README

Principles of Embedded Software Project 6

Title: FreeRTOS, DAC, ADC, and DMA

Name: Atharva Nandanwar

This repository contains source files for Principles of Embedded Software Project 6

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. dac/dac.c/h - dac functions
8. adc/adc.c/h - adc functions
9. dma/dma.c/h - dma functions
10. lookup_generator/lookup.c/h - function for generating lookup table

---

Observations:

1. FreeRTOS makes task management easy. The functions provided by FreeRTOS as APIs, if read properly can be used to make a robust system.
2. Semaphores came off as something wonderful that we can use to block access to certain part of code. I have used it for LED access.
3. Time management, Priorities, Semaphores would make the system really complicated to handle, however the system would also get really powerful by using FreeRTOS.

Installation/Execution Notes:

Compiler - gcc-arm-none-eabi

To enable detailed logging Debug mode can be compiled by added **DEBUG_CODE** in preprocessor defines.

Source Code

main.c

```c
/**
 * File -    main.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Main file
 */

#include "main.h"
;/* TODO: insert other definitions and declarations here. */
#define DACTASKPERIOD (100 / portTICK_PERIOD_MS)
#define ADCTASKPERIOD (100 / portTICK_PERIOD_MS)

// Used to log run number
uint8_t run_number = 0;

// Used to put lookup table values for DAC
uint16_t buffer[50];

// Circular buffer for ADC
circular_buffer_t* adc_buffer = NULL;

// Buffer for DSP
uint16_t dsp_buffer[64];

// Index for DAC
uint8_t dac_index = 0;

// Task Handlers for all the tasks
TaskHandle_t DAC_Task_Handler;
TaskHandle_t ADC_Task_Handler;
TaskHandle_t DSP_Task_Handler;

// Timer Handler
TimerHandle_t xLoggerTimer;
SemaphoreHandle_t xLED_Semaphore;

/*
 * @brief   Application entry point.
 */
int main(void) {
```

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

    logger.Init();
#ifdef DEBUG_CODE
    logger.Set_Log_Level(lDebug);
#else
    logger.Set_Log_Level(lNormal);
#endif
    logger.Log_Write(__func__, mError, "Starting Program");
    LED_Init();

    // Task Creation
    if(logger.Get_Log_Level() == lDebug)
      logger.Log_Write(__func__, mDebug, "Creating Tasks");
    // DAC Task on Max priority as it is critical to output Sine Wave
    xTaskCreate(DAC_Task, "DAC Task", configMINIMAL_STACK_SIZE + 50, \
            NULL, (configMAX_PRIORITIES - 1), &DAC_Task_Handler);
    // ADC Task on Minimum priority as it is not so critical task
    xTaskCreate(ADC_Task, "ADC Task", configMINIMAL_STACK_SIZE + 200, \
            NULL, (configMAX_PRIORITIES - 3), &ADC_Task_Handler);
    // DSP Task as it shall run after DMA Transfer is complete
    xTaskCreate(DSP_Task, "DSP Task", configMINIMAL_STACK_SIZE + 300, \
            NULL, (configMAX_PRIORITIES - 2), &DSP_Task_Handler);

    // Semaphore Creation
    if(logger.Get_Log_Level() == lDebug)
      logger.Log_Write(__func__, mDebug, "Creating Semaphores");
    xLED_Semaphore = xSemaphoreCreateCounting(5, 0);

    // Peripheral Initialization Routines
    dac_lookup_init(buffer);
    dac_init();
    adc_buffer = cb_init_buffer(64);
    adc_init();
    // DSP Buffer Initialization
    for(uint8_t index = 0; index < 64; index++)
      dsp_buffer[index] = 0;
    dma_init();

    // Starting Scheduler
      if(logger.Get_Log_Level() == lDebug)
            logger.Log_Write(__func__, mDebug, "Starting Scheduler");
    vTaskStartScheduler();
    while(1);
}

