Principles of Embedded Software Project 5

Readme

Principles of Embedded Software Project 5

README

Name: Atharva Nandanwar

This repository contains source files for Principles of Embedded Software Project 5 - UART and Circular Buffer

---

Source Folder:

1. main.c/h - main subroutine
2. circular_buffer/circular_buffer.c/h - functions and structure definition for circular buffer
3. led_control/led_control.c/h - functions to control LED
4. logger/logger.c/h - functions to do logging
5. logger/errno.c/h - error handling routines
6. logger/timestamp.c/h - timestamp functionality
7. test/test.c/h - test cases and test subroutine
8. uart/uart.c/h - uart subroutines, initialization, and operation drivers
9. uctest - uCunit testing files
10. common.h - common include file for system-wide implementation

---

Observations:

1. There was serious problems with how integration would work in this aspect. How do I interface interrupt service routines with circular buffer operations? How do I interface UART functions with circular buffer? How do I manage getchar and putchar functionality with everything in the system? *Making global buffers was the way to do
2. The way I had designed logger required me to implement a lot of complicated printf operations. I rather used sprintf to format data into a string and printing the string directly.
3. During Interrupt based Application Mode, the program switches to Overrun condition, but it doesn't raise the interrupt. I did not address this problem in this program.

Installation/Execution Notes:

Use basic options to compile RUN, LOG, and TEST builds from Build Targets.

Compiler - gcc-arm-none-eabi

There are different functionalities that can be configured in RUN and LOG mode:

1. UART Non-blocking Echo Mode - #define ECHO_POLLING
2. UART Non-blocking Application Mode - #define APP_POLLING
3. UART Interrupts Echo Mode - #define ECHO_IRQN
4. UART Interrupts Application Mode - #define APP_IRQN

You will have to go to common.h to edit different modes. Comment out the mode you want, and comment others.

Note:

In application mode, report is printed after every 50th character is received.

Makefile

```
# Makefile for PES Project 5
# Author : Atharva Nandanwar
# Date: 11/14/2019

################################################################################
# Build Variables

# Program for removing files
RM := rm -rf

# Program for making directories
MK := mkdir -p

# ARM compiler
ARM_CC := arm-none-eabi-gcc

# ARM linker
ARM_LL := arm-none-eabi-gcc

################################################################################
# ARM Compiler Flags
ARM_FLAGS := -c \
                    -std=c99 \
                    -O0 \
                    -g3 \
                    -ffunction-sections \
                    -fmessage-length=0 \
                    -fno-common \
                    -fdata-sections \
```

```
                    -fno-builtin \
                    -mcpu=cortex-m0plus \
                    -mthumb \
                    -Wall \
                    -Werror

# ARM Linker Flags
ARM_LL_FLAGS := -v \
                         -nostdlib \
                       -Xlinker -Map="./Debug/pes_project_5.map" \
                       -Xlinker --gc-sections \
                       -Xlinker -print-memory-usage \
                       -Xlinker --sort-section=alignment \
                       -Xlinker --cref \
                       -mcpu=cortex-m0plus \
                       -mthumb \
                       -T linkerfile.ld \
                       -o $(EXE)

# ARM Defines
ARM_DEFS := \
                -D__REDLIB__ \
                -DCPU_MKL25Z128VLK4 \
                -DCPU_MKL25Z128VLK4_cm0plus \
                -DSDK_OS_BAREMETAL \
                -DFSL_RTOS_BM \
                -DCR_INTEGER_PRINTF \
                -DPRINTF_FLOAT_ENABLE=0 \
                -DSCANF_FLOAT_ENABLE=0 \
                -DPRINTF_ADVANCED_ENABLE=0 \
                -DSCANF_ADVANCED_ENABLE=0 \
                -D__MCUXPRESSO \
                -D__USE_CMSIS \
                -DDEBUG \
                -DFRDM_KL25Z \
                -DFREEDOM \
                -specs=redlib.specs \

# Build Folders
SOURCE := ./source
DEBUG := ./Debug

# ARM Include Files
ARM_INCS := \
                -I"$(SOURCE)" \
                -I"$(SOURCE)/uart" \
                -I"$(SOURCE)/led_control" \
                -I"$(SOURCE)/logger" \
                -I"$(SOURCE)/circular_buffer" \
                -I"$(SOURCE)/test" \
                -I"$(SOURCE)/uctest" \
                -I"board" \
                -I"CMSIS" \
                -I"drivers" \
                -I"startup" \

# ARM Object Files
ARM_OBJS := \
```

```
                $(DEBUG)/source/led_control/led_control.o \
                $(DEBUG)/source/circular_buffer/circular_buffer.o \
                $(DEBUG)/source/logger/logger.o \
                $(DEBUG)/source/logger/errno.o \
                $(DEBUG)/source/logger/timestamp.o \
                $(DEBUG)/source/uart/uart.o \
                $(DEBUG)/source/uctest/System.o \
                $(DEBUG)/startup/startup_mkl25z4.o \
                $(DEBUG)/CMSIS/system_MKL25Z4.o \
                $(DEBUG)/board/board.o \
                $(DEBUG)/board/clock_config.o \
                $(DEBUG)/board/peripherals.o \
                $(DEBUG)/board/pin_mux.o \
                $(DEBUG)/drivers/fsl_clock.o \
                $(DEBUG)/drivers/fsl_common.o \
                $(DEBUG)/drivers/fsl_flash.o \
                $(DEBUG)/drivers/fsl_gpio.o \
                $(DEBUG)/drivers/fsl_lpsci.o \
                $(DEBUG)/drivers/fsl_smc.o \

# ARM Dependencies Files
ARM_DEPS := \
                $(DEBUG)/source/led_control/led_control.d \
                $(DEBUG)/source/application/application.d \
                $(DEBUG)/source/circular_buffer/circular_buffer.d \
                $(DEBUG)/source/logger/logger.d \
                $(DEBUG)/source/logger/errno.d \
                $(DEBUG)/source/logger/timestamp.d \
                $(DEBUG)/source/uart/uart.d \
                $(DEBUG)/source/uctest/System.d \
                $(DEBUG)/startup/startup_mkl25z4.d \
                $(DEBUG)/CMSIS/system_MKL25Z4.d \
                $(DEBUG)/board/board.d \
                $(DEBUG)/board/clock_config.d \
                $(DEBUG)/board/peripherals.d \
                $(DEBUG)/board/pin_mux.d \
                $(DEBUG)/drivers/fsl_clock.d \
                $(DEBUG)/drivers/fsl_common.d \
                $(DEBUG)/drivers/fsl_flash.d \
                $(DEBUG)/drivers/fsl_gpio.d \
                $(DEBUG)/drivers/fsl_lpsci.d \
                $(DEBUG)/drivers/fsl_smc.d \

# Executable file
EXE := $(DEBUG)/pes_project_5.axf

########################################################################
# Build Rules
# Rules for making all
all : $(EXE)

########################################################################
# Selecting Platform
ifeq ($(BUILD), TEST)
build_option := test
else ifeq ($(BUILD), DEBUG)
build_option := debug
else ifeq ($(BUILD), RUN)
```

```
build_option := run
endif
#########################################################################

$(EXE) : $(build_option)

#########################################################################
# Rule for compiling tests
test : directories $(ARM_OBJS) $(SOURCE)/test/test.c
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) ./source/test/test.c -o $
(DEBUG)/source/test/test.o
	@arm-none-eabi-gcc -nostdlib -Xlinker -Map="./Debug/pes_project_5.map" -
Xlinker --gc-sections -Xlinker -print-memory-usage -Xlinker --sort-
section=alignment -Xlinker --cref -mcpu=cortex-m0plus -mthumb -T linkerfile.ld -
o ./Debug/pes_project_5.axf $(ARM_OBJS) $(DEBUG)/source/test/test.o
	@echo "Testing Code Compiled"

#########################################################################
# Rule for compiling detailed debug log
debug : directories $(ARM_OBJS) $(SOURCE)/main.c
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -DDEBUG $(SOURCE)/main.c -o
$(DEBUG)/source/main.o
	@arm-none-eabi-gcc -nostdlib -Xlinker -Map="./Debug/pes_project_5.map" -
Xlinker --gc-sections -Xlinker -print-memory-usage -Xlinker --sort-
section=alignment -Xlinker --cref -mcpu=cortex-m0plus -mthumb -T linkerfile.ld -
o ./Debug/pes_project_5.axf $(ARM_OBJS) $(DEBUG)/source/main.o
	@echo "KL25Z with Debug Logging"

#########################################################################
# Rule for compiling program with normal logging
run : directories $(ARM_OBJS) $(SOURCE)/main.c
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -DRUN $(SOURCE)/main.c -o $
(DEBUG)/source/main.o
	@arm-none-eabi-gcc -nostdlib -Xlinker -Map="./Debug/pes_project_5.map" -
Xlinker --gc-sections -Xlinker -print-memory-usage -Xlinker --sort-
section=alignment -Xlinker --cref -mcpu=cortex-m0plus -mthumb -T linkerfile.ld -
o ./Debug/pes_project_5.axf $(ARM_OBJS) $(DEBUG)/source/main.o
	@echo "KL25Z with Run Configuration"

#########################################################################
# Essential Source Files
$(DEBUG)/source/circular_buffer/%.o: ./source/circular_buffer/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/source/led_control/%.o: ./source/led_control/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/source/logger/%.o: ./source/logger/%.c
	@echo 'Building file: $<'
```

```
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/source/uart/%.o: ./source/uart/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/source/uctest/%.o: ./source/uctest/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

################################################################
# Essesntial ARM Object Files
$(DEBUG)/board/%.o: ./board/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/CMSIS/%.o: ./CMSIS/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/drivers/%.o: ./drivers/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/startup/%.o: ./startup/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

$(DEBUG)/utilities/%.o: ./utilities/%.c
	@echo 'Building file: $<'
	@$(ARM_CC) $(ARM_FLAGS) $(ARM_DEFS) $(ARM_INCS) -MMD -MP -MF"./$(@:%.o=%.d)"
-MT"./$(@:%.o=%.o)" -MT"./$(@:%.o=%.d)" -o "$@" "$<"
	@echo 'Finished building: $<'
	@echo ' '

################################################################
# Making directories
```

```
.PHONY : directories
directories :
	$(MK) \
	$(DEBUG) \
	$(DEBUG)/board \
	$(DEBUG)/CMSIS \
	$(DEBUG)/drivers \
	$(DEBUG)/startup \
	$(DEBUG)/utilities \
	$(DEBUG)/source/application \
	$(DEBUG)/source/circular_buffer \
	$(DEBUG)/source/led_control \
	$(DEBUG)/source/logger \
	$(DEBUG)/source/test \
	$(DEBUG)/source/uart \
	$(DEBUG)/source/uctest

# Clean target
.PHONY : clean
clean:
	@$(RM) \
	$(DEBUG)/board \
	$(DEBUG)/CMSIS \
	$(DEBUG)/drivers \
	$(DEBUG)/startup \
	$(DEBUG)/utilities \
	$(DEBUG)/source \
	$(DEBUG)/pes_project_5.axf \
	$(DEBUG)/pes_project_5.map
	@echo "Build cleaned"
```

