```
/**
 * @file adc.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 15
 *
 * @brief Provides basic ADC functionality
 */

#include "stm32l4xx_ll_dma.h"

#include "adc.h"

/**
 * @brief Initialize DMA for the ADC
 *
 * @param values Array to store ADC values
 * @param numValues Number of values in values array
 *
 * @retval None
 */
static void ADC_Init_DMA(uint16_t * values, int numValues);

/** @brief Delay between ADC end of calibration and ADC enable */
#define ADC_DELAY_CALIB_ENABLE_CPU_CYCLES (LL_ADC_DELAY_CALIB_ENABLE_ADC_CYCLES * 32)

/** @brief Number of possible ADC channels */
#define ADC_NUM_CHANNELS 24

/** @brief Array of ADC channels */
uint32_t ADC_CHANNELS[ADC_NUM_CHANNELS] = {
    LL_ADC_CHANNEL_0,
    LL_ADC_CHANNEL_1,
    LL_ADC_CHANNEL_2,
    LL_ADC_CHANNEL_3,
    LL_ADC_CHANNEL_4,
    LL_ADC_CHANNEL_5,
    LL_ADC_CHANNEL_6,
    LL_ADC_CHANNEL_7,
    LL_ADC_CHANNEL_8,
    LL_ADC_CHANNEL_9,
    LL_ADC_CHANNEL_10,
    LL_ADC_CHANNEL_11,
    LL_ADC_CHANNEL_12,
    LL_ADC_CHANNEL_13,
    LL_ADC_CHANNEL_14,
    LL_ADC_CHANNEL_15,
    LL_ADC_CHANNEL_16,
    LL_ADC_CHANNEL_VREFINT,         // ADC1 Only, Uses Channel 0
    LL_ADC_CHANNEL_TEMPSENSOR,      // ADC1 or ADC3
    LL_ADC_CHANNEL_VBAT,            // ADC1 or ADC3
    LL_ADC_CHANNEL_DAC1CH1_ADC2,    // ADC2 Only
    LL_ADC_CHANNEL_DAC1CH2_ADC2,    // ADC2 Only
    LL_ADC_CHANNEL_DAC1CH1_ADC3,    // ADC3 Only, Uses Channel 14
    LL_ADC_CHANNEL_DAC1CH2_ADC3     // ADC3 Only, Uses Channel 15
};

/** @brief Array of ADC channel ranks */
uint32_t ADC_RANKS[16] = {
    LL_ADC_REG_RANK_1,
    LL_ADC_REG_RANK_2,
    LL_ADC_REG_RANK_3,
    LL_ADC_REG_RANK_4,
    LL_ADC_REG_RANK_5,
    LL_ADC_REG_RANK_6,
    LL_ADC_REG_RANK_7,
    LL_ADC_REG_RANK_8,
    LL_ADC_REG_RANK_9,
    LL_ADC_REG_RANK_10,
    LL_ADC_REG_RANK_11,
    LL_ADC_REG_RANK_12,
    LL_ADC_REG_RANK_13,
```

```
    LL_ADC_REG_RANK_14,
    LL_ADC_REG_RANK_15,
    LL_ADC_REG_RANK_16
};

/**
 * @brief DMA transfer value
 * 0: DMA transfer is not completed @n
 * 1: DMA transfer is completed @n
 * 2: DMA transfer has not been started yet (initial state)
 */
volatile uint8_t dmaTransferStatus = 2;

/**
 * @brief ADC group sequence conversion value
 * 0: ADC group regular sequence conversions are not completed
 * 1: ADC group regular sequence conversions are completed
 */
volatile uint8_t adcConversionStatus = 0;


/* Initialize ADC with DMA*/
void Init_ADC(uint32_t channels, uint16_t * values, int numValues)
{
    int i, j;

    // Check if the ADC is already enabled
    if (LL_ADC_IsEnabled(ADCx_BASE)) {
        return;
    }

    // Initialize DMA for the ADC
    ADC_Init_DMA(values, numValues);

    // Enable ADC interrupts
    // Set the ADC IRQ to a greater priority than the DMA IRQ
    NVIC_SetPriority(ADCx_IRQ, 0);
    NVIC_EnableIRQ(ADCx_IRQ);

    // Enable the ADC clock
    ADCx_CLK_ENABLE();

    // Set the ADC clock
    LL_ADC_SetCommonClock(__LL_ADC_COMMON_INSTANCE(ADCx_BASE), LL_ADC_CLOCK_SYNC_PCLK_DIV2);

    // Enable the internal ADC channels
    LL_ADC_SetCommonPathInternalCh(__LL_ADC_COMMON_INSTANCE(ADCx_BASE), (LL_ADC_PATH_INTERNAL_VRE
FINT | LL_ADC_PATH_INTERNAL_TEMPSENSOR | LL_ADC_PATH_INTERNAL_VBAT));

    // Delay for the internal ADC channels to stablize
    // The temperature sensor takes the longest time to stabalize
    i = ((LL_ADC_DELAY_TEMPSENSOR_STAB_US * (SystemCoreClock / (100000 * 2))) / 10);
    while (i != 0) {
        i--;
    }

    // Set the ADC to have a external timer trigger source
    LL_ADC_REG_SetTriggerSource(ADCx_BASE, LL_ADC_REG_TRIG_EXT_TIM8_TRGO);

    // Set the ADC to trigger on the rising edge
    LL_ADC_REG_SetTriggerEdge(ADCx_BASE, LL_ADC_REG_TRIG_EXT_RISING);

    // Set the ADC to perform a single conversion
    LL_ADC_REG_SetContinuousMode(ADCx_BASE, LL_ADC_REG_CONV_SINGLE);

    // Set the ADC conversion data transfer
    LL_ADC_REG_SetDMATransfer(ADCx_BASE, LL_ADC_REG_DMA_TRANSFER_UNLIMITED);

    // Set the ADC overrun behavior
    LL_ADC_REG_SetOverrun(ADCx_BASE, LL_ADC_REG_OVR_DATA_OVERWRITTEN);

    // Set the ADC sequencer length
```

```
    LL_ADC_REG_SetSequencerLength(ADCx_BASE, numValues - 1);

    // Enable specified ADC channels
    for (i = 0, j = 0; i < ADC_NUM_CHANNELS; i++) {
        if (channels & (1 << i)) {
            LL_ADC_REG_SetSequencerRanks(ADCx_BASE, ADC_RANKS[j], ADC_CHANNELS[i]);
            LL_ADC_SetChannelSamplingTime(ADCx_BASE, ADC_CHANNELS[i], ADCx_SAMPLERATE);
            j++;
        }
        if (j >= numValues || j >= 16) break;
    }

    // Enable ADC interupts for conversion completion
    LL_ADC_EnableIT_EOS(ADCx_BASE);

    // Enable ADC interupts for overrun
    LL_ADC_EnableIT_OVR(ADCx_BASE);

    // Disable the ADC deep power down mode
    LL_ADC_DisableDeepPowerDown(ADCx_BASE);

    // Enable the ADC internal voltage regulator
    LL_ADC_EnableInternalRegulator(ADCx_BASE);

    // Delay for the ADC internal voltage regulator to stabilize
    i = ((LL_ADC_DELAY_INTERNAL_REGUL_STAB_US * (SystemCoreClock / (100000 * 2))) / 10);
    while (i != 0) {
        i--;
    }

    // Start the ADC calibration
    LL_ADC_StartCalibration(ADCx_BASE, LL_ADC_SINGLE_ENDED);

    // Wait for the ADC calibration to finish
    while (LL_ADC_IsCalibrationOnGoing(ADCx_BASE) != 0);

    // Delay to allow ADC calibration to enable
    i = (ADC_DELAY_CALIB_ENABLE_CPU_CYCLES >> 1);
    while (i != 0) {
        i--;
    }

    // Enable the ADC
    LL_ADC_Enable(ADCx_BASE);

    // Wait for the ADC to be ready
    while (LL_ADC_IsActiveFlag_ADRDY(ADCx_BASE) == 0);

    // Start the ADC conversion
    LL_ADC_REG_StartConversion(ADCx_BASE);
}

static void ADC_Init_DMA(uint16_t *values, int numValues)
{
    // Enable DMA interrupts
    // Set the DMA IRQ to a lower priority than the ADC IRQ
    NVIC_SetPriority(DMAx_IRQ, 1);
    NVIC_EnableIRQ(DMAx_IRQ);

    // Enable the DMA clock
    DMAx_CLK_ENABLE();

    // Configure the DMA transfer
    LL_DMA_ConfigTransfer(DMAx_BASE, DMAx_CHANNEL,
        LL_DMA_DIRECTION_PERIPH_TO_MEMORY |
        LL_DMA_MODE_CIRCULAR |
        LL_DMA_PERIPH_NOINCREMENT |
        LL_DMA_MEMORY_INCREMENT |
        LL_DMA_PDATAALIGN_HALFWORD |
        LL_DMA_MDATAALIGN_HALFWORD |
        LL_DMA_PRIORITY_HIGH );
```

```
    // Select ADCx_BASE as the DMA transfer request
    LL_DMA_SetPeriphRequest(DMAx_BASE, DMAx_CHANNEL, LL_DMA_REQUEST_0);

    // Set the DMA transfer address source and destination
    LL_DMA_ConfigAddresses(DMAx_BASE, DMAx_CHANNEL,
        LL_ADC_DMA_GetRegAddr(ADCx_BASE, LL_ADC_DMA_REG_REGULAR_DATA),
        (uint32_t)values, LL_DMA_DIRECTION_PERIPH_TO_MEMORY);

    // Set the DMA transfer size
    LL_DMA_SetDataLength(DMAx_BASE, DMAx_CHANNEL, numValues);

    // Enable DMA transfer complete interrupts
    LL_DMA_EnableIT_TC(DMAx_BASE, DMAx_CHANNEL);

    // Enable DMA transfer error interrupts
    LL_DMA_EnableIT_TE(DMAx_BASE, DMAx_CHANNEL);

    // Enable the DMA channel
    LL_DMA_EnableChannel(DMAx_BASE, DMAx_CHANNEL);
}

/* DMA transfer complete callback */
void ADC_DMA_TransferComplete_Callback(void)
{
    // Update the DMA transfer status
    dmaTransferStatus = 1;

    // Verify the ADC conversion was completed
    if (adcConversionStatus != 1) {
        ADC_DMA_TransferError_Callback();
    }

    // Reset the ADC conversion status
    adcConversionStatus = 0;
}

/* DMA transfer error callback */
void ADC_DMA_TransferError_Callback(void)
{
    // Handle the error
    //while(1);
}

/* ADC group regular end of sequence conversions interruption callback */
void ADC_ConvComplete_Callback(void)
