**********main.c**********

```
/*
 * @file main.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/17/2019
 *
 * There are two modes of operations to choose from in the main routine
 * 1. FB_RUN: When this target is built the routine for the freedom board KL25Z
gets built.
 *                   This version prints the outputs on MCUxpresso terminal using
UART
 *                   LED indications are given on the freedom board as per the
test results
 * 2. PC_RUN: When this target is built the routine for the development
environment i.e. windows/linux gets built
 *                   This version prints the output on MCUxpresso console
 *                   Test indications are given by printing X LED ON the console
 *
 *This file calls all the routines of the program in the main function
 *
 */




#include "main.h"

// This version of the code is for operating in KL25Z freedom board
#ifdef FB_RUN


#include "Logger.h"
#include "Memory_Functions.h"
#include "LED_Blink.h"
#include "Delay_Function.h"

extern const int NUMB_OF_INVERTING_BYTES;//Number of Bytes to be Inverted
extern size_t Inverting_length;//length for which we invert the bytes

extern const int NUMB_OF_BYTES;
extern size_t length;//size of the memory in bytes.

//#define NULL ((void*)0)
extern uint8_t array5[16];
extern uint8_t array4[16];
extern uint8_t array3[16];
extern uint8_t array2[16];
extern uint32_t *ptr;//pointer for memory allocation function.

uint32_t milli_sec_val=0;

// main function calling all the function routines
```

```c
int main(void)
{

BOARD_InitBootPins();
BOARD_InitBootClocks();
BOARD_InitBootPeripherals();
BOARD_InitDebugConsole();
BOARD_InitPins();
BOARD_BootClockRUN();

Log_Enabled();
GREEN_LED_INIT();
RED_LED_INIT();
BLUE_LED_INIT();


BLUE_LED_ON();


//calling the memory allocation function.
allocate_words();

//USER INPUT FOR OFFSET VALUE
//PRINTF("\nENTER THE OFFSET VALUE:");
Log_String("\n\rENTER THE OFFSET VALUE");

int offset;

// Store user input to offset variable
SCANF("\n\r%d",&offset);

//taking the address of allocated memory
get_address(ptr,offset);

//USER INPUT FOR SEED VALUE
Log_String("\n\rENTER THE SEED VALUE:");

uint32_t seed;

// Store user input to seed variable
SCANF("\n\r%d",&seed);


//writing the randomly generated pattern to the allocated memory.
write_pattern(ptr, length,seed);
//comparing the newly generated array with the patter from pattern generation
function
verify_pattern(ptr, length, seed);
display_memory();

Log_String("\n\n\rpattern after writing the value");
//writing the value to given address.
Write_Value_To_Memory(ptr,offset, 0xFFEE);
//comparing the newly generated array with the patter from write function
verify_pattern(ptr, length, seed);

Log_String("\n\n\rpattern generated with the same seed value");
//writing the randomly generated pattern to the allocated memory.
write_pattern(ptr, length,seed);
```

```
//comparing the newly generated array with the patter from write function
display_memory();
verify_pattern(ptr, length, seed);

Log_String("\n\n\rENTER THE OFFSET VALUE FOR INVERTING BYTES FUNCTION:");
int offset2;
SCANF("\n\r%d",&offset2);
invert_block(ptr,offset2, Inverting_length);
//comparing the newly generated array with the patter from inverse function
verify_pattern(ptr, length, seed);

Log_String("\n\n\rpattern after re-inverting the blocks");
ReInvert_block(ptr,offset2, Inverting_length);
verify_pattern(ptr, length, seed);


//calling the free memory function
free_words(ptr);
Log_String("\n\n\rallocated memory FREE");
GREEN_LED_ON();
return 0;

}

#endif



//This version of the program is for running Development environment mode of
operation
#ifdef PC_RUN
#include "Logger.h"
#include "Memory_Functions.h"
#include "LED_Blink.h"
#include "Delay_Function.h"

extern const int NUMB_OF_INVERTING_BYTES;//Number of Bytes to be Inverted
extern size_t Inverting_length;//length for which we invert the bytes

extern const int NUMB_OF_BYTES;
extern size_t length;//size of the memory in bytes.