Source Files

Logger

1. logger.c

```c
/**
  * File Name           - logger.c
  * Description   - contains logger subroutines
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      - https://github.com/ntpeters/SimpleLogger/
  * URL                       -
  */
#include "logger.h"

// Struct for storing logger data
typedef struct {
            log_level_t Logger_Log_Level;
}logger_data;

logger_data thisLogger;

// Character codes for colors
// Leveraged Code - https://stackoverflow.com/questions/3585846/color-text-in-terminal-applications-in-unix
```

```c
const char* red = "\x1B[31m";
const char* green = "\x1B[32m";
const char* blue = "\x1B[34m";
const char* end = "\x1B[0m";

/**
 * Init
 * initializes logger by initializing timestamp
 */
void Init(void)
{
	vTimestamp_Init();
}

/*
 * Function - Log_Write
 * Brief - Prints a log message
 * Arguments -
 * function_name -> name of the calling function
 * message_type -> Error, Debug or Status message
 * msg, ... -> printf style argument to hold a string and format specifiers
 * Leveraged Code - https://www.ozzu.com/cpp-tutorials/tutorial-writing-custom-
printf-wrapper-function-t89166.html
 */
void Log_Write(const char* function_name, message_type_t message_type, const char
*msg, ... )
{
	// To process variable argument list
	va_list args;
	va_start(args, msg);

	// Activate color based on message type
	switch(message_type)
	{
	case mError:
		pprintf("%s", red);
		break;
	case mDebug:
		pprintf("%s", blue);
		break;
	case mStatus:
		pprintf("%s", green);
		break;
	}

	// Timestamp related routine
	timestamp_t currentTime = tTimestamp_Get_Timestamp();
	pprintf("[%02d:%02d:%02d.%d]", currentTime.hours, \
			currentTime.minutes, currentTime.seconds, \
			currentTime.deciseconds);

	// Log Level Logic
	switch(thisLogger.Logger_Log_Level)
	{
	case lTest:
		pprintf("Test:  ");
		break;
	case lDebug:
```

```
            pprintf("Debug: ");
            break;
      case lNormal:
            pprintf("Run:   ");
            break;
      }

      // Printing function names
      pprintf("%-27s:\t", function_name);


      // Message print with color termination code
      vpprintf(msg, args);
      pprintf("%s\n\r", end);
}

/*
 * Function - Get_Log_Level
 * Brief - returns the current log level
 * Return -
 * returns log_level_t enum value
 */
log_level_t Get_Log_Level (void)
{
      return thisLogger.Logger_Log_Level;
}

/*
 * Function - Set_Log_Level
 * Brief - sets the current log level
 * Arguments -
 * log_level_t enum value
 */
void Set_Log_Level (log_level_t level)
{
      thisLogger.Logger_Log_Level = level;
}

// Declaration for logger struct
logger_instance const logger = {Init, Log_Write, Set_Log_Level, Get_Log_Level};
```

