;
    uint32_t precision_width;
    //uint32_t length_modifier = 0;
#if defined(PRINTF_FLOAT_ENABLE)
    double fval;
#endif

    if (max_count == -1)
    {
        max_count = INT32_MAX - 1;
    }

    /*
     * Start parsing apart the format string and display appropriate
     * formats and data.
     */
    for (p = (char *)fmt; (c = *p) != 0; p++)
    {
        /*
         * All formats begin with a '%' marker.  Special chars like
         * '\n' or '\t' are normally converted to the appropriate
         * character by the __compiler__.  Thus, no need for this
         * routine to account for the '\' character.
         */
        if (c != '%')
        {
            n_putchar(func_ptr, c, farg, &max_count);

            count++;

            /*
             * By using 'continue', the next iteration of the loop
             * is used, skipping the code that follows.
             */
            continue;
        }
```

```
      /*
       * First check for specification modifier flags.
       */
      use_caps = true;
      flags_used = 0;
      done = false;
      while (!done)
      {
        switch (/* c = */ *++p)
        {
          case '-':
            flags_used |= FLAGS_MINUS;
            break;
          case '+':
            flags_used |= FLAGS_PLUS;
            break;
          case ' ':
            flags_used |= FLAGS_SPACE;
            break;
          case '0':
            flags_used |= FLAGS_ZERO;
            break;
          case '#':
            flags_used |= FLAGS_POUND;
            break;
          default:
            /* we've gone one char too far */
            --p;
            done = true;
            break;
        }
      }

      /*
       * Next check for minimum field width.
       */
      field_width = 0;
      done = false;
      while (!done)
      {
        switch (c = *++p)
        {
          case '0':
          case '1':
          case '2':
          case '3':
          case '4':
          case '5':
          case '6':
          case '7':
```

```
            case '8':
            case '9':
               field_width = (field_width * 10) + (c - '0');
               break;
            default:
               /* we've gone one char too far */
               --p;
               done = true;
               break;
         }
      }

      /*
       * Next check for the width and precision field separator.
       */
      precision_width = 6;
      if (/* (c = *++p) */ *++p == '.')
      {
         /* precision_used = true; */

         /*
          * Must get precision field width, if present.
          */
         precision_width = 0;
         done = false;
         while (!done)
         {
            switch (c = *++p)
            {
               case '0':
               case '1':
               case '2':
               case '3':
               case '4':
               case '5':
               case '6':
               case '7':
               case '8':
               case '9':
                  precision_width = (precision_width * 10) + (c - '0');
                  break;
               default:
                  /* we've gone one char too far */
                  --p;
                  done = true;
                  break;
            }
         }
      }
      else
```

```
    {
      /* we've gone one char too far */
      --p;
    }

    /*
     * Check for the length modifier.
     */
    /* length_modifier = 0; */
    switch (/* c = */ *++p)
    {
      case 'h':
        if (*++p != 'h')
        {
          --p;
        }
        /* length_modifier |= LENMOD_h; */
        break;
      case 'l':
        if (*++p != 'l')
        {
          --p;
        }
        /* length_modifier |= LENMOD_l; */
        break;
      case 'L':
        /* length_modifier |= LENMOD_L; */
        break;
      default:
        /* we've gone one char too far */
        --p;
        break;
    }

    /*
     * Now we're ready to examine the format.
     */
    switch (c = *++p)
    {
      case 'd':
      case 'i':
        ival = (int32_t)va_arg(ap, int32_t);
        vlen = mknumstr(vstr,&ival,true,10,use_caps);
        vstrp = &vstr[vlen];

        if (ival < 0)
        {
          schar = '-';
          ++vlen;
        }
```

```
        else
        {
          if (IS_FLAG_PLUS(flags_used))
          {
            schar = '+';
            ++vlen;
          }
          else
          {
            if (IS_FLAG_SPACE(flags_used))
            {
              schar = ' ';
              ++vlen;
            }
            else
            {
              schar = 0;
            }
          }
        }
        dschar = false;

        /*
         * do the ZERO pad.
         */
        if (IS_FLAG_ZERO(flags_used))
        {
          if (schar)
          {
            n_putchar(func_ptr, schar, farg, &max_count);
            count++;
          }
          dschar = true;

          fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
          vlen = field_width;
        }
        else
        {
          if (!IS_FLAG_MINUS(flags_used))
          {
            fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
            if (schar)
            {
              n_putchar(func_ptr, schar, farg, &max_count);
              count++;
            }
            dschar = true;
          }
        }
```

```
            /* the string was built in reverse order, now display in */
            /* correct order */
            if ((!dschar) && schar)
            {
               n_putchar(func_ptr, schar, farg, &max_count);
               count++;
            }
            goto cont_xd;
 #if defined(PRINTF_FLOAT_ENABLE)
          case 'f':
          case 'F':
            fval = (double)va_arg(ap, double);
            vlen = mkfloatnumstr(vstr,&fval,10, precision_width);
            vstrp = &vstr[vlen];

            if (fval < 0)
            {
               schar = '-';
               ++vlen;
            }
            else
            {
               if (IS_FLAG_PLUS(flags_used))
               {
                  schar = '+';
                  ++vlen;
               }
               else
               {
                  if (IS_FLAG_SPACE(flags_used))
                  {
                     schar = ' ';
                     ++vlen;
                  }
                  else
                  {
                     schar = 0;
                  }
               }
            }
            dschar = false;
            if (IS_FLAG_ZERO(flags_used))
            {
               if (schar)
               {
                  n_putchar(func_ptr, schar, farg, &max_count);
                  count++;
               }
               dschar = true;
```

```
                fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
                vlen = field_width;
            }
            else
            {
                if (!IS_FLAG_MINUS(flags_used))
                {
                    fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                    if (schar)
                    {
                        n_putchar(func_ptr, schar, farg, &max_count);
                        count++;
                    }
                    dschar = true;
                }
            }
            if (!dschar && schar)
            {
                n_putchar(func_ptr, schar, farg, &max_count);
                count++;
            }
            goto cont_xd;
#endif
        case 'x':
            use_caps = false;
        case 'X':
            uval = (uint32_t)va_arg(ap, uint32_t);
            vlen = mknumstr(vstr,&uval,false,16,use_caps);
            vstrp = &vstr[vlen];

            dschar = false;
            if (IS_FLAG_ZERO(flags_used))
            {
                if (IS_FLAG_POUND(flags_used))
                {
                    n_putchar(func_ptr, '0', farg, &max_count);
                    n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
                    count += 2;
                    /*vlen += 2;*/
                    dschar = true;
                }
                fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
                vlen = field_width;
            }
            else
            {
                if (!IS_FLAG_MINUS(flags_used))
                {
                    if (IS_FLAG_POUND(flags_used))
                    {
```

```
                vlen += 2;
              }
              fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
              if (IS_FLAG_POUND(flags_used))
              {
                n_putchar(func_ptr, '0', farg, &max_count);
                n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
                count += 2;

                dschar = true;
              }
            }
          }

          if ((IS_FLAG_POUND(flags_used)) && (!dschar))
          {
            n_putchar(func_ptr, '0', farg, &max_count);
            n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
            count += 2;
            vlen += 2;
          }
          goto cont_xd;

        case 'o':
          uval = (uint32_t)va_arg(ap, uint32_t);
          vlen = mknumstr(vstr,&uval,false,8,use_caps);
          goto cont_u;
        case 'b':
          uval = (uint32_t)va_arg(ap, uint32_t);
          vlen = mknumstr(vstr,&uval,false,2,use_caps);
          goto cont_u;
        case 'p':
          uval = (uint32_t)va_arg(ap, uint32_t);
          uval = (uint32_t)va_arg(ap, void *);
          vlen = mknumstr(vstr,&uval,false,16,use_caps);
          goto cont_u;
        case 'u':
          uval = (uint32_t)va_arg(ap, uint32_t);
          vlen = mknumstr(vstr,&uval,false,10,use_caps);

          cont_u:
            vstrp = &vstr[vlen];

            if (IS_FLAG_ZERO(flags_used))
            {
              fput_pad('0', vlen, field_width, &count, func_ptr, farg, &max_count);
              vlen = field_width;
            }
            else
            {
```

```
                if (!IS_FLAG_MINUS(flags_used))
                {
                   fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
                }
             }

          cont_xd:
             while (*vstrp)
             {
                n_putchar(func_ptr, *vstrp--, farg, &max_count);
                count++;
             }

             if (IS_FLAG_MINUS(flags_used))
             {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
             }
          break;

       case 'c':
          cval = (char)va_arg(ap, uint32_t);
          n_putchar(func_ptr, cval, farg, &max_count);
          count++;
          break;
       case 's':
          sval = (char *)va_arg(ap, char *);
          if (sval)
          {
             vlen = strlen(sval);
             if (!IS_FLAG_MINUS(flags_used))
             {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
             }
             while (*sval)
             {
                n_putchar(func_ptr, *sval++, farg, &max_count);
                count++;
             }
             if (IS_FLAG_MINUS(flags_used))
             {
                fput_pad(' ', vlen, field_width, &count, func_ptr, farg, &max_count);
             }
          }
          break;
       case 'n':
          ivalp = (int32_t *)va_arg(ap, int32_t *);
          *ivalp = count;
          break;
       default:
          n_putchar(func_ptr, c, farg, &max_count);
```

```
            count++;
            break;
        }
    }

    if (max_count)
    {
        return count;
    }
    else
    {
        return temp_count;
    }
}

/*FUNCTION*****************************************************************
*
 *
 * Function Name : _sputc
 * Description   : Writes the character into the string located by the string
 * pointer and updates the string pointer.
 *
 *END*****************************************************************/
int _sputc(int c, void * input_string)
{
   char **string_ptr = (char  **)input_string;

