**ECEN 5813**

**Chutao Wei**

**Curry Buscher**

**PES Project 6 Code pdf**

Only project6\_p2 code is shown, because project6\_p2 uses all code from project6\_p1

**readme.md**

**# cu-ecen-5813-project-6**

**\*\*Title:\*\***

PES Project 6 Readme **<**br/**>**

**\*\*Name:\*\***

Curry Buscher, Chutao Wei **<**br/**>**

**\*\*Repository Comments:\*\*** **<**br/**>**

In documents folder: **<**br/**>**

There is PES Project 6.pdf and PES Project 6 code.pdf**<**br/**>**

**<**br/**>**

In project6\_p1/source folder: **<**br/**>**

*\*project6\_p1.c/h\**: is the wrapper for freertos system.**<**br/**>**

*\*logger.c/h\**: contains debug printing function**<**br/**>**

*\*sin.c/h\**: contains code to create sin table in both float or uint16\_t format**<**br/**>**

*\*dac\_adc.c/h\**: contains code to use DAC, ADC, and summary function**<**br/**>**

**<**br/**>**

In project6\_p2/source folder: **<**br/**>**

*\*project6\_p2.c/h\**: is the wrapper for freertos system.**<**br/**>**

*\*led.c/h\**: contains RGB LED control functions**<**br/**>**

*\*gpio.c/h\**: contains gpio control functions**<**br/**>**

*\*logger.c/h\**: contains debug printing function**<**br/**>**

*\*dac\_adc.c/h\**: contains code to use DAC, ADC, DMA and DSP function**<**br/**>**

**<**br/**>**

(see more details in PES Project 6.pdf) **<**br/**>**

**<**br/**>**

**\*\*Project Comments:\*\***

Please use semihost **<**br/**>**

**### \*\*Installation/Execution/Editing Notes:\*\*<br/>**

**\*\*Language:\*\***

C**<**br/**>**

**\*\*Compiler:\*\***

GCC version 7.4.0**<**br/**>**

**\*\*IDE:\*\***

MCUExpresso**<**br/**>**

**\*\*Build Environment:\*\***

Ubuntu 16 or up**<**br/**>**

**\*\*Target Environment:\*\***

KL25Z**<**br/**>**

**\*\*License:\*\***

MIT**<**br/**>**

**project6\_p2.c (main.c)**

/\*

\* Copyright 2016-2020 NXP

\* 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 NXP 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.

\*/

/\*\*

\* **@file** project6\_p1.c

\* **@brief** Application entry point.

\*/

/\* Standard includes. \*/

#include <assert.h>

#include <stdio.h>

#include <string.h>

#include <stdio.h>

/\* Kernel includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "timers.h"

#include "semphr.h"

#include "queue.h"

#include "task.h"

/\* Freescale includes. \*/

#include "fsl\_device\_registers.h"

#include "fsl\_debug\_console.h"

#include "board.h"

#include "peripherals.h"

#include "pin\_mux.h"

#include "clock\_config.h"

#include "MKL25Z4.h"

#include "fsl\_dma.h"

#include "fsl\_dmamux.h"

/\* My Own \*/

#include "sin.h"

#include "dac\_adc.h"

#include "led.h"

#include "logger.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The software timer period. \*/

#define SW\_TIMER\_PERIOD\_MS (100 / portTICK\_PERIOD\_MS)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Prototypes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* The callback function. \*/

void SwTimerCallback**(**TimerHandle\_t xTimer**);** // highest priority 5

void DAC\_task**(**void **\***threadp**);** // priority 4

void ADC\_task**(**void **\***threadp**);** // priority 3

void DMA\_task**(**void **\***threadp**);** // priority 2

void DSP\_task**(**void **\***threadp**);** // priority 1

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Global Variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SemaphoreHandle\_t xBinary\_DAC**;**

SemaphoreHandle\_t xBinary\_ADC**;**

SemaphoreHandle\_t xBinary\_DMA**;**

SemaphoreHandle\_t xBinary\_DSP**;**

SemaphoreHandle\_t xBinary\_LED**;**

uint32\_t Hundredmsec **=** 0**;**

uint32\_t target\_Hundredmsec**;**

bool timer\_flag**;**

uint8\_t round\_count **=** 0**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* @brief Application entry point.

