Project 3 PES - Readme

Team Members: Sharan Arumugam Abhijeet Srivastava

Execution Instructions

Using custom make file with 4 build targets: -FB_DEBUG : Memory test for the KL25z with logger disabled -FB_LOG : Memory test for the KL25z with logger enabled -PC_DEBUG : Memory test for the PC with logger disabled -PC_LOG : Memory test for the KL25z with logger disabled Using make file, run

- make -r -j8 all MODE=FB_Debug *for FB_DEBUG*
- make -r -j8 all MODE=FB_Log *for FB_LOG*
- make -r -j8 all MODE=PC_Debug *for PC_DEBUG*
- make -r -j8 all MODE=PC_Log *for PC_LOG* Makefile runs ***arm-none-eabi-gcc*** as compiler for FB routines and ***gcc*** for PC

Files in Repo

- MCU XPRESSO Project Directory
    - Board Folder- Written routines for memtest, led,logger and pattern generation in memory.h, led_board.h and loggertest.h and pattern_gen.h and contains other default folder
    - Source Folder - Contains main routine- Project3.c
    - Default folders with no change -CMSIS,startup,drivers,utilities
    - Debug - Contains .o files after compilation and also contains .axf binary and .exe files
- Readme
- UML Diagrams
- PDF contains make file and code and readme

Issues faced

- Had to find a random generator which could generate same pattern with seed. Linear congruential generator fit the bill.
- Had issues on understanding what each described function had to do. Got answers from slack for most problems. Example: offset function had its parameters changed to include both base address and offset value.
- Learned to declare array as static if returning it in a function.
- Couldn't get test cases to perform properly.

References

- Slack channel for clearing up questions on function definitions
- https://mcuoneclipse.com/2017/07/22/tutorial-makefile-projects-with-eclipse/
- Makefile from Project 2 https://github.com/abhijeet-sharan/pes_fun/blob/master/Project2/makefile
- https://en.wikipedia.org/wiki/Linear_congruential_generator

Code:

Memory.h

```
#include<stdio.h>
#ifdef KL25Z
#include<board.h>
#endif
#include<stdlib.h>
#include "pattern_gen.h"
#include "led_board.h"
/* PES PROJECT 3- SHARAN ARUMUGAM AND ABIJEET
Memory Test Suite*
*/
typedef enum mem_status
{
SUCCESS = 0, // no error
FAILED // failure case
} mem_status;

uint32_t * allocate_words(size_t length);//tries to malloc a memory block of required length if not possible returns null pointer
void free_words(uint32_t* src); //fress allocation pointed to by pointer
uint8_t * display_memory(uint32_t * loc,size_t length); //returns another pointer containing the contents at loc and of size length
mem_status write_memory(uint32_t * loc, uint16_t value);//writes  value at memory address pointed by loc
```

```
mem_status invert_block(uint32_t*loc,size_t length);// toggles all the bits at memory address - loc. length gives number of bytes to toggle
mem_status write_pattern(uint32_t * loc, size_t length, int8_t seed); //writes random pattern at memory address
uint32_t * verify_pattern(uint32_t * loc, size_t length, int8_t seed);// verifies whether pattern at address matches random pattern and returns list of addresses where pattern didnt match
uint32_t *get_address (uint32_t *base_addr, uint32_t offset); //retunrs a pointer which points to memory location at base_addr+offset
```