2. logger.h

```
/**
  * File Name           - logger.h
  * Description   - header file for logger.c
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */

#ifndef LOGGER_LOGGER_H_
#define LOGGER_LOGGER_H_

// Include Files
#include <stdint.h>
#include <errno.h>
```

```c
#include <stdarg.h>
#include "timestamp.h"
#include "uart.h"

// Log Level and Message Type enums
typedef enum {lTest, lDebug, lNormal} log_level_t;
typedef enum {mError, mDebug, mStatus} message_type_t;

// Logger Instance struct
typedef struct {
	void ( * const Init )( void );
	void ( * const Log_Write )( const char* function_name, \
			message_type_t message_type, const char *msg, ... );
	void ( * const Set_Log_Level )( log_level_t level );
	log_level_t ( * const Get_Log_Level )( void );
}logger_instance;

extern logger_instance const logger;
#endif /* LOGGER_LOGGER_H_ */
```

3. timestamp.c

```c
/**
 * File - timestamp.c
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */

#include "timestamp.h"

// System Clock Macro
#define SYSCLOCK 48000000UL

// Global deciseconds count
uint32_t deciseconds = 0;

/**
 * vTimestamp_Init
 * Sets up SysTick timer with 0.1 second
 */
void vTimestamp_Init(void)
{
	SysTick_Config(SYSCLOCK/10);
}

/**
 * tTimestamp_Get_Timestamp
 * Gets time stamp data
 * @return
 *		returns a struct with timestamp information
 */
timestamp_t tTimestamp_Get_Timestamp(void)
{
	uint32_t temp;
	timestamp_t currentTime;
```

```
	currentTime.hours = deciseconds / 36000;
	temp = deciseconds % 36000;
	currentTime.minutes = temp / 600;
	temp = temp % 600;
	currentTime.seconds = temp / 10;
	currentTime.deciseconds = temp % 10;
	return currentTime;
}

/**
 * SysTick_Handler
 * Interrupt handler for systick interrupt
 */
void SysTick_Handler(void)
{
	deciseconds++;
}
```

4. timestamp.h

```
/**
 * File - timestamp.h
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */
#ifndef LOGGER_TIMESTAMP_H_
#define LOGGER_TIMESTAMP_H_
// Include files
#include "MKL25Z4.h"

// Struct for timestamp information
typedef struct {
	uint8_t hours;
	uint8_t minutes;
	uint8_t seconds;
	uint8_t deciseconds;
} timestamp_t;

// Prototype Functions
void vTimestamp_Init(void);
timestamp_t tTimestamp_Get_Timestamp(void);
#endif /* LOGGER_TIMESTAMP_H_ */
```

5. errno.c

```
/**
  * File Name           - errno.c
  * Description   - contains error enums, and related functions
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      - https://android.googlesource.com/kernel/lk/+/upstream-
master/include/errno.h
  * URL                       -
  */
```

```c
#include "errno.h"

/**
 * Get_Error_Message
 * returns with error message for particular errors
 * @param
 *          error - error code
 * @return
 *          returns error message
 */
const char* Get_Error_Message(error_t error)
{
      switch(error)
      {
      case eUART_Parity_Error:
            return "Parity Error";
            break;
      case eUART_Noise_Error:
            return "Noise Error";
            break;
      case eUART_Framing_Error:
            return "Framing Error";
            break;
      case eUART_Overrun_Error:
            return "Overrun Error";
            break;
      default:
            return "";
            break;
      }
}
```

6. errno.h

```c
/**
  * File Name           - errno.h
  * Description   - header file for errno.c
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */

#ifndef LOGGER_ERRNO_H_
#define LOGGER_ERRNO_H_
#include <stdint.h>

// Error/Event Enum
typedef enum {
      eUART_Parity_Error = 0x2001,
      eUART_Framing_Error = 0x2002,
      eUART_Noise_Error = 0x2003,
      eUART_Overrun_Error = 0x2004,
}error_t;

extern error_t errno;
```

```c
// Prototype function
const char* Get_Error_Message(error_t error);

#endif /* LOGGER_ERRNO_H_ */
```

Led Control

1. led_control.c

```c
/**
 * File - led_control.c
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */
#include "led_control.h"

/**
 * LED_Init
 * Initialization of LEDs
 */
void LED_Init(void)
{

	// Set up clock for Port peripheral
	SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK | SIM_SCGC5_PORTD_MASK;

	// Set mux for each port pins
	PORTB->PCR[18] = PORT_PCR_MUX(1);
	PORTB->PCR[19] = PORT_PCR_MUX(1);
	PORTD->PCR[1] = PORT_PCR_MUX(1);

	// Set data directions, and turn the LEDs off
	GPIOB->PDDR |= RED_LED | GREEN_LED;
	GPIOD->PDDR |= BLUE_LED;
	Turn_Off_LEDs();
}

/**
 * Turn_On_Only_LED
 * Turns on only specified color LED
 * @param
 *          LED - color of LED
 */
void Turn_On_Only_LED(led_color_t LED)
{
	switch(LED)
	{
	case Red:
		GPIOB->PCOR |= RED_LED;		// Turn On
		GPIOB->PSOR |= GREEN_LED;	// Turn Off
		GPIOD->PSOR |= BLUE_LED;	// Turn Off
		break;
	case Green:
		GPIOB->PCOR |= GREEN_LED;	// Turn On
		GPIOB->PSOR |= RED_LED;		// Turn Off
```

```
            GPIOD->PSOR |= BLUE_LED;            // Turn Off
            break;
      case Blue:
            GPIOD->PCOR |= BLUE_LED;            // Turn On
            GPIOB->PSOR |= RED_LED;             // Turn Off
            GPIOB->PSOR |= GREEN_LED;           // Turn Off
            break;
      }
}

/**
 * Turn_Off_LEDs
 * turns off all LEDs
 */
void Turn_Off_LEDs(void)
{
      GPIOD->PSOR |= (0x1 << 1U);               // Turn Off
      GPIOB->PSOR |= (0x1 << 18U);        // Turn Off
      GPIOB->PSOR |= (0x1 << 19U);        // Turn Off
}
```

2. led_control.h

```
/**
 * File - led_control.h
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */
#ifndef LED_CONTROL_LED_CONTROL_H_
#define LED_CONTROL_LED_CONTROL_H_

// Include files
#include <stdint.h>
#include "MKL25Z4.h"
#include "logger.h"

// Macros
#define RED_LED         (0x1 << 18U)
#define GREEN_LED (0x1 << 19U)
#define BLUE_LED  (0x1 << 1U)

// Enum for LED Color
typedef enum {
      Red,
      Green,
      Blue,
} led_color_t;

// Prototype functions
void LED_Init(void);
void Turn_On_Only_LED(led_color_t LED);
void Turn_Off_LEDs(void);

#endif /* LED_CONTROL_LED_CONTROL_H_ */
```