/*
 * Function - DAC_Task
 * Brief     - Task for performing DAC operations
```

```
 */
void DAC_Task(void* parameters)
{
	// Tick count used to block the task for certain amount of time
	TickType_t PreviousWakeTime = xTaskGetTickCount();
	for (;;)
	{

		// Output buffer values continuously
		dac_out(*(buffer + dac_index));
		dac_index++;
		if(dac_index == 50)
			dac_index = 0;

		// Access LED if not used by other systems
		if(uxSemaphoreGetCount(xLED_Semaphore) == 0)
		{
			Turn_Off_LED(Blue);
			Toggle_LED(Green);
		}
		else
			xSemaphoreTake(xLED_Semaphore, 0);

		// Block the task for 100 ms
		vTaskDelayUntil(&PreviousWakeTime, DACTASKPERIOD);
	}
}


/*
 * Function - ADC_Task
 * Brief    - Task for performing ADC operations
 */
void ADC_Task(void* parameters)
{

	TickType_t PreviousWakeTime = xTaskGetTickCount();
	for (;;)
	{
		// Fill the circular buffer with ADC values
		if(CB_buffer_full == cb_add_item(adc_buffer, adc_value()))
		{
			run_number++;
			logger.Log_Write(__func__, mStatus, "Run Number %d", run_number);
			logger.Log_Write(__func__, mStatus, "DMA Transfer Started");

			// Semaphore for LED Control
			xSemaphoreGive(xLED_Semaphore);
			xSemaphoreGive(xLED_Semaphore);
			xSemaphoreGive(xLED_Semaphore);
			xSemaphoreGive(xLED_Semaphore);
			Turn_Off_LED(Green);
			Turn_On_LED(Blue);

			// Start DMA Transfer
```

```
                dma_transfer(adc_buffer->pointer, dsp_buffer);
                cb_empty_buffer(adc_buffer);
            }

            // Block the task for another 100 ms
            vTaskDelayUntil(&PreviousWakeTime, ADCTASKPERIOD);
        }
}

/*
 * Function - DSP_Task
 * Brief    - Task for performing DSP operations on buffer
 */
void DSP_Task(void* parameters)
{
      for (;;)
      {

            // DSP Operations
            uint32_t mean = 0;
            uint16_t max = 0;
            uint16_t min = 0xffff;
            uint32_t standard_deviation = 0;
            float temp_mean = 0;
            float temp_sd = 0;
            for(uint8_t index = 0; index < 64; index++)
            {
                  volatile uint16_t value = *(dsp_buffer + index);
                  mean += value;
                  if(max < value)
                  {
                        max = value;
                  }
                  if(min > value)
                  {
                        min  = value;
                  }
            }
            temp_mean = mean / 64;
            mean = temp_mean * 100;

            for(uint8_t index = 0; index < 64; index++)
            {
                  volatile uint16_t value = *(dsp_buffer + index);
                  temp_sd += pow((value - temp_mean), 2);
            }
            temp_sd = temp_sd / 64;
            temp_sd = sqrt(temp_sd);
            standard_deviation = temp_sd * 100;

            // Temporary pointer used to point at string from float_string function
            char* temp = NULL;
            logger.Log_Write(__func__, mStatus, "Mean - %s", temp =
float_string(mean));
            if(mean)
```

```
                    free(temp);         // freeing the pointer as it was dynamically
allocated
              logger.Log_Write(__func__, mStatus, "Max - %d", max);
              logger.Log_Write(__func__, mStatus, "Min - %d", min);
              logger.Log_Write(__func__, mStatus, "SD - %s", temp =
float_string(standard_deviation));
              if(mean)
                    free(temp);