{
    // Update the ADC conversion status
    adcConversionStatus = 1;
}

/* ADC group regular overrun interruption callback */
void ADC_OverrunError_Callback(void)
{
    // Disable ADC overrun interrupts
    LL_ADC_DisableIT_OVR(ADCx_BASE);
}
```

```
/**
 * @file dac.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 29
 *
 * @brief Provides basic DAC functionality
 *
 */

#include "dac.h"

/* Initialize the DAC interface */
int Init_DAC(void)
{
    volatile uint32_t i = 0;

    // Enable the peripheral clock for the DAC
    DACx_CLK_ENABLE();

    // Set the DAC trigger source to software
    LL_DAC_SetTriggerSource(DACx_BASE, DAC_LMD_CHANNEL, LL_DAC_TRIG_SOFTWARE);

    // Configure the DAC output for channel 1
    LL_DAC_ConfigOutput(DACx_BASE, DAC_LMD_CHANNEL, LL_DAC_OUTPUT_MODE_NORMAL, LL_DAC_OUTPUT_BUFF
ER_ENABLE, LL_DAC_OUTPUT_CONNECT_GPIO);

    // Enable the DMA underrun interrupt for channel 1
    //LL_DAC_EnableIT_DMAUDR2(DACx_BASE);

    // Enable the DAC for channel 1
    LL_DAC_Enable(DACx_BASE, DAC_LMD_CHANNEL);

    // Delay to allow the DAC voltage to settle
    i = ((LL_DAC_DELAY_STARTUP_VOLTAGE_SETTLING_US * (SystemCoreClock / (100000 * 2))) / 10);
    while (i != 0)
    {
        i--;
    }

    // Enable the DAC
    LL_DAC_EnableTrigger(DACx_BASE, DAC_LMD_CHANNEL);

    return 0;
}

/* Adjust the DAC output value */
void DAC_SetValue(uint32_t value)
{
    // Set the DAC value
    LL_DAC_ConvertData12RightAligned(DACx_BASE, DAC_LMD_CHANNEL, 0xFFF & value);
    // Tigger a DAC conversion
    LL_DAC_TrigSWConversion(DACx_BASE, DAC_LMD_CHANNEL);
}
```

```
/**
 * @file hw_map.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 September 8
 *
 * @brief Maps hardware peripherals to board specific features
 *
 */


#include "hw_map.h"

/* Initializes hardware required by peripherals */
void HW_Init_GPIO(void)
{
    // Enable the clock for GPIO port A
    LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA);

    //-------------------------
    // ADC Setup
    //-------------------------
    // Enable the clock for GPIO port B
    LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOB);

    // Enable the clock for GPIO port E
    LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOE);


    // Configure the GPIO pin as a ADC input
    LL_GPIO_SetPinMode(ADC_BAT_GPIO_PORT, ADC_BAT_GPIO_PIN, LL_GPIO_MODE_ANALOG);
    // Map GPIO pin to the ADC
    LL_GPIO_EnablePinAnalogControl(ADC_BAT_GPIO_PORT, ADC_BAT_GPIO_PIN);

    // Configure the GPIO pin as a ADC input
    LL_GPIO_SetPinMode(ADC_LMD_GPIO_PORT, ADC_LMD_GPIO_PIN, LL_GPIO_MODE_ANALOG);
    // Map GPIO pin to the ADC
    LL_GPIO_EnablePinAnalogControl(ADC_LMD_GPIO_PORT, ADC_LMD_GPIO_PIN);

    // Configure the GPIO pin as a ADC input
    LL_GPIO_SetPinMode(ADC_RES_GPIO_PORT, ADC_RES_GPIO_PIN, LL_GPIO_MODE_ANALOG);
    // Map GPIO pin to the ADC
    LL_GPIO_EnablePinAnalogControl(ADC_RES_GPIO_PORT, ADC_RES_GPIO_PIN);

    // Configure the GPIO pin as a ADC input
    LL_GPIO_SetPinMode(ADC_CRNT_GPIO_PORT, ADC_CRNT_GPIO_PIN, LL_GPIO_MODE_ANALOG);
    // Map GPIO pin to the ADC
    LL_GPIO_EnablePinAnalogControl(ADC_CRNT_GPIO_PORT, ADC_CRNT_GPIO_PIN);

    //-------------------------
    // DAC Setup
    //-------------------------
    // Configure the GPIO pin as a DAC input
    LL_GPIO_SetPinMode(DAC_LMD_GPIO_PORT, DAC_LMD_GPIO_PIN, LL_GPIO_MODE_ANALOG);

    //-------------------------
    // PWM Setup
    //-------------------------
    // Enable the peripheral clock of GPIOs
    PWMx_GPIO_CLK_ENABLE();

    // GPIO TIM configuration
    LL_GPIO_SetPinMode(PWMx_GPIO_PORT, PWMx_GPIO_PIN, LL_GPIO_MODE_ALTERNATE);
    PWMx_SET_GPIO_AF();
    LL_GPIO_SetPinSpeed(PWMx_GPIO_PORT, PWMx_GPIO_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinOutputType(PWMx_GPIO_PORT, PWMx_GPIO_PIN, LL_GPIO_OUTPUT_PUSHPULL);
    LL_GPIO_SetPinPull(PWMx_GPIO_PORT, PWMx_GPIO_PIN, LL_GPIO_PULL_NO);

    // Enable the timer peripheral clock
    PWMx_CLK_ENABLE();

    //-------------------------
```

```
    // USART Setup
    //-------------------------
    // Enable the peripheral clock of GPIO Port
    USARTx_GPIO_CLK_ENABLE();

    // Configure Tx Pin
    LL_GPIO_SetPinMode(USARTx_TX_GPIO_PORT, USARTx_TX_PIN, LL_GPIO_MODE_ALTERNATE);
    USARTx_SET_TX_GPIO_AF();
    LL_GPIO_SetPinSpeed(USARTx_TX_GPIO_PORT, USARTx_TX_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinOutputType(USARTx_TX_GPIO_PORT, USARTx_TX_PIN, LL_GPIO_OUTPUT_PUSHPULL);
    LL_GPIO_SetPinPull(USARTx_TX_GPIO_PORT, USARTx_TX_PIN, LL_GPIO_PULL_UP);

    // Configure Rx Pin
    LL_GPIO_SetPinMode(USARTx_RX_GPIO_PORT, USARTx_RX_PIN, LL_GPIO_MODE_ALTERNATE);
    USARTx_SET_RX_GPIO_AF();
    LL_GPIO_SetPinSpeed(USARTx_RX_GPIO_PORT, USARTx_RX_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinOutputType(USARTx_RX_GPIO_PORT, USARTx_RX_PIN, LL_GPIO_OUTPUT_PUSHPULL);
    LL_GPIO_SetPinPull(USARTx_RX_GPIO_PORT, USARTx_RX_PIN, LL_GPIO_PULL_UP);

    // Enable USART peripheral clock and clock source
    USARTx_CLK_ENABLE();

    // Set clock source
    USARTx_CLK_SOURCE();

    //-------------------------
    // SPI Setup
    //-------------------------
    // Enable the peripheral clock of GPIO Port
    SPIx_GPIO_CLK_ENABLE();

    // Configure the SPI SCK pin
    LL_GPIO_SetPinMode(SPI_CJ125_SCK_PORT, SPI_CJ125_SCK_PIN, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(SPI_CJ125_SCK_PORT, SPI_CJ125_SCK_PIN, LL_GPIO_AF_5);
    LL_GPIO_SetPinSpeed(SPI_CJ125_SCK_PORT, SPI_CJ125_SCK_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinPull(SPI_CJ125_SCK_PORT, SPI_CJ125_SCK_PIN, LL_GPIO_PULL_DOWN);

    // Configure the SPI MISO pin
    LL_GPIO_SetPinMode(SPI_CJ125_MISO_PORT, SPI_CJ125_MISO_PIN, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(SPI_CJ125_MISO_PORT, SPI_CJ125_MISO_PIN, LL_GPIO_AF_5);
    LL_GPIO_SetPinSpeed(SPI_CJ125_MISO_PORT, SPI_CJ125_MISO_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinPull(SPI_CJ125_MISO_PORT, SPI_CJ125_MISO_PIN, LL_GPIO_PULL_DOWN);

    // Configure the SPI MOSI pin
    LL_GPIO_SetPinMode(SPI_CJ125_MOSI_PORT, SPI_CJ125_MOSI_PIN, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(SPI_CJ125_MOSI_PORT, SPI_CJ125_MOSI_PIN, LL_GPIO_AF_5);
    LL_GPIO_SetPinSpeed(SPI_CJ125_MOSI_PORT, SPI_CJ125_MOSI_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinPull(SPI_CJ125_MOSI_PORT, SPI_CJ125_MOSI_PIN, LL_GPIO_PULL_DOWN);

    // Configure the SPI Select pin
    LL_GPIO_SetPinMode(SPI_CJ125_SEL_PORT, SPI_CJ125_SEL_PIN, LL_GPIO_MODE_ALTERNATE);
    LL_GPIO_SetAFPin_8_15(SPI_CJ125_SEL_PORT, SPI_CJ125_SEL_PIN, LL_GPIO_AF_5);
    LL_GPIO_SetPinSpeed(SPI_CJ125_SEL_PORT, SPI_CJ125_SEL_PIN, LL_GPIO_SPEED_FREQ_HIGH);
    LL_GPIO_SetPinPull(SPI_CJ125_SEL_PORT, SPI_CJ125_SEL_PIN, LL_GPIO_PULL_DOWN);

    //---------------------------------
    // LED initialization
    //---------------------------------
    MODIFY_REG(GPIOA->MODER, GPIO_MODER_MODE8, GPIO_MODER_MODE8_0);
    MODIFY_REG(GPIOC->MODER, GPIO_MODER_MODE9, GPIO_MODER_MODE9_0);
}
```

```
/**
 * @file main.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 May 27
 *
 * @brief Runs overall control for the CJ125 and LSU4.9 sensor.
 *
 * This program uses the values provided by the CJ125 to maintain the temperature of
 * the LSU4.9 sensor. It utilizes the calibration values seen by the CJ125 to generate a
 * lookup table for both Lambda and Temperature values. This removes the need for
 * hard-coding a lookup table, at the expense of speed. This, however, results in more
 * accurate readings and control.
