```c
/*
 * Copyright 2016-2018 NXP Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    POE_Project_2_Project_2.c
 * @brief   Application entry point.
 */
#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"
/* APPLICATION HEADERS */
#include "char_counter.h"
#include "output_generator.h"
#include "ring_buffer.h"
#include "uart_adapter.h"
#include "uart_handler.h"

/* APPLCIATION DEFINES */
#define OUT_RING_SIZE    512
#define FATAL_ERROR_DEBUG

#ifdef FATAL_ERROR_DEBUG
    #define FATAL_ERROR __asm__("BKPT");
#else
    #define FATAL_ERROR    NVIC_SystemReset();
#endif

/* Declare Buffers */
char_counter input_array;
ring_buffer_struct output_ring;
unsigned char buffer[OUT_RING_SIZE];

/* Declare Flags */
volatile uint8_t schedule_flags = 0;

int main(void) {

      /* Init board hardware. */
```

```
70      BOARD_InitBootPins();            //Configures UART pins
71      BOARD_InitBootClocks();            //Configure system clocks
72
73      /* Initialize Buffers */
74      if(ring_init(&output_ring, buffer, OUT_RING_SIZE) !=  RING_SUCCESS)
75      {
76          FATAL_ERROR;
77      }
78      if(char_reset(input_array) != CHAR_SUCCESS)
79      {
80          FATAL_ERROR;
81      }
82
83      /* Initialize UART0 */
84      uart_config init_uart0 = UART_INIT_DEFAULT;
85      if(uart_init(&init_uart0) != UART_SUCCESS)
86      {
87          FATAL_ERROR;
88      }
89
90      output_error output_ret = OUTPUT_SUCCESS;
91
92      while(1)
93      {
94          GPIO_TogglePinsOutput(GPIOB, (1 << 8));
95          if(schedule_flags == 1)
96          {
97              // If the interrupt has received a character and flags main to generate new output
98              NVIC_DisableIRQ(UART0_IRQn);
99              schedule_flags = 0;                // Output generation happening, clear flag (eliminates race)
100             NVIC_EnableIRQ(UART0_IRQn);
101
102             output_ret = output_complete(input_array, &output_ring);
103         }
104         if((output_ret == OUTPUT_FULL) && !(UART0->C2 & UART_C2_TIE_MASK))
105         {
106             // If the last output generation resulted in a full buffer, and we're done transmitting,
107             // generate a clean output set with the now empty buffer
108             output_ret = output_complete(input_array, &output_ring);
109         }
110     }
111
112     return 0;
113 }
114
115 void UART0_IRQHandler(void)
116 {
117     uart_handler((UART_Type*)UART0, input_array, &output_ring);
118 }
```

```c
/*
 * ring_buffer.h
 *
 *  Created on: Nov 08, 2018
 *      Author: Dominic Doty
 *
 *    Ring buffer functions. "unsafe" versions don't check for null pointers on input
 *    It is assumed that these will only be used in functions where the input has already been confirmed non null
 */

#ifndef RING_BUFFER_H_
#define RING_BUFFER_H_

#include "stdint.h"
#include "stddef.h"
#include "stdbool.h"

typedef struct
{
    unsigned char* buffer;
    uint16_t head;
    uint16_t tail;
    uint16_t mask;
}ring_buffer_struct;

typedef enum {RING_SUCCESS, RING_NULL_PTR, RING_ILLEGAL_SIZE, RING_FULL, RING_EMPTY}ring_error;

ring_error ring_init(ring_buffer_struct* rbs, unsigned char* buffer, uint16_t size);

ring_error ring_add(ring_buffer_struct* rbs, unsigned char addition);

ring_error ring_add_unsafe(ring_buffer_struct* rbs, unsigned char addition);

ring_error ring_remove(ring_buffer_struct* rbs, unsigned char* removal);

ring_error ring_remove_unsafe(ring_buffer_struct* rbs, unsigned char* removal);

ring_error ring_element_count(ring_buffer_struct* rbs, uint16_t* count);

ring_error ring_element_count_unsafe(ring_buffer_struct* rbs, uint16_t* count);

#endif /* RING_BUFFER_H_ */
```

```c
/*
 * ring_buffer.c
 *
 *  Created on: Nov 08, 2018
 *      Author: Dominic Doty
 */

#include "ring_buffer.h"

static bool is_pow_two(uint16_t number)
{
    // NOTE: This is "Brian Kernighan Method"
    // http://graphics.stanford.edu/~seander/bithacks.html#CountBitsSetTable

    uint16_t ret;                       // ret accumulates the total bits set in v
    for (ret = 0; number; ret++)
    {
        number &= number - 1;        // clear the least significant bit set
    }
    return (ret == 1);
}


ring_error ring_init(ring_buffer_struct* rbs, unsigned char* buffer, uint16_t size)
{
    ring_error ret = RING_SUCCESS;

    if(rbs == NULL)
    {
        // Check for NULL ring ptr
        ret = RING_NULL_PTR;
    }
    else if(buffer == NULL)
    {
        // Check for NULL buffer ptr
        ret = RING_NULL_PTR;
    }
    else if(!is_pow_two(size))
    {
        ret = RING_ILLEGAL_SIZE;
    }
    else
    {
        rbs->buffer = buffer;
        rbs->head = 0;
        rbs->tail = 0;
        rbs->mask = size - 1;
    }

    return ret;
}


ring_error ring_add(ring_buffer_struct* rbs, unsigned char addition)
{
    ring_error ret = RING_SUCCESS;

    if(rbs == NULL)
    {
        // Check for NULL ring ptr
        ret = RING_NULL_PTR;
    }
    else
    {
        ret = ring_add_unsafe(rbs, addition);
    }

    return ret;
}
```

```
70
71 ring_error ring_add_unsafe(ring_buffer_struct* rbs, unsigned char addition)
72 {
73     ring_error ret = RING_SUCCESS;
74
75     rbs->buffer[rbs->head] = addition;            // Add thing to buffer
76     rbs->head = (rbs->head + 1) & rbs->mask;    // Increment the head / wrap
77     if(rbs->head == rbs->tail)                    // Check for full buffer
78     {
79         rbs->head = (rbs->head - 1) & rbs->mask;
80         ret = RING_FULL;
81     }
82
83     return ret;
84 }
85
86
87 ring_error ring_remove(ring_buffer_struct* rbs, unsigned char* removal)
88 {
89     ring_error ret = RING_SUCCESS;
90