Circular Buffer

1. circular_buffer.c

```
      /**
 * File - circular_buffer.c
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */

#include "circular_buffer.h"

/**
 * cb_init_buffer
 * Creates a circular buffer
 * @param
 *          length - size of circular buffer
 * @return
 *          pointer to circular buffer
 */
circular_buffer_t* cb_init_buffer(uint16_t length)
{
      // Allocate memory for buffer structure, and memory for buffer
      circular_buffer_t* buffer_pointer = NULL;
      buffer_pointer = (circular_buffer_t *) malloc(sizeof(circular_buffer_t));
      buffer_pointer->pointer = (uint8_t *) malloc(length);

      // Set all the parameters
      buffer_pointer->head = buffer_pointer->pointer;
      buffer_pointer->tail = buffer_pointer->pointer;
      buffer_pointer->count = 0;
      buffer_pointer->length = length;
      return buffer_pointer;
}

/**
 * cb_destroy_buffer
 * Destroys the circular buffer
 * @param
 *          buffer - pointer to circular buffer
 * @return
 *          status of operation
 */
CB_status_t cb_destroy_buffer(circular_buffer_t* buffer)
{
      // Free the memory for buffer, and buffer structure
      free(buffer->pointer);
      buffer->pointer = NULL;
      free(buffer);
      buffer = NULL;
      return CB_buffer_destroyed;
}

/**
 * cb_check_full
 * Checks if buffer is full
```

```
 * @param
 *          buffer - pointer to circular buffer
 * @return
 *          status of operation
 */
CB_status_t cb_check_full(circular_buffer_t* buffer)
{
      // Flag error
      if(buffer == NULL)
      {
            return CB_buffer_error;
      }

      // Check full
      if(buffer->count == buffer->length)
      {
            return CB_buffer_full;
      }
      else
      {
            return CB_buffer_not_full;
      }
}

/**
 * cb_check_empty
 * Checks if buffer is empty
 * @param
 *          buffer - pointer to circular buffer
 * @return
 *          status of operation
 */
CB_status_t cb_check_empty(circular_buffer_t* buffer)
{
      // Flag error
      if(buffer == NULL)
      {
            return CB_buffer_error;
      }

      // Check empty
      if(buffer->count == 0)
      {
            return CB_buffer_empty;
      }
      else
      {
            return CB_buffer_not_empty;
      }
}

/**
 * cb_add_item
 * Checks if buffer is full, and adds item if not full
 * @param
 *          buffer - pointer to circular buffer
 * @param
 *          item - data to be added into circular buffer
```

```
 * @return
 *          status of operation
 */
CB_status_t cb_add_item(circular_buffer_t* buffer, uint8_t item)
{
      // Flag error
      if(buffer == NULL)
      {
            return CB_buffer_error;
      }

      START_CRITICAL();
      // If not full, then update parameters
      if(cb_check_full(buffer) == CB_buffer_full)
      {
            return CB_buffer_full;
      }
      else
      {
            *(buffer->head) = item;
            buffer->head += 1;
            buffer->head = (uint32_t) (buffer->head - buffer->pointer) % buffer-
>length + buffer->pointer;
            buffer->count += 1;
      }
      END_CRITICAL();
      return CB_buffer_operation_success;
}


/**
 * cb_remove_item
 * Checks if circular buffer is empty, and removes the item
 * @param
 *          buffer - pointer to circular buffer
 * @return
 *          status of operation
 */
CB_status_t cb_remove_item(circular_buffer_t* buffer, uint8_t* data)
{
      // Flag error
      if(buffer == NULL)
      {
            return CB_buffer_error;
      }

      START_CRITICAL();
      // If not empty, then update parameters
      if(cb_check_empty(buffer) == CB_buffer_empty)
      {
            return CB_buffer_empty;
      }
      else
      {
            *data = *(buffer->tail);
            buffer->tail += 1;
            buffer->tail = (uint32_t) (buffer->tail - buffer->pointer) % buffer-
>length + buffer->pointer;
```

```
            buffer->count -= 1;
      }
      END_CRITICAL();
      return CB_buffer_operation_success;
}

/**
 * cb_verify_init
 * Verifies buffer initialization
 * @param
 *          buffer - pointer to circular buffer
 * @return
 *          status of operation
 */
CB_status_t cb_verify_init(circular_buffer_t* buffer)
{
      // Flag error
      if(buffer == NULL)
      {
            return CB_buffer_error;
      }

      // Checks for initialization error
      if(buffer->pointer == NULL)
      {
            return CB_buffer_error_init;
      }
      else if(buffer->head != buffer->pointer || buffer->tail != buffer->pointer)
      {
            return CB_buffer_error_init;
      }
      else if(buffer->count != 0)
      {
            return CB_buffer_error_init;
      }
      return CB_buffer_initialized;
}
```

2. circular_buffer.h

```
/**
 * File - circular_buffer.h
 * Author - Atharva Nandanwar
 * Email - atharva.nandanwar@colorado.edu
 * Principles of Embedded Software
 * University of Colorado Boulder
 */
#ifndef CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_
#define CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_

// Include files
#include <stdint.h>
#include <stdlib.h>
#include "MKL25Z4.h"

// Macros for Critical Section
#define START_CRITICAL()      __disable_irq()
```

```
#define END_CRITICAL()        __enable_irq()

// Enum for status
typedef enum {
      CB_buffer_full,
      CB_buffer_not_full,
      CB_buffer_empty,
      CB_buffer_not_empty,
      CB_buffer_initialized,
      CB_buffer_error_init,
      CB_buffer_destroyed,
      CB_buffer_error,
      CB_buffer_operation_success,
} CB_status_t;

// Structure for circular buffer
typedef struct {
      uint8_t* pointer;
      uint8_t* head;
      uint8_t* tail;
      uint16_t length;
      uint16_t count;
} circular_buffer_t;

// Prototype functions
CB_status_t cb_add_item(circular_buffer_t* buffer, uint8_t item);
CB_status_t cb_remove_item(circular_buffer_t* buffer, uint8_t* data);
CB_status_t cb_check_full(circular_buffer_t* buffer);
CB_status_t cb_check_empty(circular_buffer_t* buffer);
CB_status_t cb_verify_init(circular_buffer_t* buffer);
circular_buffer_t* cb_init_buffer(uint16_t length);
CB_status_t cb_destroy_buffer(circular_buffer_t* buffer);

#endif /* CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_ */
```