 */


#include "stm32l4xx.h"

#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_system.h"
#include "stm32l4xx_ll_utils.h"
#include "arm_math.h"
#include <math.h>

#include "hw_map.h"
#include "adc.h"
#include "dac.h"
#include "pwm.h"
#include "usart.h"
#include "spi.h"
#include "cj125.h"

void Initialize_Heater(void);
void SystemClock_Config(void);
void Generate_Lookup_Tables(void);

/**
 * @brief ADC values array.
 *
 * Array index pertains to: @n
 * 0: Internal 3V3 reference @n
 * 1: Battery voltage @n
 * 2: Lambda value from CJ125 @n
 * 3: Sensor resistance from CJ125 @n
 * 4: Current sense
 */
uint16_t adc_vals[5] = {0, 0, 0, 0, 0};

/**
 * @brief Lookup table for lambda values.
 *
 * Values are calculated and inserted into array based on all possible ADC values.
 */
uint16_t lambda_Lookup[4096];

/**
 * @brief Lookup table for temperature values.
 *
 * Values are calculated and inserted into array based on all possible ADC values.
 */
uint16_t temp_Lookup[4096];

/**
 * @brief The optimal ADC resistance value the CJ125 outputs during calibration.
 *
 * This value is used to generate the temperature lookup table and as the control for the PID.
 */
uint16_t optimal_resistance;

/**
 * @brief The optimal ADC resistance value the CJ125 outputs during calibration.
 *
```

```
 * This value is used to generate the lambda lookup table.
 */
uint16_t optimal_lambda;

/**
 * @brief The current voltage the PWM signal is using.
 *
 * Used as a reference during the PID control of the PWM signal
 */
uint32_t currentV;

/**
 * @brief Internal reference voltage of STM32.
 *
 * This calculation is highlighted on page 583 of the RM0351 Reference Manual.
 */
uint32_t VDDA;

/**
 * @def CONDENSATION
 * @brief Sensor resistance once condensation point has been reached.
 *
 * This value represents the resistance of the actual sensor as it heats up. The higher
 * the resistance, the hotter the sensor.
 */
#define CONDENSATION 3900

/**
 * @def Kp
 * @brief Proportional coefficient for the PID controller.
 *
 * The proportional control is proportional to the error between the desired sensor
 * resistance and the current resistance.
 */
#define Kp 60

/**
 * @def Ki
 * @brief Integral coefficient for the PID controller.
 *
 * The integral control is used to overcome steady state error for the P control. @n
 * In general the integral component is used for accumulating the running errors until the system
 stabilizes.
 */
#define Ki 0.8

/**
 * @def Kd
 * @brief Derivative coefficient for the PID controller.
 *
 * The derivative control is used to prevent overshoot from the P control. @n
 * In general the derivative component is used to prevent the system from overshooting the desire
d result.
 */
#define Kd 0

/**
 * @def PREAMBLE
 * @brief Preamble signifies the start of a UART transmission.
 */
#define PREAMBLE 0xFFFF

/**
 * @def Vbat3V3
 * @brief Actual measured value for the 3V3 line on the PCB.
 * This value is required to make acurate readings of the battery voltage.
 */
#define Vbat3V3 3286

/**
 * @brief Main program entrypoint.
 *
```

```
 * @return Should not return.
 */
int main(void)
{
    uint8_t i = 0;
    uint16_t response = 0;
    uint16_t lambda, lambda_adc, temp, temp_adc;
    int16_t derivative, error, change;
    int16_t integral = 0;
    int16_t last_error = 0;
    uint32_t desiredV, Vbat;
    //uint64_t t1, t2, diff; // Only used for timing
    float pwm_duty_cycle;

    // Initialize the GPIO pins
    HW_Init_GPIO();

    // Config the system clock to 8MHz
    SystemClock_Config();

    // Initialize ADC on channels 0, 8, 11, 12, and 16
    Init_ADC(0x11901, (uint16_t *)adc_vals, 5);
    // Initialize DAC
    Init_DAC();
    // Initialize PWM with duty cycle of 0%
    Init_PWM();
    // Initialize USART connection to external device
    Init_USART();
    // Initialize SPI connection to CJ125
    Init_SPI();

    // Determine actual value of the 3.3V the STM is using.
    LL_mDelay(500);
    __disable_irq();
    VDDA = VREFINT_CAL_VREF*(*VREFINT_CAL_ADDR)/adc_vals[0];
    __enable_irq();

    // Enable cycle counter; Used for timing
    //CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;
    //DWT->CYCCNT = 0;
    //DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;

    // Loop until CJ125 is ready. When CJ125 responds OK move on.
    while (response != CJ125_DIAG_REG_OK) {
        response = SPI_Transfer(CJ125_DIAG_REG);
        LL_mDelay(200);
    }

    // Enter CJ125 calibration mode
    response = SPI_Transfer(CJ125_CALIBRATE_MODE);
    // Delay to allow CJ125 to properly calibrate
    LL_mDelay(2000);

    // Store optimal resistance the CJ125 sees for the sensor
    do {
        __disable_irq();
        optimal_resistance = adc_vals[3];
        __enable_irq();
    } while (optimal_resistance > 830 || optimal_resistance < 780);

    // Store optimal lambda the CJ125 sees for the sensor
    do {
        __disable_irq();
        optimal_lambda = adc_vals[2];
        __enable_irq();
    } while (optimal_lambda > 1300 || optimal_lambda < 1100);

    // Set CJ125 into normal operation mode with an amplification of 8
    response = SPI_Transfer(CJ125_V8_MODE);
    //response = SPI_Transfer(CJ125_V17_MODE);

    // Set t1 to how many clock cycles have gone by
```

```
    //t1 = DWT->CYCCNT;

    // Initialize heater before using it
    Initialize_Heater();

    // Set t2 to how many clock cycles have gone by
    //t2 = DWT->CYCCNT;
    // Determine the amount of time passed since start
    //diff = t2-t1;

    // Continuous loop to read in values from CJ125, adjust heater, and output data.
    while(1) {
        // Read lambda and temp values from CJ125
        __disable_irq();
        lambda_adc = adc_vals[2];
        temp_adc = adc_vals[3];
        __enable_irq();

        // Find lambda and temp values in lookup table
        lambda = lambda_Lookup[lambda_adc];
        temp = temp_Lookup[temp_adc];

        // Output lambda value via DAC
        // Contract specification only calls for 0.65 to 1.36, so all other values are capped
        if (lambda >= 650 && lambda < 1360) {
            DAC_SetValue((lambda-650)*4096/710);
        } else if (lambda >= 1360) {
            DAC_SetValue(4095);
        } else if (lambda < 650) {
            DAC_SetValue(0);
        }

        // Determine battery voltage
        __disable_irq();
        Vbat = (adc_vals[1] * Vbat3V3 / 4096) * 955 / 187;
        __enable_irq();

        // Determine error between desired value and current value
        error = optimal_resistance - temp_adc;

        // Set integral term
        integral = integral + error;

        // Set derivative term
        derivative = error - last_error;

        // Calculate desired change to result in 0 error
        change = (Kp * error) + (Ki * integral) + (Kd * derivative);

        // Set current error to last error for next loop through
        last_error = error;

        // Set voltage based on desired change
        if (currentV - change > Vbat) {
            desiredV = Vbat;
        } else if (currentV - change < 0) {
            desiredV = 0;
        } else {
            desiredV = currentV - change;
        }

        // Transmit over UART every 200ms
        if (i == 20) {
            // Turn on LED before UART Transmit
            SET_BIT(GPIOC->ODR, GPIO_ODR_OD9_Msk);

            // Transmit preamble before data
            USART_Transmit((uint8_t *)PREAMBLE, 2);

            // Transmit lambda value over UART
            USART_Transmit((uint8_t *)(&lambda), 2);
```

```
            // Transmit temperature value over UART
            USART_Transmit((uint8_t *)(&temp), 2);

            // Transmit Current PWM voltage
            USART_Transmit((uint8_t *)(&desiredV), 2);

            // Turn off LED after UART transmission
            CLEAR_BIT(GPIOC->ODR, GPIO_ODR_OD9_Msk);

            i = 0;
        }

        // Calculate PWM duty cycle
        pwm_duty_cycle = pow((float)desiredV / Vbat, 2);
        //pwm_duty_cycle = pow((float)2000 / Vbat, 2);

        // Adjust PWM signal for heater so it stays at 780C
        LL_TIM_OC_SetCompareCH2(PWMx_BASE, LL_TIM_GetAutoReload(PWMx_BASE)*pwm_duty_cycle);

        i++;
        // Delay 10ms
        LL_mDelay(10);
    }
}

/**
 * @brief  System Clock Configuration.
 *
 * The system Clock is configured as follows :
 *    System Clock source            = PLL (MSI)    @n
 *    SYSCLK(Hz)                     = 80000000     @n
 *    HCLK(Hz)                       = 80000000     @n
 *    AHB Prescaler                  = 1            @n
 *    APB1 Prescaler                 = 1            @n
 *    APB2 Prescaler                 = 1            @n
 *    MSI Frequency(Hz)              = 4000000      @n
 *    PLL_M                          = 1            @n
 *    PLL_N                          = 40           @n
 *    PLL_R                          = 2            @n
 *    Flash Latency(WS)              = 4            @n
 * @retval None
 */
void SystemClock_Config(void) {
    // MSI configuration and activation
    LL_FLASH_SetLatency(LL_FLASH_LATENCY_4);
    LL_RCC_MSI_Enable();
    while(LL_RCC_MSI_IsReady() != 1);

    // Main PLL configuration and activation
    LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_MSI, LL_RCC_PLLM_DIV_1, 4, LL_RCC_PLLR_DIV_2);
    LL_RCC_PLL_Enable();
    LL_RCC_PLL_EnableDomain_SYS();
    while(LL_RCC_PLL_IsReady() != 1);

    // Sysclk activation on the main PLL
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
    while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL);

    // Set APB1 & APB2 prescaler
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
    LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);

    // Set systick to 1ms in using frequency set to 80MHz
    LL_Init1msTick(8000000);

    // Update CMSIS variable
    LL_SetSystemCoreClock(8000000);
}

/**
 * @brief Heater initialization routine
```

```
 *
 * Initializes heater to a starting voltage of less than 2V during the condensation phase.
 * Once past condensation phase heater voltage is ramped up at a rate of 0.4V/s until
 * reaching a maximum of 13V. This is highlighted in section 1.6 of the LSU 4.9 manual.