91     if(rbs == NULL)
92     {
93         // Check for NULL ring ptr
94         ret = RING_NULL_PTR;
95     }
96     else if(removal == NULL)
97     {
98         // Check for NULL return ptr
99         ret = RING_NULL_PTR;
100     }
101     else
102     {
103         ret = ring_remove_unsafe(rbs, removal);
104     }
105
106     return ret;
107 }
108
109 ring_error ring_remove_unsafe(ring_buffer_struct* rbs, unsigned char* removal)
110 {
111     ring_error ret = RING_SUCCESS;
112
113     if(rbs->head == rbs->tail)                    // Check if the buffer is empty
114     {
115         ret = RING_EMPTY;
116     }
117     else
118     {
119         *removal = rbs->buffer[rbs->tail];            // Remove the thing
120         rbs->tail = (rbs->tail + 1) & rbs->mask;     // Increment the tail and wrap
121     }
122
123     return ret;
124 }
125
126 ring_error ring_element_count(ring_buffer_struct* rbs, uint16_t* count)
127 {
128     ring_error ret = RING_SUCCESS;
129
130     if(rbs == NULL)
131     {
132         // Check for NULL ring ptr
133         ret = RING_NULL_PTR;
134     }
135     else if(count == NULL)
136     {
137         // Check for NULL return ptr
138         ret = RING_NULL_PTR;
139     }
```

```
140     else
141     {
142         ring_element_count_unsafe(rbs, count);
143     }
144 
145     return ret;
146 }
147 
148 ring_error ring_element_count_unsafe(ring_buffer_struct* rbs, uint16_t* count)
149 {
150     ring_error ret = RING_SUCCESS;
151 
152     // Calculate the number of elements
153     // head - tail, if negative, add size of ring (mask+1)
154     // need to check that this doesn't have over/underflow issues
155     uint32_t temp = (rbs->head - rbs->tail) < 0 ?
156                     rbs->head - rbs->tail + rbs->mask + 1 :
157                     rbs->head - rbs->tail;
158     *count = (uint16_t)temp;
159 
160     if(rbs->head == (rbs->tail - 1))
161     {
162         ret = RING_FULL;
163     }
164 
165     return ret;
166 }
```

```c
/*
 * uart_adapter.h
 *
 *  Created on: Nov 15, 2018
 *      Author: Dominic Doty
 */

#ifndef UART_ADAPTER_H_
#define UART_ADAPTER_H_

// UART INCLUDES
#include "stddef.h"
#include "stdbool.h"
#include "stdint.h"
#include "fsl_clock.h"
#include "fsl_uart.h"

// UART DEFINES
#define UART_PARITY_DISABLED    0x0U
#define UART_PARITY_EVEN        0x2U
#define UART_PARITY_ODD            0x3U
#define XTAL0_F                    8000000

#define UART_INIT_DEFAULT                                \
            {.port = (UART_Type*)UART0,                   \
            .clock_freq = CLOCK_GetPllFllSelClkFreq(),    \
            .baudrate = 115200,                           \
            .parity_mode = UART_PARITY_DISABLED,         \
            .enable_tx = true,                            \
            .enable_rx = true}

// UART VARIABLES
typedef struct
{
    UART_Type* port;
    uint32_t clock_freq;
    uint32_t baudrate;
    uint8_t parity_mode;
    bool enable_tx;
    bool enable_rx;
}uart_config;

typedef enum {UART_SUCCESS, UART_NULL_PTR, UART_ILLEGAL_PORT, UART_ILLEGAL_PARITY, UART_ILLEGAL_STOPBIT, UART_ILLEGAL_FREQUENCY, UART_BAUDRATE_TOO_HIGH_FOR_CLOCK, UART_F

// UART FUNCTIONS
uart_error uart_init(uart_config* init);

uart_error uart_transmit(UART_Type* uart_reg, unsigned char data);

bool uart_transmit_full(UART_Type* uart_reg);

uart_error uart_transmit_blocking(UART_Type* uart_reg, unsigned char data);

uart_error uart_receive(UART_Type* uart_reg, unsigned char* data);

bool uart_receive_full(UART_Type* uart_reg);

uart_error uart_receive_blocking(UART_Type* uart_reg, unsigned char* data);

#endif /* UART_ADAPTER_H_ */
```

```c
/*
 * uart_adapter.c
 *
 *  Created on: Nov 15, 2018
 *      Author: Dominic Doty
 */

#include "uart_adapter.h"

// UART FUNCTIONS
uart_error uart_init(uart_config* init)
{
    // NOTE THAT THIS IS HEAVILY INFLUENCED BY NXP UART DRIVER
    // IT HAS BEEN REWRITTEN, BUT IT WILL BEAR A MARKED SIMILARITY
    // CREDIT TO NXP, ORIGINAL @ fsl_uart.c
    uart_error ret = UART_SUCCESS;

    if(init == NULL)                                        //check for non-void init
    {
        ret = UART_NULL_PTR;
    }
    else if(!((init->port == (UART_Type*)UART0) ||            //check for valid port
            (init->port == (UART_Type*)UART1)     ||
            (init->port == (UART_Type*)UART2)))
    {
        ret = UART_ILLEGAL_PORT;
    }
    else if(!(    (init->parity_mode == UART_PARITY_DISABLED) ||    //check parity_mode valid
                (init->parity_mode == UART_PARITY_EVEN)        ||
                (init->parity_mode == UART_PARITY_ODD )))
    {
        ret = UART_ILLEGAL_PARITY;
    }
    else if(init->clock_freq == 0)                            //check clock frequency valid
    {
        ret = UART_ILLEGAL_FREQUENCY;
    }
    else if(init->clock_freq / (init->baudrate * 16) == 0)    //check baudrate valid
    {
        ret = UART_BAUDRATE_TOO_HIGH_FOR_CLOCK;
    }
    else
    {
        // Enable the UART Clock
        if(init->port == (UART_Type*)UART0)
        {
            CLOCK_SetLpsci0Clock(1);
            CLOCK_EnableClock(kCLOCK_Uart0);
        }
        else if(init->port == (UART_Type*)UART1)
        {
            CLOCK_EnableClock(kCLOCK_Uart1);
        }
        else if(init->port == (UART_Type*)UART2)
        {
            CLOCK_EnableClock(kCLOCK_Uart2);
        }

        // Disable TX/RX
        init->port->C2 &= ~(UART_C2_TE_MASK | UART_C2_RE_MASK);

        // Calculate the clock divisor to achieve the baud rate requested
        uint16_t baud_clock_div = init->clock_freq / (init->baudrate * 16);    //NOTE THIS IS THE SAME AS THE NXP UART DRIVER

