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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file Project2.c

\* @brief main file for the memory test project.

\*

\* This source file provides the main() function for Project2

\*

\* @author Jon Warriner

\* @date March 28, 2019

\* @version 1.0

\*

\*/

**#include** <stdio.h>

**#include** "clock\_config.h"

**#include** "uart.h"

**#include** "led.h"

**#include** "ring.h"

**#include** "disp.h"

**#include** "ps\_rand.h"

**#include** "MKL25Z4.h"

//#define UART\_TESTING

**#define** APPLICATION

**#if** !defined(UART\_BLOCKING) && defined(UART\_TESTING)

**#define** TXRX\_BUF\_SIZE 16

ring\_t \*txrx\_buf = 0;

//some test code to check how full the buffer gets

int32\_t max = 0;

**#elif** defined(APPLICATION)

**#define** RX\_BUF\_SIZE 16

**#define** TX\_BUF\_SIZE 32

**struct** randStruct rnd;

uint32\_t seed = 10;

ring\_t \*rx\_buf = 0;

ring\_t \*tx\_buf = 0;

disp\_t disp = {0};

//some test code to check how full the buffer gets

int32\_t max = 0;

**#endif**

/\*

\* @brief Application entry point.

\*/

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

**#if** defined(UART\_BLOCKING) && defined(UART\_TESTING)

**char** temp;

**#endif**

/\* Init board hardware. \*/

BOARD\_InitBootClocks();

**#if** !defined(UART\_BLOCKING) && defined(UART\_TESTING)

//Inialize the GPIO for LED blinking

LED\_init();

txrx\_buf = ring\_init(TXRX\_BUF\_SIZE);

**#elif** defined(APPLICATION)

rx\_buf = **ring\_init**(RX\_BUF\_SIZE);

tx\_buf = **ring\_init**(TX\_BUF\_SIZE);

//setup the random number generator

rnd.m = RAND\_M;

rnd.c = RAND\_C;

rnd.a = RAND\_A;

rnd.X = seed;

**#endif**

//Initialize UART0

UART\_init();

**#ifdef** APPLICATION

//Initialize the count and display module

disp\_init(&disp, rx\_buf, tx\_buf, &UART\_EN\_TX\_INT);

**#endif**

//just letting this print to the console window to let me know the code started.

printf("Hello World\n");

/\* Force the counter to be placed into memory. \*/

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

/\* Enter an infinite loop, just incrementing a counter. \*/

**while**(1) {

i++;

**#if** defined(UART\_BLOCKING) && defined(UART\_TESTING)

temp = UART\_RX\_block();

UART\_TX\_block(temp);

**#elif** !defined(UART\_BLOCKING) && defined(UART\_TESTING)

LED\_toggle();

**#elif** defined(APPLICATION)

RX\_task(&disp);

Display\_task(&disp);

//generate random numbers in our spare time

ps\_rand(&rnd);

//force the latest random number into the last slot of the counting buffer

//we'll display the most recent random number on the next display update

disp.char\_ctrs[256] = rnd.X;

**#endif**

}

**return** 0 ;

}

**#if** !defined(UART\_BLOCKING) && defined(UART\_TESTING)

**void** UART0\_DriverIRQHandler(**void**)

{

**char** temp;

//if the UART has a character available, grab it and put it in the ring buffer

**if**(UART\_RX\_full())

{

insert(txrx\_buf, UART\_RX());

}

//just some test code to see how full the buffer gets

**if**(entries(txrx\_buf) > max)

{

max = entries(txrx\_buf);

}

//if the UART is ready to transmit, and we still have data in the buffer, then grab the next character and transmit it.

**if**(UART\_TX\_rdy() && (entries(txrx\_buf) != 0))

{

extract(txrx\_buf, &temp);

UART\_TX(temp);

}

}

**#elif** defined(APPLICATION)

**void** **UART0\_DriverIRQHandler**(**void**)

{

**char** temp;

//if the UART has a character available, grab it and put it in the ring buffer

**if**(UART\_RX\_full())

{

**insert**(rx\_buf, UART\_RX());

}

//just some test code to see how full the buffer gets

**if**(**entries**(rx\_buf) > max)

{

max = **entries**(rx\_buf);

}

//if the UART is ready to transmit, and we still have data in the buffer, then grab the next character and transmit it.

**if**(UART\_TX\_rdy())

{

**if**(**entries**(tx\_buf) != 0)

{

**extract**(tx\_buf, &temp);

UART\_TX(temp);

}

**else**

{

UART\_DIS\_TX\_INT();

}

}

}

**#endif**

/\*

\* Copyright 2017-2018 NXP

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without modification,

\* are permitted provided that the following conditions are met:

\*

\* o Redistributions of source code must retain the above copyright notice, this list

\* of conditions and the following disclaimer.

\*

\* o Redistributions in binary form must reproduce the above copyright notice, this

\* list of conditions and the following disclaimer in the documentation and/or

\* other materials provided with the distribution.

\*

\* o Neither the name of NXP Semiconductor, Inc. nor the names of its

\* contributors may be used to endorse or promote products derived from this

\* software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

\* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR

\* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

\* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

\* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON

\* ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

\* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\* TEXT BELOW IS USED AS SETTING FOR TOOLS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

!!GlobalInfo

product: Clocks v4.0

\* BE CAREFUL MODIFYING THIS COMMENT - IT IS YAML SETTINGS FOR TOOLS \*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @file clock\_config.c

\* @brief Board clocks initialization file.

\*/

/\* This is a template for board specific configuration created by MCUXpresso IDE Project Wizard.\*/

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file clock\_config.c

\* @brief Configure CPU clocks

\*

\* This source file was provided as a blank template from NXP. All code was filled

\* in by me.

\*

\* @author Jon Warriner

\* @date March 28, 2019

\* @version 1.0

\*

\*/

**#include** "MKL25Z4.h"

**#include** "clock\_config.h"

/\*\*

\* @brief Set up and initialize all required blocks and functions related to the board hardware.