\*/

/\*!

\* **@brief** Main function

\*/

int main**(**void**)**

**{**

TimerHandle\_t SwTimerHandle **=** **NULL;**

/\* Init board hardware. \*/

BOARD\_InitPins**();**

BOARD\_BootClockRUN**();**

BOARD\_InitDebugConsole**();**

SystemCoreClockUpdate**();**

init\_table**();**

init\_table\_uint16**();**

DAC\_ADC\_DMA\_Init**();**

init\_LED**();**

PRINTF**(**"Hello Project 6 Problem 2\n"**);**

/\* Create the Binary \*/

xBinary\_DAC **=** xSemaphoreCreateBinary**();**

**if(**xBinary\_DAC **==** **NULL)**

**{**

PRINTF**(**"Cannot create xBinary\_DAC"**);**

**}**

xBinary\_ADC **=** xSemaphoreCreateBinary**();**

**if(**xBinary\_ADC **==** **NULL)**

**{**

PRINTF**(**"Cannot create xBinary\_ADC"**);**

**}**

xBinary\_DMA **=** xSemaphoreCreateBinary**();**

**if(**xBinary\_DMA **==** **NULL)**

**{**

PRINTF**(**"Cannot create xBinary\_DMA"**);**

**}**

xBinary\_DSP **=** xSemaphoreCreateBinary**();**

**if(**xBinary\_DSP **==** **NULL)**

**{**

PRINTF**(**"Cannot create xBinary\_DSP"**);**

**}**

xBinary\_LED **=** xSemaphoreCreateBinary**();**

**if(**xBinary\_LED **==** **NULL)**

**{**

PRINTF**(**"Cannot create xBinary\_LED"**);**

**}**

xSemaphoreGive**(**xBinary\_LED**);**

/\* Create the task, storing the handle. \*/

BaseType\_t xReturned**[**4**];**

TaskHandle\_t xHandle**[**4**]** **=** **{NULL};**

xReturned**[**0**]** **=** xTaskCreate**(**DAC\_task**,** "DAC\_task"**,**200**,NULL,**configMAX\_PRIORITIES**-**2**,** **&**xHandle**[**0**]);**

**if** **(**xReturned**[**0**]** **!=** pdPASS**)**

**{**

PRINTF**(**"Cannot create DAC\_task"**);**

**}**

xReturned**[**1**]** **=** xTaskCreate**(**ADC\_task**,** "ADC\_task"**,**200**,NULL,**configMAX\_PRIORITIES**-**3**,** **&**xHandle**[**1**]);**

**if** **(**xReturned**[**1**]** **!=** pdPASS**)**

**{**

PRINTF**(**"Cannot create ADC\_task"**);**

**}**

xReturned**[**2**]** **=** xTaskCreate**(**DMA\_task**,** "DMA\_task"**,**200**,NULL,**configMAX\_PRIORITIES**-**4**,** **&**xHandle**[**2**]);**

**if** **(**xReturned**[**1**]** **!=** pdPASS**)**

**{**

PRINTF**(**"Cannot create DMA\_task"**);**

**}**

xReturned**[**3**]** **=** xTaskCreate**(**DSP\_task**,** "DSP\_task"**,**200**,NULL,**configMAX\_PRIORITIES**-**5**,** **&**xHandle**[**3**]);**

**if** **(**xReturned**[**1**]** **!=** pdPASS**)**

**{**

PRINTF**(**"Cannot create DSP\_task"**);**

**}**

/\* Create the software timer. \*/

SwTimerHandle **=** xTimerCreate**(**"SwTimer"**,** /\* Text name. \*/

SW\_TIMER\_PERIOD\_MS**,** /\* Timer period. \*/

pdTRUE**,** /\* Enable auto reload. \*/

0**,** /\* ID is not used. \*/

SwTimerCallback**);** /\* The callback function. \*/

/\* Routine explanation \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 1. Timer task runs every 0.1 sec and release timer Binary

\* 2. DAC task runs when timer Binary release. When done, release DAC Binary .