        // Write baud divisor
        init->port->BDH = (init->port->BDH & ~UART_BDH_SBR_MASK) | (uint8_t)(baud_clock_div >> 8);
        init->port->BDL = (uint8_t)baud_clock_div;

        // Set Bit Count and Parity Mode
        uint8_t reg = init->port->C1 & ~(UART_C1_PE_MASK | UART_C1_PT_MASK | UART_C1_M_MASK);    //Pulls config register and clears all flags we want to mess with
        if(init->parity_mode != UART_PARITY_DISABLED)
        {
            reg |= UART_C1_M_MASK;                            //Sets bits per char to 9 (parity enabled adds 1 bit)
            reg |= (init->parity_mode << UART_C1_PT_SHIFT);    //Sets parity enabled and parity mode
        }
        init->port->C1 = reg;

        // Set Enable RX/TX
        init->port->C2 |= UART_C2_TE_MASK;
        init->port->C2 |= UART_C2_RE_MASK;

        // Configure Interrupts (Lifted from NXP fsl_uart.c - not original)
        uint32_t mask = (kUART_RxDataRegFullInterruptEnable) & kUART_AllInterruptsEnable;
        init->port->BDH |= mask;
        init->port->C2 |= (mask >> 8);
        init->port->C3 |= (mask >> 16);
        NVIC_EnableIRQ(UART0_IRQn);
    }
    return ret;
}

uart_error uart_transmit(UART_Type* uart_reg, unsigned char data)
{
    uart_error ret = UART_SUCCESS;
    uart_reg->D = data;
    return ret;
}

bool uart_transmit_full(UART_Type* uart_reg)
{
```

```
    return !(uart_reg->S1 & UART_S1_TDRE_MASK);
}

uart_error uart_transmit_blocking(UART_Type* uart_reg, unsigned char data)
{
    while(uart_transmit_full(uart_reg));
    return uart_transmit(uart_reg, data);
}

uart_error uart_receive(UART_Type* uart_reg, unsigned char* data)
{
    uart_error ret = UART_SUCCESS;
    *data = uart_reg->D;
    return ret;
}

bool uart_receive_full(UART_Type* uart_reg)
{
    return uart_reg->S1 & UART_S1_RDRF_MASK;
}

uart_error uart_receive_blocking(UART_Type* uart_reg, unsigned char* data)
{
    while(!uart_receive_full(uart_reg));
    return uart_receive(uart_reg, data);
}
```

```c
/*
 * uart_handler.h
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 *
 *      Functions that poll/handle UART interfaces
 */

#ifndef INCLUDE_UART_HANDLER_H_
#define INCLUDE_UART_HANDLER_H_

#include "ring_buffer.h"
#include "uart_adapter.h"
#include "char_counter.h"

/* Global Vars */
extern volatile uint8_t schedule_flags;

// Takes pointers to the UART tx/rx buffer, pointer to input buffer (char classification buffer), pointer to output buffer (ring buffer of output chars)
void uart_handler(UART_Type* uart_reg, char_counter in_buffer, ring_buffer_struct* out_buffer);

#endif /* INCLUDE_UART_HANDLER_H_ */
```

```c
/*
 * uart_handler.c
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 */

#include "uart_handler.h"

void uart_handler(UART_Type* uart_reg, char_counter in_buffer, ring_buffer_struct* out_buffer)
{
    NVIC_DisableIRQ(UART0_IRQn);
    //check for NULL pointers
    if(uart_reg == NULL)
    {
        return;
    }
    else if(out_buffer == NULL)
    {
        return;
    }
    else
    {
        unsigned char data = 0;

        // Receive a Character
        if(uart_receive_full(uart_reg))
        {
            uart_receive(uart_reg, &data);
            char_add_unsafe(in_buffer, data);
            schedule_flags = 1;                    // Flag tells main we need to generate new output
        }

        // Transmit a Character
        //&& (UART0->C2 & UART_C2_TIE_MASK)
        if(!uart_transmit_full(uart_reg))
        {
            ring_error ring_ret = ring_remove_unsafe(out_buffer, &data);
            if(ring_ret == RING_SUCCESS)
            {
                uart_transmit(uart_reg, data);
            }
            else if(ring_ret == RING_EMPTY)
            {
                // No more transmits needed, disable interrupt
                UART0->C2 &= ~UART_C2_TIE_MASK;
            }
        }
    }

    NVIC_EnableIRQ(UART0_IRQn);
}
```

```c
/*
 * char_counter.h
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 *
 *      Functions to add characters to an array that keeps track of all received chars
 *      Array must be of size 256 (0-255)
 */

#ifndef INCLUDE_CHAR_COUNTER_H_
#define INCLUDE_CHAR_COUNTER_H_

#include "stdint.h"
#include "stddef.h"

typedef volatile uint16_t char_counter[256];

typedef enum {CHAR_SUCCESS, CHAR_NULL_PTR}char_error;

char_error char_reset(char_counter array);

char_error char_add(char_counter array, unsigned char data);

char_error char_add_unsafe(char_counter array, unsigned char data);

#endif /* INCLUDE_CHAR_COUNTER_H_ */
```

```c
/*
 * char_counter.c
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 */


#include "char_counter.h"

char_error char_reset(char_counter array)
{
    char_error ret = CHAR_SUCCESS;
    if(array == NULL)
    {
        ret = CHAR_NULL_PTR;
    }
    else
    {
        for(uint16_t i = 0; i < 256; i++)
        {
            array[i] = 0;
        }
    }
    return ret;
}

char_error char_add(char_counter array, unsigned char data)
{
    char_error ret = CHAR_SUCCESS;
    if(array == NULL)
    {
        ret = CHAR_NULL_PTR;
    }
    else
    {
        char_add_unsafe(array, data);
    }
    return ret;
}

char_error char_add_unsafe(char_counter array, unsigned char data)
{
    char_error ret = CHAR_SUCCESS;
    array[(uint8_t)data]++;
    return ret;
}
```

```c
/*
 * output_generator.h
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 */

#ifndef INCLUDE_OUTPUT_GENERATOR_H_
#define INCLUDE_OUTPUT_GENERATOR_H_

#include "ring_buffer.h"
#include "char_counter.h"
#include "MKL25Z4.h"

typedef enum {OUTPUT_SUCCESS, OUTPUT_NULL_PTR, OUTPUT_FULL, OUTPUT_UNKNOWN_ERROR}output_error;