UART

1. uart.c

```
/**
  * File Name           - uart.c
  * Description   - contains drivers for uart
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */


#include "uart.h"

/**
 * uart_init
 * Initializes UART Peripheral
 * @param
 *          uart_config - pointer to uart configuration structure
 * Leveraged Code - Alexander Dean Example
 */
void uart_init(UARTConfig_t* uart_config) {
```

```
    uint16_t sbr;

    // Enable clock gating for UART0 and Port A
    SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
    SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

    // Disable TX and RX modules before changing registers
    UART0->C2 &= ~UART0_C2_TE_MASK & ~UART0_C2_RE_MASK;

    // Set UART clock to 48 MHz clock
    SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);
    SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;

    // Set PORT pins to UART TX and RX
    PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Rx
    PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Tx

    // Calculate baud rate and oversampling rate
    sbr = (uint16_t)((SYS_CLOCK)/(uart_config->baud_rate * (uart_config->osr +
1)));
    UART0->BDH &= ~UART0_BDH_SBR_MASK;
    UART0->BDH |= UART0_BDH_SBR(sbr >> 8);
    UART0->BDL = UART0_BDL_SBR(sbr);
    UART0->C4 |= UART0_C4_OSR(uart_config->osr);

    // Setting for stop bit
    UART0->BDH |= UART0_BDH_SBNS(uart_config->stop_bit);

    // Setting for parity bit
    UART0->C1 |= uart_config->parity;

    // Clear error flags
    UART0->S1 = UART0_S1_OR(1) | UART0_S1_NF(1) | UART0_S1_FE(1) | UART0_S1_PF(1);

    // Data format selection - LSB First, and No inversion
    UART0->S2 = UART0_S2_MSBF(0) | UART0_S2_RXINV(0);

    // Enable UART TX/RX
    UART0->C2 |= UART0_C2_RE(1) | UART0_C2_TE(1);

    // Do a dummy read and reset receive flag
    __attribute__((unused)) uint8_t temp = UART0->D;
    UART0->S1 &= ~UART0_S1_RDRF_MASK;
}

// Enable Interrupts only in IRQN modes
#if defined(APP_IRQN) || defined(ECHO_IRQN)
/**
 * uart_enable_irq
 * Enable UART Interrupts
 */
void uart_enable_irq(void)
{
      // Enable Interrupts
      UART0->C2 |= UART_C2_RIE(1);

    // Set up C3 register for error based interrupts
```

```
    UART0->C3 |= UART_C3_ORIE(1) | UART_C3_NEIE(1) | UART_C3_FEIE(1) |
UART_C3_PEIE(1);

    // Set up NVIC registers
      NVIC->ICPR[0] |= 1 << (UART0_IRQn);
      NVIC->ISER[0] |= 1 << (UART0_IRQn);
}
#endif

/**
 * uart_tx_available
 * Checks if TX is available
 * @return
 *          UART status
 */
UARTStatus_t uart_tx_available(void)
{
      if(UART0->S1 & UART0_S1_TDRE_MASK)
            return TX_available;
      else
            return TX_not_available;
}

/**
 * uart_tx_action
 * Send data through UART
 * @param
 *          data - 8 bit data
 */
void uart_tx_action(uint8_t data)
{
      UART0->D = data;
}

/**
 * uart_tx
 * Check if TX is available, and send the data
 * @param
 *          data - 8 bit data
 */
void uart_tx(uint8_t* data)
{
      // Logic for polling
#if defined(APP_POLLING) || defined(ECHO_POLLING)
      while(uart_tx_available() != TX_available);
      Turn_On_Only_LED(Green);
#endif
      uart_tx_action(*data);
}

/**
 * uart_rx_check
 * Check if RX is available
 * @return
 *          UART status
 */
UARTStatus_t uart_rx_check(void)
{
```

```
	if(UART0->S1 & UART0_S1_RDRF_MASK)
		return RX_available;
	else
		return RX_not_available;
}

/**
 * uart_rx_action
 * Receives data from UART
 * @return
 *		return 8 bit data
 */
uint8_t uart_rx_action(void)
{
	return UART0->D;
}

/**
 * uart_rx
 * Check if RX is available, and get data from UART
 * @return
 *		return 8 bit data
 */
void uart_rx(uint8_t* data)
{
	// Logic for polling
#if defined(APP_POLLING) || defined(ECHO_POLLING)
	while(uart_rx_check() != RX_available);
	Turn_On_Only_LED(Blue);
#endif
	*data = uart_rx_action();
}

/**
 * uart_tx_handler
 * Enables the TX Interrupt when there is data to send
 */
void uart_tx_handler(void)
{
	if(cb_check_empty(tx_buffer) == CB_buffer_not_empty)
	{
		UART0->C2 |= UART_C2_TIE_MASK;
	}
}

/**
 * uart_error_handler
 * Handles UART errors
 */
void uart_error_handler(void)
{
	if(system_info.pe_flag)
	{
		errno = eUART_Parity_Error;
		logger.Log_Write(__func__, mError, Get_Error_Message(errno));
	    // Do a dummy read and reset receive flag
	    __attribute__((unused)) uint8_t temp = UART0->D;
	    UART0->S1 &= ~UART0_S1_RDRF_MASK;
```

```
	}
	else if(system_info.ne_flag)
	{
		errno = eUART_Noise_Error;
		logger.Log_Write(__func__, mError, Get_Error_Message(errno));
	    // Do a dummy read and reset receive flag
	    __attribute__((unused)) uint8_t temp = UART0->D;
	    UART0->S1 &= ~UART0_S1_RDRF_MASK;
	}
	else if(system_info.or_flag)
	{
		errno = eUART_Overrun_Error;
		logger.Log_Write(__func__, mError, Get_Error_Message(errno));
	    // Do a dummy read and reset receive flag
	    __attribute__((unused)) uint8_t temp = UART0->D;
	    UART0->S1 &= ~UART0_S1_RDRF_MASK;
	}
	else if(system_info.fe_flag)
	{
		errno = eUART_Framing_Error;
		logger.Log_Write(__func__, mError, Get_Error_Message(errno));
	    // Do a dummy read and reset receive flag
	    __attribute__((unused)) uint8_t temp = UART0->D;
	    UART0->S1 &= ~UART0_S1_RDRF_MASK;
	}
}

/**
 * uart_echo
 * UART Echo function
 * Works with IRQ and Polling both
 */
void uart_echo(void)
{
#if defined(ECHO_POLLING)
	uint8_t temp = uart_getchar();
	uart_putchar(temp);
#elif defined(ECHO_IRQN)
	if(cb_check_empty(rx_buffer) != CB_buffer_empty)
	{
		uint8_t temp = uart_getchar();
		uart_putchar(temp);
	}
#endif
}

// Interrupt Handler for Interrupt Based Operation
#if defined(APP_IRQN) || defined(ECHO_IRQN)
/**
 * UART0_IRQHandler
 * UART Interrupt Service Routine
 */
void UART0_IRQHandler(void)
{
	// Receive interrupt
	if(UART0->S1 & UART0_S1_RDRF_MASK && UART0->C2 & UART0_C2_RIE_MASK)
	{
		uint8_t data = 0;
```

```
            uart_rx(&data);
            cb_add_item(rx_buffer, data);
            UART0->S1 &= ~UART0_S1_OR_MASK;
            Turn_On_Only_LED(Blue);
      }

      else if(UART0->S1 & UART0_S1_TDRE_MASK && UART0->C2 & UART0_C2_TIE_MASK)
      {
            uint8_t data = 0;
            uint8_t x = cb_remove_item(tx_buffer, &data);
            if(x != CB_buffer_empty)
            {
                  uart_tx(&data);
                  while(UART0->S1 & UART0_S1_TC_MASK);
            }
            Turn_On_Only_LED(Green);
            UART0->C2 &= ~UART_C2_TIE_MASK;
      }

      // Overrun Error Interrupt
      if(UART0->S1 & UART0_S1_OR_MASK)
      {
            system_info.or_flag = 1;
            Turn_On_Only_LED(Red);
      }

      // Noise Error Interrupt
      if(UART0->S1 & UART0_S1_NF_MASK)
      {
            system_info.ne_flag = 1;
            Turn_On_Only_LED(Red);
      }

      // Framing Error Interrupt
      if(UART0->S1 & UART0_S1_FE_MASK)
      {
            system_info.fe_flag = 1;
            Turn_On_Only_LED(Red);
      }

      // Parity Error Interrupt
      if(UART0->S1 & UART0_S1_PF_MASK)
      {
            system_info.pe_flag = 1;
            Turn_On_Only_LED(Red);
      }
}
#endif

/**
 * uart_putchar
 * Sends a character to UART
 * @param
 *          ch - 8 bit character
 *    Works with IRQ and Polling
 */
void uart_putchar(uint8_t ch)
{
```

```
#if defined(APP_POLLING) || defined(ECHO_POLLING)
      uart_tx(&ch);
#elif defined(APP_IRQN) || defined(ECHO_IRQN)
      cb_add_item(tx_buffer, ch);
#endif
}

/**
 * uart_getchar
 * Gets a character from UART
 * @return
 *          return 8 bit data
 * Works with IRQ and Polling
 */
uint8_t uart_getchar (void)
{
      uint8_t temp;
#if defined(APP_POLLING) || defined(ECHO_POLLING)
    uart_rx(&temp);
#elif defined(APP_IRQN) || defined(ECHO_IRQN)
    cb_remove_item(rx_buffer, &temp);
#endif
    return temp;

}
/**
 * put_string
 * Prints string
 * @param
 *          string - string to print
 */
void put_string(const char* string)
{
    uint8_t ch;
    char* pointer = (char *) string;

    // Till the end of string
    while((ch = *pointer) != '\0')
    {
        uart_putchar(ch);
        pointer++;
    }
}

/**
 * vpprintf
 * My implementation of va_list based printf
 * @param
 *          fmt - string with formatting
 * @param
 *          args - va_list
 */
void vpprintf(const char* fmt, va_list args)
{
      char* string = NULL;
      string = (char *) malloc(sizeof(char) * 200);
      vsprintf(string, fmt, args);
      put_string(string);
```

```
	free(string);
}

/**
 * pprintf
 * My implementation of printf
 * @param
 *          fmt - string with formatting
 */
void pprintf(const char* fmt, ...)
{
	va_list(args);
	va_start(args, fmt);
	vpprintf(fmt, args);
}
```