   *(*string_ptr)++ = (char)c;
   return c;
}

/*FUNCTION*****************************************************************
*
 *
 * Function Name : mknumstr
 * Description   : Converts a radix number to a string and return its length.
 *
 *END*****************************************************************/
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps)
{
    int32_t a,b,c;
    uint32_t ua,ub,uc;

    int32_t nlen;
    char *nstrp;

    nlen = 0;
    nstrp = numstr;
    *nstrp++ = '\0';
```

```
    if (neg)
    {
        a = *(int32_t *)nump;
        if (a == 0)
        {
            *nstrp = '0';
            ++nlen;
            goto done;
        }
        while (a != 0)
        {
            b = (int32_t)a / (int32_t)radix;
            c = (int32_t)a - ((int32_t)b * (int32_t)radix);
            if (c < 0)
            {
                c = ~c + 1 + '0';
            }
            else
            {
                c = c + '0';
            }
            a = b;
            *nstrp++ = (char)c;
            ++nlen;
        }
    }
    else
    {
        ua = *(uint32_t *)nump;
        if (ua == 0)
        {
            *nstrp = '0';
            ++nlen;
            goto done;
        }
        while (ua != 0)
        {
            ub = (uint32_t)ua / (uint32_t)radix;
            uc = (uint32_t)ua - ((uint32_t)ub * (uint32_t)radix);
            if (uc < 10)
            {
                uc = uc + '0';
            }
            else
            {
                uc = uc - 10 + (use_caps ? 'A' : 'a');
            }
            ua = ub;
            *nstrp++ = (char)uc;
            ++nlen;
```

```
        }
    }
    done:
    return nlen;
}

#if defined(PRINTF_FLOAT_ENABLE)
/*FUNCTION*****************************************************************
*
 *
 * Function Name : mkfloatnumstr
 * Description   : Converts a floating radix number to a string and return
 * its length, user can specify output precision width.
 *
 *END*****************************************************************/
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t
precision_width)
{
    int32_t a,b,c,i;
    double fa,fb;
    double r, fractpart, intpart;

    int32_t nlen;
    char *nstrp;
    nlen = 0;
    nstrp = numstr;
    *nstrp++ = '\0';
    r = *(double *)nump;
    if (r == 0)
    {
        *nstrp = '0';
        ++nlen;
        goto done;
    }
    fractpart = modf((double)r , (double *)&intpart);
    /* Process fractional part */
    for (i = 0; i < precision_width; i++)
    {
        fractpart *= radix;
    }
    //a = (int32_t)floor(fractpart + (double)0.5);
    if (r >= 0)
    {
        fa = fractpart + (double)0.5;
    }
    else
    {
        fa = fractpart - (double)0.5;
    }
    intpart += ((int64_t)fa - (int64_t)fractpart);
```

```
    for (i = 0; i < precision_width; i++)
    {
       fb = fa / (int32_t)radix;
       c = (int32_t)(fa - (int64_t)fb * (int32_t)radix);
       if (c < 0)
       {
          c = ~c + 1 + '0';
       }else
       {
          c = c + '0';
       }
       fa = fb;
       *nstrp++ = (char)c;
       ++nlen;
    }
    *nstrp++ = (char)'.';
    ++nlen;
    a = (int32_t)intpart;
    if(a == 0)
    {
       *nstrp++ = '0';
       ++nlen;
    }
    else
    {
       while (a != 0)
       {
          b = (int32_t)a / (int32_t)radix;
          c = (int32_t)a - ((int32_t)b * (int32_t)radix);
          if (c < 0)
          {
             c = ~c + 1 + '0';
          }else
          {
             c = c + '0';
          }
          a = b;
          *nstrp++ = (char)c;
          ++nlen;
       }
    }
    done:
    return nlen;
}
#endif

static void fput_pad(int32_t c, int32_t curlen, int32_t field_width, int32_t *count,
PUTCHAR_FUNC func_ptr, void *farg, int *max_count)
{
    int32_t i;
```

```
    for (i = curlen; i < field_width; i++)
    {
        func_ptr((char)c, farg);
        (*count)++;
    }
}

/*FUNCTION*****************************************************************
*
 *
 * Function Name : scan_prv
 * Description   : Converts an input line of ASCII characters based upon a
 * provided string format.
 *
 *END*********************************************************************/
int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
{
    uint8_t base;
    /* Identifier for the format string */
    char *c = format;
    const char *s;
    char temp;
    /* Identifier for the input string */
    const char *p = line_ptr;
    /* flag telling the conversion specification */
    uint32_t flag = 0 ;
    /* filed width for the matching input streams */
    uint32_t field_width;
    /* how many arguments are assigned except the suppress */
    uint32_t nassigned = 0;
    /* how many characters are read from the input streams */
    uint32_t n_decode = 0;

    int32_t val;
    char *buf;
    int8_t neg;

    /* return EOF error before any convernsion */
    if (*p == '\0')
    {
        return EOF;
    }

    /* decode directives */
    while ((*c) && (*p))
    {
        /* ignore all white-spaces in the format strings */
        if (scan_ignore_white_space((const char **)&c))
        {
```

```
            n_decode += scan_ignore_white_space(&p);
        }
        else if (*c != '%')
        {
            /* Ordinary characters */
            c++;
ordinary:   if (*p == *c)
            {
                n_decode++;
                p++;
                c++;
            }
            else
            {
                /* Match failure. Misalignment with C99, the unmatched
                 * characters need to be pushed back to stream. HOwever
                 * , it is deserted now. */
                break;
            }
        }
        else
        {
            /* convernsion specification */
            c++;
            if (*c == '%')
            {
                goto ordinary;
            }

            /* Reset */
            flag = 0;
            field_width = 0;
            base = 0;

            /* Loop to get full conversion specification */
            while ((*c) && (!(flag & SCAN_DEST_MASK)))
            {
                switch (*c)
                {
                    case '*':
                        if (flag & SCAN_SUPPRESS)
                        {
                            /* Match failure*/
                            return nassigned;
                        }
                        flag |= SCAN_SUPPRESS;
                        c++;
                        break;
                    case 'h':
                        if (flag & SCAN_LENGTH_MASK)
```

```
                {
                  /* Match failure*/
                  return nassigned;
                }
                flag |= SCAN_LENGTH_SHORT_INT;

                if (c[1] == 'h')
                {
                  flag |= SCAN_LENGTH_CHAR;
                  c++;
                }
                c++;
                break;
              case 'l':
                if (flag & SCAN_LENGTH_MASK)
                {
                  /* Match failure*/
                  return nassigned;
                }
                flag |= SCAN_LENGTH_LONG_INT;

                if (c[1] == 'l')
                {
                  flag |= SCAN_LENGTH_LONG_LONG_INT;
                  c++;
                }
                c++;
                break;
#if defined(ADVANCE)
              case 'j':
                if (flag & SCAN_LENGTH_MASK)
                {
                  /* Match failure*/
                  return nassigned;
                }
                flag |= SCAN_LENGTH_INTMAX;
                c++
              case 'z'
                if (flag & SCAN_LENGTH_MASK)
                {
                  /* Match failure*/
                  return nassigned;
                }
                flag |= SCAN_LENGTH_SIZE_T;
                c++;
                break;
              case 't':
                if (flag & SCAN_LENGTH_MASK)
                {
                  /* Match failure*/
```

```
                    return nassigned;
                }
                flag |= SCAN_LENGTH_PTRDIFF_T;
                c++;
                break;
#endif
#if defined(SCANF_FLOAT_ENABLE)
            case 'L':
                if (flag & SCAN_LENGTH_MASK)
                {
                    /* Match failure*/
                    return nassigned;
                }
                flag |= SCAN_LENGTH_LONG_DOUBLE;
                c++;
                break;
#endif
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                if (field_width)
                {
                    /* Match failure*/
                    return nassigned;
                }
                do {
                    field_width = field_width * 10 + *c - '0';
                    c++;
                } while ((*c >= '0') && (*c <= '9'));
                break;
            case 'd':
                flag |= SCAN_TYPE_SIGNED;
            case 'u':
                base = 10;
                flag |= SCAN_DEST_INT;
                c++;
                break;
            case 'o':
                base = 8;
                flag |= SCAN_DEST_INT;
                c++;
                break;
            case 'x':
```

```
                case 'X':
                   base = 16;
                   flag |= SCAN_DEST_INT;
                   c++;
                   break;
                case 'i':
                   base = 0;
                   flag |= SCAN_DEST_INT;
                   c++;
                   break;
#if defined(SCANF_FLOAT_ENABLE)
                case 'a':
                case 'A':
                case 'e':
                case 'E':
                case 'f':
                case 'F':
                case 'g':
                case 'G':
                   flag |= SCAN_DEST_FLOAT;
                   c++;
                   break;
#endif
                case 'c':
                   flag |= SCAN_DEST_CHAR;
                   if (!field_width)
                   {
                      field_width = 1;
                   }
                   c++;
                   break;
                case 's':
                   flag |= SCAN_DEST_STRING;
                   c++;
                   break;
#if defined(ADVANCE) /* [x]*/
                case '[':
                   flag |= SCAN_DEST_SET;
                   /*Add Set functionality */
                   break;
#endif
                default:
#if defined(SCAN_DEBUG)
                   printf("Unrecognized expression specifier: %c format: %s, number is: %d\r\n", c,
format, nassigned);
#endif
                   return nassigned;
             }
          }
```

```
        if (!(flag & SCAN_DEST_MASK))
        {
          /* Format strings are exausted */
          return nassigned;
        }

        if (!field_width)
        {
          /* Larget then length of a line */
          field_width = 99;
        }

        /* Matching strings in input streams and assign to argument */
        switch (flag & SCAN_DEST_MASK)
        {
          case SCAN_DEST_CHAR:
            s = (const char *)p;
            buf = va_arg(args_ptr, char *);
            while ((field_width--) && (*p))
            {
              if (!(flag & SCAN_SUPPRESS))
              {
                *buf++ = *p++;
              }
              else
              {
                p++;
              }
              n_decode++;
            }

            if (((!(flag)) & SCAN_SUPPRESS) && (s != p))
            {
              nassigned++;
            }
            break;
          case SCAN_DEST_STRING:
            n_decode += scan_ignore_white_space(&p);
            s = p;
            buf = va_arg(args_ptr, char *);
            while ((field_width--) && (*p != '\0') && (*p != ' ') &&
                   (*p != '\t') && (*p != '\n') && (*p != '\r') && (*p != '\v') && (*p != '\f'))
            {
              if (flag & SCAN_SUPPRESS)
              {
                p++;
              }
              else
              {
                *buf++ = *p++;
```

```
                }
                n_decode++;
            }

            if ((!(flag & SCAN_SUPPRESS)) && (s != p))
            {
                /* Add NULL to end of string */
                *buf = '\0';
                nassigned++;
            }
            break;
        case SCAN_DEST_INT:
            n_decode += scan_ignore_white_space(&p);
            s = p;
            val = 0;
            /*TODO: scope is not testsed */
            if ((base == 0) || (base == 16))
            {
                if ((s[0] == '0') && ((s[1] == 'x') || (s[1] == 'X')))
                {
                    base = 16;
                    if (field_width >= 1)
                    {
                        p += 2;
                        n_decode += 2;
                        field_width -= 2;
                    }
                }
            }

            if (base == 0)
            {
                if (s[0] == '0')
                {
                    base = 8;
                }
                else
                {
                    base = 10;
                }
            }

            neg = 1;
            switch (*p)
            {
                case '-':
                    neg = -1;
                    n_decode++;
                    p++;
                    field_width--;
```

