PES Project 6 Readme

Jack Campbell

Description

This repo contains custom sources and makefiles for Project 6 as well as adapted and generated code from MCUXpresso and the KL25Z SDK. It also contains the scope images for program 1 and program 2.

This project contains two configurations: Debug and Status. The debug build prints out all of the individual sample reads/writes.

Observations

The buffer structure I used for the ADC/DSP buffers is just a 64-integer circular buffer containing the DSP values. These are converted to floating point representations as needed. I used some optimizations for calculating the sine and for squaring values, but did leave a single square root in the program for calculating the standard deviation.

I removed the log_data and log_integer functionality from my logger, since I felt it wasn't needed with the LOG_STRING_ARGS macro I have and they were wasting SRAM with their function-static character buffers.

I also did the mutex extra credit part of the assignment by wrapping all set_led calls in the FreeRTOS semaphore functions and holding a lock on the LED when the DMA transfer begins, and releasing .5 seconds later.

This was a good exercise to get used to creating a real-time program as a series of tasks. There were hiccups when getting used to what was and wasn't allowed (any function using FreeRTOS calls needed to be a task, and all tasks are not allowed to return).

I had trouble getting the DMA interrupt to fire, so I just used a polling/callback approach to finish the other parts of the assignment.

It took me awhile to learn that I needed to physically connect the DAC and ADC pins. Once I did that, my ADC reading code all worked.

Installation/Execution Notes

These are the steps to build the project in MCUXpresso.

1. Clone the repo
2. In MCUXpresso, click `New > Project`.
3. Select `Makefile project with existing code...`

4. Unselect C++, enter a project name, browse to the directory of the repo, and select `NXP MCU Tools`, then hit next.
5. Now set up the configurations. Right click the project,
6. Hit Properties
7. Uncheck "Generate makefiles"
8. Add "Debug" to the build directory path in the same dialog.
9. Do the same for Normal and Test configurations.

Running the FB builds

1. Right click on the project name in the file hierarchy, select `Debug as > Debug configurations...`
2. Select `GDB PEMicro Interface Debugging`
3. Hit `New launch configuration`
4. Select a name for the output configuration (you need one for both Release and Debug)
5. Set the `C/C++ Application` field to the binary you want to run, either `Debug/output/kl25z_debug.axf` for Debug or `Release/output/kl25z_run.axf` for Release
6. Hit Apply
7. Hit Debug
8. The program should run in the console below, provided the board is connected successfully.