//#define NULL ((void*)0)
extern uint8_t array5[16];
extern uint8_t array4[16];
extern uint8_t array3[16];
extern uint8_t array2[16];
extern uint32_t *ptr;//pointer for memory allocation function.

uint32_t milli_sec_val=0;

// main functions calling all the function routines
int main(void)
{

BOARD_InitBootPins();
BOARD_InitBootClocks();
BOARD_InitBootPeripherals();
BOARD_InitDebugConsole();
```

```
BOARD_InitPins();
BOARD_BootClockRUN();


BLUE_LED_ON();

//calling the memory allocation function.
allocate_words();

//USER INPUT FOR OFFSET VALUE
//PRINTF("\nENTER THE OFFSET VALUE:");
printf("\n\rENTER THE OFFSET VALUE");
int offset;
//Stores the user input to offset variable
scanf("%d",&offset);
//taking the address of allocated memory
get_address(ptr,offset);

//USER INPUT FOR SEED VALUE
printf("\n\rENTER THE SEED VALUE:");
uint32_t seed;
scanf("%u",&seed);



//writing the randomly generated pattern to the allocated memory.
write_pattern(ptr, length,seed);
//comparing the newly generated array with the patter from pattern generation
function
verify_pattern(ptr, length, seed);
display_memory();

printf("\n\n\rpattern after writing the value");
//writing the value to given address.
Write_Value_To_Memory(ptr,offset, 0xFFEE);
//comparing the newly generated array with the patter from write function
verify_pattern(ptr, length, seed);

printf("\n\n\rpattern generated with the same seed value");
//writing the randomly generated pattern to the allocated memory.
write_pattern(ptr, length,seed);
//comparing the newly generated array with the patter from write function
display_memory();
verify_pattern(ptr, length, seed);

printf("\n\n\rENTER THE OFFSET VALUE FOR INVERTING BYTES FUNCTION:");
int offset2;
//Stores the user input to offset2 variable
scanf("%d",&offset2);

invert_block(ptr,offset2, Inverting_length);
//comparing the newly generated array with the patter from inverse function
verify_pattern(ptr, length, seed);
printf("\n\n\rpattern after re-inverting the blocks");
ReInvert_block(ptr,offset2, Inverting_length);
verify_pattern(ptr, length, seed);


//calling the free memory function
```

```
free_words(ptr);
printf("\n\n\rallocated memory FREE");
GREEN_LED_ON();
return 0;

}
#endif
```

**********main.h**********

```
/*
 * @file main.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/17/2019
 *
 * There are two modes of operations to choose from in the main routine
 * 1. FB_RUN: When this target is built the routine for the freedom board KL25Z
gets built.
 *                   This version prints the outputs on MCUxpresso terminal using
UART
 *                   LED indications are given on the freedom board as per the
test results
 * 2. PC_RUN: When this target is built the routine for the development
environment i.e. windows/linux gets built
 *                   This version prints the output on MCUxpresso console
 *                   Test indications are given by printing X LED ON the console
 *
 *This file contains all header files required by the main.c file
 */


#include <stdio.h>
#include <stdint.h>//for using unit32_t data type.
#include <stdlib.h>//for using malloc() function.
#include <math.h>
#include <limits.h>

#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_gpio.h"
#include "fsl_uart.h"
#include "time.h"

//#define PC_RUN
#define FB_RUN
```

**********Memory_Functions.c**********

```
/*
 * @file Memory_Functions.c
 * author: Kushagra Pandey & Vaidehi Salway
 * Date:10/20/2019
 *
 * There are two modes of operations to choose from in the main routine
 * 1. FB_RUN: When this target is built the routine for the freedom board KL25Z
gets built.