              // Exit when Run Number 5 is over
              if(run_number == 5)
              {
                    logger.Log_Write(__func__, mError, "Exiting Program");
                    vTaskDelete(DAC_Task_Handler);
                    vTaskDelete(ADC_Task_Handler);
                    vTaskDelete(DSP_Task_Handler);
              }

              vTaskSuspend(NULL);
       }
}

/*
 * Function - DMA_Callback
 * Brief     - Function to address DMA Complete Interrupt
 */
void DMA_Callback(dma_handle_t *handle, void *param)
{
       // Indicate DMA Transfer Complete
       logger.Log_Write(__func__, mStatus, "DMA Transfer Finished");

       // Resume the DSP Task
       xTaskResumeFromISR(DSP_Task_Handler);

       // Tell the DMA Peripheral that the Transfer is done
       DMA0->DMA[0].DSR_BCR |= DMA_DSR_BCR_DONE(1);
}

/*
 * Function - float_string
 * Parameter - uint32_t value - float * 100 value packed into uint32_t
 * Brief     - function to return string output for a packed float value
 * Return    - Numerical String
 */
char* float_string(uint32_t value)
{
       // Temporary variable to hold value
       uint32_t temp = value;

       // Buffer to hold individual values
       uint8_t buffer[12];
       uint8_t index = 0;

       // Logic for zero
       if(value == 0)
```

```
	{
		return "0.00";
	}

	// Extract each number
	while(temp != 0)
	{
		buffer[index] = temp % 10;
		temp = temp / 10;
		index++;
	}

	// Character buffer that will hold the numerical string
	char* char_buffer = (char *) malloc(sizeof(char) * 13);
	uint8_t char_index = 0;
	index--;
	while(index - char_index >= 2)
	{
		*(char_buffer + char_index) = buffer[index - char_index] + '0';
		char_index++;
	}

	// For values less than 1.00
	if(value < 100)
	{
		char_buffer[0] = '0';
		char_index++;
	}

	// Decimal point and decimal values
	*(char_buffer + char_index) = '.';
	char_index++;
	*(char_buffer + char_index) = buffer[1] + '0';
	char_index++;
	*(char_buffer + char_index) = buffer[0] + '0';
	char_index++;

	// Terminate with NULL
	*(char_buffer + char_index) = 0;

	// Return the string
	return char_buffer;
}
```

main.h

```
/**
 * File -    main.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Main file
 */
```

```h
#ifndef MAIN_H_
#define MAIN_H_
// Library includes
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

// Program Includes
#include "lookup.h"
#include "dac.h"
#include "adc.h"
#include "dma.h"
#include "led_control.h"
#include "logger.h"
#include "circular_buffer.h"

void DAC_Task(void* parameters);
void ADC_Task(void* parameters);
void DSP_Task(void* parameters);
char* float_string(uint32_t value);

#endif /* MAIN_H_ */
```

adc/adc.c

```c
/**
 * File -    adc.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * ADC Peripheral Implementation
 */

#include "adc.h"

/*
 * adc_init
 * Initializes ADC Peripherals
 */
void adc_init(void)
{
	if(logger.Get_Log_Level() == lDebug)
		logger.Log_Write(__func__, mStatus, "ADC Initialization Started");

	// Turn on ADC Gating Clock
	SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;
	// Port E Pin 20 Setting
	PORTE->PCR[20] |= PORT_PCR_MUX(0);
```

```
	// ADC Configuration Settings
	/*
	 * Normal Power Configuration
	 * Clock Divide Set at 8
	 * Long Sample Time
	 * 16-bit Conversion Mode
	 * Bus Clock / 2 as Clock Source
	 */
	ADC0->CFG1 = 0x7D;


	ADC0->CFG2 |= ADC_CFG2_ADHSC(1);

	/*
	 * Setting up Conversion Trigger as software
	 * Disable Compare Function
	 */
	ADC0->SC2 = 0;