\*/

**void** **BOARD\_InitBootClocks**(**void**) {

/\* The user initialization should be placed here \*/

//OUTDIV1 = 0000 - Divide-by-1

//OUTDIV4 = 001 - Divide-by-2

SIM->CLKDIV1 = SIM\_CLKDIV1\_OUTDIV1(0) | SIM\_CLKDIV1\_OUTDIV4(1);

//CLKS = 00 - Output of FLL or PLL is selected as source of MCGOUTCLK (depends on PLLS control bit).

//FRDIV = 000 - FLL reference clock divided by 1

//IREFS = 1 - select slow internal clock as source for FLL

//IRCLKEN = 1 - enable internal reference clock (IRCSCLK) as MCGIRCLK

//IREFSTEN = 0 - Internal reference clock is disabled in Stop mode

MCG->C1 = MCG\_C1\_IREFS\_MASK | MCG\_C1\_IRCLKEN\_MASK;

//LOCRE0 = 1 - Generate a reset request on a loss of OSC0 external reference clock.

//RANGE0 = 00 - Low frequency range selected for the crystal oscillator. (doesn't really matter, not using)

//HGO0 = 0 - Configure crystal oscillator for low-power operation.

//EREFS0 = 0 - External reference clock requested.

//LP = 0 - FLL or PLL is not disabled in bypass modes.

//IRCS = 1 - Fast internal reference clock selected as IRCSCLK.

MCG->C2 = MCG\_C2\_LOCRE0\_MASK | MCG\_C2\_IRCS\_MASK;

//DMX32 = 1 - reference frequency is 32.768 kHz

//DRST\_DRS = 01 - Encoding 1 — Mid range.

//The above bits configure the FLL to multiply a 32.768kHz clock by 1464 for a 47,972,352 Hz output.

//FCTRIM = XXXX - A value for FCTRIM is loaded during reset from a factory programmed location.

//SCFTRIM = X - A value for SCFTRIM is loaded during reset from a factory programmed location.

MCG->C4 |= (MCG\_C4\_DMX32\_MASK | MCG\_C4\_DRST\_DRS(1));

/\* Read core clock setting. \*/

SystemCoreClockUpdate();

}

/\*

\* Copyright 2017-2018 NXP

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without modification,

\* are permitted provided that the following conditions are met:

\*

\* o Redistributions of source code must retain the above copyright notice, this list

\* of conditions and the following disclaimer.

\*

\* o Redistributions in binary form must reproduce the above copyright notice, this

\* list of conditions and the following disclaimer in the documentation and/or

\* other materials provided with the distribution.

\*

\* o Neither the name of NXP Semiconductor, Inc. nor the names of its

\* contributors may be used to endorse or promote products derived from this

\* software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

\* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR

\* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

\* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

\* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON

\* ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

\* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*\*

\* @file clock\_config.h

\* @brief Board clocks header file.

\*/

/\* This is a template for board specific configuration created by MCUXpresso IDE Project Wizard.\*/

**#ifndef** \_CLOCK\_CONFIG\_H\_

**#define** \_CLOCK\_CONFIG\_H\_

**#if** defined(\_\_cplusplus)

**extern** "C" {

**#endif** /\* \_\_cplusplus \*/

/\*\*

\* @brief Initialize board clocks.

\*/

**void** **BOARD\_InitBootClocks**(**void**);

**#if** defined(\_\_cplusplus)

}

**#endif** /\* \_\_cplusplus \*/

**#endif** /\* \_CLOCK\_CONFIG\_H\_ \*/

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file disp.c

\* @brief count characters and build a display

\*

\* This source file provides support to count characters passed in through and input ring buffer

\* and builds a display of the count and puts it in an output ring buffer for transmission.

\*

\* @author Jon Warriner

\* @date April 9, 2019

\* @version 1.0

\*/

**#include** <stdio.h>

**#include** <string.h>

**#include** "disp.h"

int32\_t **disp\_init**(disp\_t \*d, ring\_t \*ibuf, ring\_t \*obuf, **void** (\*tx\_func)())

{

uint16\_t i;

//if any of the pointers are not initialized then return an error

**if**((d == 0) || (ibuf == 0) || (obuf == 0))

{

**return**(-1);

}

//make sure the counting buffer is zero

**for**(i = 0; i < 256; i++)

{

d->char\_ctrs[i] = 0;

}

//make sure other control variables are zero

d->trig = 0;

d->i = 0;

d->updating = 0;

//initialize pointers

d->ibuf = ibuf;

d->obuf = obuf;

d->transmit\_trig = tx\_func;

**return**(0);

}

**void** **RX\_task**(disp\_t \*d)

{

int32\_t ents,i;

uint8\_t data;

//if pointer isn't initialized return without doing anything

**if**(d == 0)

{

**return**;

}

ents = **entries**(d->ibuf);

//check if we have any entries in the RX buffer

//if we do, grab the new characters and count them

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

{

**extract**(d->ibuf,(**char** \*)&data);

d->char\_ctrs[data]++;

//tell the display task that we have data to update

d->trig = 1;

}

}

/\*\*

\* @brief Build the results display to send out the serial port

\*

\* We've got new characters and we need to update the count(s) on the display.

\*

\* @return void.

\*/

**void** **Display\_task**(disp\_t \*d)

{

size\_t i;

int32\_t ents;

//if pointer isn't initialized return without doing anything

**if**(d == 0)

{

**return**;

}

ents = **entries**(d->obuf);

// if the TX buffer isn't empty then we haven't finished the

// previous display update. Let's wait and try again later.