 *                   This version prints the outputs on MCUxpresso terminal using
UART
 *                   LED indications are given on the freedom board as per the
test results
 * 2. PC_RUN: When this target is built the routine for the development
environment i.e. windows/linux gets built
 *                   This version prints the output on MCUxpresso console
 *                   Test indications are given by printing X LED ON the console
 *
 *This file contains the functions related to memory test cycle
 *
 */



#include "memory_functions.h"
//This version is for running the code on freedom board
#ifdef FB_RUN


#include "Logger.h"
#include "LED_Blink.h"
#include "Delay_Function.h"

//#define NULL ((void*)0)
uint8_t array5[16]={0};
uint8_t array4[16]={0};
uint8_t array3[16]={0};
uint8_t array2[16]={0};
uint32_t *ptr=NULL;//pointer for memory allocation function.


//size_t MAX=16;//size of the memory in bytes.
const int NUMB_OF_BYTES=16;
size_t length=NUMB_OF_BYTES*(sizeof(int));//size of the memory in bytes.

const int NUMB_OF_INVERTING_BYTES=4;//Number of Bytes to be Inverted
size_t Inverting_length=NUMB_OF_INVERTING_BYTES*(sizeof(uint8_t));//length for
which we invert the bytes
int i=0;



//Dynamic Memory Allocaton Function
uint32_t *allocate_words()
```

```
{
    ptr=(uint32_t*)malloc(length);//dynamic memory allocation using malloc().
    if(length>SIZE_MAX)//checking for maximum allowed memory.
      {
         ptr=0;
      }

      return ptr;
}

//freeing the pointer to previously allocated memory.
void free_words(uint32_t *ptr)
{
       if(ptr==NULL)
       {
              Log_String("memory not allocated to free");

       }

       free(ptr);
}

//function to get the address of the location
uint32_t * get_address(uint32_t *base_address,int offset)
{
       uint8_t *offset_ptr2=NULL;
offset_ptr2=(uint8_t*)(base_address);

        for(i=0;i<offset;i++)
        {
              offset_ptr2++;
        }

        uint32_t *physical_address=NULL;
        physical_address=(uint32_t*)(offset_ptr2);
        Log_Data((unsigned int)physical_address);
return 0;
}


// Code for random generator
// Reference:   https://rosettacode.org/wiki/Linear_congruential_generator
void write_pattern(uint32_t * loc, size_t length,uint32_t seed)
{

  //PRINTF("\nrand max is %u\n", RANDOM_MAX);
       Log_String("rand max is");
       Log_Data(RANDOM_MAX);

       for (int i = 0; i < NUMB_OF_BYTES; i++)
   {
    seed = (seed * 13124245 + 12345 ) & RANDOM_MAX;

    array2[i]=seed;
    //writing the random number array to the allocated memory.
    loc[i]=array2[i];
    //PRINTF("%x\n",(unsigned int)*(ptr+i));
    }
}
```

```
//return the pointer for allocated memory and display its value.
uint32_t display_memory()
{
       for (int i = 0; i < NUMB_OF_BYTES; i++)
       {
         ptr[i]=array2[i];
       //PRINTF("\npattern_generated is:%x\n",(unsigned int)*(ptr+i));
       Log_Data((unsigned int)*(ptr+i));
       }
   return *ptr;
}

//Function for Writing value at a particular Memory Location.
//*loc = the memory location at which we have to write the value.
void Write_Value_To_Memory(uint32_t * base_address,int offset, uint32_t value)
{
//calculating the offset for the address in the allocated memory.
       uint32_t *offset_ptr=NULL;
       offset_ptr=base_address;
       for(i=0;i<offset;i++)
       {
   //incrementing it to the offset value required.
       offset_ptr++;
       }
    uint32_t *physical_address=NULL;
    //calculating the physical address with the help of offset pointer value
above.
    physical_address=(uint32_t*)(offset_ptr);

    *physical_address=value;
    Log_String("\nPattern after modifying the value at the given location is");

    PRINTF("\n:%x\n",(unsigned int)(*physical_address));
      uint32_t *printer=NULL;
      printer=ptr;
    for(i=0;i<NUMB_OF_BYTES;i++)
    {
    //PRINTF("\n%x", *(printer+i));
    Log_Data(*(printer+i));
    }
}

//Inverting each byte of the array of random numbers allocated.
void invert_block(uint32_t * base_address,int offset2, size_t Inverting_length)
{

  //calculating the offset for the address in the allocated memory.