	/*
	 * Turn Off Calibration
	 * Hardware Average Function On
	 * 4 Samples for Averaging
	 */
	ADC0->SC3 |= ADC_SC3_ADCO_MASK |
	             ADC_SC3_AVGE_MASK |
	             ADC_SC3_AVGS(0x00);

	logger.Log_Write(__func__, mStatus, "ADC Initialized");
}

/*
 * adc_value
 * Starts ADC conversion, and waits till the conversion is over
 * @return
 *           returns ADC value
 */
uint16_t adc_value(void)
{
	// Write to SC1 to start conversion
	ADC0->SC1[0] = 0;

	// Wait for conversion to complete
	while(!(ADC0->SC1[0] & ADC_SC1_COCO_MASK));

	// Return ADC value
	return ADC0->R[0];
}
```

adc/adc.h

```c
/**
 * File -    adc.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * ADC Peripheral Functions
 */

#ifndef ADC_H_
#define ADC_H_

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

// Prototype Functions
void adc_init(void);
uint16_t adc_value(void);

#endif /* ADC_H_ */
```

circular_buffer/circular_buffer.c

```c
/**
 * File -    circular_buffer.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Circular Buffer Implementation
 */

#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)
{
	if(logger.Get_Log_Level() == lDebug)
		logger.Log_Write(__func__, mDebug, "Circular Buffer Initialized");
	// 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 = (uint16_t *) malloc(length * 2);
```

```
	// 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, uint16_t item)
{
	// Flag error
	if(buffer == NULL)
	{
		return CB_buffer_error;
	}

	// 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;
```

```
	}
	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, uint16_t* data)
{
	// Flag error
	if(buffer == NULL)
	{
		return CB_buffer_error;
	}

	// 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;
	}
	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;
}

/**
 * cb_empty_buffer
 * Empties a circular buffer
 * @param
 *		pointer to circular buffer
 */
void cb_empty_buffer(circular_buffer_t* buffer)
{
	buffer->count = 0;
}
```

circular_buffer/circular_buffer.h

```
/**
 * File -	circular_buffer.h
 * Author-	Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Circular Buffer Implementation
 */

#ifndef CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_
#define CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_

// Include files
#include <stdint.h>
#include <stdlib.h>
#include "MKL25Z4.h"
#include "logger.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 {
	uint16_t* pointer;
	uint16_t* head;
	uint16_t* tail;
	uint16_t length;
	uint16_t count;
} circular_buffer_t;

// Prototype functions
CB_status_t cb_add_item(circular_buffer_t* buffer, uint16_t item);
CB_status_t cb_remove_item(circular_buffer_t* buffer, uint16_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);
void cb_empty_buffer(circular_buffer_t* buffer);

#endif /* CIRCULAR_BUFFER_CIRCULAR_BUFFER_H_ */
```

dac/dac.c

```
/**
 * File -    dac.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * DAC Peripheral Functions
 */

#include "dac.h"

/*
 * dac_init
 * Initializes DAC Peripherals
 */
void dac_init(void)
{
	if(logger.Get_Log_Level() == lDebug)
		logger.Log_Write(__func__, mDebug, "DAC Initialization Started");

	// Turn on Gating Clock
	SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;
	SIM->SCGC6 |= SIM_SCGC6_DAC0_MASK;
```

```
	// Set up Port Pins
	PORTE->PCR[30] |= PORT_PCR_MUX(0);

	DAC0->C0 |= DAC_C0_DACEN_MASK |		// Enable the DAC Module
			DAC_C0_DACTRGSEL_MASK |		// Select software trigger
			DAC_C0_DACRFS_MASK;		// Selecting VCCA

	// Buffer Normal Mode
	DAC0->C1 |= DAC_C1_DACBFMD_MASK;

	// DAC Buffer Settings OFF
	DAC0->C2 = 0;
	logger.Log_Write(__func__, mStatus, "DAC Initialized");
}

/*
 * dac_out
 * Outputs value in DAC0
 */
void dac_out(uint16_t data)
{
	DAC0->DAT->DATH = (data & 0xF00) >> 8;
	DAC0->DAT->DATL = data & 0x0FF;
}
```

dac/dac.h