**if**(ents != 0)

{

**return**;

}

//RX/counting task signaled that we have new data

//no matter where we are in the current display update,

//start over. We don't want to miss any characters

**if**(d->trig)

{

d->i = 0;

d->trig = 0;

d->updating = 1;

// clear the screen and print a simple header to delimit one update from the next

sprintf(d->sbuf, "%c[2J%c[H\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n", (**char**)27, (**char**)27);

//move the string to the TX buffer

**for**(i = 0; i <= **strlen**(d->sbuf); i++)

{

**insert**(d->obuf, d->sbuf[i]);

}

//kick off the transmit by enabling the interrupt

d->transmit\_trig();

}

//Here we are in the middle of a display update

**else** **if**(d->updating)

{

//we only want to print results for characters that actually have a count

**if**(d->char\_ctrs[d->i])

{

//characters up to the space (32) are not printable

//we'll display their hex value

**if**(d->i <= 32)

{

sprintf(d->sbuf, "0x%x - %d\r\n", (**char**)d->i, d->char\_ctrs[d->i]);

}

//special case to print random number

**else** **if**(d->i == 256)

{

sprintf(d->sbuf, "rand - %d\r\n", d->char\_ctrs[d->i]);

}

//characters greater than 127 are not defined

//we'll display their hex value

**else** **if**(d->i >= 127)

{

sprintf(d->sbuf, "0x%x - %d\r\n", (**char**)d->i, d->char\_ctrs[d->i]);

}

**else**

{

sprintf(d->sbuf, "%c - %d\r\n", (**char**)d->i, d->char\_ctrs[d->i]);

}

//move the string to the TX buffer

**for**(i = 0; i <= **strlen**(d->sbuf); i++)

{

**insert**(d->obuf, d->sbuf[i]);

}

//kick off the transmit by enabling the interrupt

d->transmit\_trig();

}

d->i++;

//once we've looped through all 256 possible characters (plus the random number)

//we'll shut down the display update task

**if**(d->i >= 257)

{

d->updating = 0;

d->i = 0;

}

}

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file disp.h

\* @brief An abstraction for the character counting and functions

\*

\* This header file provides an abstraction of the functions to

\* count received characters and build a display.

\*

\* @author Jon Warriner

\* @date April 9 2019

\* @version 1.0

\*

\*/

**#ifndef** DISP\_H\_

**#define** DISP\_H\_

**#include** "ring.h"

/\*\*

\* define the counter and display structured data type

\*/

**typedef** **struct**

{

**void** (\*transmit\_trig)(); //pointer to a function to trigger transmission of display data

ring\_t \*ibuf; //pointer to input ring buffer. This contains incoming characters to be counted.

ring\_t \*obuf; //pointer to output ring buffer. This contains formatted output strings to be transmitted

uint32\_t char\_ctrs[257]; //counters for each of the 256 possible characters + 1 random number

**char** sbuf[80];

uint8\_t trig; //set to 1 to trigger the display update to start or start over

uint8\_t updating; //flag indicating that the display is currently updating

uint16\_t i; //index of character count that is currently being updated

}disp\_t;

/\*\*

\* @brief Initialize the display module

\*

\* @param d pointer to a display structure

\* @param ibuf pointer to an input ring buffer

\* @param obuf pointer to an output ring buffer

\* @param tx\_func pointer to a function to trigger transmission of the output buffer

\*

\* @return pointer to ring\_t type or 0 on failure

\*/

int32\_t **disp\_init**(disp\_t \*d, ring\_t \*ibuf, ring\_t \*obuf, **void** (\*tx\_func)());

/\*\*

\* @brief Check if there are any new characters in the RX ring buffer

\*

\* If there are new characters in the ring buffer then we need to add them to

\* our tally

\*

\* @param d pointer to a display structure

\*

\* @return void.

\*/

**void** **RX\_task**(disp\_t \*d);

/\*\*

\* @brief Build the results display to send out the serial port

\*

\* We've got new characters and we need to update the count(s) on the display.

\*

\* @param d pointer to a display structure

\*

\* @return void.

\*/

**void** **Display\_task**(disp\_t \*d);

**#endif** /\* DISP\_H\_ \*/

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file led.c

\* @brief LED GPIO toggle driver

\*

\* This source file implements a GPIO driver to toggle an LED

\*

\* @author Jon Warriner

\* @date March 31, 2019

\* @version 1.0

\*

\*/

**#include** "MKL25Z4.h"

**#include** "led.h"

**void** **LED\_init**()

{

//need to enable the clock to PORTB before we can configure any GPIO

//PORTB = 1 - PORTB clock enabled

SIM->SCGC5 = SIM\_SCGC5\_PORTB(1);

//We want PTB18 to be configured as a GPIO

//MUX = 001 - PTB18 configured as ALT1 GPIO

PORTB->PCR[18] = PORT\_PCR\_MUX(1);

//We want PTB18 to be configured as an output

GPIOB->PDDR |= (1 << 18);

}

**void** **LED\_toggle**()

{

GPIOB->PTOR = (1 << 18);

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warrinerk and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file led.h

\* @brief An abstraction for the LED toggle GPIO driver

\*

\* This header file provides an abstraction of the functions to

\* implement the LED toggle driver

\*

\* @author Jon Warriner

\* @date March 31 2019

\* @version 1.0

\*

\*/

**#ifndef** LED\_H\_

**#define** LED\_H\_

/\*\*

\* @brief Initialize GPIO for LED toggle

\*

\* @return void.

\*/

**void** **LED\_init**();

/\*\*

\* @brief Toggle the LED

\*

\* @return void.

\*/

**void** **LED\_toggle**();

**#endif** /\* LED\_H\_ \*/

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file ps\_rand.c

\* @brief Generate a pseudo random number

\*

\* This source file provides support for generating a sequence

\* of pseudo random numbers given a seed value.

\*

\* @author Jon Warriner

\* @date February 26, 2019

\* @version 1.0

\*/

**#include** "ps\_rand.h"

**void** **ps\_rand**(**struct** randStruct \*r)

{

**if**(r == 0)

{

**return**;

}

//compute the next value in the sequence.

