PES Project 5 Readme

Jack Campbell

Description

This repo contains custom sources and makefiles for Project 5 as well as adapted and generated code from MCUXpresso and the KL25Z SDK.

This project contains three configurations: Test, Debug, and Normal.

I use the ECHO_MODE and USE_UART_INTERRUPTS to decided between polling/interrupt UART functions and APPLICATION vs ECHO mode for the main function.

Application mode prints the character report every REPORT_TIME tenths of a second.

Test runs a suite of unit tests covering the circular buffer functions.

I also did the extra credit for this assignment with the resizing circular buffers.

I included 2 different versions of the circular buffer push, one that errors when the buffer is full and one that resizes.

Observations

This project went smoother than the I2C one, though I did use more leveraged code for the circular buffer and UART code. Connecting between the PC and the KL25Z seemed to be more reliable than the KL25Z and the TMP102, which may or may not have been verified to work.

Switching to not use the SDK was a good exercise, since it forced me to interact with all the registers directly.

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 5
 *
 * @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 {
        uint8_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, uint8_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, uint8_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, uint8_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);

#endif

/*
 * @file delay.h
 * @brief Project 5
 *
 * @details This file contains prototypes for calculating a spin-wait
 *          on various platforms, used for delaying LED state changes.
 *
 * @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_DELAY_H
#define PES_PROJECT_4_DELAY_H

#include <stdint.h>

/**
 * delay
 *
 * @brief Blocks execution for the specified time.
 * @param inDelayMs Then time in milliseconds to block.
 */
void delay(uint64_t inDelayMs);

#endif //PES_PROJECT_4_DELAY_H
/*
 * @file handle_led.h
 * @brief Project 5
 *
 * @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 5
 *
 * @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 5
 *
 * 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_UNIT_TEST,
        LOG_MODULE_SETUP_TEARDOWN,
        LOG_MODULE_CIRCULAR_BUFFER,
        LOG_MODULE_TIME,
        LOG_MODULE_POST,
        LOG_MODULE_UART,
        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();


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_data – display in hexadecimal an address and contents of a
memory location,
 * @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 inBytes a pointer to a sequence of bytes to log
 * @param inSize Number of bytes to log
 */
void log_data(LogModule_t inModule, const char* inFuncName, LogSeverity_t
inSeverity, const uint8_t* inBytes, size_t inSize);

/**
 * @brief A macro used to wrap a log data. Used to write the function name
automatically.
 */
#define LOG_DATA(category, severity, data, length) \
{ \
    log_data(category, __FUNCTION__, severity, data, length); \
}


/**
 * @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); \
}

/**
 * @brief Logs an integer.
 * @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 inNum Integer to log.
 */
void log_integer(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity, uint64_t inNum);

/**
 * @brief A wrapper for log integer that include the function name by
default.
 */
#define LOG_INTEGER(category, severity, num) \
{ \
        log_integer(category, __func__, severity, num); \
}

#endif
/*
 * @file post.h
 * @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
 */

#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 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
 */

#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 time.h
 * @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
 */

#ifndef __timeh__
#define __timeh__

#include <stdint.h>
```

```
/**
 * @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();

#endif
/*
 * @file uart.h
 * @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
 */

#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     (1) // 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 5
 *
 * @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;
}

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 =
(uint8_t*)malloc(sizeof(uint8_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
                uint8_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, uint8_t inData)
{
        buff_err err = buff_err_invalid;

        if(bufferIsOwned(inBufHandle))
        {
                if(circular_buf_full(inBufHandle))
                        return buff_err_full;
```

```
                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_push_resize(cbuf_handle_t* inOutBufHandle, uint8_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, uint8_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;
}
/*
 * @file delay.c
 * @brief Project 5
 *
 * @details This file contains prototypes for calculating a spin-wait
 *          on the FB, used for delaying LED state changes.
 *
 * @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 "delay.h"
#include "time.h"

/**
 * delay
 *
 * @brief Blocks execution for the specified time.
 * @param inDelayMs Then time in milliseconds to block.
 */
void delay(uint64_t inDelayMs)
{
        uint64_t now = time_now();
        while(time_passed(now) < inDelayMs)
        {}
}
/*
 * @file handle_led.c
 * @brief Project 5
 *
 * 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))

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);
}

/**
 * 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)
{
```

```
	LOG_STRING_ARGS(LOG_MODULE_LED,
			LOG_SEVERITY_TEST,
			"LED %s %s", COLOR_STRINGS[inColor],
			inValue ? "ON" : "OFF");