```
                    break;
                case '+':
                    neg = 1;
                    n_decode++;
                    p++;
                    field_width--;
                    break;
                default:
                    break;
            }

            while ((*p) && (field_width--))
            {
                if ((*p <= '9') && (*p >= '0'))
                {
                    temp = *p - '0';
                }
                else if((*p <= 'f') && (*p >= 'a'))
                {
                    temp = *p - 'a' + 10;
                }
                else if((*p <= 'F') && (*p >= 'A'))
                {
                    temp = *p - 'A' + 10;
                }
                else
                {
                    break;
                }

                if (temp >= base)
                {
                    break;
                }
                else
                {
                    val = base * val + temp;
                }
                p++;
                n_decode++;
            }

            val *= neg;
            if (!(flag & SCAN_SUPPRESS))
            {
                switch (flag & SCAN_LENGTH_MASK)
                {
                    case SCAN_LENGTH_CHAR:
                        if (flag & SCAN_TYPE_SIGNED)
                        {
```

```
                    *va_arg(args_ptr, signed char *) = (signed char)val;
                  }
                  else
                  {
                    *va_arg(args_ptr, unsigned char *) = (unsigned char)val;
                  }
                  break;
                case SCAN_LENGTH_SHORT_INT:
                  if (flag & SCAN_TYPE_SIGNED)
                  {
                    *va_arg(args_ptr, signed short *) = (signed short)val;
                  }
                  else
                  {
                    *va_arg(args_ptr, unsigned short *) = (unsigned short)val;
                  }
                  break;
                case SCAN_LENGTH_LONG_INT:
                  if (flag & SCAN_TYPE_SIGNED)
                  {
                    *va_arg(args_ptr, signed long int *) = (signed long int)val;
                  }
                  else
                  {
                    *va_arg(args_ptr, unsigned long int *) = (unsigned long int)val;
                  }
                  break;
                case SCAN_LENGTH_LONG_LONG_INT:
                  if (flag & SCAN_TYPE_SIGNED)
                  {
                    *va_arg(args_ptr, signed long long int *) = (signed long long int)val;
                  }
                  else
                  {
                    *va_arg(args_ptr, unsigned long long int *) = (unsigned long long int)val;
                  }
                  break;
                default:
                  /* The default type is the type int */
                  if (flag & SCAN_TYPE_SIGNED)
                  {
                    *va_arg(args_ptr, signed int *) = (signed int)val;
                  }
                  else
                  {
                    *va_arg(args_ptr, unsigned int *) = (unsigned int)val;
                  }
                  break;
              }
              nassigned++;
```

```
                }
                break;
#if defined(SCANF_FLOAT_ENABLE)
            case SCAN_DEST_FLOAT:
                n_decode += scan_ignore_white_space(&p);
                fnum = strtod(p, (char **)&s);

                if ((fnum == HUGE_VAL) || (fnum == -HUGE_VAL))
                {
                   break;
                }

                n_decode += (int)(s) - (int)(p);
                p = s;
                if (!(flag & SCAN_SUPPRESS))
                {
                   if (flag & SCAN_LENGTH_LONG_DOUBLE)
                   {
                      *va_arg(args_ptr, double *) = fnum;
                   }
                   else
                   {
                      *va_arg(args_ptr, float *) = (float)fnum;
                   }
                   nassigned++;
                }
                break;
#endif
#if defined(ADVANCE)
            case SCAN_DEST_SET:
                break;
#endif
            default:
#if defined(SCAN_DEBUG)
                printf("ERROR: File %s  line: %d\r\n", __FILE__, __LINE__);
#endif
                return nassigned;
          }
       }
    }
    return nassigned;
}

/*FUNCTION*****************************************************************
*
 *
 * Function Name : scan_ignore_white_space
 * Description   : Scanline function which ignores white spaces.
 *
 *END*****************************************************************/
```

```
static uint32_t scan_ignore_white_space(const char **s)
{
   uint8_t count = 0;
   uint8_t c;

   c = **s;
   while ((c == ' ') || (c == '\t') || (c == '\n') || (c == '\r') || (c == '\v') || (c == '\f'))
   {
      count++;
      (*s)++;
      c = **s;
   }
   return count;
}
```

---

print_scan.h

```
#ifndef __print_scan_h__
#define __print_scan_h__

#include <stdio.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>

//#define PRINTF_FLOAT_ENABLE   1
//#define PRINT_MAX_COUNT      1
//#define SCANF_FLOAT_ENABLE    1

#ifndef HUGE_VAL
#define HUGE_VAL        (99.e99)///wrong value
#endif

typedef int (*PUTCHAR_FUNC)(int a, void *b);

/*!
 * @brief This function outputs its parameters according to a formatted string.
 *
 * @note I/O is performed by calling given function pointer using following
 * (*func_ptr)(c,farg);
 *
 * @param[in] farg      Argument to func_ptr.
 * @param[in] func_ptr  Function to put character out.
 * @param[in] max_count Maximum character count for snprintf and vsnprintf.
 * Default value is 0 (unlimited size).
 * @param[in] fmt_ptr   Format string for printf.
 * @param[in] args_ptr  Arguments to printf.
```

```
 *
 * @return Number of characters
 * @return EOF (End Of File found.)
 */
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char  *fmt, va_list ap);

/*!
 * @brief Writes the character into the string located by the string pointer and
 * updates the string pointer.
 *
 * @param[in]      c           The character to put into the string.
 * @param[in, out] input_string This is an updated pointer to a string pointer.
 *
 * @return Character written into string.
 */
int _sputc(int c, void * input_string);

/*!
 * @brief Converts an input line of ASCII characters based on a provided
 * string format.
 *
 * @param[in] line_ptr The input line of ASCII data.
 * @param[in] format   Format first points to the format string.
 * @param[in] args_ptr The list of parameters.
 *
 * @return Number of input items converted and assigned.
 * @return IO_EOF - When line_ptr is empty string "".
 */
int scan_prv(const char *line_ptr, char *format, va_list args_ptr);

#endif
```

Custom_ASCII_Counter.c

```
#include "Custom_ASCII_Counter.h"
#include "Custom_UART.h"
#include "Custom_Circular_Buffer.h"

DWord Fib_n;

Byte Counter, ASCII_Char;
DWord ASCII_Array[256], ASCII_Value;
char UART_print[50];

//Function to set the array properly
void ASCII_Counter_Init(void)
{
        //Storing all ascii values in higher 8 bits
```

```
	for(Counter = 0; Counter < 0xFF; Counter ++)		ASCII_Array[Counter] = (Counter << 24);
	ASCII_Array[Counter] = (Counter << 24);
}

//Actual application
void ASCII_Counter(void)
{
	//Give out warning if overwriting
	if(CBuffer_Instance[UART0_Rx_Buffer_ID].Status == Full)
	Output_String("\n\rOverwriting");

	//Read one byte
	CBuffer_Byte_Read(UART0_Rx_Buffer_ID, &ASCII_Char);

	//Scan for all valid ascii values
	for(Counter = 0; Counter < 0xFF; Counter ++)
	{
		//If match then increase the count by 1
		if((ASCII_Array[Counter] >> 24) == ASCII_Char)
		{
			ASCII_Value = ASCII_Array[Counter] & ASCII_Counter_Mask;
			ASCII_Value += 1;
			ASCII_Value &= ASCII_Counter_Mask;
			ASCII_Array[Counter] &= ASCII_Char_Mask;
			ASCII_Array[Counter] |= ASCII_Value;
			break;
		}
	}
	if(Counter == 0xFF)
	{
		ASCII_Value = ASCII_Array[Counter] & ASCII_Counter_Mask;
		ASCII_Value += 1;
		ASCII_Value &= ASCII_Counter_Mask;
		ASCII_Array[Counter] &= ASCII_Char_Mask;
		ASCII_Array[Counter] |= ASCII_Value;
	}

	//Printing fibonacci number
	sprintf(UART_print, "n\rFibonacci Number: %lu", Fib_n);
	Output_String(UART_print);

	//Printing report
	Output_String("\n\rNew Data\n\r");
	for(Counter = 0; Counter < 0xFF; Counter ++)
	{
		ASCII_Char = ASCII_Array[Counter] >> 24;
		ASCII_Value = ASCII_Array[Counter] & ASCII_Counter_Mask;
		if(ASCII_Value != 0)
		{
```

```
                    sprintf(UART_print, "\n\rCharacter: %c\tHex: 0x%02X\tOccurrence:
%ld", ASCII_Char, ASCII_Char, ASCII_Value);
                    Output_String(UART_print);
                }
        }
        Output_String("\n\r\n\rEnter Text:\n\r");
}
```

Custom_Circular_Buffer.c

```
#include "Custom_Circular_Buffer.h"

//Variables

CBuffer CBuffer_Instance[Maximum_Buffers];
ptr_type Location, Continuous_Read;
Byte CBuffer_Data, Error, No_of_CBuffers, cbuffer_i, cbuffer_j;
char CBuffer_Input[10];
DWord CBuffer_Instance_Length[Maximum_Buffers], value;
Byte return_value;
Byte resize = 0;


//Function to get values from string - SCANF can be used instead of FGETS

void String_to_Decimal(char *stod_ptr)
{
        if(stod_ptr)
        {
                char *stod_i;
                stod_i = stod_ptr;
                for(; *stod_ptr != 0; stod_ptr ++)
                {
                        if(isdigit(*stod_ptr) == 0)
                        {
                                Output_String("\n\r\rNon Integer Value Entered\n\r\r");
                                Error = 1;
                                value = 0;
                                break;
                        }
                }
                if(*stod_ptr == 0)
                {
                        Error = 0;
                        value = atoi(stod_i);
                }
        }
        else
```

```
        {
                Output_String("\n\rNull Pointer\n\r");
                Error = 1;
                value = 0;
        }
}

// Initializing each of the buffers

Byte CBuffer_Assign(Byte CBuffer_ID)
{
        CBuffer_Instance[CBuffer_ID].Length = CBuffer_Instance_Length[CBuffer_ID];
        CBuffer_Instance[CBuffer_ID].Elements_count = 0;
        CBuffer_Instance[CBuffer_ID].Start_ptr = (Byte
*)malloc(CBuffer_Instance_Length[CBuffer_ID]);
        if(CBuffer_Instance[CBuffer_ID].Start_ptr == 0)          return 1;
        CBuffer_Instance[CBuffer_ID].Head = 0;
        CBuffer_Instance[CBuffer_ID].Tail = 0;
        CBuffer_Instance[CBuffer_ID].Index = 0;
        CBuffer_Instance[CBuffer_ID].Status = Empty;
        return 0;
}