CODE

```
/*
 * @file circular_buffer.h
 * @brief Project 6
 *
 * @details This file contains code for a circular buffer.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *  LEVERAGED API AND IMPLEMENTATION FROM:
 *  https://embeddedartistry.com/blog/2017/05/17/creating-a-circular-
buffer-in-c-and-c/
 */


#ifndef CIRCULAR_BUFFER_H
#define CIRCULAR_BUFFER_H
#include <stdint.h>
#include <stdbool.h>

/**
 * @brief Buffer error codes.
 */
```

```
typedef enum buff_err {
        buff_err_success,
        buff_err_full,
        buff_err_empty,
        buff_err_invalid
} buff_err;

/**
 * @brief Opaque struct for circular buffer
 */
typedef struct circular_buf_t {
        uint32_t * buffer;
        size_t write;
        size_t read;
        size_t max;
        bool full;
} circular_buf_t;

/**
 * @brief Circular buffer handle type to use with free functions
 */
typedef circular_buf_t* cbuf_handle_t;

/**
 * @brief Create a new buffer
 * @param inSize Capacity of the buffer
 * @return A circular buffer handle
 */
cbuf_handle_t circular_buf_init(size_t inSize);

/**
 * @brief Free a circular buffer and all associated heap memory
 * @param inBufHandle Handle to the buffer to free
 */
void circular_buf_free(cbuf_handle_t inBufHandle);

/**
 * @brief Resize a circular buffer
 * @param inOutBufHandle The buffer to resize
 * @param inSize The size to resize to
 * @return Whether the operation succeeded
 */
buff_err circular_buf_resize(cbuf_handle_t* inOutBufHandle, size_t
inSize);

/**
 * @brief Push a new element into the circular buffer
 * @param inBufHandle Buffer to push to
 * @param inData Data to push into the buffer
 * @return Whether the operation succeeded. Errors if not.
 */
buff_err circular_buf_push(cbuf_handle_t inBufHandle, uint32_t inData);

/**
```

```
 * @brief Push a new element to the circular buffer, resizing if full
 * @param inOutBufHandle A pointer to the handle to push to
 * @param inData The data to push
 * @return Whether the operation succeeded
 */
buff_err circular_buf_push_resize(cbuf_handle_t* inOutBufHandle, uint32_t
inData);

/**
 * @brief Pop an element from the buffer
 * @param inBufHandle The buf to access
 * @param outData The data accessed
 * @return Whether the operation was successful. Error if empty.
 */
buff_err circular_buf_pop(cbuf_handle_t inBufHandle, uint32_t * outData);

/**
 * @brief Return whether the buffer is empty
 * @param inBufHandle The buffer to check
 * @return
 */
bool circular_buf_empty(cbuf_handle_t inBufHandle);

/**
 * @brief Whether the buffer is full
 * @param inBufHandle The buffer to check
 * @return
 */
bool circular_buf_full(cbuf_handle_t inBufHandle);

/**
 * @brief Capacity of the buffer
 * @param inBufHandle The buffer to check
 * @return
 */
size_t circular_buf_capacity(cbuf_handle_t inBufHandle);

/**
 * @brief The number of elements in the buffer
 * @param inBufHandle The buffer to check
 * @return
 */
size_t circular_buf_size(cbuf_handle_t inBufHandle);

/**
 * @brief Reset the bookkeeping for the buffer
 * @param inBufHandle The buffer to reset
 */
void circular_buf_reset(cbuf_handle_t cbuf);

/**
 * @brief Copies the bookkeeping from one buffer to another
 * @param fromBufHandle The buffer to copy from
 * @param toBufHandle The buffer to copy to
 */
void circular_buf_copy(cbuf_handle_t fromBufHandle,
                               cbuf_handle_t toBufHandle);

#endif

/*
 * @file dac_adc.h
 * @brief Project 6
 *
 * @details This file contains code for using the ADC and DAC.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 */

#ifndef __dach__
#define __dach__

#include <stdint.h>

/**
 * Init the DAC.
 */
void dac_init();

/**
 * Init the adc.
 */
void adc_init();

/**
 * Write a value to the DAC.
 */
void write_dac(uint32_t inVal);

/**
 * Read a value from the ADC.
 */
uint32_t read_adc();

#endif
/*
 * @file dma.h
 * @brief Project 6
 *
 * @details This file contains code for using the DMA controller.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 */

#ifndef __dmah__
#define __dmah__
#include <stdint.h>

/**
 * A callback type to pass to the DMA transfer.
 */
typedef void (*dma_callback)();

// void* so that the init can be a task

/**
 * DMA init.
 * \note This function takes a void* so that it can be run as a task.
 * I read that enabling interrupts must be done after the scheduler
starts,
 * but I am not sure whether this is true.
 */
void dma_init(void* cookie);

/**
 * Request a DMA transfer.
 * \param srcAddr The address to transfer from.
 * \param destAddr The address to transfer to.
 * \param tansferBytes Number of bytes to transfer.
 * \param inCallback A callback to fire when the transfer completes.
 */
void dma_transfer(uint32_t* srcAddr,
        uint32_t* destAddr,
        uint32_t transferBytes,
                dma_callback inCallback);

#endif
/*
 * @file handle_led.h
 * @brief Project 6
 *
 * @details Contains the prototype for handling LEDs on various platforms.
 *          This may be actually turning an LED on and off or just
printing
 *          what the LED state would be, in the absence of LEDs.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */
#ifndef PES_PROJECT_4_HANDLE_LED_H
#define PES_PROJECT_4_HANDLE_LED_H

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

/**
 * @brief Initializes the LED functions.
 */
void leds_init();

/**
 * set_led
 *
 * @brief Sets the LED state.
 * @details This function, depending on platform, may or may not
 *          control a physical LED. On PC, it will simply print the
 *          state of what the LED would be.
 * @param inValue The on/off state of the LED to set.
 * @param inColor The color of the LED to set.
 */
void set_led(uint8_t inValue, enum COLOR inColor);

#endif //PES_PROJECT_2_HANDLE_LED_H
/*
 * @file led_types.h
 * @brief Project 6
 *
 * @details Defines enumerations and constants used to describe colors and
 *          on/off states for LEDs.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef PES_PROJECT_2_LED_TYPES_H
#define PES_PROJECT_2_LED_TYPES_H

/**
 * COLOR
 *
 * @brief The possible color values of the LED.
 */
```

```
enum COLOR
{
    RED = 0,
    GREEN,
    BLUE,
    NUM_COLORS
};

/**
 * COLOR_STRINGS
 *
 * @brief String representations of the COLOR enum, used for printing.
 */
static const char * const COLOR_STRINGS[3] = {
        "RED",
        "GREEN",
        "BLUE"
};

#endif //PES_PROJECT_2_LED_TYPES_H
/*
 * @file logger.h
 * @brief Project 6
 *
 * Interface to use for logging on either PC or KL25Z.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef PES_PROJECT_3_LOGGER_H
#define PES_PROJECT_3_LOGGER_H

#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

/**
 * @brief The category in which log messages should appear.
 */
typedef enum LogSeverity
{
        LOG_SEVERITY_TEST,
        LOG_SEVERITY_DEBUG,
        LOG_SEVERITY_STATUS,
        NUM_LOG_SEVERITIES
} LogSeverity_t;
```

```
/**
 * @brief The module associated with a log message.
 */
typedef enum LogModule
{
        LOG_MODULE_MAIN,
        LOG_MODULE_LED,
        LOG_MODULE_DMA,
        LOG_MODULE_SETUP_TEARDOWN,
        LOG_MODULE_CIRCULAR_BUFFER,
        LOG_MODULE_TASKS,
        LOG_MODULE_TIME,
        LOG_MODULE_POST,
        LOG_MODULE_UART,
        LOG_MODULE_SINE,
        NUM_LOG_MODULES
} LogModule_t;

/**
 * @brief Log_enable - begin printing log messages when called
 */
void log_enable(LogSeverity_t inSeverity);

/**
 * @brief Log_disable - ignore any log messages until re-enabled
 */
void log_disable();

/**
 * Set log severity for the module.
 */
void log_set_severity(LogSeverity_t inSeverity);

/**
 * @brief Log_status - returns a flag to indicate whether the logger is
enabled or disabled
 * @return Whether the log is currently enabled.
 */
bool log_enabled();



/**
 * @brief Log a string.
 * @param inModule The module associated with this log statement.
 * @param inFuncName The function name from which we are logging.
 * @param inSeverity The severity of this log statement.
 * @param inString
 * @param ... Printf style args.
 */
void log_string(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity, const char* inString, ...);

/**
 * @brief A macro used to wrap a log_string. Includes function name
```

```
automatically and accepts printf-style args.
 */
#define LOG_STRING_ARGS(category, severity, fmt, ...) \
{ \
        log_string(category, __func__, severity, fmt, __VA_ARGS__); \
}

/**
 * @brief A macro used to wrap a log_string. Includes function name
automatically.
 */
#define LOG_STRING(category, severity, fmt) \
{ \
        log_string(category, __func__, severity, fmt); \
}


#endif
/*
 * @file post.h
 * @brief Project 6
 *
 * A power on self test.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef POSTH
#define POSTH

#include <stdbool.h>

/**
 * @brief Power on self test that checks for connection with peripherals
and board functions.
 * @return Whether the test succeeded.
 */
bool power_on_self_test();

#endif
/*
 * @file setup_teardown.h
 * @brief Project 6
 *
 * @details Contains the setup and cleanup prototypes.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef PES_PROJECT_4_SETUP_TEARDOWN_H
#define PES_PROJECT_4_SETUP_TEARDOWN_H

/**
 * initialize
 *
 * @details Initializes components needed by a particular platform,
 *          such as LEDs and UART.
 *
 */
void initialize(void);


/**
 * terminate
 *
 * @details Cleans up any required components on a particular platform.
 *
 */
void terminate(void);

#endif //PES_PROJECT_2_SETUP_TEARDOWN_H
/*
 * @file sine.h
 * @brief Project 6
 *
 * @details Contains the setup and cleanup prototypes.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef __sinelookup__
#define __sinelookup__

#include <stdint.h>

/**
 * Initialize the sine wave lookup table.
 */
void sine_init();

// get next sine sample
```

```
/**
 * Get the next sine sample in the lookup table.
 */
uint32_t get_next_sine_sample();

#endif
/*
 * @file tasks.h
 * @brief Project 6
 *
 * @details Contains the FreeRTOS tasks for this program.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#ifndef __tashsh__
#define __tashsh__

/**
 * Initialize the tasks and FreeRTOS.
 */
void tasks_init();

#endif
/*
 * @file time.h
 * @brief Project 6
 *
 * @details Contains interface for telling and initializing time.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *
 */

#ifndef __timeh__
#define __timeh__

#include <stdint.h>

/**
 * The type of a timestamp string, holding the various components of the
string.
 */
```

```
typedef struct timestamp_str
{
        char hours[4];
        char mins[4];
        char secs[4];
        char tens[4];
} timestamp_str;

/**
 * @brief Initialize the time module
 */
void time_init();

/**
 * @brief How much time has passed, in tenths of a second
 * @param since Base time to calculate current difference with
 * @return
 */
uint64_t time_passed(uint64_t since);

/**
 * @brief Return current time in tenth of a second
 */
uint64_t time_now();

/**
 * Get the current timestamp as a string.
 */
void timestamp_now(timestamp_str* outTimestamp);

#endif
/*
 * @file uart.h
 * @brief Project 6
 *
 * @details Contains interface for UART communications.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *  LEVERAGED CODE:
 *  https://github.com/alexander-g-dean/ESF/tree/master/Code/Chapter_8/Serial-Demo
 */

#ifndef __uart_H__
#define __uart_H__

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

```
/**
 * @brief Whether to use polling or interrupts for UART communication
 */
#define USE_UART_INTERRUPTS     (0) // 0 for polled UART communications, 1
for interrupt-driven

/**
 * @brief How much to oversample the uart clock
 */
#define UART_OVERSAMPLE_RATE    (16)

/**
 * @brief A define for 48000000 clock rate
 */
#define SYS_CLOCK
(48e6)

/**
 * @brief Initialize the uart module
 * @param baud_rate Bits per second to use.
 */
void uart_init(int64_t baud_rate);

/**
 * @brief Poll to get a character
 * @param outChar The character received
 * @return Whether there was a character to get.
 */
bool uart_getchar(uint8_t* outChar);