2. uart.h

```
/**
  * File Name           - uart.h
  * Description   - header for uart.c
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */

#ifndef UART_H_
#define UART_H_
// Include Files
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdarg.h>
#include "MKL25Z4.h"
#include "common.h"
#include "led_control.h"
#include "errno.h"
#include "logger.h"

// Macros
#define SYS_CLOCK 48000000UL

// Enum for Baud Rate selection
typedef enum {
	baud_115200 = 115200,
	baud_57600 = 57600,
	baud_38400 = 38400,
	baud_19200 = 19200,
	baud_14400 = 14400,
	baud_9600 = 9600,
} UARTBaudRate_t;

// Enum for Parity Setup
typedef enum {
	parity_off = 0x0,
	parity_even = 0x2,
```

```
	parity_odd = 0x3,
} UARTParityMode_t;

// Enum for Stop Bit Configuration
typedef enum {
	single_stop_bit = 0x0,
	double_stop_bit = 0x1,
} UARTStopBit_t;

// Enum for Oversampling Rate
typedef enum {
	OSR_4 = 0x03,
	OSR_5,
	OSR_6,
	OSR_7,
	OSR_8,
	OSR_9,
	OSR_10,
	OSR_11,
	OSR_12,
	OSR_13,
	OSR_14,
	OSR_15,
	OSR_16,
	OSR_17,
	OSR_18,
	OSR_19,
	OSR_20,
	OSR_21,
	OSR_22,
	OSR_23,
	OSR_24,
	OSR_25,
	OSR_26,
	OSR_27,
	OSR_28,
	OSR_29,
	OSR_30,
	OSR_31,
	OSR_32,

} UARTOSRSetting_t;

// Structure for UART Configuration
typedef struct {
	UARTBaudRate_t baud_rate;
	UARTParityMode_t parity;
	UARTStopBit_t stop_bit;
	UARTOSRSetting_t osr;
} UARTConfig_t;

// Enum for UART Status
typedef enum {
	TX_available,
	TX_not_available,
	RX_available,
	RX_not_available,
} UARTStatus_t;
```

```
// Prototype Functions
void uart_init(UARTConfig_t* uart_config);
UARTStatus_t uart_tx_available(void);
void uart_tx_action(uint8_t data);
void uart_tx(uint8_t* data);
UARTStatus_t uart_rx_check(void);
uint8_t uart_rx_action(void);
void uart_rx(uint8_t* data);
void uart_enable_irq(void);
void uart_echo(void);
void uart_tx_handler(void);
void uart_error_handler(void);

// Prototype Functions for Print operations
uint8_t uart_getchar(void);
void uart_putchar(uint8_t ch);
void vpprintf(const char* fmt, va_list args);
void pprintf(const char* fmt, ...);

#endif /* UART_H_ */
```

Test

1. test.c

```
/**
  * File Name           - test.c
  * Description   - contains test cases for the program
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */

#include "test.h"

system_t system_info = {0, 0, 0, 0, 0, 0};

static inline void delay(void)
{
      for(volatile int i = 10000; i > 0; i--);
}
/*
 * Function - unit_tests
 * Brief - Executes unit tests
 */
void unit_tests(void)
{
      logger.Init();
      logger.Set_Log_Level(lTest);
      UARTConfig_t uart_config = {
                  baud_115200,
                  parity_off,
                  single_stop_bit,
                  OSR_32,
      };
      uart_init(&uart_config);
      UCUNIT_TestcaseBegin("Starting Test Cases\n\r");
```

```
UCUNIT_TestcaseBegin("Test Case for UART\n\r");
pprintf("Here is me!\n\r");
UCUNIT_TestcaseEnd();


UCUNIT_TestcaseBegin("Test Case for Circular Buffer\n\r");
circular_buffer_t* buff1 = NULL;
buff1 = cb_init_buffer(100);
UCUNIT_CheckIsEqual(CB_buffer_initialized, cb_verify_init(buff1));
UCUNIT_CheckIsEqual(100, buff1->length);
UCUNIT_CheckIsEqual(0, buff1->count);
UCUNIT_TestcaseEnd();


uint8_t* data = (uint8_t *) malloc(1);
*data = 0x55;
UCUNIT_TestcaseBegin("Test Case for Circular Buffer Add Item\n\r");
cb_add_item(buff1, 2);
UCUNIT_CheckIsEqual(100, buff1->length);
UCUNIT_CheckIsEqual(1, buff1->count);
UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->head);
UCUNIT_CheckIsEqual(buff1->pointer, buff1->tail);
printf("Data is %d\n\r", *buff1->tail);
UCUNIT_TestcaseEnd();


UCUNIT_TestcaseBegin("Test Case for Circular Buffer Remove Item\n\r");
cb_remove_item(buff1, data);
UCUNIT_CheckIsEqual(100, buff1->length);
UCUNIT_CheckIsEqual(0, buff1->count);
UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->head);
UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->tail);
UCUNIT_TestcaseEnd();

// Filling the buffer
for(uint16_t i = 0; i < 100; i++)
{
      cb_add_item(buff1, i);
}


UCUNIT_TestcaseBegin("Test Case for Circular Buffer Full\n\r");
UCUNIT_CheckIsEqual(100, buff1->count);
UCUNIT_CheckIsEqual(CB_buffer_full, cb_add_item(buff1, 5));
UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->head);
UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->tail);
UCUNIT_TestcaseEnd();

// empty circular buffer
for(uint16_t i = 0; i <= 100; i++)
{
      printf("Data %d is %d\n\r", i, *buff1->tail);
      cb_remove_item(buff1, data);
}


UCUNIT_TestcaseBegin("Test Case for Circular Buffer Empty\n\r");
UCUNIT_CheckIsEqual(0, buff1->count);
```

```
	UCUNIT_CheckIsEqual(CB_buffer_empty, cb_remove_item(buff1, data));
	UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->head);
	UCUNIT_CheckIsEqual(buff1->pointer + 1, buff1->tail);
	UCUNIT_TestcaseEnd();


	UCUNIT_TestcaseBegin("Testing LED Functions\n\r");
	LED_Init();
	Turn_On_Only_LED(Red);
	for(volatile int i = 65535; i > 0; i--);
	Turn_On_Only_LED(Green);
	for(volatile int i = 65535; i > 0; i--);
	Turn_On_Only_LED(Blue);
	for(volatile int i = 65535; i > 0; i--);
	Turn_Off_LEDs();
	UCUNIT_TestcaseEnd();


	UCUNIT_TestcaseBegin("Testing Logger Functions\n\r");
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	logger.Log_Write(__func__, mStatus, "Testing logger");
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	logger.Log_Write(__func__, mError, "Testing logger");
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	for(volatile int i = 65535; i > 0; i--);
	logger.Log_Write(__func__, mDebug, "Testing logger");
	logger.Log_Write(__func__, mError, "Testing logger");
	UCUNIT_TestcaseEnd();
}

/*
 * Function - Main
 * Brief - Main testing routine
 */
int main(void)
{
	//Initializing board pins
	BOARD_InitBootPins();
	BOARD_InitBootClocks();
	BOARD_InitBootPeripherals();
```

```
	//Calling function to run tests
	unit_tests();

	while(1)
	{
		uart_echo();
	}
	return 0;
}
```