 *
 * @retval None
 */
void Initialize_Heater(void) {
    int i = 0;
    float pwm_duty_cycle;
    uint16_t CurADC, VbatADC, UR;
    uint16_t maxCurADC = 0;
    uint32_t Vbat, maxCur, res;

    // Warm up heater, supply <= 2V to heater until out of condensation phase
    // Uses the current sense value to determine when condensation phase is over
    do {
        // Calculate the battery voltage from the ADC
        __disable_irq();
        Vbat = (adc_vals[1] * Vbat3V3 / 4096) * 955 / 187;
        __enable_irq();

        // Set initial "warm-up" voltage to 2V
        currentV = 2000;

        // Calculate PWM duty cycle using equation found in LSU 4.9 manual
        pwm_duty_cycle = pow((float)currentV / Vbat, 2);

        // Set PWM signal to equivalent of 2Vrms
        LL_TIM_OC_SetCompareCH2(PWMx_BASE, LL_TIM_GetAutoReload(PWMx_BASE)*pwm_duty_cycle);

        // Sample current sense ADC to determine the maximum value
        for (i= 0; i < 50; i++) {
            __disable_irq();
            CurADC = adc_vals[4];
            __enable_irq();
            if (CurADC > maxCurADC) {
                maxCurADC = CurADC;
                __disable_irq();
                VbatADC = adc_vals[1];
                __enable_irq();
            }
            // Delay 10ms
            LL_mDelay(10);
        }

        // Determine the actual current based on ADC values
        maxCur = (maxCurADC * VDDA / 4096);

        // Determine the actual voltage at highest ADC value
        Vbat = (VbatADC * Vbat3V3 / 4096) * 955 / 187;

        // Determine sensor resisitance
        res = Vbat * 7 * 50 / maxCur - 23;

        // Reset max current value for next loop through
        maxCurADC = 0;

        // Delay for 500ms
        LL_mDelay(50);
    } while (res < CONDENSATION);

    // Generate lookup tables for both lambda and temperature values
    Generate_Lookup_Tables();

    //currentV = 8500;
    // Ramp up voltage at a rate of
    i = 0;
    do {
        // Get the current battery voltage
        __disable_irq();
        Vbat = (adc_vals[1] * Vbat3V3 / 4096) * 955 / 187;
```

```
        __enable_irq();

        // Transmit over UART every 200ms
        if (i == 40) {
            // Turn on LED before UART Transmit
            SET_BIT(GPIOC->ODR, GPIO_ODR_OD9_Msk);

            // Transmit preamble before data
            USART_Transmit((uint8_t *)PREAMBLE, 2);

            // Transmit lambda value over UART
            USART_Transmit((uint8_t *)(&lambda_Lookup[optimal_lambda]), 2);

            // Transmit temperature value over UART
            USART_Transmit((uint8_t *)(&temp_Lookup[optimal_resistance]), 2);

            // Transmit Current PWM voltage
            USART_Transmit((uint8_t *)(&currentV), 2);

            // Turn off LED after UART transmission
            CLEAR_BIT(GPIOC->ODR, GPIO_ODR_OD9_Msk);

            i = 0;
        }

        // Calculate duty cycle, equation from LSU 4.9 datasheet
        pwm_duty_cycle = pow((float)currentV / Vbat, 2);

        // Set PWM signal to equivalent of ramp up voltage RMS
        LL_TIM_OC_SetCompareCH2(PWMx_BASE, LL_TIM_GetAutoReload(PWMx_BASE)*pwm_duty_cycle);

        // Read resistance value of sensor; Won't be valid till 780C is hit
        __disable_irq();
        UR = adc_vals[3];
        __enable_irq();

        // Ramp up voltage by 200mV/s
        currentV += 1;

        // Delay 5ms
        LL_mDelay(5);

        i++;
    } while (currentV < 11000 && UR > optimal_resistance);
}

/**
 * @brief Generates lookup tables for lambda and temperature values
 * Equations were determined by applying best fit curves to plots present in
 * the LSU4.9 manual. @n
 * MATLAB was used to fit the plots.
 *
 * @retval None
 */
void Generate_Lookup_Tables(void) {
    uint16_t i;
    int32_t uacal, ua, urcal, ur, lambda, temp;
    float ip, o2, ri;

    // Determine resistance calibration value for calculations further on
    ur = optimal_resistance * VDDA * (365 + 187) / 365 / 4096;
    urcal = -((15.5 * 0.000158 * 301) * 1000 - ur) * 17;

    // Determine lambda calibration value for calculations further on
    uacal = optimal_lambda * VDDA * (365 + 187) / 365 / 4096;

    for (i = 0; i < 4096; i++) {
        // Calculate the Lambda value
        ua = i * VDDA * (365 + 187) / 365 / 4096;
        ip = (ua - uacal) * 1000 / (61.9 * 8); // / 1000;
        o2 = ip * 0.2095 / 2540;
        lambda = (o2 / 3 + 1) / (1 - 4.77 * o2) * 1000;
```

```
        //lambda = (492.3 * exp(-pow((ip - 3.869) / 0.772, 2)) + 2.183 * exp(-pow((ip - 2.288) /
0.714, 2)) + 1.09 * exp(-pow((ip - 2.8) / 6.656, 2)) + 1.011 * exp(-pow((ip - 1.697) / 1.112, 2))
) * 1000;

        if (lambda >= 10119 || lambda <= 0) {
            lambda = 10119;
        }
        // Set lambda lookup table value
        lambda_Lookup[i] = lambda;

        // Calculate the temperature value
        ur = i * VDDA * (365 + 187) / 365 / 4096;
        ri = (ur - urcal / 17) / (15.5 * 0.158);
        temp = 4445 * pow(ri, -0.4449) + 428.6;

        if (temp >= 7049|| temp <= 0) {
            temp = 7049;
        }
        // Set temperature lookup table value
        temp_Lookup[i] = temp;
    }
}
```

```
/**
 * @file pwm.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 September 9
 *
 * @brief Provides basic PWM functionality
 *
 */
#include "pwm.h"

/**
 * @brief Current state of the PWM signal; 1: high, 0: low.
 *
 * Value is set during PWM interrupts.
 */
uint8_t pwm_state = 0;

/* Initialize PWM */
void Init_PWM(void)
{
    // Enable PWM interrupts
    NVIC_SetPriority(PWMx_IRQ, 2);
    NVIC_EnableIRQ(PWMx_IRQ);

    // Set the pre-scaler value
    LL_TIM_SetPrescaler(PWMx_BASE, __LL_TIM_CALC_PSC(SystemCoreClock, 100000));

    // Set the auto-reload value to have a counter frequency of 100 Hz
    LL_TIM_SetAutoReload(PWMx_BASE, __LL_TIM_CALC_ARR(SystemCoreClock, LL_TIM_GetPrescaler(PWMx_B
ASE), 100));

    // Set output mode
    LL_TIM_OC_SetMode(PWMx_BASE, PWMx_CHANNEL, LL_TIM_OCMODE_PWM1);
    LL_TIM_OC_SetMode(PWMx_BASE, PWMx_IRQ_CHANNEL, LL_TIM_OCMODE_PWM1);

    // Set external trigger to output on Capture/Compare Output Channel 1
    LL_TIM_SetTriggerOutput(PWMx_BASE, LL_TIM_TRGO_OC1REF);

    // Set compare value have a 0% duty cycle
    LL_TIM_OC_SetCompareCH2(PWMx_BASE, 0);
    LL_TIM_OC_SetCompareCH1(PWMx_BASE, 1);

    // Enable PWM register preload.
    LL_TIM_OC_EnablePreload(PWMx_BASE, PWMx_CHANNEL);
    LL_TIM_OC_EnablePreload(PWMx_BASE, PWMx_IRQ_CHANNEL);

    // Enable output channel
    LL_TIM_CC_EnableChannel(PWMx_BASE, PWMx_CHANNEL);
    LL_TIM_CC_EnableChannel(PWMx_BASE, PWMx_IRQ_CHANNEL);

    // Enable outputs
    LL_TIM_EnableAllOutputs(PWMx_BASE);

    // Enable the interrupts
    LL_TIM_EnableIT_CC1(PWMx_BASE);
    LL_TIM_EnableIT_CC2(PWMx_BASE);

    // Enable counter
    LL_TIM_EnableCounter(PWMx_BASE);

    // Force update generation
    LL_TIM_GenerateEvent_UPDATE(PWMx_BASE);
}

/* Callback for Compare/Capture 1 Interrupt */
void TimerCC1_Callback(void) {
    // Turn on LED
    SET_BIT(GPIOA->ODR, GPIO_ODR_OD8_Msk);
}

/* Callback for Compare/Capture 2 Interrupt */
```

```
void TimerCC2_Callback(void) {
    // Turn off LED
    CLEAR_BIT(GPIOA->ODR, GPIO_ODR_OD8_Msk);
}
```

```
/**
 * @file spi.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 15
 *
 * @brief Provides basic SPI functionality
 *
 */

#include "spi.h"

/* Inialize the SPI interface */
void Init_SPI(void)
{
    // Enable the clock for the SPI
    SPIx_CLK_ENABLE();

    // Configure the SPI interface
    LL_SPI_SetBaudRatePrescaler(SPI_CJ125_BASE, LL_SPI_BAUDRATEPRESCALER_DIV256);
    LL_SPI_SetTransferDirection(SPI_CJ125_BASE,LL_SPI_FULL_DUPLEX);
    LL_SPI_SetClockPhase(SPI_CJ125_BASE, LL_SPI_PHASE_2EDGE);
    LL_SPI_SetClockPolarity(SPI_CJ125_BASE, LL_SPI_POLARITY_LOW);
    LL_SPI_SetDataWidth(SPI_CJ125_BASE, LL_SPI_DATAWIDTH_8BIT);
    LL_SPI_SetNSSMode(SPI_CJ125_BASE, LL_SPI_NSS_SOFT);
    LL_SPI_SetRxFIFOThreshold(SPI_CJ125_BASE, LL_SPI_RX_FIFO_TH_QUARTER);
    LL_SPI_SetMode(SPI_CJ125_BASE, LL_SPI_MODE_MASTER);

    // Enable the SPI interface
    LL_SPI_Enable(SPI_CJ125_BASE);

    // Enable select line
    MODIFY_REG(GPIOB->MODER, GPIO_MODER_MODE12, GPIO_MODER_MODE12_0);

    // Enable reset line
    MODIFY_REG(GPIOC->MODER, GPIO_MODER_MODE6, GPIO_MODER_MODE6_0);
    // Set SPI reset line high
    SET_BIT(GPIOC->ODR, GPIO_ODR_OD6_Msk);
}

/* Complete an SPI transfer */
uint16_t SPI_Transfer(uint16_t send)
{
    uint16_t recv;
    //int i;

    // Set SPI select line low
    CLEAR_BIT(GPIOB->ODR, GPIO_ODR_OD12_Msk);

    //for (i = 0; i < send_size; i++) {
        // Wait for TX buffer to be empty
        while(!LL_SPI_IsActiveFlag_TXE(SPI_CJ125_BASE));
        // Reverse byte order
        send = ((send << 8) & 0xff00) | ((send >> 8) & 0x00ff);
        // Transmit a byte
        LL_SPI_TransmitData16(SPI_CJ125_BASE, send);
        // Wait for transmit to finish
        while (SPI_CJ125_BASE->SR & SPI_SR_BSY);
    //}

    //for (i = 0; i < recv_size; i++) {
        // Wait for TX buffer to be empty
        while(!LL_SPI_IsActiveFlag_TXE(SPI_CJ125_BASE));
        // Receive a byte
        recv = LL_SPI_ReceiveData16(SPI_CJ125_BASE);
        // Reverse byte order
        recv = ((recv << 8) & 0xff00) | ((recv >> 8) & 0x00ff);
        // Wait for transmit to finish
        while (SPI_CJ125_BASE->SR & SPI_SR_BSY);
    //}

    // Set SPI select line high
```

```
    SET_BIT(GPIOB->ODR, GPIO_ODR_OD12_Msk);

    return recv;
}
```

```
/**
 * @file stm32l4xx_it.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @brief Interrupt setup for STM32L4xx
 * @date 2018-11-27
 */
#include "stm32l4xx.h"

#include "hw_map.h"

#include "adc.h"
#include "spi.h"
#include "pwm.h"

/* This function handles NMI exceptions */
void NMI_Handler(void) {
}

/* THis function handles Hard Fault exceptions */
void HardFault_Handler(void) {
    // Enter an infinite loop
    while (1);
}

/* This function handles Memory Manage exceptions */
void MemManage_Handler(void) {
    // Enter an infinite loop
    while (1);
}

/* This function handles Bus Fault exceptions */
void BusFault_Handler(void) {
    // Enter an infinite loop
    while (1);
}

/* This function handles Fault exceptions */
void UsageFault_Handler(void) {
    // Enter an infinite loop
    while (1);
}

/* This function handles SVCall exceptions */
void SVC_Handler(void) {
}

/* This function handles Debug Monitor exceptions */
void DebugMon_Handler(void) {
}

/* This function handles PendSVC exceptions */