/**
 * @brief Transmit a character over UART
 * @param ch Character to transmit
 */
void uart_putchar(char ch);

/**
 * @brief Get whether we have space to transmit a character
 */
bool uart_putchar_space_available();

/**
 * @brief Get whether a character has been received
 */
bool uart_getchar_present();

/**
 * @brief Send an entire string over UART
 * @param inChar String to send
 */
void uart_put_string(const char* inChar);

/**
```

```
 * @brief Respond to received characters by transmitting them back.
 * @param outChar Output parameter for the character received
 * @return Whether we echo'd.
 */
bool uart_echo(uint8_t* outChar);

#endif
/*
 * @file circular_buffer.c
 * @brief Project 6
 *
 * @details This file contains code for a circular buffer.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *  LEVERAGED API AND IMPLEMENTATION FROM:
 *  https://embeddedartistry.com/blog/2017/05/17/creating-a-circular-
buffer-in-c-and-c/
 */

#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
#include "circular_buffer.h"
#include <assert.h>
#include <stdlib.h>
#include <stddef.h>

/**
 * ABS
 * @details Leveraged code in this file includes the ABS macro, taken from
 *          Slide 30 of "More C Topics" lecture from ECEN 5813
 *          Link:
https://canvas.colorado.edu/courses/53078/files/folder/Class%20FIles?
preview=7085601
 *  Takes a number and returns the absolute value of that number.
 */
#define ABS(x) ((x)>0?(x):-(x))

/**
 * @brief List node for keeping track of owned buffers
 */
struct mem_list_node
{
        circular_buf_t* data;
        size_t size;
        struct mem_list_node* next;
};
```

```
// sentinel struct, always has null data
static struct mem_list_node memListHead = { NULL, 0, NULL };

/**
 * @brief Whether the given handle is owned currently or null or garbage.
 * @param inHandle Handle to check.
 * @return
 */
bool bufferIsOwned(cbuf_handle_t inHandle)
{
        circular_buf_t* buffer = (circular_buf_t*)inHandle;
        if(buffer)
        {
                // try and see if the requested location resides in owned
memory
                struct mem_list_node* iter = &memListHead;
                while (iter)
                {
                        if(iter->data == buffer)
                        {
                                return true;
                        }
                        iter = iter->next;
                }
        }
        return false;
}

void circular_buf_reset(cbuf_handle_t inBufHandle)
{
    if(bufferIsOwned(inBufHandle))
    {
        inBufHandle->read = 0;
        inBufHandle->write = 0;
        inBufHandle->full = false;
    }
}

void circular_buf_copy(cbuf_handle_t fromBufHandle,
                               cbuf_handle_t toBufHandle)
{
    if(bufferIsOwned(fromBufHandle) && bufferIsOwned(toBufHandle))
    {
        toBufHandle->read = fromBufHandle->read;
        toBufHandle->write = fromBufHandle->write;
        toBufHandle->full = fromBufHandle->full;
    }
}

cbuf_handle_t circular_buf_init(size_t inSize)
{
        assert(inSize);
        struct mem_list_node* iter = &memListHead;

        while(iter->next)
        {
                iter = iter->next;
        }
        iter->next = (struct mem_list_node*)malloc(sizeof(struct
mem_list_node));

        iter->next->data =
(circular_buf_t*)malloc(sizeof(circular_buf_t));

        assert(iter->next->data);

        iter->next->data->buffer =
(uint32_t*)malloc(sizeof(uint32_t)*inSize);
        assert(iter->next->data->buffer);
        iter->next->data->max = inSize;
        iter->next->data->write = 0;
        iter->next->data->read = 0;
        iter->next->data->full = false;

        iter->next->next = NULL;

        assert(circular_buf_empty(iter->next->data));
        return iter->next->data;
}

void circular_buf_free(cbuf_handle_t inBufHandle)
{
        circular_buf_t* buffer = (circular_buf_t*)inBufHandle;
        if(buffer)
        {
                struct mem_list_node* iter = &memListHead;

                while (iter->next)
                {
                        if(iter->next->data == buffer)
                        {
                                free(buffer->buffer);
                                free(buffer);
                                struct mem_list_node* node_to_delete =
iter->next;
                                iter->next = iter->next->next;
                                free(node_to_delete);
                                return;
                        }
                        iter = iter->next;
                }
        }
}

buff_err circular_buf_resize(cbuf_handle_t* inOutBufHandle, size_t inSize)
{
        if(inOutBufHandle &&
           bufferIsOwned(*inOutBufHandle) &&
           inSize > circular_buf_size(*inOutBufHandle))
        {
                // create new buffer
                cbuf_handle_t newBuf = circular_buf_init(inSize);

                // copy contents from old buffer
                uint32_t ch;
                while(circular_buf_pop(*inOutBufHandle, &ch) ==
buff_err_success)
                {
                        circular_buf_push(newBuf, ch);
                }

                // free old buffer
                circular_buf_free(*inOutBufHandle);

                // set output buffer
                *inOutBufHandle = newBuf;

                return buff_err_success;
        }
        return buff_err_invalid;
}

buff_err circular_buf_push(cbuf_handle_t inBufHandle, uint32_t inData)
{
        buff_err err = buff_err_success;

        if(bufferIsOwned(inBufHandle))
        {
                inBufHandle->buffer[inBufHandle->write] = inData;
                if(inBufHandle->full)
                {
                        // wrap
                        inBufHandle->read = (inBufHandle->read + 1) %
inBufHandle->max;
                        err = buff_err_full; // won't fail, just alert
about wrap
                }

                inBufHandle->write = (inBufHandle->write + 1) %
inBufHandle->max;
                inBufHandle->full = (inBufHandle->write == inBufHandle-
>read);
        }
        else
        {
                err = buff_err_invalid;
        }
        return err;
}
```

```
buff_err circular_buf_push_resize(cbuf_handle_t* inOutBufHandle, uint32_t inData)
{
        buff_err err = buff_err_invalid;
        if(inOutBufHandle)
        {
                cbuf_handle_t inBufHandle = *inOutBufHandle;

                if(bufferIsOwned(inBufHandle))
                {
                        if(circular_buf_full(inBufHandle))
                        {
                                if(circular_buf_resize(inOutBufHandle, inBufHandle->max * 2) == buff_err_success)
                                    inBufHandle = *inOutBufHandle;
                                else
                                        return err;
                        }

                        inBufHandle->buffer[inBufHandle->write] = inData;
                        if(inBufHandle->full)
                        {
                                inBufHandle->read = (inBufHandle->read + 1) % inBufHandle->max;
                        }

                        inBufHandle->write = (inBufHandle->write + 1) % inBufHandle->max;
                        inBufHandle->full = (inBufHandle->write == inBufHandle->read);
                        err = buff_err_success;

                }
        }

        return err;
}

buff_err circular_buf_pop(cbuf_handle_t inBufHandle, uint32_t * outData)
{
        buff_err err = buff_err_invalid;
        if(bufferIsOwned(inBufHandle))
        {
                if(circular_buf_empty(inBufHandle))
                        return buff_err_empty;

                *outData = inBufHandle->buffer[inBufHandle->read];
                inBufHandle->full = false;
                inBufHandle->read = (inBufHandle->read + 1) % inBufHandle->max;
                err = buff_err_success;
        }
        return err;
}
```