//m must be a power of 2 so that we can use

//& here instead of %

r->X = (uint32\_t)(((r->a \* r->X) + r->c) & (r->m - 1));

**return**;

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warrinerk and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file ps\_rand.h

\* @brief An abstraction for the ps\_rand function

\*

\* This header file provides an abstraction of the function to

\* generate a pseudo random number sequence.

\*

\* @author Jon Warriner

\* @date February 26 2019

\* @version 1.0

\*

\*/

**#ifndef** \_\_PS\_RAND\_H\_\_

**#define** \_\_PS\_RAND\_H\_\_

**#include** <stdio.h>

**#include** <stdint.h>

// define the constants that we'll use in our rand function

//'m' is the modulus. This pseudo random number generator will

//produce integers in the range of 0 to m-1. For this implementation

//'m' must be a power of 2.

**#define** RAND\_M 0x0000000100000000 //2^32

//'a' is the multiplier. The previous output will be multiplied by 'a.'

//The valid range for 'a' is 0<a<m

**#define** RAND\_A 1664525

//'c' is the increment. This is added to the previous output after it

//is multiplied by 'a.' The valid range for 'c' is 0<=c<m

**#define** RAND\_C 1013904223

//Hull-Dobell Theorem

//A full period of 'm' can be achieved for all seed values if and only if

//

//1) m and c are relatively prime. (The only common factor of the two is 1.)

//2) a-1 is divisible by all prime factors off m.

//3) a-1 is divisible by 4 if m is divisible by 4.

//define a structure for a block of allocated memory

**struct** randStruct

{

uint64\_t a;

uint64\_t c;

uint64\_t m;

uint32\_t X;

};

/\*\*

\* @brief Generate a sequence of pseudo random numbers

\*

\* Given a pointer to a char command buffer, this will display

\* the contents of a block of previously allocated memory.

\*

\* @param cmd Pointer to a command buffer

\* @param b Pointer to a memory block structure

\*

\* @return void.

\*/

**void** **ps\_rand**(**struct** randStruct \*r);

**#endif** /\* \_\_PS\_RAND\_H\_\_ \*/

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file ring.c

\* @brief ring buffer functions

\*

\* This source file provides support for manipulating a ring buffer.

\* The basis of this file was given as part of the ring buffer

\* project for ECEN 5813.

\*

\* @author Jon Warriner

\* @date March 14, 2019

\* @version 1.0

\*/

**#include** "ring.h"

**#include** <stdlib.h>

ring\_t \***ring\_init**( int32\_t length )

{

**char** \*pbuf = 0;

ring\_t \*r = 0;

// make sure the requested length makes sense

**if**(length <= 0)

{

**return** 0;

}

// create the buffer and initialize the structure

pbuf = **malloc**(length);

r = (ring\_t \*)**malloc**(**sizeof**(ring\_t));

**if**((pbuf != 0) && (r != 0))

{

r->Buffer = pbuf;

r->Length = length;

r->Ini = 0;

r->Outi = 0;

**return**(r);

}

**else**

{

**return** 0;

}

}

int32\_t **insert**( ring\_t \*ring, **char** data )

{

**if**(ring == 0)

{

**return**(-1); //invalid pointer

}

**else** **if**( ring->Ini - ring->Outi < ring->Length )

{

ring->Buffer[ring->Ini++ & (ring->Length - 1)] = data;

**return** 0;

}

**else**

{

**return**(-1); //buffer is full

}

}

int32\_t **extract**( ring\_t \*ring, **char** \*data )

{

**if**(ring == 0)

{

**return**(-1); //invalid pointer

}

**else** **if**( ring->Outi != ring->Ini )

{

\*data = ring->Buffer[ring->Outi++ & (ring->Length - 1)];

**return** 0;

}

**else**

{

**return**(-1); //buffer is empty

}

}

int32\_t **entries**( ring\_t \*ring )

{

**if**(ring == 0)

{

**return**(-1); //invalid pointer

}

**else**

{

**return**((ring->Ini - ring->Outi));

}

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file ring.h

\* @brief An abstraction for the ring buffer functions

\*

\* This header file provides an abstraction of the functions to

\* manipulate a ring buffer. The basis of this file was given as

\* part of the ring buffer project for ECEN 5813.

\*

\* @author Jon Warriner

\* @date March 14 2019

\* @version 1.0

\*

\*/

**#ifndef** RING\_H

**#define** RING\_H

**#include** <stdint.h>

/\*\*

\* define the ring buffer structured data type

\*/

**typedef** **struct**

{

**char** \*Buffer;

int32\_t Length;

int32\_t Ini;

int32\_t Outi;

}ring\_t;

/\*\*

\* @brief Create a new ring buffer of "length" chars

\*

\* Given a length (in chars) return a pointer to a new

\* ring\_t type. Return 0 on failure.

\*

\* @param length Length of buffer in chars

\*

\* @return pointer to ring\_t type or 0 on failure

\*/

ring\_t \***ring\_init**( int32\_t length );

/\*\*

\* @brief Insert a new char into the buffer

\*

\* Given a pointer to an existing ring buffer and a piece of

\* data, insert the new data into the buffer.

\*

\* @param ring\_t Pointer to an already initialized ring buffer

\* @param data New data to add to the buffer

\*

\* @return 0 on success, -1 on failure

\*/

int32\_t **insert**( ring\_t \*ring, **char** data );

/\*\*

\* @brief Extract (remove) the next char from the buffer

\*

\* Given a pointer to an existing ring buffer and a pointer to

\* a data location, extract the oldest piece of data from the

\* buffer.

\*

\* @param ring\_t Pointer to an already initialized ring buffer

\* @param data Pointer to a location to write the extracted data

\*

\* @return 0 on success, -1 on failure

\*/

int32\_t **extract**( ring\_t \*ring, **char** \*data );

/\*\*

\* @brief Return the number of entries in the buffer

\*

\* Given a pointer to an existing ring buffer return the number

\* of entries in that buffer.

