UNITTEST.C

```c
#include <stdlib.h>
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"                    //CALL THE HEADER FILES AND SETS OF LIBRARY
FUNCTIONS
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "logger.h"
#include "memory_test.h"
#include "led.h"
#include "uCUnit.h"


//PATERN GENERATION DEFINES FOR MERRAIN TWISTER RANDOM
//PATTERN GNERATION FUNCTION
#define UPPER_MASK        0x8000
#define LOWER_MASK        0x7fff
#define TEMPERING_MASK_B  0x9d2c
#define TEMPERING_MASK_C  0xefc60000
#define STATE_VECTOR_LENGTH 62
#define STATE_VECTOR_M      39 /* changes to STATE_VECTOR_LENGTH also require changes
to this */

typedef struct tagMTRand                            //defining a structure data type
of MTRand
{
    unsigned int mt[STATE_VECTOR_LENGTH];         //DEFINING THE ARRAY TO STORE THE
PATTERN
    int index;                                    //using the index to specify the
array elements
    //the value points to
} Random;

unsigned int length=4;                      // DEFINING THE SIZE OF THE BLOCK TO BE
DYNAMICALLY
//ALLOCATED

extern unsigned int pat[100];      //INITIALIZING AN ARRAY PAT TO STORE THE RANDOM
PATTERNS
//GENERATED BY RANDOM PATTERN GENERATOR
Random seedRand(unsigned int seed);
unsigned int genRandLong(Random* rand);  //PROTOTYPES OF ALL FUNCTIONS WHICH ARE
//INVOLVED IN GENERATING THE RANDOM PATTERN
unsigned int genRand(Random* rand);
void m_seedRand(Random* rand, unsigned int seed);
void gen_pattern(size_t length,unsigned int seed);


int main(void)
{
```

```
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
    /* Init FSL debug console. */
    BOARD_InitDebugConsole();
    //printf("hello");
    PRINTF("Hello World\n\r");

    LED_RED_INIT(1);
    LED_GREEN_INIT(1);                  //INITIALIZING THE LED INIT PINS
    LED_BLUE_INIT(1);




    //MEMORY TEST SUITE



    //Unit Test cases
            UCUNIT_Init(); /* initialize framework */
            UCUNIT_TestcaseBegin("Crazy Scientist");
            /* test cases ... */


    uint32_t *test_space = allocate_words(length);  //ALLOCATING THE ADDRESS OF THE
BLOCK_POINTER
    //TO TEST SPACE

    //EXECUTING THE MEMORY TESTS WHICH HAS
    //FOLLOWING FUNCTIONS
    write_pattern( test_space,  length,  10); //CALLING WRITE PATTERN WHICH WRITES
PATTERN TO
    //MEMORY LOCATION

    UCUNIT_CheckIsNotNull(test_space);

    display_memory(test_space,length);   //CALLING THE MEMORY CONTENTS
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);
    verify_pattern( test_space, length, 10);  //VERIFY PATTERN TO CHECK IF THE MEMORY
CONTENTS
    //MATCH THE CONTENTS OF THE ARRAY WHICH CONTAINS
    // THE RANDOMLY GENERATED PATTERN
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);



    write_memory(test_space,2,0xFFEE);        //WRITE TO MEMORY VALUE 0XFFEE AT
OFFSET 2
    PRINTF("\n\r");
```

```
    UCUNIT_CheckIsNotNull(test_space);
    display_memory(test_space,length);  //DISPLAY THE MEMORY CONTENTS OF THE
BLOCK_POINTER
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);
    verify_pattern( test_space, length, 10); //VERIFY PATTERN TO CHECK IF THE MEMORY
CONTENTS
    //MATCH THE CONTENTS OF THE ARRAY WHICH CONTAINS

    PRINTF("\n\r");
    PRINTF("\n\r");
    write_pattern( test_space,  length,  10);  //CALLING WRITE PATTERN WHICH WRITES
PATTERN TO
    //MEMORY LOCATION
    UCUNIT_CheckIsNotNull(test_space);
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);
    display_memory(test_space,length);     //DISPLAY THE MEMORY CONTENTS OF THE