```
bool circular_buf_empty(cbuf_handle_t inBufHandle)
{
        assert(bufferIsOwned(inBufHandle));
    return (!inBufHandle->full && (inBufHandle->write == inBufHandle->read));
}

bool circular_buf_full(cbuf_handle_t inBufHandle)
{
        assert(bufferIsOwned(inBufHandle));
    return inBufHandle->full;
}

size_t circular_buf_capacity(cbuf_handle_t inBufHandle)
{
        assert(bufferIsOwned(inBufHandle));
        return inBufHandle->max;
}

size_t circular_buf_size(cbuf_handle_t inBufHandle)
{
        assert(bufferIsOwned(inBufHandle));

        size_t size = inBufHandle->max;

        if(!inBufHandle->full)
        {
                if(inBufHandle->write >= inBufHandle->read)
                {
                        size = (inBufHandle->write - inBufHandle->read);
                }
                else
                {
                        size = (inBufHandle->max + inBufHandle->write - inBufHandle->read);
                }
        }

        return size;
}
#include "dac_adc.h"
#include "board.h"
#include "fsl_dac.h"
#include "fsl_adc16.h"
#include "pin_mux.h"
#include "clock_config.h"

void dac_init()
{
        // TAKEN FROM SDK
    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. */
    DAC_SetBufferReadPointer(DAC0, 0U); /* Make sure the read pointer to
the start. */
}

void adc_init()
{
        adc16_config_t sAdc16ConfigStruct;
    ADC16_GetDefaultConfig(&sAdc16ConfigStruct);
    ADC16_Init(ADC0, &sAdc16ConfigStruct);

    /* Make sure the software trigger is used. */
    ADC16_EnableHardwareTrigger(ADC0, false);
}

void write_dac(uint32_t inVal)
{
        DAC_SetBufferValue(DAC0, 0U, inVal);
}

uint32_t read_adc()
{
        adc16_channel_config_t sAdc16ChannelConfigStruct;
    /* Prepare ADC channel setting */
    sAdc16ChannelConfigStruct.channelNumber = 0U;
    sAdc16ChannelConfigStruct.enableInterruptOnConversionCompleted =
false;

        ADC16_SetChannelConfig(ADC0, 0U, &sAdc16ChannelConfigStruct);

        while (0U == (kADC16_ChannelConversionDoneFlag &
                                  ADC16_GetChannelStatusFlags(ADC0, 0U)))
        {
        }
        return ADC16_GetChannelConversionValue(ADC0, 0U);
}
/*
 * @file dma.h
 * @brief Project 6
 *
 * @details This file contains code for using the DMA controller.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *  Leveraged code:
 *  Baremetal DMA code from the Dean example given in class
 *  https://github.com/alexander-g-
dean/ESF/blob/master/Code/Chapter_9/DMA_Examples/Source/DMA.c
 *
 */

#include "dma.h"

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"

#include "MKL25Z4.h"
#include "logger.h"

static dma_callback sCurrentCallback = NULL;
volatile bool g_DMADoneFlag = false;
void DMA0_DriverIRQHandler(void)
{
        g_DMADoneFlag = true;
        // mark end of transfer
        // Control_RGB_LEDs(0,0,0); TODO: Turn off leds here?
        if(sCurrentCallback)
                sCurrentCallback();
}

// taken from
void dma_init(void* cookie)
{
        LOG_STRING(LOG_MODULE_DMA, LOG_SEVERITY_STATUS, "Initialize
DMA.");
        SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;
        DMA0->DMA[0].DCR = DMA_DCR_SINC_MASK | DMA_DCR_SSIZE(0) |
                DMA_DCR_DINC_MASK |     DMA_DCR_DSIZE(0);

//      NVIC_SetPriority(DMA0_IRQn, 3);
//      NVIC_ClearPendingIRQ(DMA0_IRQn);
//      NVIC_EnableIRQ(DMA0_IRQn);

//      while(1)
//      {
//              vTaskSuspend(NULL);
//      }
}

void dma_transfer(uint32_t* srcAddr,
                  uint32_t* destAddr,
                  uint32_t transferCount,
                                  dma_callback inCallback)
```

```
{
        LOG_STRING(LOG_MODULE_MAIN, LOG_SEVERITY_STATUS, "DMA transfer.");
        // initialize source and destination pointers
        DMA0->DMA[0].SAR = DMA_SAR_SAR((uint32_t) srcAddr);
        DMA0->DMA[0].DAR = DMA_DAR_DAR((uint32_t) destAddr);
        // initialize byte count
        DMA0->DMA[0].DSR_BCR =
DMA_DSR_BCR_BCR(transferCount*sizeof(uint32_t));
        // clear done flag and status flags
        DMA0->DMA[0].DSR_BCR &= ~DMA_DSR_BCR_DONE_MASK;

        sCurrentCallback = inCallback;

        // start transfer
        DMA0->DMA[0].DCR |= DMA_DCR_START_MASK;

        // wait until it is done
        while (!(DMA0->DMA[0].DSR_BCR & DMA_DSR_BCR_DONE_MASK))
                ;

        sCurrentCallback();

//      while(!g_DMADoneFlag);
//      g_DMADoneFlag = false;

}
/*
 * @file handle_led.c
 * @brief Project 6
 *
 * Functions for handling the state of an LED.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *  LEVERAGED CODE from the in-class competition activity
 *  for LEDs.
 */

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

/**
 * @brief The RED LED pin
 */
#define RED_LED_POS (18U)   // on port B

/**
```

```
 * @brief The GREEN LED pin
 */
#define GREEN_LED_POS (19U)// on port B

/**
 * @brief The BLUE LED pin
 */
#define BLUE_LED_POS (1U)    // on port D

/**
 * @brief Set a bit at a specific position
 */
#define MASK(x) (1UL << (x))

/**
 * Flag to ensure that init has been called before LED is set.
 */
static uint8_t gLedReady = 0;

void leds_init()
{
        // led init

        // make 3 pins GPIO
        PORTB->PCR[RED_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTB->PCR[RED_LED_POS] |= PORT_PCR_MUX(1);
        PORTB->PCR[GREEN_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTB->PCR[GREEN_LED_POS] |= PORT_PCR_MUX(1);
        PORTD->PCR[BLUE_LED_POS] &= ~PORT_PCR_MUX_MASK;
        PORTD->PCR[BLUE_LED_POS] |= PORT_PCR_MUX(1);

        // Set ports to outputs using the data direction register
        PTB->PDDR |= MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
        PTD->PDDR |= MASK(BLUE_LED_POS);

        PTB->PSOR = MASK(RED_LED_POS) | MASK(GREEN_LED_POS);
        PTD->PSOR = MASK(BLUE_LED_POS);

        gLedReady = 1;
}

/**
 * set_led
 *
 * @brief Sets the LED state.
 * @details This function controls a physical LED and prints
 *          debug info over UART on debug builds.
 * @param inValue The on/off state of the LED to set.
 * @param inColor The color of the LED to set.
 */
void set_led(uint8_t inValue, enum COLOR inColor)
{
        if(gLedReady)
        {
                switch(inColor)
                {
                        case RED:
                        {
                                PTB->PSOR = MASK(GREEN_LED_POS);
                                PTD->PSOR = MASK(BLUE_LED_POS);

                                if(inValue)
                                {
                                        PTB->PCOR = MASK(RED_LED_POS);
                                }
                                else
                                {
                                        PTB->PSOR = MASK(RED_LED_POS);
                                }

                                break;
                        }
                        case GREEN:
                        {
                                PTD->PSOR = MASK(BLUE_LED_POS);
                                PTB->PSOR = MASK(RED_LED_POS);