output_error output_complete(char_counter char_count_array, ring_buffer_struct* output_ring);

output_error output_single_char(char_counter char_count_array, ring_buffer_struct* output_ring);

output_error output_single_char_unsafe(char_counter char_count_array, ring_buffer_struct* output_ring);

#endif /* INCLUDE_OUTPUT_GENERATOR_H_ */
```

```c
/*
 * output_generator.c
 *
 *  Created on: Nov 14, 2018
 *      Author: Dominic Doty
 */


#include "output_generator.h"

// STATIC FUNCTIONS
static void stringify(uint16_t input, unsigned char* output_array);
static ring_error stringify_output(uint16_t input, ring_buffer_struct* output_ring);

// GLOBAL FUNCTIONS
output_error output_complete(char_counter char_count_array, ring_buffer_struct* output_ring)
{
    output_error ret = OUTPUT_SUCCESS;
    ring_error ring_ret = RING_SUCCESS;

    if(char_count_array == NULL)
    {
        ret = OUTPUT_NULL_PTR;
    }
    else if(output_ring == NULL)
    {
        ret = OUTPUT_NULL_PTR;
    }
    else
    {
        ring_ret = ring_add_unsafe(output_ring, '\f');                                  // Print form feed

        for(uint16_t index = 0; index < 256; index++)
        {
            if(char_count_array[index] != 0)                                           // Skip any 0 characters
            {
                ring_ret = ring_add_unsafe(output_ring, (unsigned char)index);          // Print the character
                ring_ret = ring_add_unsafe(output_ring, ':');
                ring_ret = stringify_output(char_count_array[index], output_ring);
                ring_ret = ring_add_unsafe(output_ring, '\r');                          // Newline/CR
                ring_ret = ring_add_unsafe(output_ring, '\n');
                UART0->C2 |= UART_C2_TIE_MASK;                                          // Output ready for transmit, enable TX int
            }
        }
    }

    if(ring_ret == RING_FULL)
    {
        ret = OUTPUT_FULL;
    }

    return ret;
}

output_error output_single_char(char_counter char_count_array, ring_buffer_struct* output_ring)
{
    output_error ret = OUTPUT_SUCCESS;

    if(output_ring == NULL)
    {
        ret = OUTPUT_NULL_PTR;
    }
    else if(char_count_array == NULL)
    {
        ret = OUTPUT_NULL_PTR;
    }
    else
    {
        ret = output_single_char_unsafe(char_count_array, output_ring);
    }
    return ret;
}

output_error output_single_char_unsafe(char_counter char_count_array, ring_buffer_struct* output_ring)
{
    output_error ret = OUTPUT_SUCCESS;
    ring_error ring_ret = RING_SUCCESS;
    static uint8_t index = 0;
    static uint8_t last_index = 0;

    last_index = index;
    index++;

```

```
84      while((char_count_array[index] == 0) && (index != last_index))        // Skip any 0 characters
85      {
86          // Increment index as long as it points at a char with 0 count
87          // If we increment all the way back to the start, break out of the loop
88          index++;
89      }
90
91      // If we've wrapped around back to zero index, print a new page character
92      if(last_index >= index)
93      {
94          ring_ret = ring_add_unsafe(output_ring, '\f');
95      }
96      // If we've found a nonzero character, print it
97      if(char_count_array[index] != 0)
98      {
99          ring_ret = ring_add_unsafe(output_ring, (unsigned char)index);        // Print the character
100         ring_ret = ring_add_unsafe(output_ring, ':');                         // Print semicolon divider
101         ring_ret = stringify_output(char_count_array[index], output_ring);
102         ring_ret = ring_add_unsafe(output_ring, '\r');                        // Newline/CR
103         ring_ret = ring_add_unsafe(output_ring, '\n');
104     }
105
106     if(ring_ret == RING_FULL){ret = OUTPUT_FULL;}
107
108     return ret;
109 }
110
111 static void stringify(uint16_t input, unsigned char* output_array)
112 {
113     uint16_t temp = 10000;    //10k used here since we are using a uint16, the highest it gets is 65,535
114     // this is fixed with the current implementation of the function, hence not using a #define for this
115
116     // divide by powers of ten starting at 10k till you find a non-zero result (leave off leading 0's)
117     while(input/temp == 0)
118     {
119         temp /= 10;
120     }
121
122     uint8_t index = 0;
123     uint8_t result = 0;
124     while(temp != 0)
125     {
126         result = input/temp;                                    // Find what the number in the ten power place is (eg 100s, 10s place)
127         input = input - (result*temp);                          // Remove that from the input number
128         output_array[index] = (unsigned char)(result + 48);    // Add the number in the ten power place to output string
129         temp /= 10;                                             // Move down to the next power place
130         index++;                                                // Move over to the next spot in output array
131     }
132     output_array[index] = '\0';                                // End of string null
133 }
134
135 static ring_error stringify_output(uint16_t input, ring_buffer_struct* output_ring)
136 {
137     unsigned char num_string_buffer[6];
138     uint8_t num_string_index = 0;
139     ring_error ring_ret = RING_SUCCESS;
140
141     stringify(input, num_string_buffer);
142     while(num_string_buffer[num_string_index] != '\0')
143     {
144         ring_ret = ring_add_unsafe(output_ring, num_string_buffer[num_string_index]);    // Print the count of character
145         num_string_index++;
146     }
147
148     return ring_ret;
149 }
```

```c
/*
 * Copyright (c) 2016, Freescale Semiconductor, Inc.