       uint32_t *offset_ptr3=NULL;
       offset_ptr3=base_address;
       for(i=0;i<offset2;i++)
       {
              //incrementing it to the offset value required.
              offset_ptr3++;
       }
    uint32_t *physical_address=NULL;
    //calculating the physical address with the help of offset pointer value
above.
    physical_address=(uint32_t*)(offset_ptr3);
```

```
    Log_String("\n\ninverted bytes are:");

    for(i=0;i<NUMB_OF_INVERTING_BYTES;i++)
    {
      //EXORing the array allocated to inverse the bits in it.
      array4[i]=(array2[i]^(0xff));
      base_address[i]=array4[i];

    }

    for(i=0;i<NUMB_OF_BYTES;i++)
    {
      // base_address[i]=array2[i];
      //PRINTF("\n%x",*(base_address+i));
      Log_Data(*(base_address+i));
    }
}

void ReInvert_block(uint32_t * base_address,int offset2, size_t Inverting_length)
{

       //calculating the offset for the address in the allocated memory.
        uint32_t *offset_ptr4=NULL;
        offset_ptr4=base_address;
              for(i=0;i<offset2;i++)
              {
          //incrementing it to the offset value required.
                    offset_ptr4++;
              }
           uint32_t *physical_address=NULL;
           //calculating the physical address with the help of offset pointer
value above.
           physical_address=(uint32_t*)(offset_ptr4);


           for(i=0;i<NUMB_OF_INVERTING_BYTES;i++)
           {
             //EXORing the array allocated to inverse the bits in it.
             array5[i]=(array4[i]^(0xff));
             base_address[i]=array5[i];

           }

           for(i=0;i<NUMB_OF_BYTES;i++)
           {
             // base_address[i]=array2[i];
             //PRINTF("\n%x",*(base_address+i));
             Log_Data(*(base_address+i));
           }


}


//verifying if the original pattern and different patterns generated match.
uint32_t * verify_pattern(uint32_t * loc, size_t length, uint32_t seed)
{
```

```
 //generating the random pattern again with the same seed value to verify
different patterns.
       for (int i = 0; i < NUMB_OF_BYTES; i++)
       {
              seed = (seed * 13124245 + 12345 ) & RANDOM_MAX;
              array3[i]=seed;
       }

    for(i=0;i<NUMB_OF_BYTES;i++)
    {
       if(loc[i]==array3[i])
       {
              BLUE_LED_ON();
              Delay_Time(200);
              BLUE_LED_OFF();
              Delay_Time(100);

      Log_String("\nallocated memories are equal");

    }

    else if(loc[i]!=array3[i])
    {
              RED_LED_ON();
              Delay_Time(200);
              RED_LED_OFF();
              Delay_Time(100);

      Log_String("\nerror generated at address");
      //PRINTF("\n%x",(unsigned int)&loc[i]);
      Log_Data((unsigned int)&loc[i]);
    }

    }
  return 0;
}
#endif


//This version is for running the code on development system i.e windows or Linux
#ifdef PC_RUN


//#include "memory_functions.h"
#include "Logger.h"
#include "LED_Blink.h"
#include "Delay_Function.h"

//#define NULL ((void*)0)
uint8_t array5[16]={0};
uint8_t array4[16]={0};
uint8_t array3[16]={0};
uint8_t array2[16]={0};
uint32_t *ptr=NULL;//pointer for memory allocation function.


//size_t MAX=16;//size of the memory in bytes.
const int NUMB_OF_BYTES=16;
size_t length=NUMB_OF_BYTES*(sizeof(int));//size of the memory in bytes.
```

```
const int NUMB_OF_INVERTING_BYTES=4;//Number of Bytes to be Inverted
size_t Inverting_length=NUMB_OF_INVERTING_BYTES*(sizeof(uint8_t));//length for
which we invert the bytes
int i=0;



//Dynamic Memory Allocaton Function
uint32_t *allocate_words()
{
    ptr=(uint32_t*)malloc(length);//dynamic memory allocation using malloc().
    if(length>SIZE_MAX)//checking for maximum allowed memory.