                                if(inValue)
                                {
                                        PTB->PCOR = MASK(GREEN_LED_POS);
                                }
                                else
                                {
                                        PTB->PSOR = MASK(GREEN_LED_POS);
                                }
                                break;
                        }
                        case BLUE:
                        {
                                PTB->PSOR = MASK(GREEN_LED_POS);
                                PTB->PSOR = MASK(RED_LED_POS);

                                if(inValue)
                                {
                                        PTD->PCOR = MASK(BLUE_LED_POS);
                                }
                                else
                                {
                                        PTD->PSOR = MASK(BLUE_LED_POS);
                                }
                                break;
                        }
                        default:
                                 break;
                }
        }
}
/*
```

```
 * @file logger.h
 * @brief Project 5
 *
 * Tools for logging.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#include "logger.h"

#include <stdint.h>
#include <stdbool.h>
#include <stdint.h>

#include <stdio.h>
#include <stdarg.h>

#include "time.h"
#include "uart.h"

/**
 * Used as a size for static char arrays.
 */
#define ARRLEN 500

/**
 * @brief Strings associated with severities.
 */
static const char* sLogSeverityStrings[NUM_LOG_SEVERITIES] =
{
        "TEST",
        "DEBUG",
        "STATUS"
};

/**
 * @brief Strings associated with modules.
 */
static const char* sLogModuleStrings[NUM_LOG_MODULES] =
{
                "MAIN",
                "LED",
                "DMA",
                "SETUP_TEARDOWN",
                "CIRCULAR_BUFFER",
                "TASKS",
                "TIME",
                "POST",
        "UART",
                "SINE"
};

/**
 * @brief Prints the current time stamp in HH:MM:SS.n format
 */
static void PRINT_TIME_STAMP()
{
        timestamp_str timestamp;
        timestamp_now(&timestamp);

        uart_put_string(timestamp.hours);
        uart_put_string(timestamp.mins);
        uart_put_string(timestamp.secs);
        uart_put_string(timestamp.tens);
}

/**
 * @brief Used to standardize the prefix before log messages.
 */
static void PRINT_LOG_PREFIX(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity)
{
        static char format_buf[ARRLEN] = {0};
        for(int i = 0; i < ARRLEN; i++) format_buf[i] = '\0';

        uart_put_string("\n\r");
        sprintf(format_buf, "%s -> %s::[%s] : ",
sLogSeverityStrings[inSeverity] , sLogModuleStrings[inModule],
inFuncName);
        PRINT_TIME_STAMP();
        uart_put_string(format_buf);
}

/**
 * @brief Static variable maintains the logging state.
 */
static bool sLoggingEnabled = false;

/**
 * @brief Static severity maintains the severity for the module.
 */
static LogSeverity_t sLogSeverity = LOG_SEVERITY_STATUS;

void log_enable(LogSeverity_t inSeverity)
{
        sLoggingEnabled = true;
        sLogSeverity = inSeverity;
}

void log_disable()
{
        sLoggingEnabled = false;
```

```
}

bool log_enabled()
{
        return sLoggingEnabled;
}

void log_string(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity, const char* inString, ...)
{
        char format_buf[ARRLEN] = {0};
        for(int i = 0; i < ARRLEN; i++) format_buf[i] = '\0';

        if (sLoggingEnabled && inSeverity >= sLogSeverity) {

            va_list argp;
            va_start(argp, inString);
            vsprintf(format_buf, inString, argp);
            va_end(argp);
            PRINT_LOG_PREFIX(inModule, inFuncName, inSeverity);

            uart_put_string(format_buf);
            uart_put_string("\n\r");
        }
}



/* Standard includes. */
#include <assert.h>
#include <stdio.h>
#include <string.h>

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"

/* Freescale includes. */
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "board.h"

#include "pin_mux.h"

#include "setup_teardown.h"
#include "dac_adc.h"
#include "sine.h"
#include "logger.h"