//Input about buffers from user

Byte CBuffer_Init(void)
{
        No_of_CBuffers = 2;
        CBuffer_Instance_Length[FGETS_Buffer_ID] = FGETS_Buffer_Length;
//      if(CBuffer_Assign(FGETS_Buffer_ID))             return 1;
        //Get length of each, store it in array called CBuffer_Instance_Length
        for(cbuffer_i = 1; cbuffer_i < No_of_CBuffers; cbuffer_i ++)
        {
                Output_String("\n\rEnter the length of ");
                Output_String("Rx Buffer: ");
                Input_String(CBuffer_Input, FGETS_Buffer_Length, stdin);
                cbuffer_j = 0;
                while(CBuffer_Input[cbuffer_j] != Enter_Detected)                cbuffer_j ++;
                CBuffer_Input[cbuffer_j] = 0;
                Output_String("\n\rLength set: ");
                String_to_Decimal(CBuffer_Input);
                if(Error)                       return 1;
                CBuffer_Instance_Length[cbuffer_i] = value;
                sprintf(CBuffer_Input, "%ld", value);
                Output_String(CBuffer_Input);
        }

        //Initialize each buffer
        for(cbuffer_i = 1; cbuffer_i < No_of_CBuffers; cbuffer_i ++)
        {
```

```
                    if(CBuffer_Assign(cbuffer_i))     return 1;
          }
          return 0;
}

//Function to write 1 byte in given buffer

Byte CBuffer_Byte_Write(Byte CBuffer_ID, Byte data)
{
          //If buffer is non empty and head meets the tail then buffer is full and overwriting
          if((CBuffer_Instance[CBuffer_ID].Status != Empty) &&
(CBuffer_Instance[CBuffer_ID].Head == CBuffer_Instance[CBuffer_ID].Tail))
          {
                    CBuffer_Instance[CBuffer_ID].Status = Full;
                    return_value = Overwriting;
          }
          else
          {
                    if(return_value != Overwriting)             return_value = Success;
          }

          //Attempt to not lose data even when buffer is full
          if(return_value == Overwriting)
          {
                    CBuffer_Instance[CBuffer_ID].Start_ptr = (Byte
*)realloc(CBuffer_Instance[CBuffer_ID].Start_ptr, CBuffer_Instance_Length[CBuffer_ID]);
                    CBuffer_Instance[CBuffer_ID].Head = CBuffer_Instance[CBuffer_ID].Length;
                    CBuffer_Instance[CBuffer_ID].Length += CBuffer_Instance_Length[CBuffer_ID];
                    CBuffer_Instance[CBuffer_ID].Status = Success;
                    return_value = Success;
          }

          //Write data using start pointer + relative address from head
          *(CBuffer_Instance[CBuffer_ID].Start_ptr + CBuffer_Instance[CBuffer_ID].Head) = data;


          //Joining end to start - that is if head is at the end, instead of incrementing, wrap it back
to the start (0)
          if(CBuffer_Instance[CBuffer_ID].Head == (CBuffer_Instance[CBuffer_ID].Length - 1))
          CBuffer_Instance[CBuffer_ID].Head = 0;
          else      CBuffer_Instance[CBuffer_ID].Head += 1;

          //Raising this flag after actual write is necessary for other functions
          if(CBuffer_Instance[CBuffer_ID].Head == CBuffer_Instance[CBuffer_ID].Tail)
          CBuffer_Instance[CBuffer_ID].Status = Full;

          //Change status flag if wrote to empty buffer
          if(CBuffer_Instance[CBuffer_ID].Status == Empty)
          CBuffer_Instance[CBuffer_ID].Status = Success;
```

```
	//Read chain broke down by writing
	Continuous_Read = 0;

	//Added element, increment count
	if(CBuffer_Instance[CBuffer_ID].Status != Full)
	CBuffer_Instance[CBuffer_ID].Elements_count += 1;
	else	CBuffer_Instance[CBuffer_ID].Elements_count =
CBuffer_Instance[CBuffer_ID].Length;
	return (return_value);
}

Byte CBuffer_Byte_Read(Byte CBuffer_ID, Byte *address)
{
	if(return_value == Overwriting)		return_value = Success;

	//If read the whole buffer then mark it as empty
	if(Continuous_Read == CBuffer_Instance[CBuffer_ID].Length)
	{
		CBuffer_Instance[CBuffer_ID].Tail = CBuffer_Instance[CBuffer_ID].Head;
		CBuffer_Instance[CBuffer_ID].Status = Empty;
		Output_String("e1");
		return (Empty);
	}
	else
	{
		//Hasn't performed maximum number of reads but, tail and head meet
		if(CBuffer_Instance[CBuffer_ID].Tail == CBuffer_Instance[CBuffer_ID].Head)
		{
			//If buffer wasn't full then it is empty now
			if(CBuffer_Instance[CBuffer_ID].Status != Full)
			{
				CBuffer_Instance[CBuffer_ID].Status = Empty;
				Output_String("e2");
				return (Empty);
			}

			//If the buffer was full then update the index to mark that this will be
the new tail starting location after reading everything
			else
			{
				CBuffer_Instance[CBuffer_ID].Index =
CBuffer_Instance[CBuffer_ID].Tail;
				CBuffer_Instance[CBuffer_ID].Status = Success;
			}
		}
	}

	//read using pointer
	*address = *(CBuffer_Instance[CBuffer_ID].Start_ptr +
CBuffer_Instance[CBuffer_ID].Tail);
```

```
	//joining end to start to make a circular path
		if(CBuffer_Instance[CBuffer_ID].Tail == (CBuffer_Instance[CBuffer_ID].Length -
1))	CBuffer_Instance[CBuffer_ID].Tail = 0;
		else	CBuffer_Instance[CBuffer_ID].Tail += 1;

	//increases continuous read variable
	Continuous_Read += 1;

	if(CBuffer_Instance[CBuffer_ID].Tail == CBuffer_Instance[CBuffer_ID].Head)
	{
		if(CBuffer_Instance[CBuffer_ID].Status != Full)
		{
			Output_String("e3");
			CBuffer_Instance[CBuffer_ID].Status = Empty;
		}
	}

	//element removed
	if(CBuffer_Instance[CBuffer_ID].Elements_count > 0)
	CBuffer_Instance[CBuffer_ID].Elements_count -= 1;
	return (Success);
}

//Function to handle both read and write
void CBuffer_Operation(Byte CBuffer_ID, Byte type, Byte data, Byte *address)
{
	if(type == Write)
	{
		//the only error is overwriting warning
		if(CBuffer_Byte_Write(CBuffer_ID, data))	Output_String("Buffer%d Full,
Overwriting\n\r", CBuffer_ID);

		//printing useful information
		Location = CBuffer_Instance[CBuffer_ID].Head;
		if(Location)		Location -= 1;
		else	Location = CBuffer_Instance[CBuffer_ID].Length - 1;
		Output_String("Wrote %c at %d in buffer%d", data, Location, CBuffer_ID);
	}
	else if(type == Read)
	{
		//only error is buffer empty
		if(CBuffer_Byte_Read(CBuffer_ID, address))
		{
			Output_String("Buffer%d Empty\n\r", CBuffer_ID);
		}
		else
		{
			Location = CBuffer_Instance[CBuffer_ID].Tail;
			if(Location)		Location -= 1;
```

```
                    else      Location = CBuffer_Instance[CBuffer_ID].Length - 1;
                    Output_String("Read return is %c from %d in buffer%d", CBuffer_Data,
Location, CBuffer_ID);
                }
        }
}

DWord CBuffer_Elements(Byte CBuffer_ID)
{
        return (CBuffer_Instance[CBuffer_ID].Elements_count);
}

//resizing existing buffer
Byte CBuffer_Resize(Byte CBuffer_ID)
{
        Output_String("\n\rEnter new length of buffer%d: ", CBuffer_ID);
        Input_String(CBuffer_Input, 10, stdin);
        cbuffer_j = 0;
        while(CBuffer_Input[cbuffer_j] != Enter_Detected)          cbuffer_j ++;
        CBuffer_Input[cbuffer_j] = 0;
        String_to_Decimal(CBuffer_Input);
        if(Error)                  return 1;
        free(CBuffer_Instance[CBuffer_ID].Start_ptr);
        CBuffer_Instance_Length[CBuffer_ID] = value;
        CBuffer_Assign(CBuffer_ID);
        return 0;
}
```

Custom_UART.c

```
#include "Custom_UART.h"
#include "Custom_Circular_Buffer.h"
volatile Byte UART0_Byte;
UART0_Operation_Type State = Normal_Operation;

volatile Byte isr_arr[50], isr_cnt = 0;
Byte led;

//UART0 Initialization Function
void Custom_UART0_Init(void)
{
        //Enabling clock first
        Enable_UART0_Clock();

        //Selecting proper Mux values for UART function
        Enable_UART0_Rx_Function();
        Enable_UART0_Tx_Function();
```

```
	//Disabling pins for configuring UART safely
	Disable_UART0_Tx();
	Disable_UART0_Rx();

	//Selecting and configuring clock source to drive UART
	Select_PLL_Clock_Divby2();
	UART0_FLL_PLL_Clock_Source();
	Set_BAUD_Rate_High_Register();
	Set_BAUD_Rate_Low_Register();

	//Addition steps for interrupt mode
	#ifdef	INTERRUPT_MODE
		Enable_Rx_Interrupt();
		NVIC_EnableIRQ(UART0_IRQn);
	#endif

	//Selecting oversampling value
	Set_Oversampling();

	//Enabling pins
	Enable_UART0_Tx();
	Enable_UART0_Rx();
}

//Polling transmitting byte function
void Custom_UART0_Tx_Byte(Byte data)
{
	//Polling flag to check for availability of UART transmitter
	UART0_Wait_for_Tx_Data_Register();

	//Putting byte in buffer/data register
	UART0_Tx_Data(data);
}

//Function to transmit strings through UART
void Custom_UART0_Tx_String(char *array)
{
  DWord uart_i;
  DWord string_length = strlen(array);