void PendSV_Handler(void) {
}

/* This function handles SysTick Handlers */
void SysTick_Handler(void) {
}

/* This function handles ADC1 interrupt requests */
void ADC1_2_IRQHandler(void) {
    // Check if the interupt is end of conversion
    if (LL_ADC_IsActiveFlag_EOS(ADCx_BASE) != 0) {
        // Clear the end of conversion flag
        LL_ADC_ClearFlag_EOS(ADCx_BASE);

        // Call the ADC conversion complete callback
        ADC_ConvComplete_Callback();
    }

    // Check if the interupt is overrun
    if (LL_ADC_IsActiveFlag_OVR(ADCx_BASE) != 0) {
```

```
        // Clear the ADC overrun flag
        LL_ADC_ClearFlag_OVR(ADCx_BASE);

        // Call the ADC conversion complete callback
        ADC_OverrunError_Callback();
    }
}

/* This function handles DMA1 interrupts requests */
void DMA1_Channel1_IRQHandler(void) {
    // Check if the DMA transfer is complete
    if (LL_DMA_IsActiveFlag_TC1(DMAx_BASE) == 1) {
        // Clear the DMA interupt flag
        LL_DMA_ClearFlag_GI1(DMAx_BASE);

        // Call the DMA transfer complete callback
        ADC_DMA_TransferComplete_Callback();
    }

    // Check if the DMA transfer casused an error
    if (LL_DMA_IsActiveFlag_TE1(DMAx_BASE) == 1) {
        // Clear the DMA error flag
        LL_DMA_ClearFlag_TE1(DMAx_BASE);

        // Call the DMA transfer error callback
        ADC_DMA_TransferError_Callback();
    }
}

/* This function handles TIM8 interrupt. */
void TIM8_CC_IRQHandler(void) {
    // Check whether CC1 interrupt is pending
    if(LL_TIM_IsActiveFlag_CC1(PWMx_BASE) == 1) {
        // Clear the update interrupt flag
        LL_TIM_ClearFlag_CC1(PWMx_BASE);

        // Timer capture/compare interrupt processing(function defined in main.c)
        TimerCC1_Callback();
    }

    // Check whether CC2 interrupt is pending
    if(LL_TIM_IsActiveFlag_CC2(PWMx_BASE) == 1) {
        // Clear the update interrupt flag
        LL_TIM_ClearFlag_CC2(PWMx_BASE);

        // Timer capture/compare interrupt processing(function defined in main.c)
        TimerCC2_Callback();
    }
}
```

```
/**
  ******************************************************************************
  * @file    system_stm32l4xx.c
  * @author  MCD Application Team
  * @brief   CMSIS Cortex-M4 Device Peripheral Access Layer System Source File
  *
  *   This file provides two functions and one global variable to be called from
  *   user application:
  *      - SystemInit(): This function is called at startup just after reset and
  *                      before branch to main program. This call is made inside
  *                      the "startup_stm32l4xx.s" file.
  *
  *      - SystemCoreClock variable: Contains the core clock (HCLK), it can be used
  *                                  by the user application to setup the SysTick
  *                                  timer or configure other parameters.
  *
  *      - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must
  *                                 be called whenever the core clock is changed
  *                                 during program execution.
  *
  *   After each device reset the MSI (4 MHz) is used as system clock source.
  *   Then SystemInit() function is called, in "startup_stm32l4xx.s" file, to
  *   configure the system clock before to branch to main program.
  *
  *   This file configures the system clock as follows:
  *=============================================================================
  *-----------------------------------------------------------------------------
  *        System Clock source                    | MSI
  *-----------------------------------------------------------------------------
  *        SYSCLK(Hz)                             | 4000000
  *-----------------------------------------------------------------------------
  *        HCLK(Hz)                               | 4000000
  *-----------------------------------------------------------------------------
  *        AHB Prescaler                          | 1
  *-----------------------------------------------------------------------------
  *        APB1 Prescaler                         | 1
  *-----------------------------------------------------------------------------
  *        APB2 Prescaler                         | 1
  *-----------------------------------------------------------------------------
  *        PLL_M                                  | 1
  *-----------------------------------------------------------------------------
  *        PLL_N                                  | 8
  *-----------------------------------------------------------------------------
  *        PLL_P                                  | 7
  *-----------------------------------------------------------------------------
  *        PLL_Q                                  | 2
  *-----------------------------------------------------------------------------
  *        PLL_R                                  | 2
  *-----------------------------------------------------------------------------
  *        PLLSAI1_P                              | NA
  *-----------------------------------------------------------------------------
  *        PLLSAI1_Q                              | NA
  *-----------------------------------------------------------------------------
  *        PLLSAI1_R                              | NA
  *-----------------------------------------------------------------------------
  *        PLLSAI2_P                              | NA
  *-----------------------------------------------------------------------------
  *        PLLSAI2_Q                              | NA
  *-----------------------------------------------------------------------------
  *        PLLSAI2_R                              | NA
  *-----------------------------------------------------------------------------
  *        Require 48MHz for USB OTG FS,          | Disabled
  *        SDIO and RNG clock                     |
  *-----------------------------------------------------------------------------
  *=============================================================================
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2017 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
```

```
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */

/** @addtogroup CMSIS
  * @{
  */

/** @addtogroup stm32l4xx_system
  * @{
  */

/** @addtogroup STM32L4xx_System_Private_Includes
  * @{
  */

#include "stm32l4xx.h"

#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000) /*!< Value of the External oscillator in Hz */
#endif /* HSE_VALUE */

#if !defined  (MSI_VALUE)
  #define MSI_VALUE    ((uint32_t)4000000) /*!< Value of the Internal oscillator in Hz*/
#endif /* MSI_VALUE */

#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)16000000) /*!< Value of the Internal oscillator in Hz*/
#endif /* HSI_VALUE */

/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_Defines
  * @{
  */

/************************ Miscellaneous Configuration ************************/
/*!< Uncomment the following line if you need to relocate your vector Table in
     Internal SRAM. */
/* #define VECT_TAB_SRAM */
#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                                   This value must be a multiple of 0x200. */
/******************************************************************************/
```

```
/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_Variables
  * @{
  */
  /* The SystemCoreClock variable is updated in three ways:
      1) by calling CMSIS function SystemCoreClockUpdate()
      2) by calling HAL API function HAL_RCC_GetHCLKFreq()
      3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
         Note: If you use this function to configure the system clock; then there
               is no need to call the 2 first functions listed above, since SystemCoreClock
               variable is updated automatically.
  */
  uint32_t SystemCoreClock = 4000000;

  const uint8_t  AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};
  const uint8_t  APBPrescTable[8] =  {0, 0, 0, 0, 1, 2, 3, 4};
  const uint32_t MSIRangeTable[12] = {100000, 200000, 400000, 800000, 1000000, 2000000, \
                                      4000000, 8000000, 16000000, 24000000, 32000000, 48000000};
/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_FunctionPrototypes
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32L4xx_System_Private_Functions
  * @{
  */

/**
  * @brief  Setup the microcontroller system.
  * @param  None
  * @retval None
  */

void SystemInit(void)
{
  /* FPU settings ------------------------------------------------------------*/
  #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
    SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
  #endif
  /* Reset the RCC clock configuration to the default reset state ------------*/
  /* Set MSION bit */
  RCC->CR |= RCC_CR_MSION;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON , HSION, and PLLON bits */
  RCC->CR &= (uint32_t)0xEAF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x00001000;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;
```

```
  /* Disable all interrupts */
  RCC->CIER = 0x00000000;

  /* Configure the Vector Table location add offset address ------------------*/
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
#endif
}

/**
  * @brief  Update SystemCoreClock variable according to Clock Register Values.
  *         The SystemCoreClock variable contains the core clock (HCLK), it can
  *         be used by the user application to setup the SysTick timer or configure
  *         other parameters.
  *
  * @note   Each time the core clock (HCLK) changes, this function must be called
  *         to update SystemCoreClock variable value. Otherwise, any configuration
  *         based on this variable will be incorrect.
  *
  * @note   - The system frequency computed by this function is not the real
  *           frequency in the chip. It is calculated based on the predefined
  *           constant and the selected clock source:
  *
  *           - If SYSCLK source is MSI, SystemCoreClock will contain the MSI_VALUE(*)
  *
  *           - If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(**)
  *
  *           - If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(***)
  *
  *           - If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(***)
  *             or HSI_VALUE(*) or MSI_VALUE(*) multiplied/divided by the PLL factors.
