PROGRAM (Project 3 – By Poorn & Rushi)

FILE: CUSTOM_MAIN.H

```
/*
 * custom_main.h
 *
 *  Created on: Oct 29, 2018
 *      Author: poorn
 */

#ifndef CUSTOM_MAIN_H_
#define CUSTOM_MAIN_H_


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

#define PROJECT_3         1

#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 uint16_t *dma_buff;
extern uint16_t *var_ptr;
extern uint8_t ready;

extern uint32_t volt;

#endif /* CUSTOM_MAIN_H_ */
```

FILE: CUSTOM_UART.H

```
/*
 * Custom_UART.h
 *
 *  Created on: Nov 14, 2018
 *      Author: poorn
 */

#ifndef CUSTOM_INCLUDES_CUSTOM_UART_H_
#define CUSTOM_INCLUDES_CUSTOM_UART_H_

#include "Custom_Main.h"

//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 Custom_UART0_Tx_Byte(Byte data);
void Custom_UART0_Tx_String(char *array);
void Custom_UART0_Rx_Byte(volatile Byte *address);


#endif /* CUSTOM_INCLUDES_CUSTOM_UART_H_ */
```

FILE: CUSTOM_UART.C

```
/*
 * Custom_UART.c
 *
 *  Created on: Nov 14, 2018
 *      Author: poorn
 */

#include "Custom_UART.h"

//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();

	//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);

	//Acutally setting up array
    for(uart_i = 0; uart_i < string_length; uart_i ++)
    {
	//Calling polling transmit byte function repeatedly
	Custom_UART0_Tx_Byte(array[uart_i]);
    }
}

//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);
}
```

FILE: PROJECT_3.H

```
/*
 * project_3.h
 *
 *  Created on: Dec 8, 2018
 *      Author: poorn
 */

#ifndef CUSTOM_INCLUDES_PROJECT_3_H_
#define CUSTOM_INCLUDES_PROJECT_3_H_

#include "Custom_Main.h"
#include "Custom_UART.h"

/* Prototypes */
void Custom_ADC_Init(void);
int Custom_ADC_Calibration(void);
int16_t Custom_ADC_Read(uint8_t chnl);

// DMA

#define  ADC_Block_Size                 64
#define  ADC_Bytes_per_Sample    2
#define ADC_Channel                     0

#define Enable_ADC_Clk()         SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK
#define Enable_PortB_Clk()       SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK

#define Configure_CFG1()         ADC0_CFG1 = (ADC_CFG1_MODE(3)  | ADC_CFG1_ADICLK(0)|
ADC_CFG1_ADIV(1))

#define Configure_Average()             ADC0_SC3 = (ADC_SC3_AVGE_MASK |
ADC_SC3_AVGS(3) | ADC_SC3_CAL_MASK)

#define Wait_for_Cailbration()   while(ADC0_SC3 & ADC_SC3_CAL_MASK)

#define Calibration_Status()            ADC0_SC3 & ADC_SC3_CALF_MASK

#define Configure_ADC_DMA()             ADC0_SC2 |= ADC_SC2_DMAEN_MASK

#define Enable_Continuous_Mode() ADC0_SC3 |= ADC_SC3_ADCO(1)

#define Enable_Differential_Mode()      ADC0_SC1A |= ADC_SC1_DIFF_MASK

#define Disable_ADC()                           ADC0_SC1A |= ADC_SC1_ADCH(31)

#define Enable_ADC()                            ADC0_SC1A = (ADC_SC1_ADCH(ADC_Channel)
| (ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK)))

void Custom_DMA_Init(void);
void DMA0_IRQHandler(void);

#define Enable_DMA_MUX_Clk()            SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK
```

```
#define Enable_DMA_Clk()                SIM_SCGC7 |= SIM_SCGC7_DMA_MASK

#define Disable_DMA_Mux_Ch()            DMAMUX0_CHCFG0 = 0x00

#define Set_DMA_Source_Addr(addr)             DMA_SAR0 = (uint32_t)addr

#define Set_DMA_Destination_Addr(addr)  DMA_DAR0 = (uint32_t)addr

#define Set_BCR(x)                                  DMA_DSR_BCR0 =
DMA_DSR_BCR_BCR(x)

#define Set_DMA_DCR()                               DMA_DCR0 |= (DMA_DCR_EINT_MASK |
DMA_DCR_ERQ_MASK | DMA_DCR_CS_MASK | \

      DMA_DCR_SSIZE(2) | DMA_DCR_DINC_MASK| DMA_DCR_DMOD(4) | DMA_DCR_DSIZE(2))

#define Set_Enable_DMA_Channel()        DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK |
DMAMUX_CHCFG_SOURCE(40)

#define Clear_DMA_DONE()                      DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK


// PIT


void pit_init(void);
void PIT_IRQHandler(void);

#endif /* CUSTOM_INCLUDES_PROJECT_3_H_ */
```