\* 3. ADC task runs when DAC Binary release, store result in a buffer

\*

\* After 50 rounds of step 1 to step 3, all tasks are ended. And a DSP summary

\* will be printed out

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Start timer. \*/

xTimerStart**(**SwTimerHandle**,** 0**);**

/\* Start scheduling. \*/

vTaskStartScheduler**();**

/\* Infinite While \*/

**while(**1**);**// should never be here

**}**

/\*!

\* **@brief** Software timer callback.xBinary\_timer

\*/

void SwTimerCallback**(**TimerHandle\_t xTimer**)**

**{**

taskENTER\_CRITICAL**();**

xSemaphoreGive**(**xBinary\_DAC**);**

Hundredmsec**++;**

**if** **(**timer\_flag **==**true**)**

**{**

**if(**Hundredmsec **==** target\_Hundredmsec**)**

**{**

timer\_flag **=** false**;**

turn\_LED\_blue**(**off**);**

xSemaphoreGive**(**xBinary\_LED**);**

**}**

**}**

taskEXIT\_CRITICAL**();**

**}**

unsigned int timerGetRunTimeHundredmsec**(**void**)**

**{**

**return** Hundredmsec**;**

**}**

/\*!

\* **@brief** DAC task

\*/

void DAC\_task**(**void **\***threadp**)**

**{**

static int i **=** 0**;**

**while(**1**)**

**{**

xSemaphoreTake**(**xBinary\_DAC**,**portMAX\_DELAY**);**

**if(**xSemaphoreTake**(**xBinary\_LED**,**1**)==**pdTRUE**)**

**{**

turn\_LED\_green**(**on**);**

DAC\_Write**(**i**);**

turn\_LED\_green**(**off**);**

xSemaphoreGive**(**xBinary\_LED**);**

**}**

**else**

**{**

DAC\_Write**(**i**);**

**}**

xSemaphoreGive**(**xBinary\_ADC**);**

**if(**i**==**50**)** i **=** 0**;**

**else** i**++;**

**if(**round\_count **==** 5**)**vTaskSuspend**(NULL);**

**}**

**}**

/\*!

\* **@brief** ADC task

\*/

void ADC\_task**(**void **\***threadp**)**

**{**

static int i **=** 0**;**

**while(**1**)**

**{**

xSemaphoreTake**(**xBinary\_ADC**,**portMAX\_DELAY**);**

ADC\_Read**(**i**);**

**if(**i**==**64**)**

**{**

xSemaphoreGive**(**xBinary\_DMA**);**

i **=** 0**;**

**}**

**else** i**++;**

**if(**round\_count **==** 5**)**vTaskSuspend**(NULL);**

**}**

**}**

/\*!

\* **@brief** ADC task

\*/

void DMA\_task**(**void **\***threadp**)**

**{**

**while(**1**)**

**{**

xSemaphoreTake**(**xBinary\_DMA**,**portMAX\_DELAY**);**

**if(**xSemaphoreTake**(**xBinary\_LED**,**1**)==**pdTRUE**)**

**{**

turn\_LED\_blue**(**on**);**

target\_Hundredmsec **=** Hundredmsec**+**3**;**

timer\_flag **=** true**;**

**}**

DMA\_Transfer**();**

xSemaphoreGive**(**xBinary\_DSP**);**

round\_count**++;**

**if(**round\_count **==** 5**)**vTaskSuspend**(NULL);**

**}**

**}**

void DSP\_task**(**void **\***threadp**)**

**{**

**while(**1**)**

**{**

xSemaphoreTake**(**xBinary\_DSP**,**portMAX\_DELAY**);**

DSP\_Summary**(**round\_count**);**

**if(**round\_count **==** 5**)** **while(**1**);**

**}**

**}**

**gpio.c**

/\*

\* gpio.c

\*

\* Created on: Feb 11, 2020

\* Author: chutao

\*

\* Minic the functions from fsl\_gpio.c

\* Still use MKL25Z4.h for hardware addresses

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** <stdio.h>

**#include** <stdint.h>

**#include** "gpio.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **set\_GPIO\_Pinout**(GPIO\_Type \*port, uint32\_t pin)

{

port->PSOR = (0x1 << pin);

}

**void** **clear\_GPIO\_Pinout**(GPIO\_Type \*port, uint32\_t pin)

{

port->PCOR = (0x1 << pin);