\*

\* @param ring\_t Pointer to an already initialized ring buffer

\*

\* @return number of buffer entries, returns -1 on error

\*/

int32\_t **entries**( ring\_t \*ring );

**#endif**

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file uart.c

\* @brief UART driver

\*

\* This source file implements a UART driver

\*

\* @author Jon Warriner

\* @date March 31, 2019

\* @version 1.0

\*

\*/

**#include** "MKL25Z4.h"

**#include** "uart.h"

**void** **UART\_init**()

{

//UART config

//UART0TXSRC = 00 - Selects the source for the UART0 transmit data as UART0\_TX pin

//UART0RXSRC = 0 - Selects the source for the UART0 receive data as UART0\_RX pin

//These should already be the default settings

//We want all other bit in this register to be 0 too

SIM->SOPT5 = 0;

//UART0SRC = 1 - MCGFLLCLK clock is the source for the UART0 transmit and receive clock

//The rest of the bits can be 0

SIM->SOPT2 = SIM\_SOPT2\_UART0SRC(1);

//UART0 = 1 - UART0 clock enabled

//Don't mess with other bits

SIM->SCGC4 |= SIM\_SCGC4\_UART0(1);

//PORTA = 1 - PORTA clock enabled

SIM->SCGC5 = SIM\_SCGC5\_PORTA(1);

//Configure UART pins

//MUX = 010 - PTA1 configured as ALT2 UART0\_RX

PORTA->PCR[1] = PORT\_PCR\_MUX(2);

//MUX = 010 - PTA2 configured as ALT2 UART0\_TX

PORTA->PCR[2] = PORT\_PCR\_MUX(2);

//Configure baudrate to 115200

//47972352/(115200 \* (15+1)) = 26.03

//BR = 26 (0x1A)

UART0->BDL = UART0\_BDL\_SBR(0x1A);

//OSR = 0xF (should alread be power-up default)

UART0->C4 = UART0\_C4\_OSR(0xF);

**#ifndef** UART\_BLOCKING

//set UART0 interrupt priority to 0

**#define** UART\_PRI 0

NVIC->IP[\_IP\_IDX(*UART0\_IRQn*)] = ((uint32\_t)(NVIC->IP[\_IP\_IDX(*UART0\_IRQn*)] & ~(0xFFUL << \_BIT\_SHIFT(*UART0\_IRQn*))) |

(((UART\_PRI << (8U - \_\_NVIC\_PRIO\_BITS)) & (uint32\_t)0xFFUL) << \_BIT\_SHIFT(*UART0\_IRQn*)));

//enable the UART IRQ

NVIC->ISER[0U] = (uint32\_t)(1UL << (((uint32\_t)(int32\_t)*UART0\_IRQn*) & 0x1FUL));

UART0->C2 |= UART0\_C2\_RIE(1);

**#endif**

//enable receiver and transmitter

UART0->C2 |= (UART0\_C2\_RE(1) | UART0\_C2\_TE(1));

}

**void** **UART\_EN\_TX\_INT**()

{

UART0->C2 |= UART0\_C2\_TIE(1);

}

**void** **UART\_DIS\_TX\_INT**()

{

UART0->C2 &= ~UART0\_C2\_TIE(1);

}

uint8\_t **UART\_TX\_rdy**()

{

**return**((UART0->S1 & UART\_S1\_TDRE\_MASK) >> UART\_S1\_TDRE\_SHIFT);

}

**void** **UART\_TX**(**char** data)

{

UART0->D = data;

}

**void** **UART\_TX\_block**(**char** data)

{

//just wait here until the TX buffer is empty

**while**(UART\_TX\_rdy() == 0);

//Now we can spit out the character

UART\_TX(data);

}

uint8\_t **UART\_RX\_full**()

{

**return**((UART0->S1 & UART\_S1\_RDRF\_MASK) >> UART\_S1\_RDRF\_SHIFT);

}

**char** **UART\_RX**()

{

**return**(UART0->D);

}

**char** **UART\_RX\_block**()

{

//just wait here until the RX buffer is full

**while**(UART\_RX\_full() == 0);

//Now we can grab the character

**return**(UART\_RX());

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warrinerk and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file uart.h

\* @brief An abstraction for the UART driver

\*

\* This header file provides an abstraction of the functions to

\* implement the UART driver

\*

\* @author Jon Warriner

\* @date March 31 2019

\* @version 1.0

\*

\*/

**#ifndef** UART\_H\_

**#define** UART\_H\_

**#include** <stdint.h>

//#define UART\_BLOCKING

/\*\*

\* @brief Initialize UART0

\*

\* Initialize UART0 for the application

\*

\* @return void.

\*/

**void** **UART\_init**();

/\*\*

\* @brief Is UART0 ready to transmit a character?

\*

\* Check the TDRE bit in S1 register to determine if the TX

\* buffer is empty or has a character that has yet to go out.

\*

\* @return uint8\_t 1 - TX buffer is empty, 0 - TX buffer is full

\*/

uint8\_t **UART\_TX\_rdy**();

/\*\*

\* @brief Enable UART0 TX Interrupt

\*

\* @return void

\*/

**void** **UART\_EN\_TX\_INT**();

/\*\*

\* @brief Disable UART0 TX Interrupt

\*

\* @return void

\*/

**void** **UART\_DIS\_TX\_INT**();

/\*\*

\* @brief Put a character in the UART0 TX buffer

\*

\* Assume the empty/full status of the buffer has already been checked

\* and place a character in the TX buffer.

\*

\* @param data character to place in the buffer

\*

\* @return void

\*/

**void** **UART\_TX**(**char** data);

/\*\*

\* @brief Use UART0 to transmit a character when the buffer is empty

\*

\* Wait (blocking) for the UART0 TX buffer to be empty then transmit

\* a character.

\*

\* @param data character to place in the buffer

\*

\* @return void

\*/

**void** **UART\_TX\_block**(**char** data);

/\*\*

\* @brief Do we have a new character in the UART0 RX buffer?