  *
  *         (*) MSI_VALUE is a constant defined in stm32l4xx_hal.h file (default value
  *             4 MHz) but the real value may vary depending on the variations
  *             in voltage and temperature.
  *
  *         (**) HSI_VALUE is a constant defined in stm32l4xx_hal.h file (default value
  *              16 MHz) but the real value may vary depending on the variations
  *              in voltage and temperature.
  *
  *         (***) HSE_VALUE is a constant defined in stm32l4xx_hal.h file (default value
  *              8 MHz), user has to ensure that HSE_VALUE is same as the real
  *              frequency of the crystal used. Otherwise, this function may
  *              have wrong result.
  *
  *         - The result of this function could be not correct when using fractional
  *           value for HSE crystal.
  *
  * @param  None
  * @retval None
  */
void SystemCoreClockUpdate(void)
{
  uint32_t tmp = 0, msirange = 0, pllvco = 0, pllr = 2, pllsource = 0, pllm = 2;

  /* Get MSI Range frequency--------------------------------------------------*/
  if((RCC->CR & RCC_CR_MSIRGSEL) == RESET)
  { /* MSISRANGE from RCC_CSR applies */
    msirange = (RCC->CSR & RCC_CSR_MSISRANGE) >> 8;
  }
  else
  { /* MSIRANGE from RCC_CR applies */
    msirange = (RCC->CR & RCC_CR_MSIRANGE) >> 4;
  }
  /*MSI frequency range in HZ*/
  msirange = MSIRangeTable[msirange];

  /* Get SYSCLK source -------------------------------------------------------*/
  switch (RCC->CFGR & RCC_CFGR_SWS)
```

```
  {
    case 0x00:  /* MSI used as system clock source */
      SystemCoreClock = msirange;
      break;

    case 0x04:  /* HSI used as system clock source */
      SystemCoreClock = HSI_VALUE;
      break;

    case 0x08:  /* HSE used as system clock source */
      SystemCoreClock = HSE_VALUE;
      break;

    case 0x0C:  /* PLL used as system clock  source */
      /* PLL_VCO = (HSE_VALUE or HSI_VALUE or MSI_VALUE/ PLLM) * PLLN
         SYSCLK = PLL_VCO / PLLR
         */
      pllsource = (RCC->PLLCFGR & RCC_PLLCFGR_PLLSRC);
      pllm = ((RCC->PLLCFGR & RCC_PLLCFGR_PLLM) >> 4) + 1 ;

      switch (pllsource)
      {
        case 0x02:  /* HSI used as PLL clock source */
          pllvco = (HSI_VALUE / pllm);
          break;

        case 0x03:  /* HSE used as PLL clock source */
          pllvco = (HSE_VALUE / pllm);
          break;

        default:    /* MSI used as PLL clock source */
          pllvco = (msirange / pllm);
          break;
      }
      pllvco = pllvco * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> 8);
      pllr = (((RCC->PLLCFGR & RCC_PLLCFGR_PLLR) >> 25) + 1) * 2;
      SystemCoreClock = pllvco/pllr;
      break;

    default:
      SystemCoreClock = msirange;
      break;
  }
  /* Compute HCLK clock frequency ----------------------------------------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> 4)];
  /* HCLK clock frequency */
  SystemCoreClock >>= tmp;
}


/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```
/**
 * @file usart.c
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 September 12
 *
 * @brief Provides basic USART functionality.
 *
 */
#include "usart.h"

/** @brief Send counter */
uint8_t ubSend = 0;

/* Initialize the USART */
void Init_USART(void) {
    // Set transfer direction (Tx/Rx)
    LL_USART_SetTransferDirection(USARTx_BASE, LL_USART_DIRECTION_TX_RX);

    // Set 8 data bits, 1 start bit, 1 stop bit, and no parity
    LL_USART_ConfigCharacter(USARTx_BASE, LL_USART_DATAWIDTH_8B, LL_USART_PARITY_NONE, LL_USART_S
TOPBITS_1);

    // Set baud rate
    LL_USART_SetBaudRate(USARTx_BASE, SystemCoreClock, LL_USART_OVERSAMPLING_16, 115200);

    // Enable USART
    LL_USART_Enable(USARTx_BASE);

    // Polling USART initialisation
    while((!(LL_USART_IsActiveFlag_TEACK(USARTx_BASE))) || (!(LL_USART_IsActiveFlag_REACK(USARTx_
BASE))));
}

/* Transmit data */
void USART_Transmit(uint8_t *send, uint8_t size) {
    // Send characters one per one, until last char to be sent
    while (ubSend < size) {
      // Wait for TXE flag to be raised
      while (!LL_USART_IsActiveFlag_TXE(USARTx_BASE));

      // If last char to be sent, clear TC flag
      if (ubSend == (size - 1)) {
        LL_USART_ClearFlag_TC(USARTx_BASE);
      }

      // Write character in Transmit Data register. TXE flag is cleared by writing data in TDR re
gister
      LL_USART_TransmitData8(USARTx_BASE, send[ubSend++]);
    }

    // Wait for TC flag to be raised for last char
    while (!LL_USART_IsActiveFlag_TC(USARTx_BASE));

    // Transmission is over, reset send amount
    ubSend = 0;
}
```

```
/**
 * @file adc.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 15
 *
 * @brief Provides ADC functionality
 */

#ifndef __ADC_H
#define __ADC_H

#include "hw_map.h"

/**
 * @brief Initialize the ADC with DMA
 *
 * Initialization of the desired channels is done by taking the reverse of
 * the binary expansion of the channels.
 * ie. Enabling channels 3, 8, 11, & 12 the representation would be:
 * - - - 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 * 0 0 0  0| 0  0  0  1| 1  0 0 1|0 0 0 0|1 0 0 0
 *
 * Which results in a hexidecimal number of 0x01908
 *
 * @param channels ADC channels to read from
 * @param values Array to store ADC values
 * @param numValues Number of values in values array
 *
 * @retval None
 */
void Init_ADC(uint32_t channels, uint16_t * values, int numValues);

/**
 * @brief DMA transfer complete callback
 *
 * @note This function is executed when the transfer complete interrupt
 *         is generated
 *
 * @retval None
 */
void ADC_DMA_TransferComplete_Callback(void);

/**
  * @brief  DMA transfer error callback
  *
  * @note   This function is executed when the transfer error interrupt
  *         is generated during DMA transfer
  *
  * @retval None
  */
void ADC_DMA_TransferError_Callback(void);

/**
 * @brief ADC group regular end of sequence conversions interruption callback
 *
 * @note This function is executed when the ADC group regular
 *         sequencer has converted all ranks of the sequence.
 *
 * @retval None
 */
void ADC_ConvComplete_Callback(void);

/**
  * @brief ADC group regular overrun interruption callback
  *
  * @note This function is executed when ADC group regular
  *         overrun error occurs.
  *
  * @retval None
  */
void ADC_OverrunError_Callback(void);
```

```
#endif // __ADC_H
```

```
/**
 * @file cj125.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @brief Definitons for the CJ125 registers
 * @date 2018-11-27
 *
 */

/**
 * @defgroup CJ125 CJ125 Register definitions
 * @{
 */

/** @brief Register value for identification request */
#define CJ125_IDENT_REG         0x4800
/** @brief Register value for diagnostic request */
#define CJ125_DIAG_REG          0x7800
/** @brief Diagnostic response from CJ125: Ready */
#define CJ125_DIAG_REG_OK       0x28ff
/** @brief Diagnostic response from CJ125: No/Low power */
#define CJ125_DIAG_REG_NOPWR    0x2855
/** @brief Diagnostic response from CJ125: No sensor */
#define CJ125_DIAG_REG_NOSNSR   0x287f
/** @brief Register value for setting CJ125 into calibration mode */
#define CJ125_CALIBRATE_MODE    0x569d
/** @brief Register value for setting CJ125 into V=8 mode */
#define CJ125_V8_MODE           0x5688
/** @brief Register value for setting CJ125 into V=17 mode */
#define CJ125_V17_MODE          0x5689
/** @} */
```

```
/**
 * @file dac.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 29
 *
 * @brief Provides basic DAC functionality
 *
 */


#ifndef __DAC_H
#define __DAC_H

#include "hw_map.h"

/**
 * @brief Initialize the DAC interface
 * @retval None
 */
int Init_DAC(void);

/**
 * @brief Adjust the DAC output value
 * @param value Value to set the DAC output to
 * @retval None
 */
void DAC_SetValue(uint32_t value);

#endif // __DAC_H
```

```
/**
 * @file hw_map.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 15
 *
 * @brief Maps hardware peripherals to board specific features
 *
 * Sets up definitions required for hardware peripherals.
 *
 * Includes: ADC, DAC, DMA, PWM, UART, SPI
 */

#ifndef __HW_MAP_H
#define __HW_MAP_H

#include "stm32l4xx_ll_gpio.h"
#include "stm32l4xx_ll_adc.h"
#include "stm32l4xx_ll_dma.h"
#include "stm32l4xx_ll_dac.h"
#include "stm32l4xx_ll_tim.h"
#include "stm32l4xx_ll_usart.h"
#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_spi.h"
#include "stm32l4xx_ll_bus.h"

/**
 * @defgroup ADC ADC Definitions
 */

/**
 * @def ADCx_BASE
 * @ingroup ADC
 * @brief ADC channel used for all.
 */
#define ADCx_BASE ADC1

/**
 * @def ADCx_SAMPLERATE
 * @ingroup ADC
 * @brief ADC sample rate
 */
#define ADCx_SAMPLERATE LL_ADC_SAMPLINGTIME_640CYCLES_5

/**
 * @def ADCx_IRQ
 * @ingroup ADC
 * @brief STM ADC interrupt
 */
#define ADCx_IRQ ADC1_2_IRQn

/**
 * @defgroup ADC_BAT ADC Battery Voltage Monitor
 * @ingroup ADC
 * @brief ADC channel used to monitor battery voltage
 *
 * These definitions are for the ADC that reads the current battery voltage.
 *
 * @{
 */
/** @brief STM ADC GPIO Pin*/
#define ADC_BAT_GPIO_PIN LL_GPIO_PIN_3
/** @brief STM ADC GPIO Port*/
#define ADC_BAT_GPIO_PORT GPIOA
/** @brief STM ADC Channel*/
#define ADC_BAT_CHANNEL LL_ADC_CHANNEL_8
/** @} */

/**
 * @defgroup ADC_LMD ADC Lambda Value Monitor
 * @ingroup ADC
 * @brief ADC channel used to read lambda value
```

```
 *
 * These definitions are for the ADC that reads the lambda value from the CJ125.