}

**void** **toggle\_GPIO\_Pinout**(GPIO\_Type \*port, uint32\_t pin)

{

port->PTOR = (0x1 << pin);

}

**void** **init\_GPIO\_Pin**(GPIO\_Type \*port, uint32\_t pin,

gpio\_pin\_direct\_t pin\_direction, uint8\_t pin\_data)

{

**if** (pin\_direction == *GPIO\_DigitalInput*)

{

// Set pin to input direction

port->PDDR &= ~(0x1 << pin);

}

**else** **if** (pin\_direction == *GPIO\_DigitalOutput*)

{

// Set pin to output direction

port->PDDR |= (0x1 << pin);

**if** (pin\_data)

{

set\_GPIO\_Pinout(port,pin);

}

**else**

{

clear\_GPIO\_Pinout(port,pin);

}

}

**else**

{

**#ifdef** LOGGING\_DEBUG

// **TODO**: Debug message

**#endif**

}

}

**led.c**

**/\***

\* led.c

\*

\* Created on: Feb 11, 2020

\* Author: chutao

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** <logger.h>

**#include** <stdint.h>

**#include** "gpio.h"

**#include** "led.h"

**#include** "timer.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global Varibles \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

led\_color\_t color = *red*;

**const** **char** \* led\_color\_string[3] ={"off","on","toggle"};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**void** **init\_LED**(**void**)

{

init\_GPIO\_Pin(LED3\_RED\_PORT, LED3\_RED\_PIN, *GPIO\_DigitalOutput*, 1);

init\_GPIO\_Pin(LED3\_GREEN\_PORT, LED3\_GREEN\_PIN, *GPIO\_DigitalOutput*, 1);

init\_GPIO\_Pin(LED3\_BLUE\_PORT, LED3\_BLUE\_PIN, *GPIO\_DigitalOutput*, 1);

}

**void** **turn\_LED**(led\_state\_t LED\_state)

{

**if** (color == *red*)

{

turn\_LED\_red(LED\_state);

}

**else** **if** (color == *green*)

{

turn\_LED\_green(LED\_state);

}

**else** **if** (color == *blue*)

{

turn\_LED\_blue(LED\_state);

}

**else**

{

LOG\_ERROR("Unexpected led\_state\_t");

}

}

**void** **change\_LED\_color**(led\_color\_t LED\_color)

{

color = LED\_color;

}

**void** **turn\_LED\_red**(led\_state\_t LED\_state)

{

LOG\_DEBUG("Turn LED red %s",led\_color\_string[LED\_state]);

color = *red*;

**if** (LED\_state == *off*)

{

set\_GPIO\_Pinout(LED3\_RED\_PORT, LED3\_RED\_PIN);

}

**else** **if** (LED\_state == *on*)

{

clear\_GPIO\_Pinout(LED3\_RED\_PORT, LED3\_RED\_PIN);

}

**else** **if** (LED\_state == *toggle*)

{

toggle\_GPIO\_Pinout(LED3\_RED\_PORT, LED3\_RED\_PIN);

}

**else**

{

LOG\_ERROR("Unexpected led\_state\_t");

}

}

**void** **turn\_LED\_green**(led\_state\_t LED\_state)

{

LOG\_DEBUG("Turn LED green %s",led\_color\_string[LED\_state]);

color = *green*;

**if** (LED\_state == *off*)

{

set\_GPIO\_Pinout(LED3\_GREEN\_PORT, LED3\_GREEN\_PIN);

}

**else** **if** (LED\_state == *on*)

{

clear\_GPIO\_Pinout(LED3\_GREEN\_PORT, LED3\_GREEN\_PIN);

}

**else** **if** (LED\_state == *toggle*)

{

toggle\_GPIO\_Pinout(LED3\_GREEN\_PORT, LED3\_GREEN\_PIN);

}

**else**

{

LOG\_ERROR("Unexpected led\_state\_t");

}

}

**void** **turn\_LED\_blue**(led\_state\_t LED\_state)

{

LOG\_DEBUG("Turn LED blue %s",led\_color\_string[LED\_state]);

color = *blue*;