FILE: PROJECT_3.C

```
/*
 * project_3.c
 *
 *  Created on: Dec 8, 2018
 *      Author: poorn
 */

#include "project_3.h"

uint16_t *DMA_Buffer;
uint16_t *var_ptr;
uint8_t ready;


void Custom_ADC_Init(void)
{
	// Enable clocks
	Enable_ADC_Clk();
	Enable_PortB_Clk();

	// Calibrate ADC
	if(Custom_ADC_Calibration())	Output_String("ADC Calibration Failed\n");
	else	Output_String("ADC Calibration Succeed\n");

	//16 bit mode, bus clock as input, divide by 2
	Configure_CFG1();

	//Route ADC sample values to DMA
	Configure_ADC_DMA();

	//Reset this register because it was set in calibration
	ADC0_SC3 = 0;

	Enable_Continuous_Mode();

	//Differential input ADC
	Enable_Differential_Mode();

	//Don't start ADC until entire init has been completed
	Disable_ADC();

	//Setting for a GPIO pin used to measure frequency
	PORTB->PCR[0] |= PORT_PCR_MUX(1);
	PTB->PDDR |= (1 << 0);
}


int Custom_ADC_Calibration(void)
{
	//16 bit mode, bus clock as input, divide by 2
	Configure_CFG1();
```

```
	//Enable hardware average, 32 samples, start calibration
	Configure_Average();

	//While loop till calibration ends
	Wait_for_Cailbration();

	if(Calibration_Status())  return 1;

	uint16_t adc_cal;// calibration variable
	adc_cal = (ADC0_CLPS + ADC0_CLP4 + ADC0_CLP3 + ADC0_CLP2 + ADC0_CLP1 +
ADC0_CLP0) >> 1;
	adc_cal |= 0x8000;
	ADC0_PG = adc_cal;

	adc_cal = (ADC0_CLMS + ADC0_CLM4 + ADC0_CLM3 + ADC0_CLM2 + ADC0_CLM1 +
ADC0_CLM0) >> 1;
	adc_cal |= 0x8000;
	ADC0_MG = adc_cal;

	return 0;
}

int16_t Custom_ADC_Read(uint8_t chnl)
{
	// Write to SC1A to start conversion
	ADC0_SC1A = (chnl & ADC_SC1_ADCH_MASK) |
			(ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK));
	while(ADC0_SC2 & ADC_SC2_ADACT_MASK);   // Conversion in progress
	while(!(ADC0_SC1A & ADC_SC1_COCO_MASK)); // Run until the conversion is
complete
	return ADC0_RA;
}

// DMA

void Custom_DMA_Init(void)
{
	//Not ready
	ready = 0;

	//Aligned Allocation (128 bytes alignment, 64 blocks)
	DMA_Buffer = (uint16_t*) memalign((ADC_Bytes_per_Sample * ADC_Block_Size),
ADC_Block_Size);

	// Enable clocks
	Enable_DMA_MUX_Clk();
	Enable_DMA_Clk();

	// Disable DMA Mux channel
	Disable_DMA_Mux_Ch();


	// Configure DMA - ADC_RA is source, Aligned Allocated buffer is destination,
BCR is 128 bytes
	Set_DMA_Source_Addr(&ADC0_RA);
```

```
	Set_DMA_Destination_Addr(DMA_Buffer);
	Set_BCR((ADC_Bytes_per_Sample * ADC_Block_Size));

	//Enable Interrupt, Peripheral Request, Source and Destination size to 16
butes, Auto Increment in Destination, 128byte Circular buffer
	Set_DMA_DCR();

	// Enable DMA channel and source as ADC0
	Set_Enable_DMA_Channel();

	// Enable interrupt
	NVIC_EnableIRQ(DMA0_IRQn);
}

/*
 * Handles DMA0 interrupt
 * Resets the BCR register and clears the DONE flag
 * */
void DMA0_IRQHandler(void)
{
	//Pin toggling to see a pulse everytime this ISR is called
	PTB->PDOR |= (1 << 0);

	//Clearing DONE flag
	Clear_DMA_DONE();

	//Setting BCR again
	Set_BCR((ADC_Bytes_per_Sample * ADC_Block_Size));
	PTB->PDOR &= ~(1 << 0);

	//Signal to main loop
	ready += 1;
}



// PIT
void pit_init(void)
{
	// Enable PIT clock
	SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;

	// Turn on PIT
	PIT_MCR = 0;

	// Configure PIT to produce an interrupt every 1s
//	PIT_LDVAL0 = 0x1312CFF;	// 1/20Mhz = 50ns	(1s/50ns)-1= 19,999,999 cycles
or 0x1312CFF
	PIT_LDVAL0 = ((0x1312CFF)/(64*2));
	PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK | PIT_TCTRL_TEN_MASK; // Enable interrupt and
enable timer

	// Enable interrupt registers ISER and ICPR
	NVIC_EnableIRQ(PIT_IRQn);
}
```

```
void PIT_IRQHandler(void)
{
	// Clear interrupt
	PIT_TFLG0 = PIT_TFLG_TIF_MASK;

	// Write to SC1A to start conversion
	ADC0_SC1A = (ADC_SC1_ADCH(ADC_Channel) |
				(ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK)));
}
```