```
/**
 * File -    dac.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * DAC Peripheral Functions
 */

#ifndef DAC_DAC_H_
#define DAC_DAC_H_

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

// Prototype Functions
void dac_init(void);
void dac_out(uint16_t data);

#endif /* DAC_DAC_H_ */
```

dma/dma.c

```
/**
 * File -    dma.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * DMA Functions
 */

#include "dma.h"

// DMA Handle for DMA Operation
dma_handle_t dma_handle;
// DMA Transfer Configuration for DMA Operation
dma_transfer_config_t dma_tx_struct;

/**
 * dma_init
 * Initializes DMA peripheral
 */
void dma_init(void)
{
	if(logger.Get_Log_Level() == lDebug)
		logger.Log_Write(__func__, mDebug, "DAC Initialization Started");
    /* Configure DMAMUX */
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, DMA_CHANNEL, DMA_SOURCE);
    DMAMUX_EnableChannel(DMAMUX0, DMA_CHANNEL);
    /* Configure DMA one shot transfer */
    DMA_Init(DMA0);
    DMA_CreateHandle(&dma_handle, DMA0, DMA_CHANNEL);

    logger.Log_Write(__func__, mStatus, "DMA Initialized");
}

/**
 * dma_transfer
 * Initiates DMA transfer
 */
void dma_transfer(uint16_t* src_buffer, uint16_t* dest_buffer)
{
    DMA_SetCallback(&dma_handle, DMA_Callback, NULL);
    DMA_PrepareTransfer(&dma_tx_struct, src_buffer, sizeof(src_buffer[0]),
dest_buffer, sizeof(dest_buffer[0]), BUFFER_LENGTH * 2,
                        kDMA_MemoryToMemory);
    DMA_SubmitTransfer(&dma_handle, &dma_tx_struct, kDMA_EnableInterrupt);
      DMA_StartTransfer(&dma_handle);
}
```

dma/dma.h

```
/**
 * File -    dma.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * DMA Functions
 */

#ifndef DMA_H_
#define DMA_H_
// Include files
#include <stdint.h>
#include "fsl_dma.h"
#include "fsl_dmamux.h"
#include "logger.h"

// Macros
#define BUFFER_LENGTH (64)
#define DMA_CHANNEL (0)
#define DMA_SOURCE (63)

// Prototype Functions
void dma_init(void);
void dma_transfer(uint16_t* src_buffer, uint16_t* dest_buffer);
void DMA_Callback(dma_handle_t *handle, void *param);

#endif /* DMA_H_ */
```

led_control/led_control.c

```
/**
 * File -    led_control.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * LED Control Library
 */
#include "led_control.h"

/**
 * LED_Init
 * Initialization of LEDs
 */
void LED_Init(void)
{
	if(logger.Get_Log_Level() == lDebug)
		logger.Log_Write(__func__, mDebug, "Starting LED Initialization");

	// 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_LED(Red);
	Turn_Off_LED(Green);
	Turn_Off_LED(Blue);
}

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

/**
 * Turn_Off_LED
 * Turns off specified color LED
 * @param
 *		LED - color of LED
 */
void Turn_Off_LED(led_color_t LED)
{
	switch(LED)
	{
	case Red:
		GPIOB->PSOR |= RED_LED;		// Turn On
		break;
	case Green:
		GPIOB->PSOR |= GREEN_LED;	// Turn On
		break;
	case Blue:
		GPIOD->PSOR |= BLUE_LED;	// Turn On
		break;
	}
```

```
}

/**
 * Toggle_LED
 * Toggled specified color LED
 * @param
 *           LED - color of LED
 */
void Toggle_LED(led_color_t LED)
{
	switch(LED)
	{
	case Red:
		GPIOB->PTOR |= RED_LED;                 // Turn On
		break;
	case Green:
		GPIOB->PTOR |= GREEN_LED;        // Turn On
		break;
	case Blue:
		GPIOD->PTOR |= BLUE_LED;         // Turn On
		break;
	}
}
```

led_control/led_control.h