memory.c

```
#include "memory.h"
#include "led_board.h"


uint32_t * allocate_words(size_t length)
{
LED_PROCESS();//switches blue led on
uint32_t *p = NULL;
 p=malloc(length * sizeof(uint8_t));//malloc a memeory block depending on input number of bytes
if(p==NULL)//checking if malloc was successfull
{
    printf("Memory not allocated\n\r");
    LED_FAIL();//switches red led on
}
else
{
    printf("Memory allocated = %p\n",p);
    LED_PASS();//switches green led on
    // for now let the memory be
}
return p;
}
void free_words(uint32_t* src)
{
      LED_PROCESS();
if(NULL==src)
    {
    printf("Error: Cannot free unallocated memory!\n");//if pointer is already at null doesnt free
    LED_FAIL();//switches on red led
    }
else
    {
free(src);//freeing block of memory
src=NULL;
LED_PASS();
    }
}
uint8_t * display_memory(uint32_t *loc,size_t length)
```

```
{
	LED_PROCESS();
	volatile uint8_t *ptr=NULL;
		ptr=(uint8_t *)loc;//casting loc to uint8_t type pointer
    static uint8_t *arr=NULL;//as returning arr defined as static
    arr=malloc(length*sizeof(uint8_t));//setting a block depending on size of bits
	  for(int i=0;i<length;i++)
	  {
		arr[i]=*ptr;//getting values at memory address loc and assigning to arr
block
		ptr++;
	  }
return arr;//returning pointer

}
mem_status write_memory(uint32_t *loc,uint16_t value)
{LED_PROCESS();
    uint16_t *ptr=NULL;
    ptr=(uint16_t *)loc;//casting to uint16 pointer and then writing value
    *ptr = value;
    return SUCCESS;
}

mem_status invert_block(uint32_t * loc, size_t length)
{LED_PROCESS();
	uint8_t *ptr=NULL;//casting to uint8_t pointer
	ptr=(uint8_t*)loc;
	for(int i=0;i<length;i++)
	{
		ptr[i]^=0xFF; //toggling all bits in that specific byte
	}
	return SUCCESS;
}

mem_status write_pattern(uint32_t * loc, size_t length, int8_t seed)
{ LED_PROCESS();
	uint8_t *patternwrite=NULL;//random patterns are of type int8_t
	patternwrite=(uint8_t *)loc;//casting to uint8 pointer


	gen_pattern(patternwrite,length,seed);//calls pattern generating function

	return SUCCESS;
	}

 uint32_t * verify_pattern(uint32_t * loc, size_t length, int8_t seed)
{LED_PROCESS();
	uint8_t storepattern[length];
	gen_pattern(storepattern,length,seed);//creates a array containg standard
random pattern
	uint8_t *testptr=NULL;
	testptr=(uint8_t *)loc;
	  uint32_t *errorlist=NULL;
	  errorlist=malloc(length*sizeof(uint32_t));//list containg addresses where
values dont match
```

```
	for(int i=0;i<length;i++) //goes through memory block and tests with pattern.
if not matching stores address in memory block else sets as 0
	{
		if((*(testptr+i))!=storepattern[i])
		errorlist[i]=(uint32_t)(testptr+i);
		else
			errorlist[i]=0;

	}
	return errorlist;
}
uint32_t *get_address (uint32_t *base_addr, uint32_t offset) //returns uint32_t* type
pointer to base address +offset
{
LED_PROCESS();
uint8_t *offset_address=NULL;
offset_address=(uint8_t *)base_addr;//casts base adress to uint8 ptr
while(offset!=0)
{
	offset_address++;offset--;//uses poiner math to increment address by offset
value
}
uint32_t * return_offset_address=NULL;
return_offset_address=(uint32_t *)offset_address;//casts pointer uint32 type to match
return value.
return return_offset_address;
}
```

Led_board.h

```
//PES Project3 - Sharan Arumugam and Abhijeet ( LEd routines)
#include<stdio.h>
#include<stdint.h>
#ifdef KL25Z //include files only when platform is kl25z
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "MKL25Z4.h"
#endif
//all functions dependent on platform
void LED_Initialise(); //depending on platform led initialises or does nothing
void ALLLED_OFF();//switches all led off
void LED_PASS();//switches on green led and all others off
void LED_PROCESS();//switches on blue led and others off
void LED_FAIL();//switches on red led and others off
void DELAY();//delay to create halt in order to observe led status
```

led_board.c

```c
#include "led_board.h"
#ifdef KL25Z
#include "board.h"
#endif
void delay()
{
volatile uint64_t i=0;
while(i!=390000) //holds execution till delay is achieved
{ __asm("NOP");
i++;
  }
}
//only for kl25z
#ifdef KL25Z
void LED_Initialise()
{
	LED_BLUE_INIT(1);
	LED_RED_INIT(1);
	LED_GREEN_INIT(1);
}
void ALLLED_OFF()
{
	LED_BLUE_OFF();
	LED_RED_OFF();
	LED_GREEN_OFF();
}
void LED_PASS()
{
ALLLED_OFF();
LED_GREEN_ON();
delay();
}
void LED_PROCESS()
{
ALLLED_OFF();
LED_BLUE_ON();
delay();
}
void LED_FAIL()
{
	ALLLED_OFF();
	LED_RED_ON();
	delay();
}
#endif
//when platform is pc
#ifdef PC
void LED_Initialise()//does nothing
{
	;
}
void ALLLED_OFF()//does nothing
{
	;
}
```

```
void LED_PASS()//prints green led is on
{
printf("green led on\n");
delay();
}
void LED_PROCESS()//prints blue led on //delay too less as clock on pc is much faster
{
printf("blue led on\n");
delay();
}
void LED_FAIL()
{
	printf("red led on\n");
	delay();
}
#endif
```