BLOCK_POINTER
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);
    verify_pattern( test_space, length, 10); //VERIFY PATTERN TO CHECK IF THE MEMORY
CONTENTS
    //MATCH THE CONTENTS OF THE ARRAY WHICH CONTAINS
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);

    PRINTF("\n\r");                  //INVERT THE BUFFER_POINTER AT THE LOCATION WITH
1 OFFSET
    invert(test_space,1);
    UCUNIT_CheckIsNotNull(test_space);
    PRINTF("\n\r");
    display_memory(test_space,length);  //DISPLAY THE MEMORY CONTENTS OF THE
BLOCK_POINTER
    PRINTF("\n\r");
    UCUNIT_CheckIsNotNull(test_space);
    verify_pattern(test_space, length, 10); //VERIFY PATTERN TO CHECK IF THE MEMORY
CONTENTS
    //MATCH THE CONTENTS OF THE ARRAY WHICH CONTAINS

    PRINTF("\n\r");

    PRINTF("\n\r");
    invert(test_space,1);         //INVERT THE BUFFER_POINTER AT THE LOCATION WITH 1
OFFSET
    PRINTF("\n\r");
    display_memory(test_space,length); //DISPLAY THE MEMORY CONTENTS OF THE
BLOCK_POINTER
    PRINTF("\n\r");
    verify_pattern( test_space, length, 10); //VERIFY PATTERN TO CHECK IF THE MEMORY
CONTENTS
    //MATCH THE CONTENTS OF THE ARRAY WHICH CONTAINS

    PRINTF("\n\r");
```

```
    PRINTF("\n\r");
    get_address(test_space,3);      //GET ADDRESS OF A PARTICULAR OFFSET FROM
LOCATION
    PRINTF("\n\r");
    free_words(test_space,length); //FREE BLOCK_POINTER
    LED_BLUE_OFF();
    LED_GREEN_ON();               //TURN THE LED GREEN ON
    LED_RED_OFF();

    PRINTF("\n\r");
    printf("\n\r");
////////////////////////////////////////////////////////////////////////////////
////////


//
//        UCUNIT_CheckIsEqual(read_return(test_space,len,1),2 );
//        UCUNIT_CheckIsEqual(read_return(test_space,len,2),3 );
//        UCUNIT_CheckIsEqual(read_return(test_space,len,3),4 );


     //   blink(ERR);


    //    UCUNIT_CheckIsEqual(x, 0);          /* check if x == 0 */
    //    UCUNIT_CheckIsInRange(x, 0, 10);    /* check 0 <= x <= 10 */
    //    UCUNIT_CheckIsBitSet(x, 7);         /* check if bit 7 set */
    //    UCUNIT_CheckIsBitClear(x, 7);             /* check if bit 7 cleared */
    //    UCUNIT_CheckIs8Bit(x);              /* check if not larger then 8 bit */
    //    UCUNIT_CheckIs16Bit(x);             /* check if not larger then 16 bit */
    //    UCUNIT_CheckIs32Bit(x);             /* check if not larger then 32 bit */
    //    UCUNIT_CheckIsNull(p);              /* check if p == NULL */
    //    UCUNIT_CheckIsNotNull(s);           /* check if p != NULL */
    //    UCUNIT_Check((*s)==’\0’, "Missing termination", "s");   /* generic check:
condition, msg, args */



       UCUNIT_TestcaseEnd();
       UCUNIT_WriteSummary();
        UCUNIT_Shutdown();




    return 0 ;
}
```

```
void gen_pattern(size_t length,unsigned int seed)
{
    unsigned int i;            //GENERATE PATTERN FUNCTION WHICH STORES THE RANDOM
PATTERN
    //IN THE PARTICULAR ARRAY

    Random r = seedRand(seed);
    for(i=0; i<length; i++)
    {
        pat[i]=genRand(&r);   //STORE THE RANDOM PATTERN IN ARRAY PAT

    }

}


void m_seedRand(Random* rand, unsigned int seed)
{
    rand->mt[0] = seed & 0xff;        //TAKE THE FIRST ELEMENT OF THE ARRAY MT OF
VARIABLE
    //RAND AND DO BITWISE OPERATIONS TO GET A PARTICULAR SET OF VALUE
    for(rand->index=1; rand->index<STATE_VECTOR_LENGTH; rand->index++)
    {
        rand->mt[rand->index] = (6069 * rand->mt[rand->index-1]) & 0xff;
    }
}

/*
 Creates a new random number generator from a given seed.
