```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file         : main.c
 * @brief        : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2025 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdint.h>
#include "stm32f0xx.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
#define MAX_ITER 100
#define LED_START GPIO_PIN_0
#define LED_END GPIO_PIN_1
#define LED_PORT GPIOB

// Test dimensions
const uint16_t image_dimensions[] = {128, 160, 192, 224, 256};
const uint8_t array_size = sizeof(image_dimensions) / sizeof(image_dimensions[0]);
/* 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 ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
// Global variables for debug monitoring
volatile uint32_t start_time, end_time;
volatile float exec_time_sec[5]; // Execution times in seconds
volatile uint64_t checksum_float_fpu[5]; // Float Checksums with FPU
```

```
volatile uint64_t checksum_double_fpu[5]; // Double Checksums with FPU
volatile uint64_t checksum_float_nofpu[5]; // Float Checksums without FPU
volatile uint64_t checksum_double_nofpu[5]; // Double Checksums without FPU
volatile uint8_t current_test = 0; // Test number
volatile uint8_t current_dimension = 0; // Current dimension index
volatile uint8_t test_complete = 0; // Test completion flag
volatile uint64_t current_checksum = 0; // Current checksum for monitoring
volatile float current_time_sec = 0; // Current time for monitoring in seconds
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
/* USER CODE BEGIN PFP */
uint64_t calculate_mandelbrot_float(int width, int height, int max_iterations);
uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations);
uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations);
void enable_fpu(void);
void disable_fpu(void);
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* 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();
 /* USER CODE BEGIN 2 */

 // Test 1: Float precision
 current_test = 1;
 for (int i = 0; i < array_size; i++) {
    current_dimension = i;
    int width = image_dimensions[i];
```

```
    int height = width;

    HAL_GPIO_WritePin(LED_PORT, LED_START, GPIO_PIN_SET);
    start_time = HAL_GetTick();
    current_checksum = calculate_mandelbrot_float(width, height, MAX_ITER);
    checksum_float_fpu[i] = current_checksum;
    end_time = HAL_GetTick();
    current_time_sec = (end_time - start_time) / 1000.0f;
    exec_time_sec[i] = current_time_sec;

    HAL_GPIO_WritePin(LED_PORT, LED_START, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_PORT, LED_END, GPIO_PIN_SET);
    HAL_Delay(300);
    HAL_GPIO_WritePin(LED_PORT, LED_END, GPIO_PIN_RESET);
    HAL_Delay(200);
  }

  // Test 2: Double precision
  current_test = 2;
  for (int i = 0; i < array_size; i++) {
    current_dimension = i;
    int width = image_dimensions[i];
    int height = width;

    HAL_GPIO_WritePin(LED_PORT, LED_START, GPIO_PIN_SET);
    start_time = HAL_GetTick();
    current_checksum = calculate_mandelbrot_double(width, height, MAX_ITER);
    checksum_double_fpu[i] = current_checksum;
    end_time = HAL_GetTick();
    current_time_sec = (end_time - start_time) / 1000.0f;
    exec_time_sec[i] = current_time_sec;

    HAL_GPIO_WritePin(LED_PORT, LED_START, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LED_PORT, LED_END, GPIO_PIN_SET);
    HAL_Delay(300);
    HAL_GPIO_WritePin(LED_PORT, LED_END, GPIO_PIN_RESET);
    HAL_Delay(200);
  }

  // Signal completion
  test_complete = 1;
  HAL_GPIO_WritePin(LED_PORT, LED_START | LED_END, GPIO_PIN_SET);

  while (1) {
    HAL_Delay(1000);
  }
  /* USER CODE END 2 */
  /* Infinite loop */
```

```
 /* USER CODE BEGIN WHILE */
 while (1)
 {
   /* USER CODE END WHILE */
   /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}
/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 /** Initializes the RCC Oscillators according to the specified parameters
 * in the RCC_OscInitTypeDef structure.