      {
         ptr=0;
      }

      return ptr;
}

//freeing the pointer to previously allocated memory.
void free_words(uint32_t *ptr)
{
      if(ptr==NULL)
      {
            printf("memory not allocated to free");
      }
            free(ptr);
}


uint32_t * get_address(uint32_t *base_address,int offset)
{
      uint8_t *offset_ptr2=NULL;
      offset_ptr2=(uint8_t*)(base_address);

      for(i=0;i<offset;i++)
      {
            offset_ptr2++;
      }
      uint32_t *physical_address=NULL;
      physical_address=(uint32_t*)(offset_ptr2);
      printf("\n\n\r physical address is:%x",(unsigned int)physical_address);
      return 0;
}


// Code for random generator
// Reference:   https://rosettacode.org/wiki/Linear_congruential_generator
void write_pattern(uint32_t * loc, size_t length,uint32_t seed)
{

      printf("\n\r rand max is %u\n", RANDOM_MAX);

      for (int i = 0; i < NUMB_OF_BYTES; i++)
      {
            seed = (seed * 13124245 + 12345 ) & RANDOM_MAX;

            array2[i]=seed;
```

```
            //writing the random number array to the allocated memory.
            loc[i]=array2[i];
    }
}
//return the pointer for allocated memory and display its value.
uint32_t display_memory()
{
      for (int i = 0; i < NUMB_OF_BYTES; i++)
      {
            printf("\n\r pattern_generated is:%x\n",(unsigned int)*(ptr+i));
      }
   return *ptr;
}

//Function for Writing value at a particular Memory Location.
//*loc = the memory location at which we have to write the value.
void Write_Value_To_Memory(uint32_t * base_address,int offset, uint32_t value)
{
      //calculating the offset for the address in the allocated memory.
      uint32_t *offset_ptr=NULL;
      offset_ptr=base_address;
      for(i=0;i<offset;i++)
      {
            //incrementing it to the offset value required.
            offset_ptr++;
      }
    uint32_t *physical_address=NULL;
    //calculating the physical address with the help of offset pointer value
above.
    physical_address=(uint32_t*)(offset_ptr);

    *physical_address=value;
    printf("\n\r Pattern after modifying the value at the given location is");

    printf("\n\r:%x\n",(unsigned int)(*physical_address));
    uint32_t *printer=NULL;
    printer=ptr;
    for(i=0;i<NUMB_OF_BYTES;i++)
    {
       printf("\n\r%x", *(printer+i));
    }
}

//Inverting each byte of the array of random numbers allocated.
void invert_block(uint32_t * base_address,int offset2, size_t Inverting_length)
{

  //calculating the offset for the address in the allocated memory.
 uint32_t *offset_ptr3=NULL;
 offset_ptr3=base_address;
 for(i=0;i<offset2;i++)
 {
   //incrementing it to the offset value required.
  offset_ptr3++;
 }
    uint32_t *physical_address=NULL;
    //calculating the physical address with the help of offset pointer value
above.
    physical_address=(uint32_t*)(offset_ptr3);
```

```
    printf("\n\n\rinverted bytes are:");

    for(i=0;i<NUMB_OF_INVERTING_BYTES;i++)
    {
      //EXORing the array allocated to inverse the bits in it.
      array4[i]=(array2[i]^(0xff));
      base_address[i]=array4[i];

    }

    for(i=0;i<NUMB_OF_BYTES;i++)
    {
      // base_address[i]=array2[i];
      printf("\n\r%x",*(base_address+i));
    }
}

void ReInvert_block(uint32_t * base_address,int offset2, size_t Inverting_length)
{

       //calculating the offset for the address in the allocated memory.
        uint32_t *offset_ptr4=NULL;
        offset_ptr4=base_address;
        for(i=0;i<offset2;i++)
        {
          //incrementing it to the offset value required.
              offset_ptr4++;
        }
           uint32_t *physical_address=NULL;
           //calculating the physical address with the help of offset pointer
value above.
           physical_address=(uint32_t*)(offset_ptr4);

           printf("\n\n\rinverted bytes are:");

           for(i=0;i<NUMB_OF_INVERTING_BYTES;i++)
           {
             //EXORing the array allocated to inverse the bits in it.