 * Copyright 2016-2017 NXP
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of the copyright holder nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef _PIN_MUX_H_
#define _PIN_MUX_H_

#include "fsl_gpio.h"

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

/*! @brief Direction type  */
typedef enum _pin_mux_direction
{
  kPIN_MUX_DirectionInput = 0U,         /* Input direction */
  kPIN_MUX_DirectionOutput = 1U,        /* Output direction */
  kPIN_MUX_DirectionInputOrOutput = 2U  /* Input or output direction */
} pin_mux_direction_t;

/*!
 * @addtogroup pin_mux
 * @{
 */

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

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

/*!
 * @brief Configures pin routing and optionally pin electrical features.
 *
 */
void BOARD_InitPins(void);

/*!
 * @brief Calls initialization functions.
 *
```

```
 */
void BOARD_InitBootPins(void);

#if defined(__cplusplus)
}
#endif

/*!
 * @}
 */
#endif /* _PIN_MUX_H_ */

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

```c
/*
 * Copyright (c) 2016, Freescale Semiconductor, Inc.
 * Copyright 2016-2017 NXP
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of the copyright holder nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/*
 * TEXT BELOW IS USED AS SETTING FOR THE PINS TOOL *****************************
PinsProfile:
- !!product 'Pins v2.0'
- !!processor 'MKL25Z128xxx4'
- !!package 'MKL25Z128VLK4'
- !!mcu_data 'ksdk2_0'
- !!processor_version '1.1.0'
 * BE CAREFUL MODIFYING THIS COMMENT - IT IS YAML SETTINGS FOR THE PINS TOOL ***
 */

#include "fsl_common.h"
#include "fsl_port.h"
#include "pin_mux.h"

#define PIN1_IDX                         1u   /*!< Pin number for pin 1 in a port */
#define PIN2_IDX                         2u   /*!< Pin number for pin 2 in a port */
#define SOPT5_UART0RXSRC_UART_RX      0x00u   /*!< UART0 receive data source select: UART0_RX pin */
#define SOPT5_UART0TXSRC_UART_TX      0x00u   /*!< UART0 transmit data source select: UART0_TX pin */

/*
 * TEXT BELOW IS USED AS SETTING FOR THE PINS TOOL *****************************
BOARD_InitPins:
- options: {coreID: singlecore, enableClock: 'true'}
- pin_list:
  - {pin_num: '28', peripheral: UART0, signal: TX, pin_signal: TSI0_CH3/PTA2/UART0_TX/TPM2_CH1}
  - {pin_num: '27', peripheral: UART0, signal: RX, pin_signal: TSI0_CH2/PTA1/UART0_RX/TPM2_CH0}
 * BE CAREFUL MODIFYING THIS COMMENT - IT IS YAML SETTINGS FOR THE PINS TOOL ***
*/

/*FUNCTION**********************************************************************
 *
 * Function Name : BOARD_InitBootPins
 * Description   : Calls initialization functions.
 *
 *END**********************************************************************/
void BOARD_InitBootPins(void) {
  BOARD_InitPins();
}


```

```
/*FUNCTION**********************************************************************
 *
 * Function Name : BOARD_InitPins
 * Description   : Configures pin routing and optionally pin electrical features.