**if** (LED\_state == *off*)

{

set\_GPIO\_Pinout(LED3\_BLUE\_PORT, LED3\_BLUE\_PIN);

}

**else** **if** (LED\_state == *on*)

{

clear\_GPIO\_Pinout(LED3\_BLUE\_PORT, LED3\_BLUE\_PIN);

}

**else** **if** (LED\_state == *toggle*)

{

toggle\_GPIO\_Pinout(LED3\_BLUE\_PORT, LED3\_BLUE\_PIN);

}

**else**

{

LOG\_ERROR("Unexpected led\_state\_t");

}

}

**logger.c**

/\*

\* log.c

\*

\* Created on: Dec 18, 2018

\* Author: Chutao Wei

\*

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Include \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdint.h>

#include "logger.h"

#include "project6\_p2.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Global \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

log\_status\_t log\_status **=** disable**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* **@return** a timestamp value for the logger, typically based on a free running timer.

\* This will be printed at the beginning of each log message.

\*/

unsigned int loggerGetTimestamp**(**void**)**

**{**

**return** **(**unsigned int**)(**timerGetRunTimeHundredmsec**());**

**}**

#ifdef HMS\_FORMAT

uint32\_t loggerGetTimestampHour**()**

**{**

uint64\_t msec **=** timerGetRunTimeMilliseconds**();**

**return** **(**uint32\_t**)(**msec**/**360000**);**

**}**

uint32\_t loggerGetTimestampMinute**()**

**{**

uint64\_t msec **=** timerGetRunTimeMilliseconds**();**

**return** **(**uint32\_t**)(**msec**/**6000**);**

**}**

uint32\_t loggerGetTimestampSecond**()**

**{**

uint64\_t msec **=** timerGetRunTimeMilliseconds**();**

**return** **(**uint32\_t**)(**msec**/**1000**);**

**}**

uint32\_t loggerGetTimestampTenthSec**()**

**{**

uint64\_t msec **=** timerGetRunTimeMilliseconds**();**

**return** **(**uint32\_t**)(**msec**/**100**);**

**}**

#endif

**dac\_adc.c**

/\*

\* dac\_adc.c

\*

\* Created on: Apr 27, 2020

\* Author: chutao

\*/

/\* Kernel includes. \*/

#include "FreeRTOS.h"

#include "task.h"

#include "timers.h"

#include "semphr.h"

#include "queue.h"

#include "task.h"

#include "dac\_adc.h"

#include "fsl\_dac.h"

#include "fsl\_adc16.h"

#include "fsl\_debug\_console.h"

#include "fsl\_dma.h"

#include "fsl\_dmamux.h"

#include "MKL25Z4.h"

#include "sin.h"

#include "project6\_p2.h"

#include "math.h"

#include "logger.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Defines

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define DEMO\_ADC16\_CHANNEL\_GROUP 0U

#define DEMO\_ADC16\_USER\_CHANNEL 0U /\* PTE20, ADC0\_SE0 \*/

#define NUM\_POINTS 50

#define DMA\_CHANNEL 0

#define DMA\_SOURCE 63

#define BUF\_SIZE 64

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Global Variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

volatile bool g\_Adc16ConversionDoneFlag **=** false**;**

volatile uint32\_t g\_Adc16ConversionValue **=** 0**;**

adc16\_channel\_config\_t g\_adc16ChannelConfigStruct**;**

dma\_handle\_t g\_DMA\_Handle**;**

volatile bool g\_Transfer\_Done **=** false**;**

uint16\_t ADC\_buf**[**BUF\_SIZE**];**

uint16\_t DSP\_buf**[**BUF\_SIZE**];**

unsigned int DMA\_start\_time**;**

unsigned int DMA\_end\_time**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// code provided by the fsl library

uint16\_t average**(**uint16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int average **=** arr**[**0**];**

**for** **(**i **=** 0**;** i **<** n**;** i**++)**

average **=** average **+** arr**[**i**];**

**return** **(**average**/**n**);**

**}**

// source: https://www.geeksforgeeks.org/c-program-find-largest-element-array/

uint16\_t max**(**uint16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int max **=** arr**[**0**];**