*/
Random seedRand(unsigned int seed)
{
    //TAKE PARTICULAR VARIABLE OF STRUCTURE NAMED RAND AND
    //DO THE FOLLOWING OPERATIONS
    Random rand;
    m_seedRand(&rand, seed);  //CALL M_SEED TO DO A PARTICULAR SET OF OPERATIONS FOR
PATTERN
    //GENERATION
    return rand;
}


/**
 * Generates a pseudo-randomly generated long.
 */
unsigned int genRandLong(Random* rand)
{

    unsigned int y;
    static unsigned int mag[2] = {0x0, 0x99};      /* mag[x] = x * 0x99 for x = 0,1
*/
    if(rand->index >= STATE_VECTOR_LENGTH || rand->index < 0)
    {
```

```
        /* generate STATE_VECTOR_LENGTH words at a time */
        int kk;
        if(rand->index >= STATE_VECTOR_LENGTH+1 || rand->index < 0)
        {
            //CHECK IF RAND VARIABELS INDEX FALLS BETWEEN A
            //PARTICULAR RANGE AND DO PARTICULAR BITWSIE OPERATIONS
            //ON   THE ARRAY MT OF VARIABLE RAND
            m_seedRand(rand, 43);
        }
        for(kk=0; kk<STATE_VECTOR_LENGTH-STATE_VECTOR_M; kk++)
        {
            y = (rand->mt[kk] & UPPER_MASK) | (rand->mt[kk+1] & LOWER_MASK);
            rand->mt[kk] = rand->mt[kk+STATE_VECTOR_M] ^ (y >> 1) ^ mag[y & 0x1];
        }                                           //CHECK FOR THE ARRAY MT IF IT
FALLS BETWEEN A SET OF VALUES
        //IF IT DOES THEN WE DO BITWISE OPERATION AT THOSE VALUES
        for(; kk<STATE_VECTOR_LENGTH-1; kk++)
        {
            //CHECK FOR THE ARRAY MT IF IT FALLS BETWEEN A SET OF VALUES
            //IF IT DOES THEN WE DO BITWISE OPERATION AT THOSE VALUES
            y = (rand->mt[kk] & UPPER_MASK) | (rand->mt[kk+1] & LOWER_MASK);
            rand->mt[kk] = rand->mt[kk+(STATE_VECTOR_M-STATE_VECTOR_LENGTH)] ^ \
                           (y >> 1) ^ mag[y & 0x1];
        }                                                   //CHECK FOR THE ARRAY MT IF IT
FALLS BETWEEN A SET OF VALUES
        //IF IT DOES THEN WE DO BITWISE OPERATION AT THOSE VALUES
        y = (rand->mt[STATE_VECTOR_LENGTH-1] & UPPER_MASK) | (rand->mt[0] &
LOWER_MASK);
        rand->mt[STATE_VECTOR_LENGTH-1] = rand->mt[STATE_VECTOR_M-1] ^ (y >> 1) ^
mag[y & 0x1];
        rand->index = 0;
    }
    y = rand->mt[rand->index++];                    //DO THE MERRIAN TWISTER
ALGORITHM ON THE SET OF VALUES OF THE ARRAY
    y ^= (y >> 11);
    y ^= (y << 7) & TEMPERING_MASK_B;
    y ^= (y << 15) & TEMPERING_MASK_C;
    y ^= (y >> 18);
    return y;
}

/**
 * Generates a pseudo-randomly generated double in the range [0..1].
 */
unsigned int genRand(Random* rand)                          //CONVERT THE GENRANDLONG
INTO AN UNSIGNED  FORM AND GET THE 8 BIT
//RANDOM PATTERN
{
    return(genRandLong(rand) / 0xfffff);
}
```

UNITTEST.H

```
/*
 * uCUnit.h
 *
 *  Created on: Oct 21, 2019
 *      Author: Utkarsh Dviwedi
 */

#ifndef UCUNIT_H_
#define UCUNIT_H_



#endif /* UCUNIT_H_ */



/****************************************************************************
 *                                                                          *
 *  uCUnit - A unit testing framework for microcontrollers                  *
 *                                                                          *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                      *
 *                  https://www.ucunit.org                                  *
 *                                                                          *
 *  File        : uCUnit-v1.0.h                                             *
 *  Description : Macros for Unit-Testing                                   *
 *  Author      : Sven Stefan Krauss                                        *
 *  Contact     : www.ucunit.org                                            *
 *                                                                          *
 ****************************************************************************/

/*
 * This file is part of ucUnit.