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

```
***************************************************************************
****/

#define ever (;;);

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

***************************************************************************
****/
/*!
 * @brief Main function
 */
int main(void)
{
        initialize();

    for ever
}

/*
 * @file post.c
 * @brief Project 5
 *
 * A power on self test.
 *
 * @author Jack Campbell
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

#include "handle_led.h"
#include "post.h"
#include "logger.h"
#include "MKL25Z4.h"

bool power_on_self_test()
{
        // Check delays and LED timing
    set_led(1, RED);

    set_led(1, GREEN);

    set_led(1, BLUE);

    return true;
}
/*
 * @file setup_teardown.h
 * @brief Project 5
 *
 * @details Contains the setup and cleanup prototypes.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 */

//TODO: trim down includes
#include "board.h"
#include "peripherals.h"
#include "clock_config.h"
#include "pin_mux.h"


/* Freescale includes. */
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"

#include "tasks.h"
#include "post.h"
#include "circular_buffer.h"
#include "logger.h"
#include <stdlib.h>
#include "handle_led.h"
#include "time.h"
#include "sine.h"
#include "dac_adc.h"
#include "MKL25Z4.h"
#include "uart.h"
#include "dma.h"

#define UART_BAUD_RATE 115200

void initialize()
{
    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    SystemCoreClockUpdate();

        /* Init board hardware. */
         /* Enable all of the port clocks. */
        SIM->SCGC5 |= (SIM_SCGC5_PORTA_MASK
                           | SIM_SCGC5_PORTB_MASK
                           | SIM_SCGC5_PORTC_MASK
                           | SIM_SCGC5_PORTD_MASK
                           | SIM_SCGC5_PORTE_MASK );
```

```
#ifdef DEBUG
        log_enable(LOG_SEVERITY_DEBUG);
#else
        log_enable(LOG_SEVERITY_STATUS);
#endif
        uart_init(UART_BAUD_RATE); // todo define this
        time_init();
    leds_init();
    sine_init();
    dac_init();
    adc_init();
    dma_init(NULL);
    power_on_self_test();

    tasks_init();
}

/**
 * terminate
 *
 * @details Print "program end" in debug builds.
 *          Shows that the program successfully completed.
 *
 */
void terminate()
{
#ifdef DEBUG
        LOG_STRING(LOG_MODULE_SETUP_TEARDOWN, LOG_SEVERITY_DEBUG, "program
end");
#endif
}
#include "sine.h"
#include "logger.h"
#include "handle_led.h"
#include <math.h>
#define NUM_SINE_SAMPLES 50
#define INV_THREE_FACTORIAL (1/6)
#define INV_FIVE_FACTORIAL (1/120)
#define INV_SEVEN_FACTORIAL (1/5040)

#define M_PI 3.14159265358979323846

// derived from voltRead = (float)(g_Adc16ConversionValue * (VREF_BRD /
SE_12BIT));
// in dac adc example
const static float sDigitalConversionFactor = 1/(3.30/4096.0);

// lookup table
// TODO Generate this table..
static uint32_t sSineLookup[NUM_SINE_SAMPLES] = {0};

float sinef(float x)
{
        // LEVERAGED FROM WHITE BOOK
        float xSq = x * x;

        return x * (1 - xSq * (INV_THREE_FACTORIAL + xSq *
                   (INV_FIVE_FACTORIAL - INV_SEVEN_FACTORIAL * xSq)));
}

// populate lookup
void sine_init()
{
        // make a local sine function with taylor series
        // try to figure out how to generate the lookup table above
        uint64_t voltage_offset = 2;
        // generate a sine wave, between 1V and 3V: sin(x) = A*sin(x)+2
        LOG_STRING(LOG_MODULE_SINE, LOG_SEVERITY_STATUS, "Calculate and
create a lookup table to represent the values in a sine wave that runs
from 1V to 3V.");
        for(int x =0; x < NUM_SINE_SAMPLES; x++)
        {
                sSineLookup[x] = (sin((2.0 * M_PI * (x/(float)
(NUM_SINE_SAMPLES)))) + voltage_offset) * sDigitalConversionFactor;
                if(sSineLookup[x] > 4095)
                {
                        set_led(1, RED);
                        LOG_STRING_ARGS(LOG_MODULE_SINE,
LOG_SEVERITY_STATUS, "Value [%d] out of range. Setting to 0.",
sSineLookup[x]);
                        sSineLookup[x] = 0;
                }
                else
                {
                        LOG_STRING_ARGS(LOG_MODULE_SINE,
LOG_SEVERITY_DEBUG, "Writing [%d] to the temp table.", sSineLookup[x]);
                }

        }

        LOG_STRING(LOG_MODULE_SINE, LOG_SEVERITY_STATUS, "Sine
initialized.");
}

// get next sine sample
uint32_t get_next_sine_sample()
{
        static uint32_t sNextSample = 0;
        return sSineLookup[sNextSample++ % NUM_SINE_SAMPLES]; // TODO: mod
is expensive!
}
/*
 * @file tasks.h
 * @brief Project 6
 *
 * @details Contains the FreeRTOS tasks for this program.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *  LEVERAGED CODE: STANDARD DEVIATION https://www.sanfoundry.com/c-program-mean-variance-standard-deviation/
 *  Used to calculate standard deviation.
 */

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "semphr.h"

#include "post.h"
#include "tasks.h"
#include "circular_buffer.h"
#include "logger.h"
#include "handle_led.h"
#include "sine.h"
#include "dac_adc.h"
#include "dma.h"
#include "time.h"
#include <float.h>
#include <math.h>

SemaphoreHandle_t xMutex;

/**
 * Define this to run program 1 or program 2
 */
//#define PROGRAM_1

/**
 * The timer handle for writing to the DAC.
 */
static TimerHandle_t writeTimerHandle = NULL;

/**
 * The timer handler for reading from the ADC.
 */
static TimerHandle_t readTimerHandle = NULL;

/**
 * Timestamp string for the start time of the last DMA transfer.
 */
static timestamp_str sLastDMAStart;

/**
 * Timestamp string for the finish time of the last DMA transfer.
 */
static timestamp_str sLastDMAFinish;
```

```
/**
 * Voltage reference for reading from the ADC.
 */
#define VREF_BRD 3.300

/**
 * Max 12 bit float val for reading from the ADC.
 */
#define SE_12BIT 4096.0

/**
 * Global buffer struct for the ADC and DSP buffers.
 */
struct Buffers
{
        cbuf_handle_t adcBuffer;
        cbuf_handle_t dspBuffer;
} sBuffers;

/**
 * Size of ADC and DSP buffers.
 */
#define BUFFER_CAPACITY 64

/**
 * Number of runs for program 2.
 */
#define NUM_RUNS 5

/**
 * Prototypes
 */

/**
 * Task to write to the DAC.
 */
void write_dac0_task(TimerHandle_t xTimer);

/**
 * Task to read from the ADC.
 */
void read_adc0_task(TimerHandle_t xTimer);


/**
 * One shot timer task to turn off the blue LED.
 */
void turn_off_dma_led(void *pvParameters)
{
        LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Turn off blue
LED triggered by DMA transfer, release mutex.");
        set_led(0, BLUE);
        xSemaphoreGive(xMutex);
```

```
}

/**
 * A task that analyzes the DSP buffer after each DMA transfer.
 */
void dsp_callback(void *pvParameters)
{
        static uint8_t sRunNumber = 0;
        static float sMaxVoltage = 0;
        static float sMinVoltage = FLT_MAX;
        static float sAverageVoltage = 0;
        static float sVoltagesCumulative = 0;
        static float sNumVoltagesRecorded = 0;

        static float sVarianceSum = 0;
        static float sVariance = 0;
        static float sStDeviationVoltage = 0;

        sRunNumber++;

        float voltages[BUFFER_CAPACITY] = {0.0f};

        /**
         * Calculate the following floating point values
         * from the ADC register values:  maximum,  minimum, average,
         * and standard deviation of voltage levels.
         */

        uint32_t data;
        uint32_t i = 0;
        while(circular_buf_pop(sBuffers.dspBuffer, &data) ==
buff_err_success)
        {
                sNumVoltagesRecorded++;

        // convert to float
                float voltage = (float)(data * (VREF_BRD / SE_12BIT));

                // calc max
                if(voltage > sMaxVoltage)
                {
                        sMaxVoltage = voltage;
                }

                // calc min
                if(voltage < sMinVoltage)
                {
                        sMinVoltage = voltage;
                }
                // calc avg
                sVoltagesCumulative += voltage;
                sAverageVoltage =
(sVoltagesCumulative)/sNumVoltagesRecorded;