 *
 *END**********************************************************************/
void BOARD_InitPins(void) {
  CLOCK_EnableClock(kCLOCK_PortA);                         /* Port A Clock Gate Control: Clock enabled */

  PORT_SetPinMux(PORTA, PIN1_IDX, kPORT_MuxAlt2);          /* PORTA1 (pin 27) is configured as UART0_RX */
  PORT_SetPinMux(PORTA, PIN2_IDX, kPORT_MuxAlt2);          /* PORTA2 (pin 28) is configured as UART0_TX */
  SIM->SOPT5 = ((SIM->SOPT5 &
    (~(SIM_SOPT5_UART0TXSRC_MASK | SIM_SOPT5_UART0RXSRC_MASK))) /* Mask bits to zero which are setting */
      | SIM_SOPT5_UART0TXSRC(SOPT5_UART0TXSRC_UART_TX)     /* UART0 transmit data source select: UART0_TX pin */
      | SIM_SOPT5_UART0RXSRC(SOPT5_UART0RXSRC_UART_RX)     /* UART0 receive data source select: UART0_RX pin */
    );

  /* Set Up GPIO Pin for Toggling */
  gpio_pin_config_t fig = {kGPIO_DigitalOutput, 0};
  GPIO_PinInit(GPIOB, 8, &fig);
}

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

```
Tests are contained in this folder

Running make with no args will build and run all tests

Running make with args thing_test will build that test
    ring_test
    char_test

Running make with args thing_test_rn will build and run that test
    ring_test_run
    ring_test_run

Make clean deletes all build stuff
```

```
# Based on:
# http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

CC = gcc
LIBRARIES = -llibcunit
BUILDDIR = build
INCLUDE = -I../include -Icunit_headers
CFLAGS = $(INCLUDE) $(LIBRARIES) -Wall

unittest:all

#========================================================================

SOURCE_RING = ../source/ring_buffer.c ring_test.c
BINNAME_RING = ring_test
BINRUN_RING = $(addsuffix _run, $(BINNAME_RING))
_RING_DEPS = ring_buffer.h
RING_DEPS = $(patsubst %, $(INCLUDE)/%, $(_RING_DEPS))
_RING_OBJ = ring_buffer.o ring_buffer_test.o
RING_OBJ = $(patsubst %, $(BUILDDIR)/%, $(_RING_OBJ))

$(BUILDDIR)/%.o: $(SOURCE_RING) $(RING_DEPS)
    mkdir -p $(BUILDDIR)
    $(CC) -c -o $@ $< $(CFLAGS)

$(BINNAME_RING): $(SOURCE_RING)
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

$(BINRUN_RING): $(BINNAME_RING)
    ./$(BINNAME_RING)

#========================================================================

SOURCE_CHAR = ../source/char_counter.c char_test.c
BINNAME_CHAR = char_test
BINRUN_CHAR = $(addsuffix _run, $(BINNAME_CHAR))
_CHAR_DEPS = char_counter.h
CHAR_DEPS = $(patsubst %, $(INCLUDE)/%, $(_CHAR_DEPS))
_CHAR_OBJ = char_counter.o char_test.o
CHAR_OBJ = $(patsubst %, $(BUILDDIR)/%, $(_CHAR_OBJ))

$(BUILDDIR)/%.o: $(SOURCE_CHAR) $(CHAR_DEPS)
    mkdir -p $(BUILDDIR)
    $(CC) -c -o $@ $< $(CFLAGS)

$(BINNAME_CHAR): $(SOURCE_CHAR)
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

$(BINRUN_CHAR): $(BINNAME_CHAR)
    ./$(BINNAME_CHAR)

#========================================================================

SOURCE_OUTPUT = ../source/output_generator.c output_test.c ../source/ring_buffer.c ../source/char_counter.c
BINNAME_OUTPUT = output_test
BINRUN_OUTPUT = $(addsuffix _run, $(BINNAME_OUTPUT))
_OUTPUT_DEPS = output_generator.h ring_buffer.h char_counter.h
OUTPUT_DEPS = $(patsubst %, $(INCLUDE)/%, $(_OUTPUT_DEPS))
_OUTPUT_OBJ = output_generator.o output_test.o ring_buffer.o char_counter.o
OUTPUT_OBJ = $(patsubst %, $(BUILDDIR)/%, $(_OUTPUT_OBJ))

$(BUILDDIR)/%.o: $(SOURCE_OUTPUT) $(OUTPUT_DEPS)
    mkdir -p $(BUILDDIR)
    $(CC) -c -o $@ $< $(CFLAGS)

$(BINNAME_OUTPUT): $(SOURCE_OUTPUT)
    $(CC) -o $@ $^ $(CFLAGS) $(LIBS)

$(BINRUN_OUTPUT): $(BINNAME_OUTPUT)
```

```
70      ./$(BINNAME_OUTPUT)
71
72  #=============================================================================
73
74  all: $(BINNAME_RING) $(BINNAME_CHAR) $(BINNAME_OUTPUT)
75      ./$(BINNAME_RING)
76      ./$(BINNAME_CHAR)
77      ./$(BINNAME_OUTPUT)
78
79  .PHONY: clean
80
81  clean:
82      rm -f $(BUILDDIR)/*.o *~ core $(INCDIR)/*~
83      rm $(BINNAME_RING)
84      rm $(BINNAME_CHAR)
85      rm $(BINNAME_OUTPUT)
```

```c
/*
 * ring_buffer_test.c
 *
 *  Created on: Nov 08, 2018
 *      Author: Dominic Doty
 *  Based on: http://cunit.sourceforge.net/example.html
 */

#include "Basic.h"            // CUnit Test Framework
#include "ring_buffer.h"

#define ZERO        0
#define SIXTEEN        16
#define SEVENTEEN    17

ring_buffer_struct ringo;
unsigned char buffer[SIXTEEN];


void test_ring_init()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(ring_init(NULL, buffer, SIXTEEN), RING_NULL_PTR);            // Test NULL struct ptr
    CU_ASSERT_EQUAL(ring_init(&ringo, NULL, SIXTEEN), RING_NULL_PTR);            // Test NULL buffer ptr

    /* BAD SIZES */
    CU_ASSERT_EQUAL(ring_init(&ringo, buffer, SEVENTEEN), RING_ILLEGAL_SIZE);    // Test non power 2 size
    CU_ASSERT_EQUAL(ring_init(&ringo, buffer, ZERO), RING_ILLEGAL_SIZE);        // Test zero size

    /* SUCCESS */