 *
 * You can redistribute and/or modify it under the terms of the
 * Common Public License as published by IBM Corporation; either
 * version 1.0 of the License, or (at your option) any later version.
 *
 * uCUnit is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * Common Public License for more details.
 *
 * You should have received a copy of the Common Public License
 * along with uCUnit.
 *
 * It may also be available at the following URL:
 *       http://www.opensource.org/licenses/cpl1.0.txt
 *
 * If you cannot obtain a copy of the License, please contact the
 * author.
 */
```

```c
#ifndef UCUNIT_0101_H_
#define UCUNIT_0101_H_

#include "System.h"

/*****************************************************************************/
/****** Customizing area ******/
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_WriteString(msg)
 *
 * @Description: Encapsulates a function which is called for
 *               writing a message string to the host computer.
 *
 * @param msg:   Message which shall be written.
 *
 * @Remarks:     Implement a function to write an integer to a host
 *               computer.
 *
 *               For most microcontrollers a special implementation of
 *               printf is available for writing to a serial
 *               device or network. In some cases you will have
 *               also to implement a putch(char c) function.
 */
#define UCUNIT_WriteString(msg)    System_WriteString(msg)

/**
 * @Macro:       UCUNIT_WriteInt(n)
 *
 * @Description: Encapsulates a function which is called for
 *               writing an integer to the host computer.
 *
 * @param n:     Integer number which shall be written
 *
 * @Remarks:     Implement a function to write an integer to a host
 *               computer.
 *
 *               For most microcontrollers a special implementation of
 *               printf is available for writing to a serial
 *               device or network. In some cases you will have
 *               also to implement a putch(char c) function.
 */
#define UCUNIT_WriteInt(n)    System_WriteInt(n)

/**
 * @Macro:       UCUNIT_Safestate()
 *
 * @Description: Encapsulates a function which is called for
 *               putting the hardware to a safe state.
 *
 * @Remarks:     Implement a function to put your hardware into
 *               a safe state.
 *
```

```
 *               For example, imagine a motor controller
 *               application:
 *                 1. Stop the motor
 *                 2. Power brake
 *                 3. Hold the brake
 *                 4. Switch warning lamp on
 *                 5. Wait for acknowledge
 *                    ...
 *
 */
#define UCUNIT_Safestate()         System_Safestate()

/**
 * @Macro:       UCUNIT_Recover()
 *
 * @Description: Encapsulates a function which is called for
 *               recovering the hardware from a safe state.
 *
 * @Remarks:     Implement a function to recover your hardware from
 *               a safe state.
 *
 *               For example, imagine our motor controller
 *               application:
 *                 1. Acknowledge the error with a key switch
 *                 2. Switch warning lamp off
 *                 3. Reboot
 *                    ...
 *
 */
#define UCUNIT_Recover()           System_Reset()

/**
 * @Macro:       UCUNIT_Init()
 *
 * @Description: Encapsulates a function which is called for
 *               initializing the hardware.
 *
 * @Remarks:     Implement a function to initialize your microcontroller
 *               hardware. You need at least to initialize the
 *               communication device for transmitting your results to
 *               a host computer.
 *
 */
#define UCUNIT_Init()              System_Init()

/**
 * @Macro:       UCUNIT_Shutdown()
 *
 * @Description: Encapsulates a function which is called to
 *               stop the tests if a checklist fails.
 *
 * @Remarks:     Implement a function to stop the execution of the
 *               tests.
 *
 */
```

```
#define UCUNIT_Shutdown()          System_Shutdown()

/**
 * Verbose Mode.
 * UCUNIT_MODE_SILENT: Checks are performed silently.
 * UCUNIT_MODE_NORMAL: Only checks that fail are displayed
 * UCUNIT_MODE_VERBOSE: Passed and failed checks are displayed
 */
//#define UCUNIT_MODE_NORMAL
#define UCUNIT_MODE_VERBOSE

/**
 * Max. number of checkpoints. This may depend on your application
 * or limited by your RAM.