\*

\* Check the RDRF bit in S1 register to determine if the RX

\* buffer is empty or has a new character available.

\*

\* @return uint8\_t 1 - RX buffer is full, 0 - RX buffer empty

\*/

uint8\_t **UART\_RX\_full**();

/\*\*

\* @brief Get a character from the UART0 RX buffer

\*

\* Assume the empty/full status of the buffer has already been checked

\* and get a character from the RX buffer.

\*

\* @return character read from the buffer

\*/

**char** **UART\_RX**();

/\*\*

\* @brief Receive a character from UART0 when the buffer is full

\*

\* Wait (blocking) for the UART0 RX buffer to be full then read

\* a character.

\*

\* @return character read from the buffer

\*/

**char** **UART\_RX\_block**();

**#endif** /\* UART\_H\_ \*/

**Linux Unit testing code**

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Copyright (C) 2019 by Jon Warriner

#

# Redistribution, modification or use of this software in source or binary

# forms is permitted as long as the files maintain this copyright. Users are

# permitted to modify this and use it to learn about the field of embedded

# software. Jon Warriner and the University of Colorado are not liable for

# any misuse of this material.

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# @file makefile

# @brief makefile for the ring buffer test project

#

# This file defines the make steps to build the

# ring buffer test project.

#

# @author Jon Warriner

# @date March 14, 2019

# @version 1.0

#

**INC** = ../inc/

**SRC** = ../src/

ringtest : unittest.o ring.o

cc -o ringtest unittest.o ring.o -lcunit

longtest : unittest\_long.o ring.o

cc -o longtest unittest\_long.o ring.o -lcunit

unittest.o : unittest.c $(INC)ring.h

cc -Wall -Werror -I$(INC) -c unittest.c

ring.o : $(SRC)ring.c $(INC)ring.h

cc -Wall -Werror -I$(INC) -c $(SRC)ring.c

unittest\_long.o : unittest\_long.c $(INC)ring.h

cc -Wall -Werror -I$(INC) -c unittest\_long.c

clean :

rm ringtest unittest.o ring.o unittest\_long.o

test : ringtest

./ringtest

test\_long : longtest

./longtest

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file ring\_test.c

\* @brief Ring buffer unit testing

\*

\* This source file provides support for unit testing a ring buffer.

\* The basis of this file was given as part of the ring buffer

\* project for ECEN 5813.

\*

\* @author Jon Warriner

\* @date March 14, 2019

\* @version 1.0

\*/

**#include** "ring.h"

**#include** <stdio.h>

**#include** <CUnit/Basic.h>

//#define DEBUG //uncomment this to display detailed buffer info for each step of the tests

**#define** TEST\_LENGTH 8 //Buffer length must be a power of 2.

ring\_t \*ring = 0;

**#ifdef** DEBUG

// Debug function to display the ring buffer parameters and contents

**void** **dispBuf**(ring\_t \*r)

{

int32\_t j;

printf("\r\nLength: %d\r\n", r->Length);

printf("Ini: %d\r\n", r->Ini);

printf("Outi: %d\r\n", r->Outi);

printf("Buffer: \r\n");

**for**(j = 0; j < TEST\_LENGTH; j++)

{

printf("%d\r\n", r->Buffer[j]);

}

printf("Entries: %d\r\n\r\n", **entries**(r));

}

**#endif**

/\* The suite initialization function.

\* Nothing to do in here for now.

\* Returns zero on success, non-zero otherwise.

\*/

**int** **init\_suite1**(**void**)

{

**return** 0;

}

/\* The suite cleanup function.

\* Nothing to do in here for now.

\* Returns zero on success, non-zero otherwise.

\*/

**int** **clean\_suite1**(**void**)

{

**return** 0;

}

/\* Simple test of init().

\* Attempt an invalid and then valid initialization of a ring buffer.

\*/

**void** **testINIT**(**void**)

{

// first try a length of 0. This is invalid and should return 0

ring = **ring\_init**(0);

CU\_ASSERT(0 == ring);

// now try a valid length

// check that we got a non-zero pointer

ring = **ring\_init**(TEST\_LENGTH);

CU\_ASSERT(0 != ring);

// check that the structure members make sense after initialization

CU\_ASSERT(ring->Buffer != 0);

CU\_ASSERT(ring->Length == TEST\_LENGTH);

CU\_ASSERT(ring->Ini == 0);

CU\_ASSERT(ring->Outi == 0);

}

/\* Simple test of insert().

\* First, we will verify the the function correctly returns an error

\* for an invalide pointer.

\* Next, we will attempt to fill the buffer and verify each insertion.

\* Finally, we will attempt one more insertion and verify that we get

\* an error for attempting to insert into a full buffer.

\*

\*/

**void** **testINSERT**(**void**)

{

int32\_t i;

int32\_t err;

int32\_t ents;

**char** data = 1;

// first, pass a NULL pointer for ring. This should return -1

err = **insert**((ring\_t \*)0, data);

CU\_ASSERT(err == -1);

// attempt to fill the buffer with incrementing data

// check the elements of the structure and number of

// entries after each insertion.

**#ifdef** DEBUG

// Display the ring buffer parameters and contents before we start filling it.

dispBuf(ring);

**#endif**

**for**(i = 0; i < TEST\_LENGTH; i++)

{

// first insert the data and make sure no errors are returned.

err = **insert**(ring, data);

CU\_ASSERT(err == 0);

// check the structure members that should NOT have changed

CU\_ASSERT(ring->Length == TEST\_LENGTH);

CU\_ASSERT(ring->Outi == 0);

// now check that the input elememt pointer has incremented

CU\_ASSERT(ring->Ini == (i + 1));

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == (i + 1));

// now check that our data got written to the buffer

CU\_ASSERT(ring->Buffer[i] == data);

data++;

**#ifdef** DEBUG

// Display the ring buffer parameters and contents during each iteration.

dispBuf(ring);

**#endif**

}

// at this point we should have a full buffer

// let's try to add one more entry. We should get a -1

err = **insert**(ring, data);

CU\_ASSERT(err == -1);

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == TEST\_LENGTH);

}

/\* Simple test of extract().