             array5[i]=(array4[i]^(0xff));
             base_address[i]=array5[i];

           }

           for(i=0;i<NUMB_OF_BYTES;i++)
           {
             // base_address[i]=array2[i];
             printf("\n\r%x",*(base_address+i));
           }


}


//verifying if the original pattern and different patterns generated match.
uint32_t * verify_pattern(uint32_t * loc, size_t length, uint32_t seed)
{
```

```
 //generating the random pattern again with the same seed value to verify
different patterns.
      for (int i = 0; i < NUMB_OF_BYTES; i++)
   {
             seed = (seed * 13124245 + 12345 ) & RANDOM_MAX;

             array3[i]=seed;
   }

    for(i=0;i<NUMB_OF_BYTES;i++)
    {
      if(loc[i]==array3[i])
      {
             BLUE_LED_ON();
             printf("\n\rallocated memories are equal");

      }

      else if(loc[i]!=array3[i])
      {
             RED_LED_ON();
             printf("\n\rerror generated at address:");
             PRINTF("%x",(unsigned int)&loc[i]);
      }
    }
  return 0;
}
#endif
```

**********Memory_Functions.h**********

```
/*
 * @file Memory_Functions.h
 * author: Kushagra Pandey & Vaidehi Salway
 * Date:10/20/2019
 *
 * There are two modes of operations to choose from in the main routine
 * 1. FB_RUN: When this target is built the routine for the freedom board KL25Z
gets built.
 *                   This version prints the outputs on MCUxpresso terminal using
UART
 *                   LED indications are given on the freedom board as per the
test results
 * 2. PC_RUN: When this target is built the routine for the development
environment i.e. windows/linux gets built
 *                   This version prints the output on MCUxpresso console
 *                   Test indications are given by printing X LED ON the console
 *
 *This file contains header files, define statements, and function prototypes for
Memory_Function.c file
 *
 */
```

```c
#include "main.h"

//Function prototypes for FB_RUN MODE
#ifdef FB_RUN



#define RANDOM_MAX ((1U << 7) - 1)


uint32_t *allocate_words();
void free_words(uint32_t *ptr);
uint32_t * get_address(uint32_t *base_address,int offset);
void write_pattern(uint32_t * loc, size_t length,uint32_t seed);
uint32_t display_memory();
void Write_Value_To_Memory(uint32_t * base_address,int offset, uint32_t value);
void invert_block(uint32_t * base_address,int offset2, size_t Inverting_length);
uint32_t * verify_pattern(uint32_t * loc, size_t length, uint32_t seed);
void ReInvert_block(uint32_t * base_address,int offset2, size_t Inverting_length);

#endif

#ifdef PC_RUN

//Function prototypes for PC_RUN MODE
#define RANDOM_MAX ((1U << 7) - 1)



uint32_t *allocate_words();
void free_words(uint32_t *ptr);
uint32_t * get_address(uint32_t *base_address,int offset);
void write_pattern(uint32_t * loc, size_t length,uint32_t seed);
uint32_t display_memory();
void Write_Value_To_Memory(uint32_t * base_address,int offset, uint32_t value);
void invert_block(uint32_t * base_address,int offset2, size_t Inverting_length);
uint32_t * verify_pattern(uint32_t * loc, size_t length, uint32_t seed);
void ReInvert_block(uint32_t * base_address,int offset2, size_t Inverting_length);



#endif
```

**********LED_Blink.c**********

```c
/*
 * @file LED_Blink.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/16/2019
 *
 * This .c file contains two modes of operations
 * 1. FB_RUN: When running on frdm board
 *                    The LED on the board turn ON/OFF
 * 2. PC_RUN: When running on development environment i.e. windows/linux
```

```
 *                    Message on the console prints indicating the state of the
LED ON/OFF
 *
 *Different modes of operations will run based on which target is built
 *
 *
 */



#include "LED_Blink.h"

// This mode of operation is for running on KL25Z frdm board
#ifdef FB_RUN