2. test.h

```
/**
  * File Name          - test.h
  * Description   - header file for test.c
  * Author             - Atharva Nandanwar
  * Tools              - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code     -
  * URL                      -
  */

#ifndef TEST_TEST_H_
#define TEST_TEST_H_

// Include Files
#include <stdint.h>

#include "pin_mux.h"
#include "peripherals.h"
#include "clock_config.h"
#include "board.h"

// Includes for test functions
#include "System.h"
#include "uCUnit.h"

// Includes for functions
#include "logger.h"
#include "errno.h"
#include "uart.h"
#include "circular_buffer.h"
#include "led_control.h"
#include "common.h"

void application(void);
void print_report(uint8_t *char_array);

#endif /* TEST_TEST_H_ */
```

Main Routine

1. common.h

```
/**
  * File Name          - common.h
```

```c
  * Description   - file commonly to be included across system
  * Author              - Atharva Nandanwar
  * Tools               - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code      -
  * URL                       -
  */

#ifndef COMMON_H_
#define COMMON_H_
// Include Files
#include <stdint.h>
#include "MKL25Z4.h"
#include "circular_buffer.h"

// RX and TX buffer globally available
extern circular_buffer_t* rx_buffer;
extern circular_buffer_t* tx_buffer;


// Structure to hold system status
typedef struct {
      uint8_t tx_ready_flag;
      uint8_t rx_ready_flag;
      uint8_t or_flag;
      uint8_t ne_flag;
      uint8_t fe_flag;
      uint8_t pe_flag;
} system_t;

// Macros for system info
#define TX_FLAG_RESET   0
#define RX_FLAG_RESET   0
#define OR_FLAG_RESET   0
#define NE_FLAG_RESET   0
#define FE_FLAG_RESET   0
#define PE_FLAG_RESET   0

// Extern for system_info global variable
extern system_t system_info;



/*-----------------------------*/
/*|     Application Mode      |*/
/*-----------------------------*/
/*  #define ECHO_POLLING      1  */
/*  #define APP_POLLING       1  */
/*  #define ECHO_IRQN         1  */
/*    #define APP_IRQN        1  */
/*-----------------------------*/
/*|     Application Mode      |*/
/*-----------------------------*/


#endif /* COMMON_H_ */
```

2. main.c

```c
/*
 * File - main.c
 * Brief -
 * Author - Atharva Nandanwar
 * University of Colorado Boulder
 * Principles of Embedded Software
 */

#include "main.h"

system_t system_info = {
            TX_FLAG_RESET,
            RX_FLAG_RESET,
            OR_FLAG_RESET,
            NE_FLAG_RESET,
            FE_FLAG_RESET,
            PE_FLAG_RESET
};

circular_buffer_t* rx_buffer = NULL;
circular_buffer_t* tx_buffer = NULL;

#if defined(APP_IRQN) || defined(APP_POLLING)
application_t application_data = {
            0,
            NULL,
};
#endif

int main(void)
{
      // All initialization functions - Logging Disabled
      // Due to reliance on UART peripheral
      BOARD_InitBootPins();
      BOARD_InitBootClocks();
      BOARD_InitBootPeripherals();

      // RX/TX buffer initialization
      rx_buffer = cb_init_buffer(500);
      tx_buffer = cb_init_buffer(500);

      // Initializing Logger
      logger.Init();
#ifdef DEBUG
      logger.Set_Log_Level(lDebug);
#endif
#ifdef RUN
      logger.Set_Log_Level(lNormal);
#endif

      // Initializing UART
      UARTConfig_t uart_config = {
                  baud_115200,
                  parity_off,
                  single_stop_bit,
```

```
                  OSR_32,
      };
      uart_init(&uart_config);

      // LED Initialization - Logging can be used from here
      LED_Init();
      logger.Log_Write(__func__, mStatus, "Starting Program");

#if defined(APP_IRQN) || defined(APP_POLLING)
      application_init();
#endif

#if defined(APP_IRQN) || defined(ECHO_IRQN)
      uart_enable_irq();
#endif

#if defined(APP_IRQN)
      logger.Log_Write(__func__, mStatus, "Starting in Application Mode in IRQ");
#elif defined(ECHO_IRQN)
      logger.Log_Write(__func__, mStatus, "Starting in Echo Mode in IRQ");
#elif defined(APP_POLLING)
      logger.Log_Write(__func__, mStatus, "Starting in Application Mode in
Polling");
#elif defined(ECHO_POLLING)
      logger.Log_Write(__func__, mStatus, "Starting in Echo Mode in Polling");
#endif

      while(1)
      {
#if defined(APP_IRQN)
            application();
            uart_tx_handler();
            uart_error_handler();
#elif defined(ECHO_IRQN)
            uart_echo();
            uart_tx_handler();
#elif defined(APP_POLLING)
            application();
#elif defined(ECHO_POLLING)
            uart_echo();
#endif
      }
}

// Application Mode Functions
#if defined(APP_POLLING) || defined(APP_IRQN)

/**
 * reset_array
 * Resets character counts
 */
static inline void reset_array(uint8_t* char_array)
{
      if(logger.Get_Log_Level() == lDebug)
            logger.Log_Write(__func__, mDebug, "Resetting character count");
      for(int i = 0; i < 128; i++)
      {
            *(char_array + i) = 0;
```

```
	}
}

/**
 * application_init
 * Finishes initialization for application
 */
void application_init(void)
{
	// Make an array and reset it
	application_data.char_array = (uint8_t *) malloc(128);
	reset_array(application_data.char_array);
}

/**
 * application
 * Code for application
 */
void application(void)
{
// Different implementations for polling and IRQ, IRQ is based on circular buffers
#if defined(APP_POLLING)
	*(application_data.char_array + uart_getchar()) += 1;
	application_data.count++;
	if(application_data.count % 50 == 0)
	{
			print_report(application_data.char_array);
			reset_array(application_data.char_array);
			application_data.count = 0;
	}
#elif defined(APP_IRQN)
	if(cb_check_empty(rx_buffer) != CB_buffer_empty)
	{
		*(application_data.char_array + uart_getchar()) += 1;
		application_data.count++;
		if(application_data.count % 50 == 0)
		{
			print_report(application_data.char_array);
			reset_array(application_data.char_array);
		}
	}
#endif
}

// Print report function
void print_report(uint8_t *char_array)
{
	logger.Log_Write(__func__, mStatus, "Printing Debug Report");
	pprintf("UART Report\n\r");
	// Look up table for escape characters in ascii
	char* LookUp[34] = {"NULL", "SOH", "STX", \
			"ETX", "EOT", "ENQ", "ACK", "BEL", \
			"BS", "HT", "LF", "VT", "FF", "CR", \
			"SO", "SI", "DLE", "DC1", "DC2", "DC3", \
			"DC4", "NAK", "SYN", "ETB", "CAN", \
			"EM", "SUB", "ESC", "FS", "GS", "RS", \
			"US", "SPACE", "DEL"
	};
```