  //For interrupt mode, set a array which is shared between ISR and this function
  //Cleaning up that array here
	#ifdef	INTERRUPT_MODE
  while(isr_cnt != 0);
	for(uart_i = 0; uart_i < 50; uart_i ++)		isr_arr[uart_i] = 0;
	#endif
	//Acutally setting up array
  for(uart_i = 0; uart_i < string_length; uart_i ++)
  {
```

```
            //Calling polling transmit byte function repeatedly
                        #ifdef    POLLING_MODE
            Custom_UART0_Tx_Byte(array[uart_i]);
                        #else
            //Filling data in shared array
            isr_arr[uart_i] = array[uart_i];
                        #endif
    }
    //Enable transmitter buffer empty interrupt
    //It should be kept disable normally to avoid going into ISR infinitely, continuously, and
instantly
            #ifdef    INTERRUPT_MODE
    Enable_TxE_Interrupt();
            #endif
}

//Polling function to receive a byte
void Custom_UART0_Rx_Byte(volatile Byte *address)
{
            //Polling flag to see if any data has been received
            UART0_Wait_for_Rx_Data_Register();

            //Storing that byte using the pointer of the variable
            UART0_Rx_Data(address);
}

void UART0_IRQHandler(void)
{
            //Check whether Rx interrupt has caused the code to go in ISR or Tx empty interrupt
    if(UART0_Rx_Interrupt())
    {
            //First store the byte in a variable
            UART0_Rx_Data(&UART0_Byte);

#if                 APPLICATION

            //To see whether it's FGETS running or normal one
            if(State == Normal_Operation)
                        {
                        //Write byte in the UART Rx circular buffer
                        CBuffer_Byte_Write(UART0_Rx_Buffer_ID, UART0_Byte);

                        //Code for fun :P
            if(led < 7)                 led += 1;
            else      led = 0;
                        }
            else if(State == FGETS_Operation)
                        {
                        //Echo byte to actually see what's being typed in
                        UART0_Tx_Data(UART0_Byte);
```

```
                    //Write byte in FGETS circular buffer
                    CBuffer_Byte_Write(FGETS_Buffer_ID, UART0_Byte);

                    //Leave FGETS function is the user presses enter or the buffer is filled up
                    if((CBuffer_Instance[FGETS_Buffer_ID].Status == Full) || (Enter_Detected ==
UART0_Byte)) State = Normal_Operation;
                    }

#else
          UART0_Tx_Data(UART0_Byte);
#endif

   }
   else if(UART0_TxE_Interrupt())
   {
          //Transmit one byte if the index in shared array hasn't reached null
          if(isr_arr[isr_cnt] != 0)   UART0_Tx_Data(isr_arr[isr_cnt ++]);

          //If the index is pointing to null then reset index, and disable Tx empty interrupt
          else
                    {
                    Disable_TxE_Interrupt();
                    isr_cnt = 0;
                    }
   }
}
```

Custom_ASCII_Counter.c

```
#ifndef CUSTOM_INCLUDES_CUSTOM_ASCII_COUNTER_H_
#define CUSTOM_INCLUDES_CUSTOM_ASCII_COUNTER_H_

#include "Custom_Main.h"

//A single 32bit array is used for both - to store all ascii values and to store counts of them
//Each double word will have both. MSB 8 bits holding ascii char and lower 24 bits holding
counts
#define ASCII_Char_Mask                    0xFF000000
#define ASCII_Counter_Mask          0x00FFFFFF

extern Byte Counter, ASCII_Char;
extern DWord ASCII_Array[256], ASCII_Value;

void ASCII_Counter_Init(void);
void ASCII_Counter(void);

#endif /* CUSTOM_INCLUDES_CUSTOM_ASCII_COUNTER_H_ */
```

Custom_Circular_Buffer.h

```
#ifndef CUSTOM_CIRCULAR_BUFFER_H_
#define CUSTOM_CIRCULAR_BUFFER_H_

#include "Custom_Main.h"

#define Success                 0
#define Overwriting             1
#define Empty                   2
#define Full                    3
#define Write                   0
#define Read                    1

#define FGETS_Buffer_ID             0
#define UART0_Rx_Buffer_ID  1
#define UART0_Tx_Buffer_ID  2

#define FGETS_Buffer_Length         10


typedef struct
{
        Byte *Start_ptr;
        Byte Status;
        DWord Length;
        DWord Elements_count;
        ptr_type Head;
        ptr_type Tail;
        ptr_type Index;
}CBuffer;

#define Maximum_Buffers             10

extern CBuffer CBuffer_Instance[Maximum_Buffers];
extern Byte CBuffer_Data, Error, No_of_CBuffers;
extern ptr_type Location, Continuous_Read;
extern char CBuffer_Input[10];
extern DWord CBuffer_Instance_Length[Maximum_Buffers], value;


void String_to_Decimal(char *stod_ptr);
Byte CBuffer_Assign(Byte CBuffer_ID);
Byte CBuffer_Init(void);
Byte CBuffer_Byte_Write(Byte CBuffer_ID, Byte data);
Byte CBuffer_Byte_Read(Byte CBuffer_ID, Byte *address);
void CBuffer_Operation(Byte CBuffer_ID, Byte type, Byte data, Byte *address);
```

```
DWord CBuffer_Elements(Byte CBuffer_ID);
Byte CBuffer_Resize(Byte CBuffer_ID);




#endif /* CUSTOM_CIRCULAR_BUFFER_H_ */
```

Custom_Main.h

```
#ifndef CUSTOM_MAIN_H_
#define CUSTOM_MAIN_H_


#include<stdint.h>
#include<inttypes.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>


#include "Custom_Sys_Identifier.h"

#ifdef    FRDM
//#include "core_cm0plus.h"
#include "MKL25Z4.h"
//#include "board.h"
#include "fsl_debug_console.h"
//#include "fsl_os_abstraction_bm.h"
#endif

#define Invalid()           output_string("\nInvalid Command\n")
#define Null_Ptr()          output_string("\nNull Pointer. Can't proceed. Abort.\n")

typedef uint8_t Byte;
typedef uint16_t Word;
typedef uint32_t DWord;
typedef volatile uint8_t vuint8_t;
typedef volatile uint32_t          vuint32_t;

extern char UART_print[50];
extern DWord Fib_n;
extern Byte led;


void FGETS(char *array_to_write, Byte bytes, FILE *stream);

#endif /* CUSTOM_MAIN_H_ */
```

Custom_Sys_Identifier.h

```
#ifndef __CUSTOM_SYS_IDENTIFIER_H__
#define __CUSTOM_SYS_IDENTIFIER_H__

/*
 * This header file decides if the device being targeted is a host machine or an embedded system.
 * It will check if the target is Linux, Windows, or a Mac. Else, it is an embedded system.
 * It will then adjust the code accordingly.
 *


 */

/* Compiler flag */
#define TARGET_DEVICE (__linux__ || _WIN64 || __APPLE__ )
#if TARGET_DEVICE
   #define HOST
#else
   #define FRDM
#endif

#ifdef   HOST
         #define Output_String  printf
         #define Input_String   fgets
         typedef uint64_t       ptr_type;
         #define exit_function  system_exit
         #define Enter_Detected         '\n'
#else
         #define Output_String  Custom_UART0_Tx_String
         #define Input_String   FGETS
         typedef volatile uint32_t      ptr_type;
         #define exit_function  embedded_exit
         #define Enter_Detected         0x0D
#endif


#endif /* CUSTOM_SYSTEM_IDENTIFIER_H_ */
```

Custom_UART.h

```
#ifndef CUSTOM_INCLUDES_CUSTOM_UART_H_
#define CUSTOM_INCLUDES_CUSTOM_UART_H_

#include "Custom_Main.h"

//Defines and conditions for switching between polling and interrupt mode
#define POLLING                 0
```

```
#define INTERRUPT      1

#define UART_MODE    INTERRUPT

#if UART_MODE
        #define INTERRUPT_MODE
        #define APPLICATION             1
#else
        #define POLLING_MODE
#endif

//Defines for clocking UART
#define Clock_Gating_Register_4             SCGC4
#define System_Integration_Module    SIM
#define UART0_Clock_Gate_Bit                10
#define Enable_UART0_Clock()          (System_Integration_Module->Clock_Gating_Register_4
|= (1 << UART0_Clock_Gate_Bit))

//Define for function selection
#define UART0_Port              PORTA
#define Pin_Control_Register    PCR
#define UART0_Rx_Pin            1
#define UART0_Tx_Pin            2
#define UART0_Function          0x02    //page 162 in ref manual
#define Pin_Function_Select(x)          PORT_PCR_MUX(x)

//Macros for function select
#define Enable_UART0_Rx_Function()          (UART0_Port-
>Pin_Control_Register[UART0_Rx_Pin] |= \

        Pin_Function_Select(UART0_Function))
#define Enable_UART0_Tx_Function()          (UART0_Port-
>Pin_Control_Register[UART0_Tx_Pin] |= \

        Pin_Function_Select(UART0_Function))

//Defines for UART Interrupt configuration
#define UART0_TxE_Interrupt_Bit                 7
#define UART0_Rx_Interrupt_Bit                  5
#define UART0_Transmitter_Enable_Bit    3
#define UART0_Receiver_Enable_Bit       2
#define UART0_Control_Register_2        C2
#define UART0_Register_Handler                  UART0

//Macros for UART Interrupt support
#define Disable_UART0_Tx()                      (UART0_Register_Handler-
>UART0_Control_Register_2 &= \
                                                                        ~(1 <<
UART0_Transmitter_Enable_Bit))
```

```
#define Disable_UART0_Rx()                    (UART0_Register_Handler->UART0_Control_Register_2 &= \
                                                                                    ~(1 << UART0_Receiver_Enable_Bit))
#define Enable_UART0_Tx()                     (UART0_Register_Handler->UART0_Control_Register_2 |= \
                                                                                    (1 << UART0_Transmitter_Enable_Bit))
#define Enable_UART0_Rx()                     (UART0_Register_Handler->UART0_Control_Register_2 |= \
                                                                                    (1 << UART0_Receiver_Enable_Bit))
#define Enable_Rx_Interrupt()                 (UART0_Register_Handler->UART0_Control_Register_2 |= \
                                                                                    (1 << UART0_Rx_Interrupt_Bit))
#define Enable_TxE_Interrupt()                (UART0_Register_Handler->UART0_Control_Register_2 |= \
                                                                                    (1 << UART0_TxE_Interrupt_Bit))
#define Disable_TxE_Interrupt()               (UART0_Register_Handler->UART0_Control_Register_2 &= \
                                                                                    ~(1 << UART0_TxE_Interrupt_Bit))