 *
 * @{
 */
/** @brief STM ADC GPIO Pin*/
#define ADC_LMD_GPIO_PIN LL_GPIO_PIN_6
/** @brief STM ADC GPIO Port*/
#define ADC_LMD_GPIO_PORT GPIOA
/** @brief STM ADC Channel*/
#define ADC_LMD_CHANNEL LL_ADC_CHANNEL_11
/** @} */

/**
 * @defgroup ADC_RES ADC Resistance Monitor
 * @ingroup ADC
 * @brief ADC channel used to read sensor resitance
 *
 * These definitions are for the ADC that reads the oxygen sensor resistance
 * from the CJ125.
 *
 * @{
 */
/** @brief STM ADC GPIO Pin */
#define ADC_RES_GPIO_PIN LL_GPIO_PIN_7
/** @brief STM ADC GPIO Port */
#define ADC_RES_GPIO_PORT GPIOA
/** @brief STM ADC Channel */
#define ADC_RES_CHANNEL LL_ADC_CHANNEL_12
/** @} */

/**
 * @defgroup ADC_CRNT ADC PWM Current Sense
 * @ingroup ADC
 * @brief ADC channel used to sense current
 *
 * @{
 */
/** @brief STM ADC GPIO Pin */
#define ADC_CRNT_GPIO_PIN LL_GPIO_PIN_1
/** @brief STM ADC GPIO Port */
#define ADC_CRNT_GPIO_PORT GPIOB
/** @brief STM ADC Channel */
#define ADC_CRNT_CHANNEL LL_ADC_CHANNEL_16
/** @} */

/**
 * @def ADCx_CLK_ENABLE()
 * @ingroup ADC
 * @brief ADC definition for enabling peripheral clock.
 */
#define ADCx_CLK_ENABLE() LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_ADC)

/**
 * @defgroup DMA DMA Definitions
 */

/**
 * @defgroup DMA_ADC ADC DMA
 * @ingroup DMA
 * @{
 */
/** @brief STM DMA */
#define DMAx_BASE DMA1
/** @brief STM DMA Channel */
#define DMAx_CHANNEL LL_DMA_CHANNEL_1
/** @brief DMA Interrupt */
#define DMAx_IRQ DMA1_Channel1_IRQn
/** @} */

/**
 * @def DMAx_CLK_ENABLE()
```

```
 * @ingroup DMA
 * @brief Enable DMA peripheral clock
 */
#define DMAx_CLK_ENABLE() LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_DMA1);


/**
 * @defgroup DAC DAC Definitions
 */

/**
 * @def DACx_BASE
 * @ingroup DAC
 * @brief STM DAC
 */
#define DACx_BASE DAC1

/**
 * @defgroup DAC_LMD DAC Lambda Output
 * @ingroup DAC
 * @{
 */
/** @brief STM DAC GPIO Pin */
#define DAC_LMD_GPIO_PIN LL_GPIO_PIN_5
/** @brief STM DAC GPIO Port */
#define DAC_LMD_GPIO_PORT GPIOA
/** @brief STM DAC Channel */
#define DAC_LMD_CHANNEL LL_DAC_CHANNEL_2
/** @} */

/**
 * @defgroup DAC_CLK DAC Clock
 * @ingroup DAC
 * @{
 */
/** @brief Enable DAC peripheral clock */
#define DACx_CLK_ENABLE() LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_DAC1)
/** @} */


/**
 * @defgroup PWM PWM Definitions
 * @{
 */
/** @brief STM PWM timer */
#define PWMx_BASE TIM8
/** @brief STM PWM Timer channel */
#define PWMx_CHANNEL LL_TIM_CHANNEL_CH2
/** @brief STM PWM Timer Interrupt */
#define PWMx_IRQ TIM8_CC_IRQn
/** @brief STM PWM Interrupt channel */
#define PWMx_IRQ_CHANNEL LL_TIM_CHANNEL_CH1
/** @brief Enable PWM peripheral clock */
#define PWMx_CLK_ENABLE() LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_TIM8)

/** @brief Enable GPIO clock */
#define PWMx_GPIO_CLK_ENABLE() LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOC)
/** @brief STM PWM GPIO Pin */
#define PWMx_GPIO_PIN LL_GPIO_PIN_7
/** @brief STM PWM GPIO Port */
#define PWMx_GPIO_PORT GPIOC
/** @brief Set PWM alternative function */
#define PWMx_SET_GPIO_AF() LL_GPIO_SetAFPin_0_7(GPIOC, LL_GPIO_PIN_7, LL_GPIO_AF_3)
/** @} */

/**
 * @defgroup USART USART Definitions
 * @{
 */
/** @brief STM USART */
#define USARTx_BASE USART1
/** @brief Enable peripheral clock */
```

```
#define USARTx_CLK_ENABLE() LL_APB2_GRP1_EnableClock(LL_APB2_GRP1_PERIPH_USART1)
/** @brief Set USART clock source */
#define USARTx_CLK_SOURCE() LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_PCLK2)

/** @brief Enable GPIO clock */
#define USARTx_GPIO_CLK_ENABLE() LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOA)
/** @brief STM USART Transmit GPIO Pin */
#define USARTx_TX_PIN LL_GPIO_PIN_9
/** @brief STM USART Transmit GPIO Port */
#define USARTx_TX_GPIO_PORT GPIOA
/** @brief Set Transmit alternative function */
#define USARTx_SET_TX_GPIO_AF() LL_GPIO_SetAFPin_8_15(GPIOA, LL_GPIO_PIN_9, LL_GPIO_AF_7)
/** @brief STM USART Receive GPIO Pin */
#define USARTx_RX_PIN LL_GPIO_PIN_10
/** @brief STM USART Receive GPIO Port */
#define USARTx_RX_GPIO_PORT GPIOA
/** @brief Set Receive alternative function */
#define USARTx_SET_RX_GPIO_AF() LL_GPIO_SetAFPin_8_15(GPIOA, LL_GPIO_PIN_10, LL_GPIO_AF_7)
/** @} */

/**
 * @defgroup SPI SPI Definitions
 * @{
 */
/** @brief Enable GPIO clock */
#define SPIx_GPIO_CLK_ENABLE() LL_AHB2_GRP1_EnableClock(LL_AHB2_GRP1_PERIPH_GPIOB)
/** @brief Enable peripheral clock */
#define SPIx_CLK_ENABLE() LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_SPI2);
/** @brief STM SPI */
#define SPI_CJ125_BASE SPI2
/** @brief STM SPI Clock GPIO Pin */
#define SPI_CJ125_SCK_PIN LL_GPIO_PIN_13
/** @brief STM SPI Clock GPIO Port */
#define SPI_CJ125_SCK_PORT GPIOB
/** @brief STM SPI MISO GPIO Pin */
#define SPI_CJ125_MISO_PIN LL_GPIO_PIN_14
/** @brief STM SPI MISO GPIO Port */
#define SPI_CJ125_MISO_PORT GPIOB
/** @brief STM SPI MOSI GPIO Pin */
#define SPI_CJ125_MOSI_PIN LL_GPIO_PIN_15
/** @brief STM SPI MOSI GPIO Port */
#define SPI_CJ125_MOSI_PORT GPIOB
/** @brief STM SPI Select GPIO Pin */
#define SPI_CJ125_SEL_PIN LL_GPIO_PIN_12
/** @brief STM SPI Select GPIO Port */
#define SPI_CJ125_SEL_PORT GPIOB
/** @} */

/**
 * @brief Initializes hardware required by peripherals
 * @retval None
 */
void HW_Init_GPIO(void);

#endif // __HW_MAP_H
```

```
/**
 * @file pwm.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 September 9
 *
 * @brief Provides basic PWM functionality
 *
 */
#ifndef __PWM_H
#define __PWM_H

#include "hw_map.h"

/**
 * @brief Initialize PWM
 *
 * Sets up the PWM peripheral. The PWM has a counter rate of 100Hz.
 * Because the PWM signal is used to control the sensor heater, it
 * is initialized to a 0% duty cycle. The Timer for PWM is also used
 * to trigger ADC conversions so the Capture/Compare Channel 1 is set
 * accordingly.
 *
 * @retval None
 */
void Init_PWM(void);

/**
 * @brief This function handles timer Compare/Capture 1 interrupt.
 *
 * This handler simply turns on the LED on PA8.
 *
 * @retval None
 */
void TimerCC1_Callback(void);

/**
 * @brief  This function handles timer Compare/Capture 2 interrupt.
 *
 * This handler simply turns off the LED on PA8.
 *
 * @retval None
 */
void TimerCC2_Callback(void);

#endif /* __PWM_H */
```

```
/**
 * @file spi.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 July 15
 *
 * @brief Provides basic SPI functionality
 *
 */

#ifndef __SPI_H
#define __SPI_H

#include "hw_map.h"

/**
 * @brief Initialize the SPI Interface
 *
 * @retval None
 */
void Init_SPI(void);

/**
 * @brief Complete an SPI Transfer
 *
 * @param send Value to send
 *
 * @returns Received data
 */
uint16_t SPI_Transfer(uint16_t send);

#endif // __SPI_H
```

```
/**
 * @file stm32l4xx_it.h
 *
 */
#ifndef __STM32L4XX_IT_H
#define __STM32L4XX_IT_H


/**
 * @brief   This function handles NMI exception.
 * @retval None
 */
void NMI_Handler(void);

/**
 * @brief  This function handles Hard Fault exception.
 * @retval None
 */
void HardFault_Handler(void);

/**
 * @brief  This function handles Memory Manage exception.
 * @retval None
 */
void MemManage_Handler(void);

/**
 * @brief  This function handles Bus Fault exception.
 * @retval None
 */
void BusFault_Handler(void);

/**
 * @brief  This function handles Usage Fault exception.
 * @retval None
 */
void UsageFault_Handler(void);

/**
 * @brief  This function handles SVCall exception.
 * @retval None
 */
void SVC_Handler(void);

/**
 * @brief  This function handles Debug Monitor exception.
 * @retval None
 */
void DebugMon_Handler(void);

/**
 * @brief  This function handles PendSVC exception.
 * @retval None
 */
void PendSV_Handler(void);

/**
 * @brief  This function handles SysTick Handler.
 * @retval None
 */
void SysTick_Handler(void);

/**
  * @brief  This function handles ADC1 interrupt request.