	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 2048

/**
 * @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",
                "UNIT_TEST",
                "SETUP_TEARDOWN",
```

```
                "CIRCULAR_BUFFER",
                "TIME",
                "POST",
        "UART"
};

/**
 * @brief Prints the current time stamp in HH:MM:SS.n format
 */
static void PRINT_TIME_STAMP()
{
        static char format_buf[ARRLEN] = {0};
        for(int i = 0; i < ARRLEN; i++) format_buf[i] = '\0';

        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(format_buf, "%02d:",  hours);
        uart_put_string(format_buf);
        sprintf(format_buf, "%02d:",  minutes);
        uart_put_string(format_buf);
        sprintf(format_buf, "%02d",  seconds);
        uart_put_string(format_buf);
        sprintf(format_buf, ".%1d ",  tenths_seconds%10);
        uart_put_string(format_buf);
}

/**
 * @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_data(LogModule_t inModule, const char* inFuncName, LogSeverity_t
inSeverity, const uint8_t* inBytes, size_t inSize)
{
        static char format_buf[ARRLEN] = {0};
        for(int i = 0; i < ARRLEN; i++) format_buf[i] = '\0';

        if (sLoggingEnabled && inSeverity >= sLogSeverity)
        {
                PRINT_LOG_PREFIX(inModule, inFuncName, inSeverity);
                sprintf(format_buf, "\n\rBytes at address
%p:\n\r========================\n\r", inBytes);
                uart_put_string(format_buf);
                for(int i = 0; i < inSize; i++)
                {
                        sprintf(format_buf, "%2x ", inBytes[i]);
                        uart_put_string(format_buf);

                        if((i+1)%4 == 0)
                        {
                                uart_put_string("\n\r");
                        }
                }
                uart_put_string("\n\r========================\n\r");
        }
}


void log_string(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity, const char* inString, ...)
{
        static 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");
        }
}


void log_integer(LogModule_t inModule, const char* inFuncName,
LogSeverity_t inSeverity, uint64_t inNum)
{
        static char format_buf[ARRLEN] = {0};
        for(int i = 0; i < ARRLEN; i++) format_buf[i] = '\0';

        if (sLoggingEnabled && inSeverity >= sLogSeverity)
        {
                PRINT_LOG_PREFIX(inModule, inFuncName, inSeverity);
                sprintf(format_buf, "%lld\n\r", inNum);
                uart_put_string(format_buf);
                uart_put_string("\n\r");
        }
}
/*
 * @file main.c
 * @brief Project 5
 *
 * @details Main controller for the echo and application modes.
 *          Echo mode just transmits back any data received from the PC.
 *          Application mode makes reports for characters send over UART.
 *
 * @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 <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

#include "handle_led.h"
#include "setup_teardown.h"
```

```
#include "logger.h"
#include "post.h"
#include "delay.h"
#include "uart.h"
#include "time.h"
#include "circular_buffer.h"

/**
 * @brief Define this to enter echo mode, undef to use application mode.
 */
#define ECHO_MODE

struct CharCounts_t
{
        uint32_t mSeen[0xFF+1];
};

static struct CharCounts_t sCharCounts = {{0}};


/**
 * @brief How often to print the character report in application mode
 */
#define REPORT_TIMER 17

#ifndef ECHO_MODE
/**
 * Used as a size for static char arrays.
 */
#define ARRLEN 2048

static char format_buf[ARRLEN] = {0};
#endif

int main(void) {

        initialize();
    uint8_t outChar;

    for(int i =0; i < 0xFF+1;i++)
    {
        sCharCounts.mSeen[i] =0x0;
    }
#ifdef ECHO_MODE
        while(1)
        {
            uart_echo(&outChar);
        }
#else // application mode

        while(1)
        {
                if(uart_echo(&outChar) == true)
                {
```

```
                        sCharCounts.mSeen[outChar]++;
                }

                if((time_now() % REPORT_TIMER) == 0)
                {
                        // print the report!
                        LOG_STRING(LOG_MODULE_MAIN,
                                        LOG_SEVERITY_STATUS,
                                        "Application mode character
report: ");

                        for(uint32_t c = 0; c < 0xFF+1; c++)
                        {
                                if(sCharCounts.mSeen[c] > 0)
                                {
                                        sprintf(format_buf, "%c - %d; ",
c, sCharCounts.mSeen[c]);
                                        uart_put_string(format_buf);