#include "Memory_Functions.h"



//Configuring Pin direction and initial digital output value
gpio_pin_config_t LED_config=
      {
                  kGPIO_DigitalOutput, 1,
      };

//initializing green led GPIO Pin
void GREEN_LED_INIT()
      {

            GPIO_PinInit(BOARD_LED_GREEN_GPIO, BOARD_LED_GREEN_GPIO_PIN,
&LED_config);

      }
//setting ON green led GPIO Pin
void GREEN_LED_ON()
      {

            GPIO_ClearPinsOutput(BOARD_LED_GREEN_GPIO, 1U <<
BOARD_LED_GREEN_GPIO_PIN);

      }
//setting OFF green led GPIO Pin
void GREEN_LED_OFF()
      {

            GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1U <<
BOARD_LED_GREEN_GPIO_PIN);

      }

//initializing red led GPIO Pin
void RED_LED_INIT()
      {

            GPIO_PinInit(BOARD_LED_RED_GPIO, BOARD_LED_RED_GPIO_PIN,
&LED_config);

      }
```

```
//setting ON red led GPIO Pin
void RED_LED_ON()
	{

		GPIO_ClearPinsOutput(BOARD_LED_RED_GPIO, 1U <<
BOARD_LED_RED_GPIO_PIN);

	}
//setting OFF red led GPIO Pin
	void RED_LED_OFF()
	{

		GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1U <<
BOARD_LED_RED_GPIO_PIN);

	}


//initializing blue led GPIO Pin
	void BLUE_LED_INIT()
	{

		GPIO_PinInit(BOARD_LED_BLUE_GPIO, BOARD_LED_BLUE_GPIO_PIN,
&LED_config);

	}
//setting ON blue led GPIO Pin
void BLUE_LED_ON()
	{

		 GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1U <<
BOARD_LED_BLUE_GPIO_PIN);

	}
//setting OFF blue led GPIO Pin
void BLUE_LED_OFF()
	{

		 GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1U <<
BOARD_LED_BLUE_GPIO_PIN);

	}

#endif


// This mode of operation is for running on the development environment
#ifdef PC_RUN

#include "Memory_Functions.h"


//Configuring Pin direction and initial digital output value
gpio_pin_config_t LED_config=
	{
		kGPIO_DigitalOutput, 1,
	};
```

```
//setting ON green led GPIO Pin
void GREEN_LED_ON()
	{

		printf("\n\rGREEN_LED_ON");
	}
//setting OFF green led GPIO Pin
void GREEN_LED_OFF()
	{

		printf("\n\rGREEN_LED_OFF");

	}


//setting ON red led GPIO Pin
void RED_LED_ON()
	{

		printf("\n\rRED_LED_ON");

	}
//setting OFF red led GPIO Pin
void RED_LED_OFF()
	{

		printf("\n\rRED_LED_OFF");

	}



//setting ON blue led GPIO Pin
void BLUE_LED_ON()
	{

		printf("\n\rBLUE_LED_ON");

	}
//setting OFF blue led GPIO Pin
void BLUE_LED_OFF()
	{

		printf("\n\rBLUE_LED_OFF");

	}


#endif
```

**********LED_Blink.h**********

```
/*
 * @file LED_Blink.h
 * author: kushagra Pandey & Vaidehi Salway
```

```
 * Date:10/16/2019
 *
 * This is the header file to the LED_Blink.c
 * This contains Function definitions for two modes of operation of the program
 * 1. FB_RUN: to run on KL25Z frdm board
 * 2. PC_RUN: to run on development environment such as windows and linux
 */


#include "main.h"

// Defining functions for the Kl25Z frdm board mode of operation
#ifdef FB_RUN



void GREEN_LED_INIT();
void GREEN_LED_ON();
void GREEN_LED_OFF();

void RED_LED_INIT();
void RED_LED_ON();
void RED_LED_OFF();

void BLUE_LED_INIT();
void BLUE_LED_ON();
void BLUE_LED_OFF();

#endif


// Defining functions for Development environment i.e. windows/linux mode of
operation
#ifdef PC_RUN


void GREEN_LED_ON();
void GREEN_LED_OFF();

void RED_LED_ON();
void RED_LED_OFF();

void BLUE_LED_ON();
void BLUE_LED_OFF();

#endif
```

**********Delay_Function.c**********

```c
/*
 * @file Delay_Function.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/14/2019
 *
 * This .c file contains a function to generate a delay
 */

#include "Delay_Function.h"
#include "Memory_Functions.h"


// Constant multiplier based on the clock frequency of frdm board
uint32_t cons_val=4000000;
uint32_t j=0;
extern uint32_t milli_sec_val;

/* This is a delay function with a parameter milli_sec_val
 * delay calculations are based on the clock frequency to
 * generate a delay equivalent to user input through delay function.