Loggertest.c

```
#include"loggertest.h"

uint8_t Log_enable()
{
 return 1;
}

uint8_t Log_disable()
{
return 0;
}



void Log_data(uint32_t* pointer,uint8_t length)
{

	uint8_t* tempp = NULL;
	tempp=(uint8_t*)pointer;//to print out characters byte by byte
	for(uint8_t i = 0;i<length;i++)
	{
		printf("address = %p ",(tempp+i));
		printf("data= %d \n",*(tempp+i));

	}
	printf("\n");
}


int Log_status(uint8_t x)
{
if(x)
return 1;
else
```

```
return 0;
}


void log_string(char *letter)
{
printf("%s",letter);
printf("\n");
}

void log_integer(uint32_t integer)
{
printf("%x\n",integer);
}
```

Loggertest.h

```
//PES Project 3- for printing out log data and address while execution By- Sharan and Abhijeet
#ifdef KL25Z
#include "fsl_debug_console.h"
#endif
#include <stdio.h>
#include<stdint.h>
uint8_t Log_enable();//enables logger
uint8_t Log_disable();//disables logger
void Log_data(uint32_t* pointer,uint8_t length);//prints data and address of memory block
int Log_status(uint8_t x);//checks if logger is enabled
void log_string(char *letter);//prints string
void log_integer(uint32_t integer);//prints integer
```

pattern_gen.h

```
#include<stdlib.h>
#include<stdint.h>
void gen_pattern(uint8_t * pattern, size_t length, int8_t seed);
/* PES PROJECT 3- SHARAN ARUMUGAM AND ABIJEET
function takes a seed and generates a random pattern 'length' bytes long using a linear congruential generator
returns pointer/array containing the pattern at pointer pattern
*/
```

Patter_gen.c

```
#include "pattern_gen.h"

void gen_pattern(uint8_t *pattern, size_t length, int8_t seed)// linear congruential generator
//https://en.wikipedia.org/wiki/Linear_congruential_ghttps://en.wikipedia.org/wiki/Linear_congruential_generatorenerator
{
               uint16_t a=34509;//random constant
               uint32_t m=1653656;//modulus
               uint16_t c=664;//constant
               *pattern=seed;//first value is seed itself
for(int i=1;i<length;i++)
               {
                 *(pattern+i)=(a*(*(pattern+(i-1)))+c)%m;//random value depends on previous value also
               }

}
```

Unittest.c

Performed unit testing in a separate project where memory.h and pattern_gen.h were included. Functions from memory.h were tested using test functions for testing two individually.

```
void Testallocate(uint32_t* pointer) {
    uint32_t* test = NULL;
    UCUNIT_Init();
    UCUNIT_TestcaseBegin("Memory Allocate");
    test = allocate_words(16);
    UCUNIT_CheckIsEqual(test,pointer);
    UCUNIT_CheckIsInRange(*test,536866816,536870911);
    UCUNIT_CheckIsInRange(*test,536870912,536883199);
    UCUNIT_TestcaseEnd();

    UCUNIT_WriteSummary();
    }

void Testfree(uint32_t* pointer) {
    //uint32_t* test = NULL;
    UCUNIT_Init();
    UCUNIT_TestcaseBegin("Memory Free");
    UCUNIT_CheckIsEqual(pointer,NULL);
    //UCUNIT_CheckIsInRange(*test,536866816,536870911);
    //UCUNIT_CheckIsInRange(*test,536870912,536883199);
    UCUNIT_TestcaseEnd();

    UCUNIT_WriteSummary();
    UCUNIT_Shutdown();


    }


int main(void) {


    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
          /* Init FSL debug console. */
    BOARD_InitDebugConsole();

    //PRINTF("ucUnit Testing");
    printf("ucUnit Testing");
   memory = allocate_words(16);
   Testallocate(memory);
   free_words(memory);
   Testfree(memory);
    return 0 ;


}
```