//Defines and macros for clock source selection and configuration for UART
#define System_Option_Register_2              SOPT2
#define PLL_FLL_Clock_Select_Bit              16
#define Select_PLL_Clock_Divby2()             (System_Integration_Module->System_Option_Register_2 |= \
                                                                                    (1 << PLL_FLL_Clock_Select_Bit))
#define Select_FLL_Clock()                    (System_Integration_Module->System_Option_Register_2 &= \
                                                                                    ~(1 << PLL_FLL_Clock_Select_Bit))
#define UART0_Clock_Source_Offset             26
#define UART0_Clock_Souce_FLL_PLL             1
#define UART0_FLL_PLL_Clock_Source()          (System_Integration_Module->System_Option_Register_2 |= \
                                                    (UART0_Clock_Souce_FLL_PLL << UART0_Clock_Source_Offset))

//Defines and macros for BAUD rate
#define BAUD_Rate                 115200UL
#define System_Clock   48000000UL
#define Oversampling   16
#define BAUD_Rate_Setting_Value               (Word)(System_Clock / (BAUD_Rate * Oversampling))
#define BAUD_Rate_High_Mask                   0x1F00
```

```
#define BAUD_Rate_Low_Mask                          0x00FF
#define BAUD_Rate_High_Register             BDH
#define BAUD_Rate_Low_Register              BDL
#define Set_BAUD_Rate_High_Register()               (UART0_Register_Handler-
>BAUD_Rate_High_Register = \

        (BAUD_Rate_Setting_Value & BAUD_Rate_High_Mask))
#define Set_BAUD_Rate_Low_Register()                (UART0_Register_Handler-
>BAUD_Rate_Low_Register = \

        (BAUD_Rate_Setting_Value & BAUD_Rate_Low_Mask))

//Defines and macros for oversampling
#define UART0_Control_Register_4            C4
#define Oversampling_16                             0x0F
#define Set_Oversampling()                          (UART0_Register_Handler-
>UART0_Control_Register_4 = \

        Oversampling_16)

//Defines and macros for polling UART functions
#define UART0_Status_Register_1                     S1
#define Tx_Data_Register_Empty_Flag_Bit             7       //1 means empty
#define Tx_Data_Transmission_Complete_Flag_Bit      6       //1 means complete
#define UART0_Tx_Empty_Flag_Status()                ((UART0_Register_Handler-
>UART0_Status_Register_1) & \

        (1 << Tx_Data_Register_Empty_Flag_Bit))
#define UART0_Wait_for_Tx_Data_Register()   while((!UART0_Tx_Empty_Flag_Status()))

#define Rx_Data_Register_Full_Flag_Bit              5       //1 means full
#define UART0_Wait_for_Rx_Data_Register()   while(!((UART0_Register_Handler-
>UART0_Status_Register_1) & \

        (1 << Rx_Data_Register_Full_Flag_Bit)))

//Defines and macros for interrupt UART functions
#define UART0_Data_Register                         D
#define UART0_Tx_Data(x)                            (UART0_Register_Handler-
>UART0_Data_Register = x)
#define UART0_Rx_Data(addr)                         (*addr = UART0_Register_Handler-
>UART0_Data_Register)

#define UART0_Rx_Interrupt()                (UART0_Register_Handler-
>UART0_Status_Register_1 & \

                                                                        (1 <<
Rx_Data_Register_Full_Flag_Bit))

#define UART0_TxE_Interrupt()               (UART0_Register_Handler-
>UART0_Status_Register_1 & \
```

```
                                                                    (1 <<
Tx_Data_Register_Empty_Flag_Bit))

//Currently only 2 modes - can be increased easily in future
typedef enum
{
        FGETS_Operation,
        Normal_Operation
}UART0_Operation_Type;

//Function initializations
void Custom_UART0_Init(void);
void ASCII_Counter2(void);
void Custom_UART0_Tx_Byte(Byte data);
void Custom_UART0_Tx_String(char *array);
void Custom_UART0_Rx_Byte(volatile Byte *address);
void UART0_IRQHandler(void);

//Variables
extern UART0_Operation_Type State;
extern volatile Byte UART0_Byte;

#endif /* CUSTOM_INCLUDES_CUSTOM_UART_H_ */
```

board.c

```
#include "board.h"
#include "fsl_clock_manager.h"
#include "fsl_smc_hal.h"
#include "fsl_debug_console.h"
#include "pin_mux.h"

/* Configuration for enter VLPR mode. Core clock = 4MHz. */
const clock_manager_user_config_t g_defaultClockConfigVlpr =
{
  .mcgConfig =
  {
    .mcg_mode         = kMcgModeBLPI,   // Work in BLPI mode.
    .irclkEnable      = true,  // MCGIRCLK enable.
    .irclkEnableInStop  = false, // MCGIRCLK disable in STOP mode.
    .ircs             = kMcgIrcFast, // Select IRC4M.
    .fcrdiv           = 0U,    // FCRDIV is 0.

    .frdiv   = 0U,
    .drs     = kMcgDcoRangeSelLow,  // Low frequency range
    .dmx32   = kMcgDmx32Default,    // DCO has a default range of 25%

    .pll0EnableInFllMode      = false,  // PLL0 disable
```

```
      .pll0EnableInStop  = false,  // PLL0 disalbe in STOP mode
      .prdiv0          = 0U,
      .vdiv0           = 0U,
    },
    .simConfig =
    {
      .pllFllSel = kClockPllFllSelFll, // PLLFLLSEL select FLL.
      .er32kSrc  = kClockEr32kSrcLpo,    // ERCLK32K selection, use LPO.
      .outdiv1   = 0U,
      .outdiv4   = 4U,
    },
    .oscerConfig =
    {
      .enable      = true,  // OSCERCLK enable.
      .enableInStop = false, // OSCERCLK disable in STOP mode.
    }
};

/* Configuration for enter RUN mode. Core clock = 48MHz. */
const clock_manager_user_config_t g_defaultClockConfigRun =
{
    .mcgConfig =
    {
      .mcg_mode          = kMcgModePEE,   // Work in PEE mode.
      .irclkEnable       = true,  // MCGIRCLK enable.
      .irclkEnableInStop  = false, // MCGIRCLK disable in STOP mode.
      .ircs              = kMcgIrcSlow, // Select IRC32k.
      .fcrdiv            = 0U,    // FCRDIV is 0.

      .frdiv   = 3U,
      .drs     = kMcgDcoRangeSelLow,  // Low frequency range
      .dmx32   = kMcgDmx32Default,    // DCO has a default range of 25%

      .pll0EnableInFllMode       = false,  // PLL0 disable
      .pll0EnableInStop  = false,  // PLL0 disalbe in STOP mode
      .prdiv0          = 0x1U,
      .vdiv0           = 0x0U,
    },
    .simConfig =
    {
      .pllFllSel = kClockPllFllSelPll,    // PLLFLLSEL select PLL.
      .er32kSrc  = kClockEr32kSrcLpo,     // ERCLK32K selection, use LPO.
      .outdiv1   = 1U,
      .outdiv4   = 3U,
    },
    .oscerConfig =
    {
      .enable      = true,  // OSCERCLK enable.
      .enableInStop = false, // OSCERCLK disable in STOP mode.
    }
```

```
};

/* Function to initialize OSC0 base on board configuration. */
void BOARD_InitOsc0(void)
{
  // OSC0 configuration.
  osc_user_config_t osc0Config =
  {
    .freq              = OSC0_XTAL_FREQ,
    .hgo               = MCG_HGO0,
    .range             = MCG_RANGE0,
    .erefs             = MCG_EREFS0,
    .enableCapacitor2p  = OSC0_SC2P_ENABLE_CONFIG,
    .enableCapacitor4p  = OSC0_SC4P_ENABLE_CONFIG,
    .enableCapacitor8p  = OSC0_SC8P_ENABLE_CONFIG,
    .enableCapacitor16p = OSC0_SC16P_ENABLE_CONFIG,
  };

  CLOCK_SYS_OscInit(0U, &osc0Config);
}

/* Function to initialize RTC external clock base on board configuration. */
void BOARD_InitRtcOsc(void)
{

}

static void CLOCK_SetBootConfig(clock_manager_user_config_t const* config)
{
  CLOCK_SYS_SetSimConfigration(&config->simConfig);

  CLOCK_SYS_SetOscerConfigration(0, &config->oscerConfig);

#if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
  CLOCK_SYS_BootToBlpi(&config->mcgConfig);
 #else
  CLOCK_SYS_BootToPee(&config->mcgConfig);
 #endif

  SystemCoreClock = CORE_CLOCK_FREQ;
}

/* Initialize clock. */
void BOARD_ClockInit(void)
{
  /* Set allowed power mode, allow all. */
  SMC_HAL_SetProtection(SMC, kAllowPowerModeAll);

  /* Setup board clock source. */
  // Setup OSC0 if used.
```

```
    // Configure OSC0 pin mux.
    PORT_HAL_SetMuxMode(EXTAL0_PORT, EXTAL0_PIN, EXTAL0_PINMUX);
    PORT_HAL_SetMuxMode(XTAL0_PORT, XTAL0_PIN, XTAL0_PINMUX);
    BOARD_InitOsc0();

    /* Set system clock configuration. */
#if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
    CLOCK_SetBootConfig(&g_defaultClockConfigVlpr);
#else
    CLOCK_SetBootConfig(&g_defaultClockConfigRun);
#endif
}

void dbg_uart_init(void)
{
    configure_lpsci_pins(BOARD_DEBUG_UART_INSTANCE);

    // Select different clock source for LPSCI. */
#if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
    CLOCK_SYS_SetLpsciSrc(BOARD_DEBUG_UART_INSTANCE, kClockLpsciSrcMcgIrClk);
#else
    CLOCK_SYS_SetLpsciSrc(BOARD_DEBUG_UART_INSTANCE, kClockLpsciSrcPllFllSel);
#endif

    DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUD,
kDebugConsoleLPSCI);
}
/*****************************************************************************
 *
 *   @name     usb_device_board_init
 *
 *   @brief    This function is to handle board-specified initialization
 *
 *   @param    controller_id:      refer to CONTROLLER_INDEX defined in usb_misc.h
 *                         "0" stands for USB_CONTROLLER_KHCI_0.
 *   @return   status
 *                         0 : successful
 *                         1 : failed
 **
 *****************************************************************************/
uint8_t usb_device_board_init(uint8_t controller_id)
{
    int8_t ret = 0;

    if (0 == controller_id)
    {
        /* TO DO */
        /*add board initialization code if have*/
    }
    else
```