 */
#define UCUNIT_MAX_TRACEPOINTS 16

/*****************************************************************************/
/* **** End of customizing area *****                                        */
/*****************************************************************************/

/*****************************************************************************/
/* Some useful constants                                                     */
/*****************************************************************************/

#define UCUNIT_VERSION "v1.0" /* Version info */

#ifndef NULL
#define NULL (void *)0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

/* Action to take if check fails */
#define UCUNIT_ACTION_WARNING   0 /* Goes through the checks
                                     with message depending on level */
#define UCUNIT_ACTION_SHUTDOWN  1 /* Stops on the end of the checklist
                                     if any check has failed */
#define UCUNIT_ACTION_SAFESTATE 2 /* Goes in safe state if check fails */

/*****************************************************************************/
/* Variables */
/*****************************************************************************/

/* Variables for simple statistics */
static int ucunit_checks_failed = 0; /* Number of failed checks */
static int ucunit_checks_passed = 0; /* Number of passed checks */

static int ucunit_testcases_failed = 0; /* Number of failed test cases */
```

```c
static int ucunit_testcases_passed = 0; /* Number of passed test cases */
static int ucunit_testcases_failed_checks = 0; /* Number of failed checks in a
testcase */
static int ucunit_checklist_failed_checks = 0; /* Number of failed checks in a
checklist */
static int ucunit_action = UCUNIT_ACTION_WARNING; /* Action to take if a check fails
*/
static int ucunit_checkpoints[UCUNIT_MAX_TRACEPOINTS]; /* Max. number of tracepoints
*/
static int ucunit_index = 0; /* Tracepoint index */

/*************************************************************************/
/* Internal (private) Macros                                             */
/*************************************************************************/

/**
 * @Macro:       UCUNIT_DefineToStringHelper(x)
 *
 * @Description: Helper macro for converting a define constant into
 *               a string.
 *
 * @Param x:     Define value to convert.
 *
 * @Remarks:     This macro is used by UCUNIT_DefineToString().
 *
 */
#define UCUNIT_DefineToStringHelper(x)    #x

/**
 * @Macro:       UCUNIT_DefineToString(x)
 *
 * @Description: Converts a define constant into a string.
 *
 * @Param x:     Define value to convert.
 *
 * @Remarks:     This macro uses UCUNIT_DefineToStringHelper().
 *
 */
#define UCUNIT_DefineToString(x)   UCUNIT_DefineToStringHelper(x)

#ifdef UCUNIT_MODE_VERBOSE
/**
 * @Macro:       UCUNIT_WritePassedMsg(msg, args)
 *
 * @Description: Writes a message that check has passed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_VERBOSE.
 *
 */
```

```
#define UCUNIT_WritePassedMsg(msg, args)                        \
    do                                                          \
    {                                                           \
        UCUNIT_WriteString(__FILE__);                           \
        UCUNIT_WriteString(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(": passed:");                        \
        UCUNIT_WriteString(msg);                                \
        UCUNIT_WriteString("(");                                \
        UCUNIT_WriteString(args);                               \
        UCUNIT_WriteString(")\n");                              \
    } while(0)
#else
#define UCUNIT_WritePassedMsg(msg, args)
#endif

#ifdef UCUNIT_MODE_SILENT
#define UCUNIT_WriteFailedMsg(msg, args)
#else
/**
 * @Macro:       UCUNIT_WriteFailedMsg(msg, args)
 *
 * @Description: Writes a message that check has failed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_WriteFailedMsg(msg, args)                        \
    do                                                          \
    {                                                           \
        UCUNIT_WriteString(__FILE__);                           \
        UCUNIT_WriteString(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(": failed:");                        \
        UCUNIT_WriteString(msg);                                \
        UCUNIT_WriteString("(");                                \
        UCUNIT_WriteString(args);                               \
        UCUNIT_WriteString(")\n");                              \
    } while(0)
#endif

/**
 * @Macro:       UCUNIT_FailCheck(msg, args)
 *
 * @Description: Fails a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
```

```
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_FailCheck(msg, args)                 \
    do                                              \
    {                                               \
        if (UCUNIT_ACTION_SAFESTATE==ucunit_action)  \
        {                                           \
            UCUNIT_Safestate();                     \
        }                                           \
        UCUNIT_WriteFailedMsg(msg, args);           \
        ucunit_checks_failed++;                     \
        ucunit_checklist_failed_checks++;           \
    } while(0)

/**
 * @Macro:       UCUNIT_PassCheck(msg, args)
 *
 * @Description: Passes a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *               Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:     This macro is used by UCUNIT_Check(). A message will
 *               only be written if verbose mode is set
 *               to UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_PassCheck(message, args)             \
    do                                              \
    {                                               \
        UCUNIT_WritePassedMsg(message, args);       \
        ucunit_checks_passed++;                     \
    } while(0)

/****************************************************************************/
/* Checklist Macros                                                         */
/****************************************************************************/

/**
 * @Macro:       UCUNIT_ChecklistBegin(action)
 *
 * @Description: Begin of a checklist. You have to tell what action
 *               shall be taken if a check fails.