```

```
                voltages[i] = voltage;
                i++;
        }

    /*  Compute  variance  and standard deviation  */
    for (int iter = 0; iter < BUFFER_CAPACITY; iter++)
    {
        float voltage_entry = (voltages[iter] - sAverageVoltage);
                sVarianceSum = sVarianceSum + (voltage_entry *
voltage_entry); // cheap square
    }
    sVariance = sVarianceSum / (float)sNumVoltagesRecorded;
    sStDeviationVoltage = sqrt(sVariance); //TODO remove if we can't
afford this

        /**
         * Report those values along with an incremented run number
         * starting at 1 and the start time and end time for the last DMA
transfer.
         */
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Run #%d",
                        sRunNumber);

        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Last DMA
start: [ %s%s%s%s ]:",
                        sLastDMAStart.hours,
                        sLastDMAStart.mins,
                        sLastDMAStart.secs,
                        sLastDMAStart.tens);

        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Last DMA
finish: [ %s%s%s%s ]:",
                        sLastDMAFinish.hours,
                        sLastDMAFinish.mins,
                        sLastDMAFinish.secs,
                        sLastDMAFinish.tens);

        // report max
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Maximum
voltage: %f", sMaxVoltage);

        // report min
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Minimum
voltage: %f", sMinVoltage);

        // report avg
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Average
voltage: %f", sAverageVoltage);

        // report st deviation
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Standard
deviation voltage: %f", sStDeviationVoltage);

        /**
```

```
         * Once run number 5 is completed and reported, terminate the
         * DAC and ADC tasks, and terminate this task to end the program.
         */
        if(sRunNumber >= NUM_RUNS)
        {
                LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS,
"Exiting app.", sRunNumber);

                xTimerStop(writeTimerHandle, 0);
                xTimerStop(readTimerHandle, 0);
        }

        vTaskDelete(NULL);
}

/**
 * Callback for when the DMA transfer has completed.
 */
void DMA_Callback()
{
        // copies buffer state, not data
        circular_buf_copy(sBuffers.adcBuffer, sBuffers.dspBuffer);
        circular_buf_reset(sBuffers.adcBuffer);
        LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "DMA Transfer
completed.");

        timestamp_now(&sLastDMAFinish);

        if(xTaskCreate(dsp_callback, "DSP Callback",
configMINIMAL_STACK_SIZE + 512, NULL, (configMAX_PRIORITIES - 1), NULL) !=
pdPASS)
        {
                LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "DSP
task creation failed.");
            xSemaphoreTake(xMutex, pdMS_TO_TICKS(1000));
                set_led(1, RED);
                xSemaphoreGive(xMutex);
        }
}

/**
 * Init all the tasks for FreeRTOS.
 */
void tasks_init()
{

    LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Create .1 second
timer to write sine values to the DAC.");
    /* Create the software timer. */
    writeTimerHandle = xTimerCreate("DAC Write Timer",          /* Text
name. */
                                     pdMS_TO_TICKS(100), /* Timer period.
*/
                                 pdTRUE,             /* Enable auto
reload. */
                                 0,                  /* ID is not used. */

write_dac0_task);   /* The callback function. */
    xTimerStart(writeTimerHandle, 0);

    // program 1 only cares about writing to the DAC0_OUT
#ifndef PROGRAM_1
    xMutex = xSemaphoreCreateMutex();

    LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Create .1 second
timer to read sine values from the ADC.");
    /* Create the software timer. */
    readTimerHandle = xTimerCreate("ADC READ Timer",          /* Text
name. */
                                     pdMS_TO_TICKS(100), /* Timer period.
*/
                                 pdTRUE,             /* Enable auto
reload. */
                                 0,                  /* ID is not used. */

read_adc0_task);   /* The callback function. */
    xTimerStart(readTimerHandle, 0);

    LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Create DSP and ADC
buffers.");
    sBuffers.adcBuffer = circular_buf_init(BUFFER_CAPACITY);
    sBuffers.dspBuffer = circular_buf_init(BUFFER_CAPACITY);
    // need to init post-scheduler start
    //xTaskCreate(dma_init, "DMA Init", configMINIMAL_STACK_SIZE + 512,
NULL, (configMAX_PRIORITIES - 1), NULL);
#endif
    /* Start scheduling. */
    vTaskStartScheduler();
}

void write_dac0_task(TimerHandle_t xTimer)
{
        static int ledVal = 0;
        uint32_t sineVal = get_next_sine_sample();
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_DEBUG, "Writing %d
to the DAC.", sineVal);
    /**
     * apply the values from the lookup table to DAC0_OUT (pin J10-11)
every .1 second,
     * repeating from the beginning of the table once the last value is
applied.
     * Toggle a Blue LED on and off for each visit to the timer callback.
     */
        write_dac(sineVal);

#ifdef PROGRAM_1
        LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_DEBUG, "Toggle blue led
for program 1 DAC write.");
```

```
        set_led(ledVal, BLUE);
#else
        // todo : add synchronization primitives here
        LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_DEBUG, "Toggle green led
for program 2 DAC write.");

    xSemaphoreTake(xMutex, pdMS_TO_TICKS(1000));
        set_led(ledVal, GREEN);
        xSemaphoreGive(xMutex);
#endif
        ledVal = !ledVal;
}

void read_adc0_task(TimerHandle_t xTimer)
{


        /*
         Create a FreeRTOS Task to periodically (every .1 seconds)
         read the DAC0 value via ADC0 and store it in a circular buffer.

         The ADC buffer will be 64 samples long and should contain the
         raw ADC register values from each read.
    */

        uint32_t sample = read_adc();
        LOG_STRING_ARGS(LOG_MODULE_TASKS, LOG_SEVERITY_DEBUG, "Reading %d
from the ADC.", sample);
        if(circular_buf_push(sBuffers.adcBuffer, sample) == buff_err_full)
        {
                 // When the buffer is full, initiate a DMA transfer from
the ADC buffer to a second
                 // buffer (called the DSP buffer).
                /*
                 When the DMA Transfer is about to start, toggle the LED
to Blue for .5 seconds.
                 During this period, the LED cannot be used by other
tasks.


                 Capture a time stamp at the start and completion of the
DMA transfer.
                 */
                LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "Turn on
blue LED triggered by DMA transfer, acquire mutex.");
            xSemaphoreTake(xMutex, pdMS_TO_TICKS(1000));
                set_led(1, BLUE);
            TimerHandle_t startTransferTimer = NULL;
            startTransferTimer = xTimerCreate("Start DMA timer",
/* Text name. */
                                                  pdMS_TO_TICKS(500), /*
Timer period. */
                                              pdFALSE,            /*
Enable auto reload. */
                                              0,                 /* ID is
not used. */
turn_off_dma_led);   /* The callback function. */
            xTimerStart(startTransferTimer, 0);

            timestamp_now(&sLastDMAStart);
            LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_STATUS, "DMA
Transfer started.");
            dma_transfer(sBuffers.adcBuffer->buffer,
                             sBuffers.dspBuffer->buffer,
                                         BUFFER_CAPACITY,
                                         DMA_Callback);

            LOG_STRING(LOG_MODULE_TASKS, LOG_SEVERITY_DEBUG, "DMA Transfer
claimed to complete.");
        }

        /*
         Clear (or overwrite) the ADC buffer with incoming DAC values and
continue sampling
         until the next series of samples are collected.

         You will need to consider the size and data width requirements
for the ADC
         buffer and the DSP buffer.
         */
}
/*
 * @file time.c
 * @brief Project 5
 *
 * @details Contains interface for telling and initializing time.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *
 *  LEVERAGED CODE: This time-passed and time_now pseudocode
 *  is taken from the White book, p140.
 */

#include "MKL25Z4.h"
#include <stdbool.h>
/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "time.h"
#include <stdio.h>

/* The software timer period. */
#define SW_TIMER_PERIOD_MS ( 10 / portTICK_PERIOD_MS )