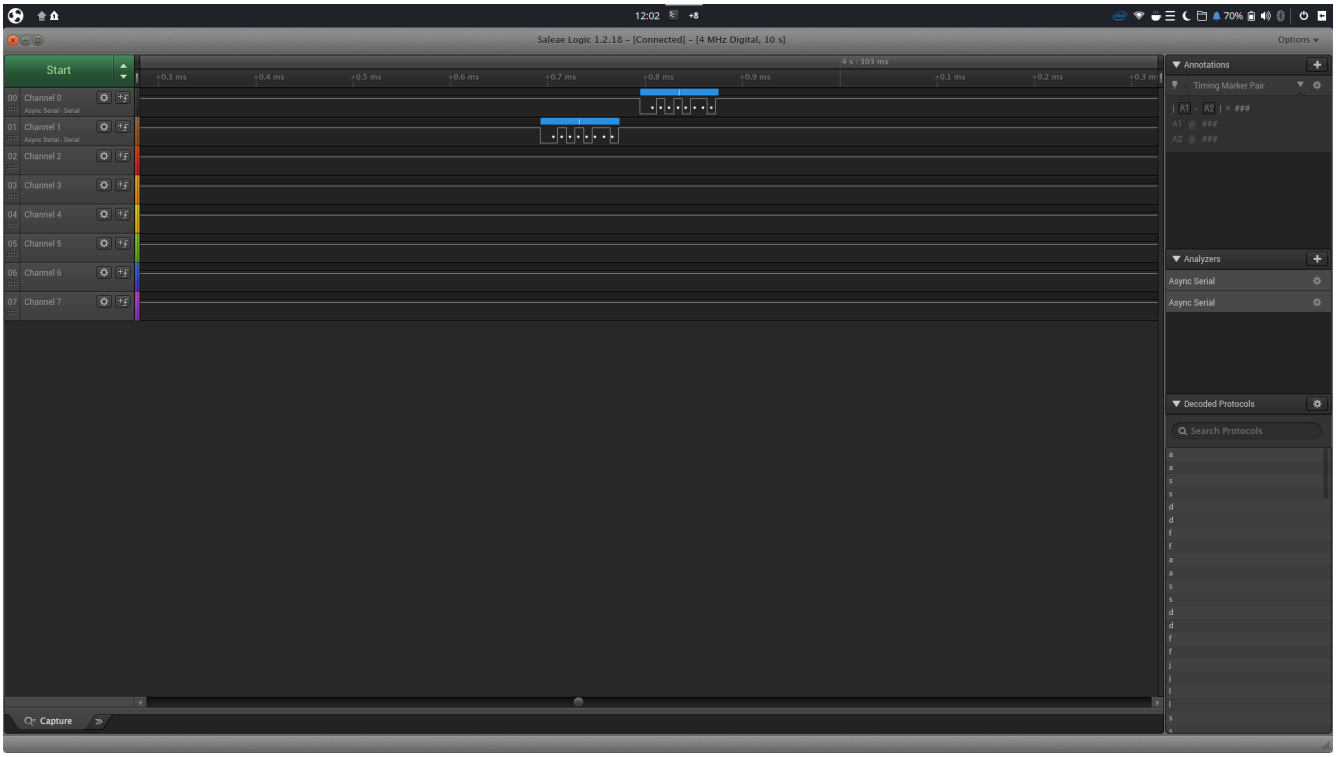
```
	for(uint8_t i = 0; i < 128; i++)
	{
		if(*(char_array + i))
		{
			// Print escape characters from Lookup table
			if(i <= 0x20 || i == 0x7F)
			{
				pprintf("%-5s - %3d\n\r", LookUp[i], *(char_array + i));
			}
			// Print rest of them normally
			else
			{
				pprintf("%-5c - %3d\n\r", i, *(char_array + i));
			}
		}
	}
	pprintf("UART Report Ends\n\r");
}
#endif
```

3. main.h

```
/**
  * File Name          - main.h
  * Description   -
  * Author             - Atharva Nandanwar
  * Tools              - GNU C Compiler / ARM Compiler Toolchain
  * Leveraged Code     -
  * URL                      -
  */

#ifndef MAIN_H_
#define MAIN_H_

#include "clock_config.h"
#include "peripherals.h"
#include "board.h"
#include "pin_mux.h"
#include "MKL25Z4.h"
#include "common.h"
#include "logger.h"
#include "uart.h"
#include "led_control.h"
#include "circular_buffer.h"

#if defined(APP_POLLING) || defined(APP_IRQN)

static inline void reset_array(uint8_t* char_array);
void print_report(uint8_t *char_array);
void application_init(void);
void application(void);

typedef struct {
	uint8_t count;
	uint8_t* char_array;
} application_t;
```

```
#endif

#endif /* MAIN_H_ */
```

UART Traces from Logic Analyzer



Sending ‘j’ and receiving ‘j’

FMEA

Item Number	Function/Variable	Potential Failure Mode	Potential Failure Effect	Severity	Potential Cause	Occurence	Controls	Detection	Risk Priority Number
1	UART0_IRQHandler	Interrupts firing too often	Disrupts echo program flow	7	TX flag is always set when TX buffer is empty	10	Disabling TX Interrupt till we want to transmit the characters	1	70
2	Generic Software Integration	Unable to integrate the software because subsystems aren’t compatible	Program breaks without showing functionality	10	Function prototypes are not compatible with each other	10	Global buffers	3	300
3	UART0_IRQHandler	Receive Interrupt Not firing	Program works till a certain point and then stops	9	Overrun flag is not being set	10	No solution found yet	8	540
4	printf	Hardfault when sending data – polling	Printing not happening	8	LED subroutine called before initialization	6	removing logging from LED subroutines	5	240
5	Circular Buffer	Circular Buffer wrap around error	Buffer is unable to fill more data into the allocated memory	7	Wrap around code not working	5	Used modulo logic to wrap around and contain the address value between buffer start and end	4	140
6	Circular Buffer	Data fill	Unable to fill data into buffer	4	Calling function which checks the buffer empty	3	Did extensive add item testing once program logic was changed	2	24