 *
 * @Param action: Action to take. This can be:
 *                * UCUNIT_ACTION_WARNING:   A warning message will be printed
 *                                           that a check has failed
 *                * UCUNIT_ACTION_SHUTDOWN:  The system will shutdown at
 *                                           the end of the checklist.
 *                * UCUNIT_ACTION_SAFESTATE: The system goes into the safe state
```

```
 *                                          on the first failed check.
 * @Remarks:     A checklist must be finished with UCUNIT_ChecklistEnd()
 *
 */
#define UCUNIT_ChecklistBegin(action)                           \
    do                                                          \
    {                                                           \
        ucunit_action = action;                                 \
        ucunit_checklist_failed_checks = 0;                     \
    } while (0)

/**
 * @Macro:       UCUNIT_ChecklistEnd()
 *
 * @Description: End of a checklist. If the action was UCUNIT_ACTION_SHUTDOWN
 *               the system will shutdown.
 *
 * @Remarks:     A checklist must begin with UCUNIT_ChecklistBegin(action)
 *
 */
#define UCUNIT_ChecklistEnd()                       \
    if (ucunit_checklist_failed_checks!=0)          \
    {                                               \
        UCUNIT_WriteFailedMsg("Checklist","");      \
        if (UCUNIT_ACTION_SHUTDOWN==ucunit_action)  \
        {                                           \
            UCUNIT_Shutdown();                      \
        }                                           \
    }                                               \
    else                                            \
    {                                               \
        UCUNIT_WritePassedMsg("Checklist","");      \
    }

/*****************************************************************************/
/* Check Macros                                                              */
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_Check(condition, msg, args)
 *
 * @Description: Checks a condition and prints a message.
 *
 * @Param msg:   Message to write.
 * @Param args:  Argument list as string
 *
 * @Remarks:     Basic check. This macro is used by all higher level checks.
 *
 */
#define UCUNIT_Check(condition, msg, args)          \
    if ( (condition) ) { UCUNIT_PassCheck(msg, args); } else { UCUNIT_FailCheck(msg,
args); }

/**
 * @Macro:       UCUNIT_CheckIsEqual(expected,actual)
```

```
 *
 * @Description: Checks that actual value equals the expected value.
 *
 * @Param expected: Expected value.
 * @Param actual: Actual value.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsEqual(expected,actual)         \
    UCUNIT_Check( (expected) == (actual), "IsEqual", #expected "," #actual )

/**
 * @Macro:       UCUNIT_CheckIsNull(pointer)
 *
 * @Description: Checks that a pointer is NULL.
 *
 * @Param pointer: Pointer to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsNull(pointer)                  \
    UCUNIT_Check( (pointer) == NULL, "IsNull", #pointer)

/**
 * @Macro:       UCUNIT_CheckIsNotNull(pointer)
 *
 * @Description: Checks that a pointer is not NULL.
 *
 * @Param pointer: Pointer to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsNotNull(pointer)               \
    UCUNIT_Check( (pointer) != NULL, "IsNotNull", #pointer)

/**
 * @Macro:       UCUNIT_CheckIsInRange(value, lower, upper)
 *
 * @Description: Checks if a value is between lower and upper bounds (inclusive)
 *               Mathematical: lower <= value <= upper
 *
 * @Param value: Value to check.
 * @Param lower: Lower bound.