```

```
// tenths of a second
static uint64_t gSystemTime = 0;

void time_ticker()
{
        gSystemTime++;
}


void time_init()
{
    TimerHandle_t SwTimerHandle = NULL;

    /* Create the software timer. */
    SwTimerHandle = xTimerCreate("TimeTicker",          /* Text name. */
                                     pdMS_TO_TICKS(100), /* Timer period.
*/
                                 pdTRUE,             /* Enable auto
reload. */
                                 0,                  /* ID is not used. */

time_ticker);   /* The callback function. */
    /* Start timer. */
    xTimerStart(SwTimerHandle, 0);
}


// taken from the white book
uint64_t time_passed(uint64_t since)
{
        // used to rollover the time
        static const uint64_t gTimeMax = ~0;
        uint64_t now = gSystemTime;

        if(now >= since)
        {
                return now - since;
        }

        // rollover has occurred
        return (now + (gTimeMax-since));
}

uint64_t time_now()
{
        return gSystemTime;
}

void timestamp_now(timestamp_str* outTimestamp)
{
        uint64_t tenths_seconds = time_now();
        float now = tenths_seconds / 10;
        uint64_t hours = (uint64_t)(now/3600)%60;
        uint64_t minutes = (uint64_t)(now/60)%60;
        uint64_t seconds = (uint64_t)(now)%60;
        sprintf((outTimestamp->hours), "%02d:",  hours);
        sprintf((outTimestamp->mins),  "%02d:",  minutes);
        sprintf((outTimestamp->secs),  "%02d",  seconds);
        sprintf((outTimestamp->tens),  ".%1d ",  tenths_seconds%10);
}

/*
 * @file uart.c
 * @brief Project 5
 *
 * @details Contains interface for UART communications.
 *
 * @tools  PC Compiler: GNU gcc 8.3.0
 *         PC Linker: GNU ld 2.32
 *         PC Debugger: GNU gdb 8.2.91.20190405-git
 *         ARM Compiler: GNU gcc version 8.2.1 20181213
 *         ARM Linker: GNU ld 2.31.51.20181213
 *         ARM Debugger: GNU gdb 8.2.50.20181213-git
 *  LEVERAGED CODE:
 *  https://github.com/alexander-g-
dean/ESF/tree/master/Code/Chapter_8/Serial-Demo
 */

#include "uart.h"
#include "handle_led.h"
#include "circular_buffer.h"
#include <stddef.h>

/**
 * How many characters we want to initially request for the circular
buffer
 */
#define UART_CAPACITY 100

/**
 * Transmit circular buffer
 */
static cbuf_handle_t sTxBuffer = NULL;

/**
 * Receive circular buffer
 */
static cbuf_handle_t sRxBuffer = NULL;

/**
 * Enable interrupt macro
 */
#define ENABLE_IRQ NVIC_EnableIRQ(UART0_IRQn);

/**
 * Disable interrupt macro
 */
```

```
#define DISABLE_IRQ NVIC_DisableIRQ(UART0_IRQn);

void uart_init(int64_t baud_rate)
{
         set_led(1, BLUE);

        uint16_t sbr;
        uint8_t temp;

        // Enable clock gating for UART0 and Port A
        SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

        // Make sure transmitter and receiver are disabled before init
        UART0->C2 &= ~UART0_C2_TE_MASK & ~UART0_C2_RE_MASK;

        // Set UART clock to 48 MHz clock
        SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);
        SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;

        // Set pins to UART0 Rx and Tx
        PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Rx
        PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Tx

        // Set baud rate and oversampling ratio
        sbr = (uint16_t)((SYS_CLOCK/2)/(baud_rate *
UART_OVERSAMPLE_RATE));
        UART0->BDH &= ~UART0_BDH_SBR_MASK;
        UART0->BDH |= UART0_BDH_SBR(sbr>>8);
        UART0->BDL = UART0_BDL_SBR(sbr);
        UART0->C4 |= UART0_C4_OSR(UART_OVERSAMPLE_RATE-1);

        // Disable interrupts for RX active edge and LIN break detect,
select one stop bit
        UART0->BDH |= UART0_BDH_RXEDGIE(0) | UART0_BDH_SBNS(0) |
UART0_BDH_LBKDIE(0);

        // Don't enable loopback mode, use 8 data bit mode, don't use
parity
        UART0->C1 = UART0_C1_LOOPS(0) | UART0_C1_M(0) | UART0_C1_PE(0);
        // Don't invert transmit data, don't enable interrupts for errors
        UART0->C3 = UART0_C3_TXINV(0) | UART0_C3_ORIE(0)| UART0_C3_NEIE(0)
                        | UART0_C3_FEIE(0) | UART0_C3_PEIE(0);

        // Clear error flags
        UART0->S1 = UART0_S1_OR(1) | UART0_S1_NF(1) | UART0_S1_FE(1) |
UART0_S1_PF(1);

        // Try it a different way
        UART0->S1 |= UART0_S1_OR_MASK | UART0_S1_NF_MASK |

UART0_S1_FE_MASK | UART0_S1_PF_MASK;

        // Send LSB first, do not invert received data
        UART0->S2 = UART0_S2_MSBF(0) | UART0_S2_RXINV(0);

#if USE_UART_INTERRUPTS
        // Enable interrupts. Listing 8.11 on p. 234
        sTxBuffer = circular_buf_init(UART_CAPACITY);
        sRxBuffer = circular_buf_init(UART_CAPACITY);

        NVIC_SetPriority(UART0_IRQn, 2); // 0, 1, 2, or 3
        NVIC_ClearPendingIRQ(UART0_IRQn);
        ENABLE_IRQ

        // Enable receive interrupts but not transmit interrupts yet
        // also turn on error interrupts
        UART0->C2 |= UART_C2_RIE(1) |
                             UART_C2_TIE(1);

        UART0->C3 |= UART_C3_ORIE(1) |
                                 UART_C3_NEIE(1) |
                                 UART_C3_PEIE(1) |
                                 UART_C3_FEIE(1);
#endif

        // Enable UART receiver and transmitter
        UART0->C2 |= UART0_C2_RE(1) | UART0_C2_TE(1);

        // Clear the UART RDRF flag
        temp = UART0->D;
        UART0->S1 &= ~UART0_S1_RDRF_MASK;

}

bool uart_putchar_space_available ()
{
        set_led(1, GREEN);
    return (UART0->S1 & UART0_S1_TDRE_MASK);
}

bool uart_getchar_present ()
{
        //set_led(1, BLUE);
    return (UART0->S1 & UART0_S1_RDRF_MASK);
}

void uart_putchar (char ch)
{
         set_led(1, GREEN);
    /* Wait until space is available in the FIFO */
    while(!(UART0->S1 & UART0_S1_TDRE_MASK));

    /* Send the character */
    UART0->D = (uint8_t)ch;
}

bool uart_getchar(uint8_t* outChar)
```

```
{
         set_led(1, BLUE);

        /* Return the 8-bit data from the receiver */
        if((UART0->S1 & UART0_S1_RDRF_MASK))
        {
                *outChar = UART0->D;
                return true;
        }
        return false;
}

// taken from DEAN
void uart_put_string(const char* str) {
        // enqueue string
        while (*str != '\0') { // Send characters up to null terminator
                uart_putchar(*str++);
        }
}

bool uart_echo(uint8_t* outChar)
{
#if USE_UART_INTERRUPTS
        if(circular_buf_pop(sRxBuffer, outChar) == buff_err_success)
        {
                circular_buf_push_resize(&sTxBuffer, *outChar);
                UART0->C2 |= UART0_C2_TIE_MASK;
                return true;
        }
#else
        uint8_t ch;

        if(uart_getchar(&ch))
        {
                *outChar = ch;
                uart_putchar(ch);
                return true;
        }

#endif
        return false;
}

// UART0 IRQ Handler. Listing 8.12 on p. 235
void UART0_IRQHandler(void) {
        DISABLE_IRQ

        uint8_t ch;

        // error handling
        if (UART0->S1 & (UART_S1_OR_MASK |UART_S1_NF_MASK |
                             UART_S1_FE_MASK | UART_S1_PF_MASK))
        {
                        // clear the error flags
                        UART0->S1 |= UART0_S1_OR_MASK | UART0_S1_NF_MASK |
                                                 UART0_S1_FE_MASK |
UART0_S1_PF_MASK;
                        // read the data register to clear RDRF
                        ch = UART0->D;

                         set_led(1, RED);
        }

        // received a character
        if (UART0->S1 & UART0_S1_RDRF_MASK)
        {

                ch = UART0->D;
                if (!circular_buf_full(sRxBuffer))
                {
                        circular_buf_push_resize(&sRxBuffer, ch);
                }
                else
                {
                        // error - queue full.
                        // discard character
                }

                 set_led(1, BLUE);
        }

        // transmitter interrupt enabled and tx buffer empty
        if ( (UART0->C2 & UART0_C2_TIE_MASK) &&
                        (UART0->S1 & UART0_S1_TDRE_MASK) )
        {
                // can send another character
                if (!circular_buf_empty(sTxBuffer))
                {
                        uint32_t outCh = -1;
                        if(circular_buf_pop(sTxBuffer, &outCh) ==
buff_err_success)
                        {
                          UART0->D = outCh;
                        }
                }
                else
                {
                        // queue is empty so disable transmitter interrupt
                        UART0->C2 &= ~UART0_C2_TIE_MASK;
                }
                 set_led(1, GREEN);
        }

        ENABLE_IRQ
}
```