    CU_ASSERT_EQUAL(ring_init(&ringo, buffer, SIXTEEN), RING_SUCCESS);            // Good init

    /* CHECK STRUCTURE VALUES */
    CU_ASSERT_EQUAL(&ringo.buffer[0], &buffer[0]);                                    // Check values after init
    CU_ASSERT_EQUAL(ringo.head, 0);
    CU_ASSERT_EQUAL(ringo.tail, 0);
    CU_ASSERT_EQUAL(ringo.mask, SIXTEEN - 1);
}

void test_ring_add()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(ring_add(NULL, SIXTEEN), RING_NULL_PTR);                    // Test NULL struct ptr
}

void test_ring_remove()
{
    /* NULL POINTERS */
    unsigned char temp = 0;
    CU_ASSERT_EQUAL(ring_remove(NULL, &temp), RING_NULL_PTR);                    // Test NULL struct ptr
    CU_ASSERT_EQUAL(ring_remove(&ringo, NULL), RING_NULL_PTR);                    // Test NULL return ptr
}

void test_ring_element_count()
{
    /* NULL POINTERS */
    uint16_t temp = 0;
    CU_ASSERT_EQUAL(ring_element_count(NULL, &temp), RING_NULL_PTR);            // Test NULL struct ptr
    CU_ASSERT_EQUAL(ring_element_count(&ringo, NULL), RING_NULL_PTR);            // Test NULL return ptr

    /* RING COUNTS WITH 0 INDEX */
    ring_add(&ringo, 'A');
    ring_add(&ringo, 'A');
    ring_add(&ringo, 'A');
    ring_element_count(&ringo, &temp);
    CU_ASSERT_EQUAL(temp, 3)

    /* RESET RING BUFFER */
    CU_ASSERT_EQUAL(ring_init(&ringo, buffer, SIXTEEN), RING_SUCCESS);            // Good init
}
```

```
71
72 void test_add_remove_count()
73 {
74     /* CONFIRM A SINGLE ADDITION */
75     CU_ASSERT_EQUAL(ring_add(&ringo, 'A'), RING_SUCCESS);
76     CU_ASSERT_EQUAL(&ringo.buffer[0], &buffer[0]);
77     CU_ASSERT_EQUAL(ringo.head, 1);
78     CU_ASSERT_EQUAL(ringo.tail, 0);
79     CU_ASSERT_EQUAL(ringo.mask, SIXTEEN - 1);
80
81     /* ADD TILL FULL */
82     for(unsigned char i = 'B'; i < 'B'+14; i++)
83     {
84         CU_ASSERT_EQUAL(ring_add(&ringo, i), RING_SUCCESS);
85     }
86     CU_ASSERT_EQUAL(ring_add(&ringo, 'X'), RING_FULL);
87
88     /* CONFIRM A SINGLE REMOVAL */
89     unsigned char remove = 0;
90     CU_ASSERT_EQUAL(ring_remove(&ringo, &remove), RING_SUCCESS);
91     CU_ASSERT_EQUAL(remove, 'A');
92     CU_ASSERT_EQUAL(&ringo.buffer[0], &buffer[0]);
93     CU_ASSERT_EQUAL(ringo.head, 15);
94     CU_ASSERT_EQUAL(ringo.tail, 1);
95     CU_ASSERT_EQUAL(ringo.mask, SIXTEEN - 1);
96
97     /* ADD TO WRAP */
98     CU_ASSERT_EQUAL(ring_add(&ringo, 'P'), RING_SUCCESS);
99     CU_ASSERT_EQUAL(&ringo.buffer[0], &buffer[0]);
100    CU_ASSERT_EQUAL(ringo.head, 0);
101    CU_ASSERT_EQUAL(ringo.tail, 1);
102    CU_ASSERT_EQUAL(ringo.mask, SIXTEEN - 1);
103
104    /* ELEMENT COUNT WRAPPED FULL */
105    uint16_t count = 0;
106    CU_ASSERT_EQUAL(ring_element_count(&ringo, &count), RING_SUCCESS);
107    CU_ASSERT_EQUAL(count, 15);
108
109    /* REMOVE TILL EMPTY */
110    for(uint8_t i = 'B'; i < 'B'+15; i++)
111    {
112        CU_ASSERT_EQUAL(ring_remove(&ringo, &remove), RING_SUCCESS);
113        CU_ASSERT_EQUAL(remove, i);
114    }
115    CU_ASSERT_EQUAL(ring_remove(&ringo, &remove), RING_EMPTY);
116    CU_ASSERT_EQUAL(ring_element_count(&ringo, &count), 0);
117 }
118
119 int main()
120 {
121     CU_pSuite pSuite = NULL;
122
123     /* initialize the CUnit test registry */
124     if (CUE_SUCCESS != CU_initialize_registry())
125         return CU_get_error();
126
127     /* add a suite to the registry */
128     pSuite = CU_add_suite("Ring Buffer Tests", NULL, NULL);
129     if (NULL == pSuite)
130     {
131         CU_cleanup_registry();
132         return CU_get_error();
133     }
134
135     /* add the tests to the suite */
136     if ((NULL == CU_add_test(pSuite, "test of ring init", test_ring_init)) ||
137     (NULL == CU_add_test(pSuite, "test of ring add args", test_ring_add)) ||
138     (NULL == CU_add_test(pSuite, "test of ring remove args",test_ring_remove)) ||
139     (NULL == CU_add_test(pSuite, "test of ring element args",test_ring_element_count)) ||
140     (NULL == CU_add_test(pSuite, "test of ring add/remove/count",test_add_remove_count)))
141     {
```

```
142         CU_cleanup_registry();
143         return CU_get_error();
144     }
145 
146     /* Run all tests using the CUnit Basic interface */
147     CU_basic_set_mode(CU_BRM_VERBOSE);
148     CU_basic_run_tests();
149     CU_cleanup_registry();
150     return CU_get_error();
151 }
```

```c
/*
 * char_test.c
 *
 *  Created on: Nov 08, 2018
 *      Author: Dominic Doty
 *  Based on: http://cunit.sourceforge.net/example.html
 */

#include "Basic.h"            // CUnit Test Framework
#include "char_counter.h"

char_counter count;

void test_char_reset()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(char_reset(NULL), CHAR_NULL_PTR);

    /* INTRODUCE VALUES TO TEST RESET */
    count[50] = 'b';
    count[195] = 'f';
    count[255] = 'g';

    /* SUCCESS */