 * @Param upper: Upper bound.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsInRange(value, lower, upper)   \
    UCUNIT_Check( ( (value>=lower) && (value<=upper) ), "IsInRange", #value ","
#lower "," #upper)
```

```
/**
 * @Macro:       UCUNIT_CheckIs8Bit(value)
 *
 * @Description: Checks if a value fits into 8-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs8Bit(value)                       \
    UCUNIT_Check( value==(value & 0xFF), "Is8Bit", #value )

/**
 * @Macro:       UCUNIT_CheckIs16Bit(value)
 *
 * @Description: Checks if a value fits into 16-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs16Bit(value)                      \
    UCUNIT_Check( value==(value & 0xFFFF), "Is16Bit", #value )

/**
 * @Macro:       UCUNIT_CheckIs32Bit(value)
 *
 * @Description: Checks if a value fits into 32-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs32Bit(value)                      \
    UCUNIT_Check( value==(value & 0xFFFFFFFF), "Is32Bit", #value )

/**
 * Checks if bit is set
 */
/**
 * @Macro:       UCUNIT_CheckIsBitSet(value, bitno)
 *
 * @Description: Checks if a bit is set in value.
 *
 * @Param value: Value to check.
 * @Param bitno: Bit number. The least significant bit is 0.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsBitSet(value, bitno) \
    UCUNIT_Check( (1==(((value)>>(bitno)) & 0x01) ), "IsBitSet", #value "," #bitno)
```

```
/**
 * @Macro:       UCUNIT_CheckIsBitClear(value, bitno)
 *
 * @Description: Checks if a bit is not set in value.
 *
 * @Param value: Value to check.
 * @Param bitno: Bit number. The least significant bit is 0.
 *
 * @Remarks:     This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsBitClear(value, bitno) \
    UCUNIT_Check( (0==(((value)>>(bitno)) & 0x01) ), "IsBitClear", #value "," #bitno)

/*****************************************************************************/
/* Testcases */
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_TestcaseBegin(name)
 *
 * @Description: Marks the beginning of a test case and resets
 *               the test case statistic.
 *
 * @Param name:  Name of the test case.
 *
 * @Remarks:     This macro uses UCUNIT_WriteString(msg) to print the name.
 *
 */
#define UCUNIT_TestcaseBegin(name)                                      \
    do                                                                  \
    {                                                                   \
        UCUNIT_WriteString("\n--------------------------------------\n"); \
        UCUNIT_WriteString(name);                                       \
        UCUNIT_WriteString("\n--------------------------------------\n"); \
        ucunit_testcases_failed_checks = ucunit_checks_failed;          \
    }                                                                   \
    while(0)

/**
 * @Macro:       UCUNIT_TestcaseEnd()
 *
 * @Description: Marks the end of a test case and calculates
 *               the test case statistics.
 *
 * @Remarks:     This macro uses UCUNIT_WriteString(msg) to print the result.
 *
 */
#define UCUNIT_TestcaseEnd()                                            \
    do                                                                  \
    {                                                                   \
        UCUNIT_WriteString("======================================\n");  \
        if( 0==(ucunit_testcases_failed_checks - ucunit_checks_failed) ) \
        {                                                               \
```

```
            UCUNIT_WriteString("Testcase passed.\n");               \
            ucunit_testcases_passed++;                              \
        }                                                           \
        else                                                        \
        {                                                           \
            UCUNIT_WriteFailedMsg("EndTestcase","");                \
            ucunit_testcases_failed++;                              \
        }                                                           \
        UCUNIT_WriteString("====================================\n"); \
    }                                                               \
    while(0)

/*****************************************************************************/
/* Support for code coverage */
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_Tracepoint(index)
 *
 * @Description: Marks a trace point.
 *               If a trace point is executed, its coverage state switches
 *               from 0 to the line number.
  *              If a trace point was never executed, the state
 *               remains 0.
 *
 * @Param index: Index of the tracepoint.
 *
 * @Remarks:     This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_Tracepoint(index)                        \
    if(index<UCUNIT_MAX_TRACEPOINTS)                    \
    {                                                   \
        ucunit_checkpoints[index] = __LINE__;           \
    }                                                   \
    else                                                \
    {                                                   \
        UCUNIT_WriteFailedMsg("Tracepoint index", #index);     \
    }

/**
 * @Macro:       UCUNIT_ResetTracepointCoverage()
 *
 * @Description: Resets the trace point coverage state to 0.
 *
 * @Param index: Index of the trace point.
 *
 * @Remarks:     This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_ResetTracepointCoverage()                    \
    for (ucunit_index=0; ucunit_index<UCUNIT_MAX_TRACEPOINTS; ucunit_index++) \
    {                                                       \
        ucunit_checkpoints[ucunit_index]=0;                 \
    }
```

```
/**
 * @Macro:       UCUNIT_CheckTracepointCoverage(index)
 *
 * @Description: Checks if a trace point was covered.
