```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  * @name                                  : Chase Westlake
  *
  * Description: Program uses DMA to assign 100 samples of the internal temp and thermistor temp to
  *                                 their respective arrays. The samples are then averaged and calculated to
produce
  *                                 two seperate temperatures in Celcius.
  *
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <math.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc3;
DMA_HandleTypeDef hdma_adc1;
DMA_HandleTypeDef hdma_adc3;

RNG_HandleTypeDef hrng;

/* USER CODE BEGIN PV */
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_RNG_Init(void);
static void MX_ADC3_Init(void);
/* USER CODE BEGIN PFP */

double dieVal();
double thermVal();
void blinkOn();
void blinkOff();

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint32_t thermArray [100];
double thermValue;
uint32_t dieArray[100];
double dieValue;
float C;
float intTemp;
float internalTemperature;
float Vsense;
```

```
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
```

```
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC1_Init();
MX_RNG_Init();
MX_ADC3_Init();
/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc3, thermArray, 100);
HAL_ADC_Start_DMA(&hadc1, dieArray, 100);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

	//HAL_ADC_START(&hadc1);
	blinkOn();
	thermValue = thermVal();
	dieValue = dieVal();
	blinkOff();

	/*
	 * Acquire the C value with the following math:
	 *
	 * seriesResistance = 100k ohms
	 * ThermistorResistance = 10k ohms
 * Coefficient = 3950
 * temperatureNominal = 25
 *
 * thermValue = 4095 / thermvalue - 1
 * thermValue = seriesResistance / thermValue  --- Provides thermistor resistance
 *
 *
 * C = thermValue / ThermistorResistance
 * C = log(c)
 * C = C / Coefficient
 * C = C + 1 / (temperatureNominal + 273.15) -- converts to kelvin
 * C = 1 / C                    -- inverts
 * C = C - 273.15                         -- produces C value
 */

//Does math required to produce celcius value from thermistor
C = (1 / ((log((100000/(4095/thermValue))/100000)/3950) + (1/(25+273.15)))) - 273.15;
printf("Temperature from thermistor is: %.1f\n", C); //Prints C



/*internalTemperature reads the internal die value, then converts that value
 * to a voltage. That voltage is then converted to Celcius.
 *
 * voltage at 25C is 0.76V
 * average slope is 2.5mV/degree
 *
 * temperature = ((Vsense - V25)/avg_slope) + 25
```

```
         * V25 = Vsense value for 25C
         *
         * Vsense = 4095/die, 4095/die converts die value to voltage
         */
        internalTemperature = (((4095/dieValue) - 0.76)/2.5) + 25;
        printf("Internal temperature is: %.1f\n\n", internalTemperature); //prints internal temp.

        HAL_Delay(500);
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_84CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief ADC3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC3_Init(void)
{

  /* USER CODE BEGIN ADC3_Init 0 */
  /* USER CODE END ADC3_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC3_Init 1 */

  /* USER CODE END ADC3_Init 1 */
  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
  */
  hadc3.Instance = ADC3;
  hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc3.Init.Resolution = ADC_RESOLUTION_12B;
  hadc3.Init.ScanConvMode = DISABLE;
  hadc3.Init.ContinuousConvMode = ENABLE;
  hadc3.Init.DiscontinuousConvMode = DISABLE;
  hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc3.Init.NbrOfConversion = 1;
  hadc3.Init.DMAContinuousRequests = ENABLE;
  hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc3) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_11;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_84CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc3, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC3_Init 2 */

  /* USER CODE END ADC3_Init 2 */

}

/**
  * @brief RNG Initialization Function
  * @param None
  * @retval None
  */
static void MX_RNG_Init(void)
{

  /* USER CODE BEGIN RNG_Init 0 */

  /* USER CODE END RNG_Init 0 */

  /* USER CODE BEGIN RNG_Init 1 */

  /* USER CODE END RNG_Init 1 */
  hrng.Instance = RNG;
  if (HAL_RNG_Init(&hrng) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RNG_Init 2 */

  /* USER CODE END RNG_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA2_Stream0_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
  /* DMA2_Stream4_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA2_Stream4_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream4_IRQn);

}
```

```
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, GreenLED_Pin|OrangeLED_Pin|RedLED_Pin|BlueLED_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin : Button_Pin */
  GPIO_InitStruct.Pin = Button_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Button_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : GreenLED_Pin OrangeLED_Pin RedLED_Pin BlueLED_Pin */
  GPIO_InitStruct.Pin = GreenLED_Pin|OrangeLED_Pin|RedLED_Pin|BlueLED_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

double dieVal(){
	//Iterates through dieArrary, adding to dieValue
	for(int i = 0 ; i < 100 ; i++){
	  dieValue += dieArray[i];
	}
	//Averages the values from dieArray in dieValue.
	dieValue /= 100;

	return dieValue;
}

double thermVal(){
	//Iterates through thermArrary, adding to thermValue
	for(int i = 0 ; i < 100 ; i++){
	  thermValue += thermArray[i];
	}
	//Averages the values from thermArray in thermValue.
	thermValue /= 100;

	return thermValue;
```

```
}

void blinkOn(){
	HAL_GPIO_WritePin(OrangeLED_GPIO_Port, OrangeLED_Pin, GPIO_PIN_SET);
	HAL_Delay(250);
}

void blinkOff(){
	HAL_GPIO_WritePin(OrangeLED_GPIO_Port, OrangeLED_Pin, GPIO_PIN_RESET);
	HAL_Delay(250);
}

//Enables printf
int __io_putchar(int ch)
{
	ITM_SendChar(ch);
	return 0;
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

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