    CU_ASSERT_EQUAL(char_reset(count), CHAR_SUCCESS);

    /* CHECK ARRAY VALUES */
    CU_ASSERT_EQUAL(count[50], 0);
    CU_ASSERT_EQUAL(count[195], 0);
    CU_ASSERT_EQUAL(count[255], 0);
}

void test_char_add()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(char_add(NULL, 'b'), CHAR_NULL_PTR);

    /* TEST SUCCESSFUL ADDS */
    CU_ASSERT_EQUAL(char_add(count, 'a'), CHAR_SUCCESS);
    CU_ASSERT_EQUAL(char_add(count, 'a'), CHAR_SUCCESS);
    CU_ASSERT_EQUAL(char_add(count, 'f'), CHAR_SUCCESS);
    CU_ASSERT_EQUAL(char_add(count, '!'), CHAR_SUCCESS);

    /* CHECK ARRAY VALUES */
    CU_ASSERT_EQUAL(count[(uint8_t)'a'], 2);
    CU_ASSERT_EQUAL(count[(uint8_t)'f'], 1);
    CU_ASSERT_EQUAL(count[(uint8_t)'!'], 1);
    CU_ASSERT_EQUAL(count[(uint8_t)'b'], 0);
}


int main()
{
    CU_pSuite pSuite = NULL;

    /* initialize the CUnit test registry */
    if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

    /* add a suite to the registry */
    pSuite = CU_add_suite("Char Counter Tests", NULL, NULL);
    if (NULL == pSuite)
    {
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* add the tests to the suite */
    if ((NULL == CU_add_test(pSuite, "test of char reset", test_char_reset)) ||
```

```
70      (NULL == CU_add_test(pSuite, "test of char add", test_char_add)))
71      {
72          CU_cleanup_registry();
73          return CU_get_error();
74      }
75
76      /* Run all tests using the CUnit Basic interface */
77      CU_basic_set_mode(CU_BRM_VERBOSE);
78      CU_basic_run_tests();
79      CU_cleanup_registry();
80      return CU_get_error();
81  }
```

```c
/*
 * output_test.c
 *
 *  Created on: Nov 08, 2018
 *      Author: Dominic Doty
 *  Based on: http://cunit.sourceforge.net/example.html
 */

#include "Basic.h"            // CUnit Test Framework
#include "output_generator.h"
#include "ring_buffer.h"
#include "char_counter.h"

#define RING_SIZE 32

char_counter count;
ring_buffer_struct ring;
unsigned char buffer[RING_SIZE];

static bool check_ring(ring_buffer_struct* rbs, unsigned char character, uint8_t quant)
{
    bool ret = false;
    uint32_t i = 0;

    // Find the index of the char in question
    for(; i < 65537; i++)
    {
        if(rbs->buffer[i] == character)
        {
            break;
        }
    }

    // Couldn't find the character in the output
    if(i >= 65536)
    {
        ret = false;
    }

    // Confirm the formatting and count is correct
    bool test = (    (rbs->buffer[i] == character) &&
                    (rbs->buffer[i+1] == ':') &&
                    (rbs->buffer[i+2] == quant) &&
                    (rbs->buffer[i+3] == '\r') &&
                    (rbs->buffer[i+4] == '\n') );
    // If we find the right formatted string with the right count return true
    if(test)
    {
        ret = true;
    }

    return ret;
}

void test_output_single_char()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(output_single_char(NULL, &ring), OUTPUT_NULL_PTR);
    CU_ASSERT_EQUAL(output_single_char(count, NULL), OUTPUT_NULL_PTR);
}

void test_output_complete()
{
    /* NULL POINTERS */
    CU_ASSERT_EQUAL(output_complete(NULL, &ring), OUTPUT_NULL_PTR);
    CU_ASSERT_EQUAL(output_complete(count, NULL), OUTPUT_NULL_PTR);

    /* SET SOME THINGS IN THE COUNT */
    char_add(count, 'd');
```

```
    char_add(count, 'd');
    char_add(count, 'd');
    char_add(count, '?');
    char_add(count, '?');
    char_add(count, 'f');
    char_add(count, 'a');

    /* GENERATE THE OUTPUT */
    CU_ASSERT_EQUAL(output_complete(count, &ring), OUTPUT_SUCCESS);

    /* CHECK THE OUTPUT */
    CU_ASSERT_EQUAL(ring.buffer[0], (unsigned char)12);
    CU_ASSERT_EQUAL(check_ring(&ring, 'd', 3), true);
    CU_ASSERT_EQUAL(check_ring(&ring, '?', 2), true);
    CU_ASSERT_EQUAL(check_ring(&ring, 'f', 1), true);
    CU_ASSERT_EQUAL(check_ring(&ring, 'a', 1), true);

    /* GENERATE OUTPUT TILL FULL BUFFER */
    CU_ASSERT_EQUAL(output_complete(count, &ring), OUTPUT_FULL);
}


int main()
{
    ring_init(&ring, buffer, RING_SIZE);
    char_reset(count);

    CU_pSuite pSuite = NULL;

    /* initialize the CUnit test registry */
    if (CUE_SUCCESS != CU_initialize_registry())
        return CU_get_error();

    /* add a suite to the registry */
    pSuite = CU_add_suite("Output Generator Test", NULL, NULL);
    if (NULL == pSuite)
    {
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* add the tests to the suite */
    if ((NULL == CU_add_test(pSuite, "test of single char output", test_output_single_char)) ||
    (NULL == CU_add_test(pSuite, "test of full output", test_output_complete)))
    {
        CU_cleanup_registry();
        return CU_get_error();
    }

    /* Run all tests using the CUnit Basic interface */
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return CU_get_error();
}
```