**for** **(**i **=** 1**;** i **<** n**;** i**++)**

**if** **(**arr**[**i**]** **>** max**)**

max **=** arr**[**i**];**

**return** max**;**

**}**

uint16\_t min**(**uint16\_t arr**[],** uint32\_t n**)**

**{**

int i**;**

int min **=** arr**[**0**];**

**for** **(**i **=** 1**;** i **<** n**;** i**++)**

**if** **(**arr**[**i**]** **<** min**)**

min **=** arr**[**i**];**

**return** min**;**

**}**

// from https://www.programiz.com/c-programming/examples/standard-deviation

uint16\_t SD**(**uint16\_t arr**[],** uint16\_t mean**,** uint32\_t n**)**

**{**

uint32\_t variance **=** 0**;**

int i**;**

**for** **(**i **=** 0**;** i **<** 10**;** **++**i**)**

variance **+=** pow**(**arr**[**i**]** **-** mean**,** 2**);**

**return** sqrt**(**variance **/** n**);**

**}**

// interrupt hanlder

void ADC0\_IRQHandler**(**void**)**

**{**

g\_Adc16ConversionDoneFlag **=** true**;**

/\* Read conversion result to clear the conversion completed flag. \*/

g\_Adc16ConversionValue **=** ADC16\_GetChannelConversionValue**(**ADC0**,** DEMO\_ADC16\_CHANNEL\_GROUP**);**

**}**

/\* User callback function for DMA transfer. \*/

void DMA\_Callback**(**dma\_handle\_t **\***handle**,** void **\***param**)**

**{**

g\_Transfer\_Done **=** true**;**

**}**

// init function

void DAC\_ADC\_DMA\_Init**(**void**)**

**{**

EnableIRQ**(**ADC0\_IRQn**);**

adc16\_config\_t adc16ConfigStruct**;**

dac\_config\_t dacConfigStruct**;**

/\* Configure the DAC. \*/

/\*

\* dacConfigStruct.referenceVoltageSource = kDAC\_ReferenceVoltageSourceVref2;

\* dacConfigStruct.enableLowPowerMode = false;

\*/

DAC\_GetDefaultConfig**(&**dacConfigStruct**);**

DAC\_Init**(**DAC0**,** **&**dacConfigStruct**);**

DAC\_Enable**(**DAC0**,** true**);** /\* Enable output. \*/

/\* Configure the ADC16. \*/

/\*

\* adc16ConfigStruct.referenceVoltageSource = kADC16\_ReferenceVoltageSourceVref;

\* adc16ConfigStruct.clockSource = kADC16\_ClockSourceAsynchronousClock;

\* adc16ConfigStruct.enableAsynchronousClock = true;

\* adc16ConfigStruct.clockDivider = kADC16\_ClockDivider8;

\* adc16ConfigStruct.resolution = kADC16\_ResolutionSE12Bit;

\* adc16ConfigStruct.longSampleMode = kADC16\_LongSampleDisabled;

\* adc16ConfigStruct.enableHighSpeed = false;

\* adc16ConfigStruct.enableLowPower = false;

\* adc16ConfigStruct.enableContinuousConversion = false;

\*/

ADC16\_GetDefaultConfig**(&**adc16ConfigStruct**);**

#if defined(BOARD\_ADC\_USE\_ALT\_VREF)

adc16ConfigStruct**.**referenceVoltageSource **=** kADC16\_ReferenceVoltageSourceValt**;**

#endif

ADC16\_Init**(**ADC0**,** **&**adc16ConfigStruct**);**

/\* Make sure the software trigger is used. \*/

ADC16\_EnableHardwareTrigger**(**ADC0**,** false**);**

#if defined(FSL\_FEATURE\_ADC16\_HAS\_CALIBRATION) && FSL\_FEATURE\_ADC16\_HAS\_CALIBRATION

**if** **(**kStatus\_Success **==** ADC16\_DoAutoCalibration**(**ADC0**))**

**{**

//PRINTF("\r\nADC16\_DoAutoCalibration() Done.");

**}**

**else**

**{**

//PRINTF("ADC16\_DoAutoCalibration() Failed.\r\n");