                                }
                        }
                        uart_put_string("\r");
                        LOG_STRING(LOG_MODULE_MAIN,
                                        LOG_SEVERITY_STATUS,
                                        "Application mode character report
complete.");
                }

        }

#endif

    return 0 ;
}
/*
 * @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"
#include "uart.h"
```

```
#include "delay.h"

bool power_on_self_test()
{
        LOG_STRING( LOG_MODULE_POST, LOG_SEVERITY_DEBUG, "Starting power-
on self test." );

        // Check delays and LED timing
    set_led(1, RED);
    delay(10);
    set_led(1, GREEN);
    delay(10);
    set_led(1, BLUE);
    delay(10);

    // Log will succeed if uart is enabled correctly
    LOG_STRING( LOG_MODULE_POST, LOG_SEVERITY_DEBUG, "POST SUCCESS!");

    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
 */

#include "board.h"
#include "peripherals.h"
#include "clock_config.h"
#include "pin_mux.h"



#include "post.h"
#include "logger.h"
#include <stdlib.h>
#include "uart.h"
#include "handle_led.h"
#include "time.h"

#include "MKL25Z4.h"

#define UART_BAUD_RATE 115200

void initialize()
{
```

```
    BOARD_InitBootClocks();

        /* 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 );


        uart_init(UART_BAUD_RATE);
        time_init();
    leds_init();

#ifdef DEBUG
        log_enable(LOG_SEVERITY_TEST);
        LOG_STRING(LOG_MODULE_SETUP_TEARDOWN, LOG_SEVERITY_DEBUG, "program
start");
#else
        log_enable(LOG_SEVERITY_STATUS);
#endif

        if(!power_on_self_test())
                exit(-1);
}

/**
 * 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
}
/*
 * @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"

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


void time_init()
{
        SysTick->LOAD = 48000000UL/100;

        NVIC_SetPriority(SysTick_IRQn, 3); // 0, 1, 2, or 3
        NVIC_ClearPendingIRQ(SysTick_IRQn);
        NVIC_EnableIRQ(SysTick_IRQn);

        SysTick->VAL = 0;
        SysTick->CTRL = SysTick_CTRL_TICKINT_Msk |
SysTick_CTRL_ENABLE_Msk;
}

void SysTick_Handler()
{
        gSystemTime++;
}

// 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;
}

/*
 * @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>

#define UART_CAPACITY 256

static cbuf_handle_t sTxBuffer = NULL;
static cbuf_handle_t sRxBuffer = NULL;

#define ENABLE_IRQ NVIC_EnableIRQ(UART0_IRQn);

#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)/(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))
                {
                        uint8_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
}


#include "uCUnit.h"
#include <stdint.h>
#include "logger.h"
#include "setup_teardown.h"
#include "circular_buffer.h"
#include "handle_led.h"

#define HANDLE_ERROR(num_errors) if(num_errors > 0) set_led(1, RED); else
set_led(1, GREEN);

#define TEST_BUF_SIZE 16

int main()
{
        initialize();
        set_led(1, BLUE);

        {
            UCUNIT_TestcaseBegin("Create a buffer");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
        UCUNIT_CheckIsNotNull(buf->buffer);
        UCUNIT_CheckIsEqual(buf->write, 0);
        UCUNIT_CheckIsEqual(buf->read, 0);
        UCUNIT_CheckIsEqual(buf->max, TEST_BUF_SIZE);
        UCUNIT_CheckIsEqual(buf->full, false);
            UCUNIT_TestcaseEnd();
            circular_buf_free(buf);
        }

        {
                UCUNIT_TestcaseBegin("Destroy buffer");
                cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
                circular_buf_free(buf);
```

```
            uint8_t outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_invalid);
                UCUNIT_TestcaseEnd();
        }

        {
                UCUNIT_TestcaseBegin("Destroy the same buffer twice");
                cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
                circular_buf_free(buf);
                circular_buf_free(buf);
            uint8_t outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_invalid);
                UCUNIT_TestcaseEnd();
        }

        {
                UCUNIT_TestcaseBegin("Test data access");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            uint8_t outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xAA);
            circular_buf_free(buf);
                UCUNIT_TestcaseEnd();
        }