```
/**
 * File -    led_control.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * LED Control Library
 */
#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;
```

```c
// Prototype functions
void LED_Init(void);
void Turn_On_LED(led_color_t LED);
void Turn_Off_LED(led_color_t LED);
void Toggle_LED(led_color_t LED);

#endif /* LED_CONTROL_LED_CONTROL_H_ */
```

logger/logger.c

```c
/**
 * File -    logger.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Logger Implemenation
 */

#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
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:
		printf("%s", red);
		break;
	case mDebug:
		printf("%s", blue);
		break;
	case mStatus:
		printf("%s", green);
		break;
	}

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

	// Log Level Logic
	switch(thisLogger.Logger_Log_Level)
	{
	case lDebug:
		printf("Debug: ");
		break;
	case lNormal:
		printf("Run:   ");
		break;
	}

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


	// Message print with color termination code
	vprintf(msg, args);
	printf("%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};
```

logger/logger.h

```
/**
 * File -    logger.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Logger Implemenation
 */

#ifndef LOGGER_LOGGER_H_
#define LOGGER_LOGGER_H_

// Include Files
#include <stdio.h>
#include <stdint.h>
#include <errno.h>
#include <stdarg.h>
#include "timestamp.h"

// Log Level and Message Type enums
typedef enum {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_ */
```

logger/errno.c

```c
/**
 * File -    errno.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Error Enum
 */

#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)
	{
	default:
		return "";
		break;
	}
}
```

logger/errno.h

```c
/**
 * File -    errno.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Error Enum
 */

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

// Error/Event Enum
typedef enum {
	StartSchedulerError,
}error_t;

extern error_t errno;

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

```
#endif /* LOGGER_ERRNO_H_ */
```

logger/timestamp.c

```
/**
 * File -    timestamp.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Timestamp Implementation
 */

#include "timestamp.h"

// Global deciseconds count
uint32_t deciseconds = 0;

/*
 * vTimestamp_Init
 * Initializes Time Stamp
 */
void vTimestamp_Init(void)
{
	TimerHandle_t xLoggerTimer;
    xLoggerTimer = xTimerCreate("Logger Timer", (100/ portTICK_PERIOD_MS), pdTRUE,
            (void *) 0, vLoggerTimerCallback);

	xTimerStart(xLoggerTimer, 0);
}

/**
 * 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;
}

/*
 * vLoggerTimerCallback
 * Timer callback function
```

```
 * @param
 *           xLoggerTimer - timer handle for logger timer
 */
void vLoggerTimerCallback(TimerHandle_t xLoggerTimer)
{
	deciseconds++;
}
```

logger/timestamp.h

```
/**
 * File -    timestamp.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Timestamp Header File
 */

#ifndef LOGGER_TIMESTAMP_H_
#define LOGGER_TIMESTAMP_H_
// Include files
#include "MKL25Z4.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "semphr.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);
void vLoggerTimerCallback(TimerHandle_t xLoggerTimer);
#endif /* LOGGER_TIMESTAMP_H_ */
```

lookup_generator/lookup.c

```
/**
 * File -    lookup.c
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Lookup Table Generator
 */
```

```c
#include "lookup.h"

/*
 * Function - dac_lookup_init
 * Parameter - pointer to buffer
 * Brief     - Fills up the buffer with lookup table values
 */
void dac_lookup_init(uint16_t* buffer)
{
    if(logger.Get_Log_Level() == lDebug)
      logger.Log_Write(__func__, mDebug, "Initialization Lookup Table");

      for(uint8_t index = 0; index < 50; index++)
      {
            buffer[index] = (uint16_t)((sin(2*PI* (float) index/(float) 50) + 2) *
(float) 4096 / 3.3);
      }
      logger.Log_Write(__func__, mStatus, "Lookup Table Initialized");
}
```