FILE: MAIN.C

```
/*
 * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////

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

```c
#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"

uint16_t *DMA_Buffer = NULL;
uint16_t *var_ptr = NULL;
uint32_t volt = 0;
uint8_t ready = 0;

#define Decay0.99

#include "Custom_Main.h"
#include "Custom_UART.h"
#include "project_3.h"


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

	char adc_str[30];
	uint32_t i;
	int32_t	avg;
	avg = 0;
	ready = 0;
	int32_t peak1 = 0;
	uint32_t peak_volt = 0;
	volatile uint32_t peak2 = 0;
	volatile uint32_t pre_peak = 0;

	Custom_UART0_Init();
	Custom_DMA_Init();
	Custom_ADC_Init();
//	pit_init();  //Can be used to slow down ADC and watch its exact behavior


	Output_String("Program Start\n\r");

	Enable_ADC();

//	ADC0_SC1A = (ADC_SC1_ADCH(ADC_Channel) |
//				(ADC0_SC1A & (ADC_SC1_AIEN_MASK | ADC_SC1_DIFF_MASK)));


	while(1)
	{
		if(ready > 0)//Is new data availalbe?
		{
//			ADC0_SC1A |= ADC_SC1_ADCH(31); // Disable module
```

```
                    var_ptr = DMA_Buffer;

                    //Average calculation, Peak calculation, peak of buffer = peak2,
each sample = peak1
                    for(i = 0; i < ADC_Block_Size; i++, var_ptr++)
                    {
//                          sprintf(adc_str, "ADC_val %d : %d\n\r", i,
(int16_t)*var_ptr);
//                          Output_String(adc_str);
                            peak1 = (int16_t)*var_ptr;
                            if(peak1 < 0)peak1 *= -1;
                            if(peak1 > peak2)   peak2 = peak1;
                            avg += (int16_t)*var_ptr;
                    }
                    var_ptr = DMA_Buffer;

                    //Average and voltage calculation
                    avg /= ADC_Block_Size;
                    volt = ((avg * 3300) / 32767);

                    //Either Decay or Peak Update
                    if(pre_peak > peak2)
                    {
                            pre_peak *= Decay;
                            peak_volt = ((pre_peak * 3300) / 32767);
                            sprintf(adc_str, "ADC avg milliVolts: %d   ADC Peak: %d
Peak mV: %d\n", volt, pre_peak, peak_volt);
                            Output_String(adc_str);
                    }
                    else
                    {
                            pre_peak = peak2;
                            peak_volt = ((peak2 * 3300) / 32767);
                            sprintf(adc_str, "ADC avg milliVolts: %d   ADC Peak: %d
Peak mV: %d\n", volt, peak2, peak_volt);
                            Output_String(adc_str);
                    }

                    //Resetting variables
                    peak2 = 0;
                    avg = 0;
                    ready = 0;

//                  ADC0_SC1A = (ADC_SC1_ADCH(ADC_Channel) |
//                                          (ADC0_SC1A & (ADC_SC1_AIEN_MASK |
ADC_SC1_DIFF_MASK)));
            }
      }
}
```