  * @retval None
  */
void ADC1_2_IRQHandler(void);

/**
  * @brief  This function handles DMA1 interrupt request.
  * @retval None
  */
```

```
void DMA1_Channel1_IRQHandler(void);

/**
* @brief  This function handles TIM2 interrupt.
* @retval None
*/
void TIM8_CC_IRQHandler(void);

#endif // __STM32L4XX_IT_H
```

```
/**
 * @file usart.h
 * @author Kyle Bernier
 * @author Daeghan Elkin
 * @date 2018 September 12
 *
 * @brief Provides basic USART functionality.
 *
 */

#ifndef __USART_H
#define __USART_H

#include "hw_map.h"

/**
 * @brief Initializes the USART.
 *
 * The USART initialization sets the UART to a TX/RX transmisson.
 * The UART has the typical 8 data, 1 start, and 1 stop bits with no parity.
 * The baudrate for the UART is set to 115200.
 *
 * @retval None
 */
void Init_USART();

/**
 * @brief Transfers data over the transmit line.
 *
 * The UART is set up to handle both transmit and receive, however,
 * nothing is done with the received data. The receive line is
 * available for future changes.
 *
 * @param send Array of desired send data
 * @param size Size of data array
 * @retval None
 */
void USART_Transmit(uint8_t *send, uint8_t size);

#endif /* __USART_H */
```

```
# @file Makefile
# @author Kyle Bernier and Daeghan Elkin
# @date 2018 May 27
#
# @brief Makefile to compile STM32L4 project with dependencies


# System configuration
CC = arm-none-eabi-gcc
RM = rm -f
MKDIR = mkdir -p

# Hardware architecture
ARCH = STM32L475xx
ARCH_LOW = stm32l475xx

# Assembler, Compiler and Linker flags and linker script settings
LINKER_FLAGS = -lm -mthumb -mhard-float -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -Wl,--gc-sections -T$(L
INK_SCRIPT) -static -Wl,--start-group -lc -lm -Wl,--end-group -specs=nano.specs -specs=nosys.spec
s
LINK_SCRIPT = linker_script.ld
ASSEMBLER_FLAGS = -c -g -O0 -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -mthumb -mhard-float -specs=nano.sp
ecs -D"$(ARCH)" -D"ARM_MATH_CM4" -D"__FPU_PRESENT" -x assembler-with-cpp $(INCLUDE_FLAGS)
COMPILER_FLAGS = -c -g -mcpu=cortex-m4 -mfpu=fpv4-sp-d16 -O0 -Wall -ffunction-sections -fdata-sec
tions -mthumb -mhard-float -specs=nano.specs -D"$(ARCH)" -D"STM32L4" -D"USBD_SOF_DISABLED" -D"ARM
_MATH_CM4" -D"__FPU_PRESENT" $(INCLUDE_FLAGS)

# Define directories
BIN_DIR = bin
SRC_DIR = src
LIB_DIR = lib

# CMSIS directories
CMSIS_DIR = $(LIB_DIR)/cmsis
CMSIS_DEV_DIR = $(CMSIS_DIR)/device
CMSIS_INC_DIR = $(CMSIS_DIR)/inc
CMSIS_SRC_DIR = $(CMSIS_DIR)/src

# FreeRTOS directories
#FREERTOS_DIR = $(LIB_DIR)/freertos
#FREERTOS_INC_DIR = $(FREERTOS_DIR)/include
#FREERTOS_PRT_DIR = $(FREERTOS_DIR)/portable

# HAL directories
HAL_DIR = $(LIB_DIR)/hal
HAL_INC_DIR = $(HAL_DIR)/inc
HAL_SRC_DIR = $(HAL_DIR)/src

# libUSB_stm32 directories
LIBUSB_DIR = $(LIB_DIR)/libusb_stm32
LIBUSB_INC_DIR = $(LIBUSB_DIR)/inc
LIBUSB_SRC_DIR = $(LIBUSB_DIR)/src

# Define include flags
INCLUDE_FLAGS = -I$(SRC_DIR) -I$(CMSIS_DEV_DIR) -I$(CMSIS_INC_DIR) -I$(CMSIS_SRC_DIR) -I$(HAL_DIR
) -I$(HAL_INC_DIR) -I$(HAL_SRC_DIR) -I$(LIBUSB_DIR) -I$(LIBUSB_INC_DIR) -I$(LIBUSB_SRC_DIR)

# Define sources
SRCS := $(wildcard $(HAL_SRC_DIR)/*.c) \
        $(wildcard $(HAL_SRC_LEG_DIR)/*.c) \
        $(wildcard $(SRC_DIR)/*.c) \
        $(wildcard $(CMSIS_SRC_DIR)/*.c) \
        $(wildcard $(LIBUSB_SRC_DIR)/*.c)
SRCSASM := $(CMSIS_SRC_DIR)/startup_$(ARCH_LOW).s

HDRS := $(wildcard $(HAL_INS_DIR)/*.h) \
        $(wildcard $(SRC_DIR)/*.h) \
        $(wildcard $(CMSIS_INC_DIR)/*.h) \
        $(wildcard $(CMSIS_DEV_DIR)/*.h) \
        $(wildcard $(LIBUSB_INC_DIR)/*.h)

# Define objects
```

```
OBJS := $(SRCSASM:%.s=$(BIN_DIR)/%.o) $(SRCS:%.c=$(BIN_DIR)/%.o)

# Define the target file
TARGET = $(BIN_DIR)/main

all: $(TARGET)

# Link the objects into the target
$(TARGET): $(OBJS) $(HDRS)
        $(CC) -o $@ $(OBJS) $(LINKER_FLAGS)

# Compile c files
$(BIN_DIR)/%.o: %.c
        $(MKDIR) -p $(dir $@) 2> /dev/null
        $(CC) $(COMPILER_FLAGS) $< -o $@

# Compile assembly files
$(BIN_DIR)/%.o: %.s
        $(MKDIR) -p $(dir $@) 2> /dev/null
        $(CC) $(ASSEMBLER_FLAGS) $< -o $@

# Remove only src directory object files
clean:
        $(RM) $(wildcard $(BIN_DIR)/$(SRC_DIR)/*.o)

# Remove all object files and the target
cleaner:
        $(RM) $(OBJS) $(TARGET)
```

```
import serial
import struct
import statistics as stat
import matplotlib.pyplot as plt
import matplotlib.animation as anim
from collections import deque

# Connect to the USB serial device
ser = serial.Serial('/dev/ttyUSB0')
ser.baudrate = 115200
ser.flushInput()

# Create the lambda and temp arrays for graphing
lamb = deque([0] * 100);
temp = deque([0] * 100);
pwmv = deque([0] * 100);
x = range(100);

# Create the plots
fig, ax1 = plt.subplots()

# Plot Lambda
ax1.set_xlabel('time (s)')
ax1.set_ylabel('Lambda (mLam)', color='tab:blue')
ax1.plot(x, lamb, color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')
ax1.set_ylim(0, 10200)

# Create a second axes
ax2 = ax1.twinx()
# Plot Temperature
ax2.set_ylabel('Temperature (C)', color='tab:red')
ax2.plot(x, temp, color='tab:red')
ax2.tick_params(axis='y', labelcolor='tab:red')
ax2.set_ylim(0, 1200)

# Create a third axes
ax3 = ax1.twinx()
# Plot Temperature
ax3.set_ylabel('PWM Vrms(V)', color='tab:green')
ax3.plot(x, pwmv, color='tab:green')
ax3.tick_params(axis='y', labelcolor='tab:green')
ax3.set_ylim(0, 12)
ax3.spines["right"].set_position(("axes", 1.2))
ax3.spines["right"].set_visible(True)

fig.tight_layout()
fig.show()

# Flush the serial input of buffered data
ser.flushInput()
# Graph data forever
while True:
    # Check for preamble
    while True:
        rdbytes = ser.read(1)
        if rdbytes == b'\xff':
            rdbytes = ser.read(1)
            if rdbytes == b'\xff':
                break
    # Read data
    rdbytes = ser.read(6)
    # Convert bytes to integers
    values = struct.unpack('<HHH', rdbytes)
    # Insert the lambda value into the array
    lamb.popleft()
    lamb.append(values[0]/1000)
    # Insert the temperature value into the array
    temp.popleft()
    temp.append(values[1])
    # Insert the PWM Vrms value into the array
    pwmv.popleft()
```

```
    pwmv.append(values[2]/1000)
    # Print the lambda value the temperature and the PWM Vrms
    print(values)
    # Clear the plots and replot
    ax1.cla()
    ax2.cla()
    ax3.cla()
    ax1.set_ylabel('Lambda Value', color='tab:blue')
    ax2.set_ylabel('Temperature (C)', color='tab:red')
    ax3.set_ylabel('PWM Vrms (V)', color='tab:green')
    ax1.plot(x, lamb, color='tab:blue')
    ax2.plot(x, temp, color='tab:red')
    ax3.plot(x, pwmv, color='tab:green')
    # Reset the axes
    ax1.set_ylim(0, max(lamb)+0.5)
    ax2.set_ylim(0, max(temp)+200)
    ax3.set_ylim(0, max(pwmv)+1)
    ax3.spines["right"].set_position(("axes", 1.2))
    ax3.spines["right"].set_visible(True)
    # Print imformation about each plot
    ax1.text(0.01, 0.04, 'Current Lambda: ' + str(lamb[-1]), verticalalignment='bottom', horizont
alalignment='left', transform=ax2.transAxes, color='blue', fontsize=12)
    ax1.text(0.01, 0.01, 'Mean Lambda: ' + str(stat.mean(lamb)), verticalalignment='bottom', hori
zontalalignment='left', transform=ax2.transAxes, color='blue', fontsize=12)
    ax2.text(0.01, 0.13, 'Current Temp: ' + str(temp[-1]) + 'C', verticalalignment='bottom', hori
zontalalignment='left', transform=ax2.transAxes, color='red', fontsize=12)
    ax2.text(0.01, 0.10, 'Max Temp: ' + str(max(temp)) + 'C', verticalalignment='bottom', horizon
talalignment='left', transform=ax2.transAxes, color='red', fontsize=12)
    ax2.text(0.01, 0.07, 'Min Temp: ' + str(min(temp)) + 'C', verticalalignment='bottom', horizon
talalignment='left', transform=ax2.transAxes, color='red', fontsize=12)
    ax3.text(0.01, 0.16, 'Current PWM Vrms: ' + str(pwmv[-1]) + 'V', verticalalignment='bottom',
horizontalalignment='left', transform=ax2.transAxes, color='green', fontsize=12)
    # Draw the updated plot
    fig.canvas.draw()
    fig.canvas.flush_events()
```