lookup_generator/lookup.h

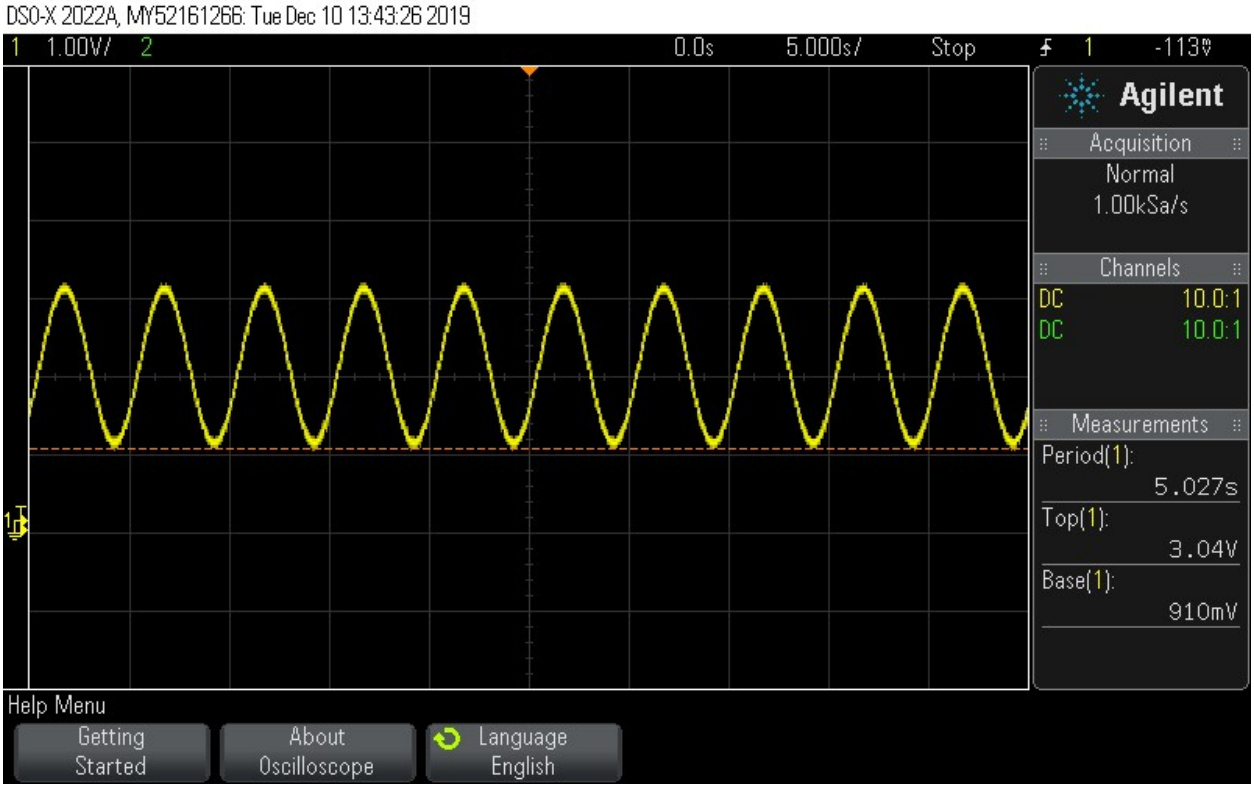
```c
/**
 * File -    lookup.h
 * Author-   Atharva Nandanwar
 * Principles of Embedded Software Project 6
 * University of Colorado Boulder
 * Lookup Table Generator
 */

#ifndef LOOKUP_H_
#define LOOKUP_H_
// Include files
#include <stdint.h>
#include <arm_math.h>
#include "logger.h"

// Prototype Function
void dac_lookup_init(uint16_t* buffer);

#endif /* LOOKUP_H_ */
```

Program 1 Oscilloscope



Program 2