**}**

#endif /\* FSL\_FEATURE\_ADC16\_HAS\_CALIBRATION \*/

/\* Prepare ADC channel setting \*/

g\_adc16ChannelConfigStruct**.**channelNumber **=** DEMO\_ADC16\_USER\_CHANNEL**;**

g\_adc16ChannelConfigStruct**.**enableInterruptOnConversionCompleted **=** true**;**

#if defined(FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE) && FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE

g\_adc16ChannelConfigStruct**.**enableDifferentialConversion **=** false**;**

#endif /\* FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE \*/

/\* Configure DMAMUX \*/

DMAMUX\_Init**(**DMAMUX0**);**

DMAMUX\_SetSource**(**DMAMUX0**,** DMA\_CHANNEL**,** DMA\_SOURCE**);**

DMAMUX\_EnableChannel**(**DMAMUX0**,** DMA\_CHANNEL**);**

/\* Configure DMA one shot transfer \*/

DMA\_Init**(**DMA0**);**

DMA\_CreateHandle**(&**g\_DMA\_Handle**,** DMA0**,** DMA\_CHANNEL**);**

DMA\_SetCallback**(&**g\_DMA\_Handle**,** DMA\_Callback**,** **NULL);**

**}**

void DAC\_Write**(**int i**)**

**{**

LOG\_DEBUG**(**"DAC\_Write\n"**);**

DAC\_SetBufferValue**(**DAC0**,** 0U**,** sin\_lookup\_table\_uint16**[**i**]);**

**}**

void ADC\_Read**(**int i**)**

**{**

LOG\_DEBUG**(**"ADC\_Read\n"**);**

g\_Adc16ConversionDoneFlag **=** false**;**

ADC16\_SetChannelConfig**(**ADC0**,** DEMO\_ADC16\_CHANNEL\_GROUP**,** **&**g\_adc16ChannelConfigStruct**);**

**while** **(!**g\_Adc16ConversionDoneFlag**);**

ADC\_buf**[**i**]** **=** g\_Adc16ConversionValue**;**

**}**

void DMA\_Transfer**(**void**)**

**{**

LOG\_DEBUG**(**"DMA\_Transfer\n"**);**

DMA\_start\_time **=** timerGetRunTimeHundredmsec**();**

dma\_transfer\_config\_t transferConfig**;**

DMA\_PrepareTransfer**(&**transferConfig**,** ADC\_buf**,** **sizeof(**ADC\_buf**[**0**]),**

DSP\_buf**,** **sizeof(**DSP\_buf**[**0**]),** **sizeof(**ADC\_buf**),** kDMA\_MemoryToMemory**);**

DMA\_SubmitTransfer**(&**g\_DMA\_Handle**,** **&**transferConfig**,** kDMA\_EnableInterrupt**);**

DMA\_StartTransfer**(&**g\_DMA\_Handle**);**

**while** **(**g\_Transfer\_Done **!=** true**);**

DMA\_end\_time **=** timerGetRunTimeHundredmsec**();**

**}**

void DSP\_Summary**(**uint8\_t round\_count**)**

**{**

PRINTF**(**"DSP Summary Report %d\t \n"**,**round\_count**);**

uint16\_t DSP\_ave **=** average**(**DSP\_buf**,** BUF\_SIZE**);**

uint16\_t DSP\_max **=** max**(**DSP\_buf**,** BUF\_SIZE**);**

uint16\_t DSP\_min **=** min**(**DSP\_buf**,** BUF\_SIZE**);**

uint16\_t standard\_deviation **=** SD**(**DSP\_buf**,**DSP\_ave**,**BUF\_SIZE**);**

float DSP\_ave\_f **=** 3.3**/**4096**\***DSP\_ave**;**

float DSP\_max\_f **=** 3.3**/**4096**\***DSP\_max**;**

float DSP\_min\_f **=** 3.3**/**4096**\***DSP\_min**;**

float standard\_deviation\_f **=** 3.3**/**4096**\***standard\_deviation**;**

PRINTF**(**"Ave\t\tMax\t\tMin\t\tSD\t\t \n"**);**

PRINTF**(**"%f\t%f\t%f\t%f\t \n"**,**DSP\_ave\_f**,**DSP\_max\_f**,**DSP\_min\_f**,**standard\_deviation\_f**);**

PRINTF**(**"\t \n"**);**

**}**