```
    {
        ret = 1;
    }

    return ret;

}
/*****************************************************************************
 *
 *  @name       usb_host_board_init
 *
 *  @brief      This function is to handle board-specified initialization
 *
 *  @param    controller_id:       refer to CONTROLLER_INDEX defined in usb_misc.h
 *                          "0" stands for USB_CONTROLLER_KHCI_0.
 *  @return       status
 *                          0 : successful
 *                          1 : failed
 **
 *****************************************************************************/

uint8_t usb_host_board_init(uint8_t controller_id)
{
    int8_t ret = 0;
    /*"0" stands for USB_CONTROLLER_KHCI_0 */
    if (0 == controller_id)
    {

    }
    else
    {
      ret = 1;
    }

    return ret;


}
/*****************************************************************************
 * EOF
 *****************************************************************************/
```

board.h

```
#if !defined(__BOARD_H__)
#define __BOARD_H__

#include <stdint.h>
```

```
#include "pin_mux.h"
#include "gpio_pins.h"

/* The board name */
#define BOARD_NAME                  "FRDM-KL25Z"

#define CLOCK_VLPR 1U
#define CLOCK_RUN  2U
#define CLOCK_NUMBER_OF_CONFIGURATIONS 3U

#ifndef CLOCK_INIT_CONFIG
#define CLOCK_INIT_CONFIG CLOCK_RUN
#endif

#if (CLOCK_INIT_CONFIG == CLOCK_RUN)
#define CORE_CLOCK_FREQ 48000000U
#else
#define CORE_CLOCK_FREQ 4000000U
#endif

/* OSC0 configuration. */
#define OSC0_XTAL_FREQ 8000000U
#define OSC0_SC2P_ENABLE_CONFIG  false
#define OSC0_SC4P_ENABLE_CONFIG  false
#define OSC0_SC8P_ENABLE_CONFIG  false
#define OSC0_SC16P_ENABLE_CONFIG false
#define MCG_HGO0   kOscGainLow
#define MCG_RANGE0 kOscRangeVeryHigh
#define MCG_EREFS0 kOscSrcOsc

/* EXTAL0 PTA18 */
#define EXTAL0_PORT   PORTA
#define EXTAL0_PIN    18
#define EXTAL0_PINMUX kPortPinDisabled

/* XTAL0 PTA19 */
#define XTAL0_PORT   PORTA
#define XTAL0_PIN    19
#define XTAL0_PINMUX kPortPinDisabled

/* The UART to use for debug messages. */
#ifndef BOARD_DEBUG_UART_INSTANCE
   #define BOARD_DEBUG_UART_INSTANCE   0
   #define BOARD_DEBUG_UART_BASEADDR   UART0
#endif
#ifndef BOARD_DEBUG_UART_BAUD
   #define BOARD_DEBUG_UART_BAUD       115200
#endif

/* This define to use for power manager demo */
```

```
#define BOARD_LOW_POWER_UART_BAUD      9600

#define BOARD_USE_LPSCI
#define PM_DBG_UART_IRQ_HANDLER       UART0_IRQHandler
#define PM_DBG_UART_IRQn              UART0_IRQn

/* Define print statement to inform user which switch to press for
 * power_manager_hal_demo and power_manager_rtos_demo
 */
#define PRINT_LLWU_SW_NUM \
  PRINTF(" PTD6 J2-17 to VSS J9-14")

/* Defines the llwu pin number for board switch which is used in power_manager_demo. */
#define BOARD_SW_HAS_LLWU_PIN       1
#define BOARD_SW_LLWU_EXT_PIN       15
/* Switch port base address and IRQ handler name. Used by power_manager_demo */
#define BOARD_SW_LLWU_PIN           6
#define BOARD_SW_LLWU_BASE          PORTD
#define BOARD_SW_LLWU_IRQ_HANDLER   PORTD_IRQHandler
#define BOARD_SW_LLWU_IRQ_NUM       PORTD_IRQn

#define BOARD_I2C_GPIO_SCL          GPIO_MAKE_PIN(GPIOE_IDX, 24)
#define BOARD_I2C_GPIO_SDA          GPIO_MAKE_PIN(GPIOE_IDX, 25)

#define HWADC_INSTANCE              0
#define ADC_IRQ_N                   ADC0_IRQn

/* The instances of peripherals used for dac_adc_demo */
#define BOARD_DAC_DEMO_DAC_INSTANCE    0U
#define BOARD_DAC_DEMO_ADC_INSTANCE    0U
#define BOARD_DAC_DEMO_ADC_CHANNEL     0U

/* The i2c instance used for i2c DAC demo */
#define BOARD_DAC_I2C_INSTANCE         1

/* The i2c instance used for i2c connection by default */
#define BOARD_I2C_INSTANCE             1

/* The spi instance used for spi example */
#define BOARD_SPI_INSTANCE             0

/* The TPM instance/channel used for board */
#define BOARD_TPM_INSTANCE             0
#define BOARD_TPM_CHANNEL              1

/* The bubble level demo information */
#define BOARD_BUBBLE_TPM_INSTANCE      2
#define BOARD_TPM_X_CHANNEL            0
#define BOARD_TPM_Y_CHANNEL            1
#define BOARD_MMA8451_ADDR             0x1D
```

```
#define BOARD_ACCEL_ADDR              BOARD_MMA8451_ADDR
#define BOARD_ACCEL_BAUDRATE          100
#define BOARD_ACCEL_I2C_INSTANCE      0

/* board led color mapping */
#define BOARD_GPIO_LED_BLUE           kGpioLED3
#define BOARD_GPIO_LED_RED            kGpioLED2
#define BOARD_GPIO_LED_GREEN          kGpioLED1

#define BOARD_TSI_ELECTRODE_CNT       2
#define BOARD_TSI_ELECTRODE_1         9
#define BOARD_TSI_ELECTRODE_2         10

#define LED1_EN (GPIO_DRV_OutputPinInit(&ledPins[0]))       /*!< Enable target LED1 */
#define LED2_EN (GPIO_DRV_OutputPinInit(&ledPins[1]))       /*!< Enable target LED2 */
#define LED3_EN (GPIO_DRV_OutputPinInit(&ledPins[2]))       /*!< Enable target LED3 */

#define LED1_DIS (PORT_HAL_SetMuxMode(PORTB, 19, kPortMuxAsGpio))   /*!< Enable target
LED1 */
#define LED2_DIS (PORT_HAL_SetMuxMode(PORTB, 18, kPortMuxAsGpio))       /*!< Enable
target LED2 */
#define LED3_DIS (PORT_HAL_SetMuxMode(PORTD,  1, kPortMuxAsGpio))       /*!< Enable
target LED3 */

#define LED1_OFF (GPIO_DRV_WritePinOutput(ledPins[0].pinName, 1))       /*!< Turn off target
LED1 */
#define LED2_OFF (GPIO_DRV_WritePinOutput(ledPins[1].pinName, 1))       /*!< Turn off target
LED2 */
#define LED3_OFF (GPIO_DRV_WritePinOutput(ledPins[2].pinName, 1))       /*!< Turn off target
LED3 */

#define LED1_ON (GPIO_DRV_WritePinOutput(ledPins[0].pinName, 0))        /*!< Turn on target
LED1 */
#define LED2_ON (GPIO_DRV_WritePinOutput(ledPins[1].pinName, 0))        /*!< Turn on target
LED2 */
#define LED3_ON (GPIO_DRV_WritePinOutput(ledPins[2].pinName, 0))        /*!< Turn on target
LED3 */

#define LED1_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[0].pinName))      /*!< Toggle on
target LED1 */
#define LED2_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[1].pinName))      /*!< Toggle on
target LED2 */
#define LED3_TOGGLE (GPIO_DRV_TogglePinOutput(ledPins[2].pinName))      /*!< Toggle on
target LED3 */

#define BOARD_HAS_ONLY_MULTIPLE_COLOR_LED
#define LED_RTOS_EN       LED1_EN
#define LED_RTOS_TOGGLE   LED1_TOGGLE

/* The CMP instance used for board. */
```

```
#define BOARD_CMP_INSTANCE          0
/* The CMP channel used for board. */
#define BOARD_CMP_CHANNEL           0

/* The rtc instance used for rtc_func */
#define BOARD_RTC_FUNC_INSTANCE     0

#if defined(__cplusplus)
extern "C" {
#endif /* __cplusplus */

void hardware_init(void);
void dbg_uart_init(void);
/*This function to used for power manager demo*/
void disable_unused_pins(void);
void enable_unused_pins(void);
/* Function to initialize clock base on board configuration. */
void BOARD_ClockInit(void);

/* Function to initialize OSC0 base on board configuration. */
void BOARD_InitOsc0(void);

/* Function to initialize RTC external clock base on board configuration. */
void BOARD_InitRtcOsc(void);

/*Function to handle board-specified initialization*/
uint8_t usb_device_board_init(uint8_t controller_id);
/*Function to handle board-specified initialization*/
uint8_t usb_host_board_init(uint8_t controller_id);

#if defined(__cplusplus)
}
#endif /* __cplusplus */

#endif /* __BOARD_H__ */
```

gpio_pins.c

```
#include <stdint.h>
#include <stdlib.h>
#include "gpio_pins.h"

/*!
 * Definitions
 *!/

gpio_input_pin_user_config_t switchPins[] = {
    {
```

```
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};

/* Declare Output GPIO pins */
gpio_output_pin_user_config_t ledPins[] = {
    {
        .pinName = kGpioLED1,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = kGpioLED2,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = kGpioLED3,
        .config.outputLogic = 1,
        .config.slewRate = kPortSlowSlewRate,
        .config.driveStrength = kPortLowDriveStrength,
    },
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};


/* END gpio_pins. */
/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

gpio_pins.h

```
#ifndef __FSL_GPIO_PINS_H__
#define __FSL_GPIO_PINS_H__

#include "fsl_gpio_driver.h"

/*! @file */
/*!*/
/*! This file contains gpio pin definitions used by gpio peripheral driver.*/
/*! The enums in _gpio_pins map to the real gpio pin numbers defined in*/
/*! gpioPinLookupTable. And this might be different in different board.*/

/*******************************************************************************
 * Definitions
 ******************************************************************************/

/*! @brief gpio pin names.*/
/*!*/
/*! This should be defined according to board setting.*/
enum _gpio_pins
{
   kGpioLED1       = GPIO_MAKE_PIN(GPIOB_IDX, 19),   /* FRDM-KL25Z4 Green LED */
   kGpioLED2       = GPIO_MAKE_PIN(GPIOB_IDX, 18),   /* FRDM-KL25Z4 Red LED */
   kGpioLED3       = GPIO_MAKE_PIN(GPIOD_IDX,  1),   /* FRDM-KL25Z4 Blue LED */
   kGpioSW1        = GPIO_MAKE_PIN(GPIOD_IDX,  6),   /* FRDM-KL25Z4 power manager */


};

extern gpio_input_pin_user_config_t switchPins[];
extern gpio_output_pin_user_config_t ledPins[];

#endif /* __FSL_GPIO_PINS_H__ */

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

hardware_init.c