        {
                UCUNIT_TestcaseBegin("Test data multiple buffers");
            cbuf_handle_t buf1 = circular_buf_init(TEST_BUF_SIZE);
            UCUNIT_CheckIsEqual(circular_buf_push(buf1, 0xAA),
buff_err_success);
            uint8_t outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf1, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xAA);

            cbuf_handle_t buf2 = circular_buf_init(TEST_BUF_SIZE);
            UCUNIT_CheckIsEqual(circular_buf_push(buf2, 0xAA),
buff_err_success);
            outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf2, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xAA);


            circular_buf_free(buf1);
            circular_buf_free(buf2);
                UCUNIT_TestcaseEnd();
        }
```

```
        // test that your buffer can wrap around the edge boundary and add
to the front
        {
                UCUNIT_TestcaseBegin("Verify wrap write");

            cbuf_handle_t buf = circular_buf_init(4);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            UCUNIT_CheckIsEqual(buf->write, 1);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xBB),
buff_err_success);
            UCUNIT_CheckIsEqual(buf->write, 2);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xCC),
buff_err_success);
            UCUNIT_CheckIsEqual(buf->write, 3);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xDD),
buff_err_success);
            UCUNIT_CheckIsEqual(buf->write, 0);

            circular_buf_free(buf);

                UCUNIT_TestcaseEnd();
        }

        // test that your buffer tail point can wrap around the edge
boundary when a remove occurs at the boundary
        {
                UCUNIT_TestcaseBegin("Verify wrap read");

            cbuf_handle_t buf = circular_buf_init(4);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xBB),
buff_err_success);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xCC),
buff_err_success);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xDD),
buff_err_success);

            uint8_t outbyte = 0x0;
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xAA);
            UCUNIT_CheckIsEqual(buf->read, 1);
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xBB);
            UCUNIT_CheckIsEqual(buf->read, 2);
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xCC);
            UCUNIT_CheckIsEqual(buf->read, 3);
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
```

```
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xDD);
            UCUNIT_CheckIsEqual(buf->read, 0);

            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xEE),
buff_err_success);
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xEE);
            UCUNIT_CheckIsEqual(buf->read, 1);

            circular_buf_free(buf);

                UCUNIT_TestcaseEnd();
        }

        // test that your buffer fails when too many items are added
        {
                UCUNIT_TestcaseBegin("Overfill");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
            for(int i = 0; i < TEST_BUF_SIZE; i++)
            {
                UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            }
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_full);
            circular_buf_free(buf);
                UCUNIT_TestcaseEnd();
        }

        // test that destructive overload resizes buffer when it is full
        {
                UCUNIT_TestcaseBegin("Overfill resize");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
            for(int i = 0; i < TEST_BUF_SIZE; i++)
            {
                UCUNIT_CheckIsEqual(circular_buf_push_resize(&buf, 0xAA),
buff_err_success);
            }
            UCUNIT_CheckIsEqual(circular_buf_push_resize(&buf, 0xAA),
buff_err_success);
            UCUNIT_CheckIsEqual(buf->max, TEST_BUF_SIZE * 2);
            circular_buf_free(buf);
                UCUNIT_TestcaseEnd();
        }

        // test that your buffer fails to remove an item when empty
        {
                UCUNIT_TestcaseBegin("Over empty");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
            UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            uint8_t outbyte = 0x0;
```

```
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_success);
            UCUNIT_CheckIsEqual(outbyte, 0xAA);
            UCUNIT_CheckIsEqual(circular_buf_pop(buf, &outbyte),
buff_err_empty);
            circular_buf_free(buf);
                UCUNIT_TestcaseEnd();
        }


        {
                UCUNIT_TestcaseBegin("Resize buffer");
            cbuf_handle_t buf = circular_buf_init(TEST_BUF_SIZE);
            for(int i = 0; i < TEST_BUF_SIZE; i++)
            {
                UCUNIT_CheckIsEqual(circular_buf_push(buf, 0xAA),
buff_err_success);
            }

            circular_buf_resize(&buf, TEST_BUF_SIZE*2);
            UCUNIT_CheckIsEqual(buf->max, TEST_BUF_SIZE * 2);

            uint8_t ch;
            for(int i = 0; i < TEST_BUF_SIZE; i++)
            {
                UCUNIT_CheckIsEqual(circular_buf_pop(buf, &ch),
buff_err_success);
                UCUNIT_CheckIsEqual(ch, 0xAA);
            }

            circular_buf_free(buf);
                UCUNIT_TestcaseEnd();
        }


        HANDLE_ERROR(ucunit_testcases_failed)

        terminate();

        return 0;
}
```