 *
 * @Param index: Index of the trace point.
 *
 * @Remarks:     This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_CheckTracepointCoverage(index)    \
    UCUNIT_Check( (ucunit_checkpoints[index]!=0), "TracepointCoverage", #index);

/*****************************************************************************/
/* Testsuite Summary                                                         */
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_WriteSummary()
 *
 * @Description: Writes the test suite summary.
 *
 * @Remarks:     This macro uses UCUNIT_WriteString(msg) and
 *               UCUNIT_WriteInt(n) to write the summary.
 *
 */
#define UCUNIT_WriteSummary()                                             \
{                                                                         \
    UCUNIT_WriteString("\n**************************************");    \
    UCUNIT_WriteString("\nTestcases: failed: ");                          \
    UCUNIT_WriteInt(ucunit_testcases_failed);                             \
    UCUNIT_WriteString("\n           passed: ");                          \
    UCUNIT_WriteInt(ucunit_testcases_passed);                             \
    UCUNIT_WriteString("\nChecks:    failed: ");                          \
    UCUNIT_WriteInt(ucunit_checks_failed);                                \
    UCUNIT_WriteString("\n           passed: ");                          \
    UCUNIT_WriteInt(ucunit_checks_passed);                                \
    UCUNIT_WriteString("\n**************************************\n"); \
}

#endif /*UCUNIT_H_*/
```

SYSTEM.C

```
/*
 * system.c
 *
```

```
 *  Created on: Oct 21, 2019
 *      Author: Utkarsh Dviwedi
 */
/*****************************************************************************
 *                                                                           *
 *  uCUnit - A unit testing framework for microcontrollers                   *
 *                                                                           *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                       *
 *                  https://www.ucunit.org                                   *
 *                                                                           *
 *  File        : System.c                                                   *
 *  Description : System dependent functions used by uCUnit.                 *
 *                This file runs with arm-elf-run                            *
 *  Author      : Sven Stefan Krauss                                         *
 *  Contact     : www.ucunit.org                                             *
 *                                                                           *
 *****************************************************************************/

/*
 * This file is part of ucUnit.
 *
 * You can redistribute and/or modify it under the terms of the
 * Common Public License as published by IBM Corporation; either
 * version 1.0 of the License, or (at your option) any later version.
 *
 * uCUnit is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * Common Public License for more details.
 *
 * You should have received a copy of the Common Public License
 * along with uCUnit.
 *
 * It may also be available at the following URL:
 *       http://www.opensource.org/licenses/cpl1.0.txt
 *
 * If you cannot obtain a copy of the License, please contact the
 * author.
 */
#include <stdio.h>
#include <stdlib.h>
#include "System.h"
#include "fsl_debug_console.h"

void System_exit(int val) {
      for(;;) {
            /* we don't do a shutdown with exit(), as this is will affect debugging.
             * Instead, we stay here in an endless loop.
             */
            __asm("nop"); /* burning some CPU cycles here */
      }
      // exit(val);
}

/* Stub: Initialize your hardware here */
```

```c
void System_Init(void)
{

	PRINTF("Init of hardware finished.\n");
}

/* Stub: Shutdown your hardware here */
void System_Shutdown(void)
{

	/* asm("\tSTOP"); */
	PRINTF("System shutdown.\n");
	System_exit(0);
}

/* Stub: Recover the system */
void System_Recover(void)
{
	/* Stub: Recover the hardware */
	/* asm("\tRESET"); */
	PRINTF("System reset.\n");
	System_exit(0);
}

/* Stub: Put system in a safe state */
void System_Safestate(void)
{
	/* Disable all port pins */
	/* PORTA = 0x0000; */
	/* PORTB = 0x0000; */
	/* PORTC = 0x0000; */

	/* Disable interrupts */
	/* DIE(); */

	/* Put processor into idle state */
	/* asm("\tIDLE"); */
	PRINTF("System safe state.\n");
	System_exit(0);
}

/* Stub: Transmit a string to the host/debugger/simulator */
void System_WriteString(char * msg)
{
	PRINTF(msg);
}

void System_WriteInt(int n)
{
	PRINTF("%d", n);
}
```