 */
void Delay_Time(uint32_t milli_sec_val)
{
	uint32_t Ticks_value= (cons_val * milli_sec_val)/(500);

for( j=0;j<=Ticks_value;j++);


}
```

**********Delay_Function.h**********

```c
/*
 * @file Delay_Function.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/14/2019
 *
 * This .h file includes all the header files required for the Delay_Function.c
file
 */

#include "main.h"

// defining the delay function
void Delay_Time(uint32_t milli_sec_val);
```

**********Logger.c**********

```c
/*
 * @file Logger.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/18/2019
 *
 * This .c file contains the logger statements for cross-platform
 * There are two modes of operating this file
 * 1. Logger Enable
 * 2. Logger Disable
 *
 * Enable or Disable the logger by un-commenting #define logging_init or #define
logging_notinit respectively from Logger.h
 *
 */


#include "Logger.h"


uint8_t log_status;

// Function log enable, when called in main returns log_status 1
void Log_Enabled()
	{
		log_status=1;
	}

// Function log disable, when called in main returns log status 0
void Log_Disabled()
	{
		PRINTF("\nLOGGERS ARE DISABLED");
		log_status=0;
	}

// checking condition to enable logging
// Status function called when logging_init is defined in logger.h
#ifdef logging_init

// Function Status calls log_Enabled
uint8_t Status()
	{
		Log_Enabled();
		return log_status;
	}
#endif


// checking condition to disable logging
// Status function called when logging_notinit is defined in logger.h
#ifdef logging_notinit

//Function Status calls log_Disabled
uint8_t Status()
	{
		Log_Disabled();
```

```
		return log_status;
	}
#endif

// Log Data function enables printing data on the terminal when running in freedom
board
void Log_Data(uint32_t data)
	{
		Status();
		if(log_status==1)
			{
				PRINTF("\n\rLOG_DATA:%x",data);
			}

	}


// Log Data function enables printing Strings on the terminal when running in
freedom board
void Log_String(char *statement)
	{
		Status();

		if(log_status==1)
			{
				PRINTF("\n\rLOG_STRING:%s",statement);
			}

	}


//
//void Log_User_Input(int user_input)
//{
//Status();
//
//if(log_status==1)
//{
//SCANF("\n\rLOG_STRING:%d",&user_input);
//}
//
//}


// Log Data function enables printing integer values on the terminal when running
in freedom board
void Log_Integer(int integer_value)
	{
		Status();
		if(log_status==1)
			{
				PRINTF("\n\rLOG_INTEGER:%d",integer_value);
			}

	}
```

**********Logger.h**********

```
/*
 * @file Logger.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/18/2019
 *
 * This .h file contains the header files requred for Logger.c file
 * There are two modes of operating this file
 * 1. Logger Enable
 * 2. Logger Disable
 *
 * Enable or Disable the logger by un-commenting #define logging_init or #define
logging_notinit respectively from Logger.h
 *
 *Defining the funcyions used in Logger.c file
 */


#include "main.h"
#include "string.h"

// Un-comment logging_init and comment logging_notinit to enable logging
#define logging_init

// Comment logging_init and un-comment logging_notinit to enable logging
//#define logging_notinit

void Log_Enabled();
void Log_Disabled();
uint8_t Status();
void Log_Data(uint32_t data);
void Log_String(char *statement);
void Log_Integer(int integer_value);
```

**********The End**********