```
#include "board.h"
#include "pin_mux.h"
#include "fsl_clock_manager.h"
#include "fsl_debug_console.h"

void hardware_init(void) {

  /* enable clock for PORTs */
  CLOCK_SYS_EnablePortClock(PORTA_IDX);
  CLOCK_SYS_EnablePortClock(PORTB_IDX);
  CLOCK_SYS_EnablePortClock(PORTC_IDX);
  CLOCK_SYS_EnablePortClock(PORTD_IDX);
  CLOCK_SYS_EnablePortClock(PORTE_IDX);

  /* Init board clock */
  BOARD_ClockInit();
//  dbg_uart_init();
//  configure_dac_pins(0U);
}

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.4 [05.10]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

pin_mux.c

```
#include "fsl_device_registers.h"
#include "fsl_port_hal.h"
#include "pin_mux.h"

void configure_gpio_pins(uint32_t instance)
{
  switch(instance) {
    case PORTA_IDX:                     /* PTA */
      /* PORTA_PCR14  MMA8451 - INT1 */
      PORT_HAL_SetMuxMode(PORTA,14u,kPortMuxAsGpio);
      /* PORTA_PCR15  MMA8451 - INT2 */
      PORT_HAL_SetMuxMode(PORTA,15u,kPortMuxAsGpio);
      break;
    case PORTB_IDX:                     /* PTB */
```

```
    /* PORTB_PCR19 LED1 - Green */
    PORT_HAL_SetMuxMode(PORTB,19u,kPortMuxAsGpio);
    /* PORTB_PCR18 LED2 - Red */
    PORT_HAL_SetMuxMode(PORTB,18u,kPortMuxAsGpio);
    break;
   case PORTD_IDX:                  /* PTD */
    /* PORTD_PCR1 LED3 - Blue */
    PORT_HAL_SetMuxMode(PORTD,1u,kPortMuxAsGpio);
    /* PORTD_PCR6    LLWU_P15 SW1 - Power Manager demo */
    PORT_HAL_SetMuxMode(PORTD,6u,kPortMuxAsGpio);
    break;
   default:
    break;
  }
}

void configure_i2c_pins(uint32_t instance)
{
  switch(instance) {
   case I2C0_IDX:                  /* I2C0 */
    /* PORTB_PCR2 */
    PORT_HAL_SetMuxMode(PORTB,2u,kPortMuxAlt2);
    PORT_HAL_SetPullCmd(PORTB,2u,true);
    PORT_HAL_SetPassiveFilterCmd(PORTB,2u,false);
    /* PORTB_PCR3 */
    PORT_HAL_SetMuxMode(PORTB,3u,kPortMuxAlt2);
    PORT_HAL_SetPullCmd(PORTB,3u,true);
    PORT_HAL_SetPassiveFilterCmd(PORTB,3u,false);
    break;
   case I2C1_IDX:                  /* I2C1 */
    /* PORTC_PCR1 */
    PORT_HAL_SetMuxMode(PORTC,1u,kPortMuxAlt2);
    PORT_HAL_SetPullCmd(PORTC,1u,true);
    PORT_HAL_SetPassiveFilterCmd(PORTC,1u,false);
    /* PORTC_PCR2 */
    PORT_HAL_SetMuxMode(PORTC,2u,kPortMuxAlt2);
    PORT_HAL_SetPullCmd(PORTC,2u,true);
    PORT_HAL_SetPassiveFilterCmd(PORTC,2u,false);
    break;
   default:
    break;
  }
}

void configure_rtc_pins(uint32_t instance)
{
  /* PORTE_PCR0 */
  PORT_HAL_SetMuxMode(PORTE,0u,kPortMuxAlt4);
}
```

```
void configure_lpsci_pins(uint32_t instance)
{
  switch(instance) {
    case UART0_IDX:                    /* LPSCI0 */
      /* PORTA_PCR1 */
      PORT_HAL_SetMuxMode(PORTA,1u,kPortMuxAlt2);
      /* PORTA_PCR2 */
      PORT_HAL_SetMuxMode(PORTA,2u,kPortMuxAlt2);
      break;
    default:
      break;
  }
}

void configure_uart_pins(uint32_t instance)
{
  switch(instance) {
    case UART1_IDX:                    /* UART1 */
      /* PORTE_PCR0 */
      PORT_HAL_SetMuxMode(PORTE,0u,kPortMuxAlt3);
      /* PORTE_PCR1 */
      PORT_HAL_SetMuxMode(PORTE,1u,kPortMuxAlt3);
      break;
    default:
      break;
  }
}

/* Set-up TSI pins for on board electrodes */
void configure_tsi_pins(uint32_t instance)
{
  switch(instance) {
    case TSI0_IDX:                   /* TSI0 */
      /* PORTB_PCR16 */
      PORT_HAL_SetMuxMode(PORTB,16u,kPortPinDisabled);
      /* PORTB_PCR17 */
      PORT_HAL_SetMuxMode(PORTB,17u,kPortPinDisabled);
      break;
    default:
      break;
  }
}

void configure_spi_pins(uint32_t instance)
{
  switch(instance) {
    case SPI0_IDX:                 /* SPI0 */
      /* PORTC_PCR6 */
      PORT_HAL_SetMuxMode(PORTC,6u,kPortMuxAlt2); /* MOSI */
      /* PORTC_PCR7 */
```

```
    PORT_HAL_SetMuxMode(PORTC,7u,kPortMuxAlt2); /* MISO */
    /* PORTC_PCR5 */
    PORT_HAL_SetMuxMode(PORTC,5u,kPortMuxAlt2); /* SCK */
    /* PORTC_PCR4 */
    PORT_HAL_SetMuxMode(PORTC,4u,kPortMuxAlt2); /* PCS0 */
    break;
   case SPI1_IDX:                    /* SPI1 */
    /* PORTD_PCR6 */
    PORT_HAL_SetMuxMode(PORTD,6u,kPortMuxAlt2); /* MOSI */
    /* PORTD_PCR7 */
    PORT_HAL_SetMuxMode(PORTD,7u,kPortMuxAlt2); /* MISO */
    /* PORTD_PCR5 */
    PORT_HAL_SetMuxMode(PORTD,5u,kPortMuxAlt2); /* SCK */
    /* PORTD_PCR4 */
    PORT_HAL_SetMuxMode(PORTD,4u,kPortMuxAlt2); /* PCS0 */
    break;
   default:
    break;
  }
}

void configure_tpm_pins(uint32_t instance)
{
  switch(instance) {
   case TPM0_IDX:                    /* TPM0 */
     /* PTD_PCR1 TPM0 channel 1 */
     PORT_HAL_SetMuxMode(PORTD,1u,kPortMuxAlt4);
    break;
   default:
    break;
  }
}

void configure_cmp_pins(uint32_t instance)
{
  switch (instance) {
   case CMP0_IDX:
    PORT_HAL_SetMuxMode(PORTC,6u,kPortPinDisabled); /* PTC6 - CMP0_IN0. */
    break;
   default:
    break;
  }
}

void configure_dac_pins(uint32_t instance)
{
  switch (instance) {
   case DAC0_IDX:
    PORT_HAL_SetMuxMode(PORTE,30u,kPortPinDisabled);
    break;
```

```
    default:
      break;
  }
}

/* END pin_mux. */
/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.5 [05.21]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

pin_mux.h

```
#ifndef pin_mux_H_
#define pin_mux_H_

/* MODULE pin_mux. */

/*
** ===================================================================
**     Method     :  configure_gpio_pins (component PinSettings)
*/
/*!
**     @brief
**         GPIO method sets registers according routing settings. Call
**         this method code to route desired pins into:
**         PTA, PTB, PTC, PTD, PTE
**         peripherals.
**     @param
**         uint32_t instance - GPIO instance number 0..4
*/
/* ===================================================================*/
void configure_gpio_pins(uint32_t instance);
/*
** ===================================================================
**     Method     :  configure_i2c_pins(component PinSettings)
*/
/*!
**     @brief
**         I2C method sets registers according routing settings. Call
**         this method code to route desired pins into:
```

```
**       I2C0, I2C1, I2C2
**       peripherals.
**     @param
**       uint32_t instance - I2C instance number 0..2
*/
/* ===================================================================*/
void configure_i2c_pins(uint32_t instance);
/*
** ===================================================================
**     Method     :  configure_rtc_pins(component PinSettings)
*/
/*!
**     @brief
**       RTC method sets registers according routing settings. Call
**       this method code to route desired pins into RTC periphery.
**     @param
**       uint32_t instance - RTC instance number (0 is expected)
*/
/* ===================================================================*/

void configure_rtc_pins(uint32_t instance);
/*
** ===================================================================
**     Method     :  configure_uart_pins(component PinSettings)
*/
/*!
**     @brief
**       UART method sets registers according routing settings. Call
**       this method code to route desired pins into:
**       UART0, UART1, UART2, UART3, UART4, UART5
**       peripherals.
**     @param
**       uint32_t instance - UART instance number 0..5
*/
/* ===================================================================*/
void configure_uart_pins(uint32_t instance);
void configure_lpsci_pins(uint32_t instance);
void configure_spi_pins(uint32_t instance);
void configure_tpm_pins(uint32_t instance);
void configure_cmp_pins(uint32_t instance);

/*
** ===================================================================
**     Method     :  configure_tsi_pins (component PinSettings)
*/
/*!
**     @brief
**       TSI method sets registers according routing settings. Call
**       this method code to route desired pins into:
**       TSI0
```

```
**       peripheral.
**    @param
**       uint32_t instance - TSI instance number 0
*/
/* ========================================================================*/
void configure_tsi_pins(uint32_t instance);

void configure_dac_pins(uint32_t instance);
/* END pin_mux. */
#endif /* #ifndef __pin_mux_H_ */
/*!
** @}
*/
/*
** ###################################################################
**
**    This file was created by Processor Expert 10.5 [05.21]
**    for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```