\* First, we will verify the the function correctly returns an error

\* for an invalide pointer.

\* Next, we will attempt to empty the buffer and verify each extraction.

\* Finally, we will attempt one more extraction and verify that we get

\* an error for attempting to remove from and empty buffer.

\*

\*/

**void** **testREMOVE**(**void**)

{

int32\_t i;

int32\_t err;

int32\_t ents;

**char** data;

// first, pass a NULL pointer for ring. This should return -1

err = **extract**((ring\_t \*)0, &data);

CU\_ASSERT(err == -1);

// attempt to empty the buffer and check the data

// check the elements of the structure and number of

// entries after each extraction.

**#ifdef** DEBUG

// Display the ring buffer parameters and contents before we start emptying it.

dispBuf(ring);

**#endif**

**for**(i = 0; i < TEST\_LENGTH; i++)

{

// first extract the data and make sure no errors are returned.

err = **extract**(ring, &data);

CU\_ASSERT(err == 0);

// check the structure members that should NOT have changed

CU\_ASSERT(ring->Length == TEST\_LENGTH);

CU\_ASSERT(ring->Ini == TEST\_LENGTH);

// now check that the Output elememt pointer has incremented

CU\_ASSERT(ring->Outi == (i + 1));

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == (TEST\_LENGTH - (i + 1)));

// now check that we extracted the expected data

// remember, we inserted incrementing data in the previous test

CU\_ASSERT(data == (i+1));

**#ifdef** DEBUG

// Display the ring buffer parameters and contents during each iteration.

dispBuf(ring);

printf("Data: %d\r\n", data);

**#endif**

}

// at this point we should have an empty buffer

// let's try extracting one more time. We should get a -1

err = **extract**(ring, &data);

CU\_ASSERT(err == -1);

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == 0);

}

/\* Simple test of insert() with wrap-around.

\* Attempt to fill the buffer and verify each insertion. We previously

\* filled and emptied the buffer so the first insertion should cause

\* the indices to wrap-around.

\* Finally, we will attempt one more insertion and verify that we get

\* an error for attempting to insert into a full buffer.

\*

\*/

**void** **testINSERT\_wrap**(**void**)

{

int32\_t i;

int32\_t err;

int32\_t ents;

**char** data = 11;

// attempt to fill the buffer with incrementing data

// check the elements of the structure and number of

// entries after each insertion.

**#ifdef** DEBUG

// Display the ring buffer parameters and contents before we start filling it.

dispBuf(ring);

**#endif**

**for**(i = 0; i < TEST\_LENGTH; i++)

{

// first insert the data and make sure no errors are returned.

err = **insert**(ring, data);

CU\_ASSERT(err == 0);

// check the structure members that should NOT have changed

CU\_ASSERT(ring->Length == TEST\_LENGTH);

CU\_ASSERT(ring->Outi == ring->Length);

// now check that the input elememt pointer has incremented

CU\_ASSERT(ring->Ini == (i + (TEST\_LENGTH + 1)));

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == (i + 1));

// now check that our data got written to the buffer

CU\_ASSERT(ring->Buffer[i] == data);

data++;

**#ifdef** DEBUG

// Display the ring buffer parameters and contents during each iteration.

dispBuf(ring);

**#endif**

}

// at this point we should have a full buffer

// let's try to add one more entry. We should get a -1

err = **insert**(ring, data);

CU\_ASSERT(err == -1);

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == TEST\_LENGTH);

}

/\* Simple test of extract() with wrap-around.

\* We will attempt to empty the buffer and verify each extraction. We

\* previously filled, emptied then refilled the buffer. The first

\* extraction should cause the out index to wrap.

\* Finally, we will attempt one more extraction and verify that we get

\* an error for attempting to remove from and empty buffer.

\*

\*/

**void** **testREMOVE\_wrap**(**void**)

{

int32\_t i;

int32\_t err;

int32\_t ents;

**char** data;

// attempt to empty the buffer and check the data

// check the elements of the structure and number of

// entries after each extraction.

**#ifdef** DEBUG

// Display the ring buffer parameters and contents before we start emptying it.

dispBuf(ring);

**#endif**

**for**(i = 0; i < TEST\_LENGTH; i++)

{

// first extract the data and make sure no errors are returned.

err = **extract**(ring, &data);

CU\_ASSERT(err == 0);

// check the structure members that should NOT have changed

CU\_ASSERT(ring->Length == TEST\_LENGTH);

CU\_ASSERT(ring->Ini == (TEST\_LENGTH << 1));

// now check that the Output elememt pointer has incremented

CU\_ASSERT(ring->Outi == (TEST\_LENGTH + (i + 1)));

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == (TEST\_LENGTH - (i + 1)));

// now check that we extracted the expected data

// remember, we inserted incrementing data in the previous test

CU\_ASSERT(data == (i+11));

**#ifdef** DEBUG

// Display the ring buffer parameters and contents during each iteration.

dispBuf(ring);

printf("Data: %d\r\n", data);

**#endif**

}

// at this point we should have an empty buffer

// let's try extracting one more time. We should get a -1

err = **extract**(ring, &data);

CU\_ASSERT(err == -1);

// check that the entries() function works

ents = **entries**(ring);

CU\_ASSERT(ents == 0);

}

/\* The main() function for setting up and running the tests.

\* Returns a CUE\_SUCCESS on successful running, another

\* CUnit error code on failure.