 */
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
 RCC_OscInitStruct.HSIState = RCC_HSI_ON;
 RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 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_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
```

```
/* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOB_CLK_ENABLE();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1, GPIO_PIN_RESET);
 /*Configure GPIO pins : PB0 PB1 */
 GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
// Enable FPU
void enable_fpu(void) {
#ifdef __ARM_FP
  SCB->CPACR |= (3UL << 10*2) | (3UL << 11*2);
  __DSB();
  __ISB();
#endif
}

// Disable FPU
void disable_fpu(void) {
#ifdef __ARM_FP
  SCB->CPACR &= ~((3UL << 10*2) | (3UL << 11*2));
  __DSB();
  __ISB();
#endif
}

// Mandelbrot using single-precision float
uint64_t calculate_mandelbrot_float(int width, int height, int max_iterations) {
  uint64_t mandelbrot_sum = 0;

  for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
      // Convert pixel coordinates to complex numbers (float)
      float x0 = (x / (float)width) * 3.5f - 2.5f;
      float y0 = (y / (float)height) * 2.0f - 1.0f;

      float xi = 0.0f;
      float yi = 0.0f;
      uint32_t iteration = 0;
```

```
        while (iteration < max_iterations) {
          float xi_sq = xi * xi;
          float yi_sq = yi * yi;

          if (xi_sq + yi_sq > 4.0f) {
            break;
          }

          float temp = xi_sq - yi_sq;
          yi = 2.0f * xi * yi + y0;
          xi = temp + x0;

          iteration++;
        }

        mandelbrot_sum += iteration;
      }
    }

    return mandelbrot_sum;
}

//Mandelbrot using variable type integers and fixed point arithmetic
uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations){
  const int64_t MULT = 1000000LL; // fixed-point multiplier
  const int64_t MULT_2P0 = 2 * MULT;
  const int64_t MULT_3P5 = 7 * MULT / 2;   // 3.5 * MULT
  const int64_t MULT_2P5 = 5 * MULT / 2;   // 2.5 * MULT
  const int64_t MULT_4P0 = 4 * MULT;
  uint64_t mandelbrot_sum = 0;

  for (int y = 0; y < height; y++) {
        int64_t y0 = ((int64_t)y * MULT_2P0) / height - MULT; // (y/height)*2.0 - 1.0
        for (int x = 0; x < width; x++) {
          int64_t x0 = ((int64_t)x * MULT_3P5) / width - MULT_2P5;
          int64_t xi = 0;
          int64_t yi = 0;
          int iteration = 0;
          while (iteration < max_iterations) {
            int64_t xi_sq = (xi * xi) / MULT;
            int64_t yi_sq = (yi * yi) / MULT;
            if ((xi_sq + yi_sq) > MULT_4P0) break;
            int64_t xtemp = xi_sq - yi_sq;
            yi = ((2 * xi * yi) / MULT) + y0;
            xi = xtemp + x0;
            iteration++;
          }
          mandelbrot_sum += iteration;
```

```
        }
      }
  return mandelbrot_sum;
}

//Mandelbrot using variable type double
uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations){
  uint64_t mandelbrot_sum = 0;

  for (int y = 0; y < height; y++) {
        double y0 = ((double)y * 2.0 / (double)height) - 1.0;
        for (int x = 0; x < width; x++) {
          double x0 = ((double)x * 3.5 / (double)width) - 2.5;
          double xi = 0.0;
          double yi = 0.0;
          int iteration = 0;
          while (iteration < max_iterations && (xi * xi + yi * yi <= 4.0)) {
            double xtemp = xi * xi - yi * yi;
            yi = 2.0 * xi * yi + y0;
            xi = xtemp + x0;
            iteration++;
          }
          mandelbrot_sum += iteration;
        }
      }
  return mandelbrot_sum;
}
/* 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 */
 __disable_irq();
 while (1)
 {
 }
 /* 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,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
 /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```