\*/

**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("Suite\_1", init\_suite1, clean\_suite1);

**if** (NULL == pSuite) {

CU\_cleanup\_registry();

**return** CU\_get\_error();

}

/\* add the tests to the suite \*/

/\* NOTE - ORDER IS IMPORTANT - MUST TEST init() THEN insert() THEN remove() THEN insert() with wrap \*/

/\* finally remove() with wrap. \*/

**if** ((NULL == CU\_add\_test(pSuite, "test of init()", testINIT)) ||

(NULL == CU\_add\_test(pSuite, "test of insert()", testINSERT)) ||

(NULL == CU\_add\_test(pSuite, "test of remove()", testREMOVE)) ||

(NULL == CU\_add\_test(pSuite, "test of insert() with wrap-around", testINSERT\_wrap)) ||

(NULL == CU\_add\_test(pSuite, "test of remove() with wrap-around", testREMOVE\_wrap))

)

{

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();

}

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

\* Copyright (C) 2019 by Jon Warriner

\*

\* Redistribution, modification or use of this software in source or binary

\* forms is permitted as long as the files maintain this copyright. Users are

\* permitted to modify this and use it to learn about the field of embedded

\* software. Jon Warriner and the University of Colorado are not liable for

\* any misuse of this material.

\*

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

/\*\*

\* @file unittest\_long.c

\* @brief Ring buffer long, random unit testing

\*

\* This source file provides support for unit testing a ring buffer.

\* The basis of this file was given as part of the ring buffer

\* project for ECEN 5813.

\*

\* @author Jon Warriner

\* @date March 14, 2019

\* @version 1.0

\*/

**#include** "ring.h"

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**#include** <CUnit/Basic.h>

//#define DEBUG //uncomment this to display detailed buffer info for each step of the tests

**#define** TEST\_LENGTH 64 //Buffer length must be a power of 2.

**#define** TEST\_DURATION 1000000

ring\_t \*ring = 0;

ring\_t \*ring2 = 0;

uint32\_t g\_inserts = 0;

uint32\_t f\_inserts = 0;

uint32\_t g\_extracts = 0;

uint32\_t f\_extracts = 0;

**#ifdef** DEBUG

// Debug function to display the ring buffer parameters and contents

**void** **dispBuf**(ring\_t \*r)

{

int32\_t j;

printf("\r\nLength: %d\r\n", r->Length);

printf("Ini: %d\r\n", r->Ini);

printf("Outi: %d\r\n", r->Outi);

printf("Buffer: \r\n");

**for**(j = 0; j < TEST\_LENGTH; j++)

{

printf("%d\r\n", r->Buffer[j]);

}

printf("Entries: %d\r\n\r\n", **entries**(r));

}

**#endif**

/\* The suite initialization function.

\* Nothing to do in here for now.

\* Returns zero on success, non-zero otherwise.

\*/

**int** **init\_suite1**(**void**)

{

**return** 0;

}

/\* The suite cleanup function.

\* Nothing to do in here for now.

\* Returns zero on success, non-zero otherwise.

\*/

**int** **clean\_suite1**(**void**)

{

**return** 0;

}

/\* Simple test of init().

\* Attempt an invalid and then valid initialization of a ring buffer.

\*/

**void** **testLONG**(**void**)

{

int32\_t i;

int32\_t ents;

int32\_t r;

int32\_t err;

**char** wdata = 5;

**char** rdata = 0;

**char** rdata2 = 0;

// Create a buffer

// check that we got a non-zero pointer

ring = **ring\_init**(TEST\_LENGTH);

CU\_ASSERT(0 != ring);

// Create a second buffer

// this one is intended to have the same data as the first

// we'll use this to compare and check

ring2 = **ring\_init**(TEST\_LENGTH);

CU\_ASSERT(0 != ring2);

**srand**(**time**(0));

**for**(i = 0; i < TEST\_DURATION; i++)

{

//get the number of current entries in the buffer

//we will use this to determine whether or not to expect

//an error when we insert or extract to/from the buffer

ents = **entries**(ring);

r = **rand**();

//if r is odd, insert a character into the buffer

//if r is even, extract a character from the buffer

**if**(r & 0x1)

{

wdata = (**char**)(**rand**() & 0x00FF);

err = **insert**(ring, wdata);

//if the buffer is full we should expect a non-zero error

**if**(ents < TEST\_LENGTH)

{

CU\_ASSERT(err == 0);

// assume everything went ok and insert data into the second buffer as well

err = **insert**(ring2, wdata);

g\_inserts++;

}

**else**

{

CU\_ASSERT(err != 0);

f\_inserts++;

}

}

**else**

{

err = **extract**(ring, &rdata);

//if the buffer is empty we should expect a non-zero error

**if**(ents != 0)

{

CU\_ASSERT(err == 0);

//assume the extract went ok and extract from the second buffer as well.

//the dat from the first and second buffers should match

err = **extract**(ring2, &rdata2);

CU\_ASSERT(rdata == rdata2);

g\_extracts++;

}

**else**

{

CU\_ASSERT(err != 0);

f\_extracts++;

}

}

}

}

/\* The main() function for setting up and running the tests.

\* Returns a CUE\_SUCCESS on successful running, another

\* CUnit error code on failure.

\*/

**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("Suite\_1", init\_suite1, clean\_suite1);

**if** (NULL == pSuite) {

CU\_cleanup\_registry();

**return** CU\_get\_error();

}

/\* add the tests to the suite \*/

/\* NOTE - ORDER IS IMPORTANT - MUST TEST init() THEN insert() THEN remove() THEN insert() with wrap \*/

/\* finally remove() with wrap. \*/

**if** ((NULL == CU\_add\_test(pSuite, "Long running random test", testLONG)))

{

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();

//print out some of my own statistics

printf("%d successful insertions\r\n", g\_inserts);

printf("%d failed insertions\r\n", f\_inserts);

printf("%d successful extractions\r\n", g\_extracts);

printf("%d failed extractions\r\n", f\_extracts);

printf("total = %d\r\n", g\_inserts + f\_inserts + g\_extracts + f\_extracts);

**return** CU\_get\_error();

}