```c
/* 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
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
// Add these stubs to silence warnings
int _close(int file) { return -1; }
int _lseek(int file, int ptr, int dir) { return 0; }
int _read(int file, char *ptr, int len) { return 0; }
int _write(int file, char *ptr, int len) { return len; }

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
//TODO: Define and initialize the global variables required
// Setting the dimensions for the Mandelbrot calculation
// change these values for each test run (128, 160, 192, 224, 256)
  const int IMAGE_WIDTH = 192; // Width of the image
  const int IMAGE_HEIGHT = 192; // Height of the image

// These variables store the timing information.
// HAL_GetTick() returns the number of milliseconds since the system started (32-bit unsigned
  uint32_t start_time = 0;
  uint32_t end_time = 0;
  uint32_t execution_time = 0;
```

```
62
63 // This variable will hold the checksum of the Mandelbrot calculation
64   uint64_t checksum = 0; //: should be uint64_t
65   //initial width and height maybe or you might opt for an array??
66
67
68 /* USER CODE END PV */
69
70 /* Private function prototypes -----------------------------------------------*/
71 void SystemClock_Config(void);
72 static void MX_GPIO_Init(void);
73 /* USER CODE BEGIN PFP */
74 uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations);
75 uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations);
76
77
78 /* USER CODE END PFP */
79
80 /* Private user code ---------------------------------------------------------*/
81 /* USER CODE BEGIN 0 */
82
83 /* USER CODE END 0 */
84
85 /**
86   * @brief  The application entry point.
87   * @retval int
88   */
89 int main(void)
90 {
91   /* USER CODE BEGIN 1 */
92
93   /* USER CODE END 1 */
94
95   /* MCU Configuration--------------------------------------------------------*/
96
97   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
98   HAL_Init();
99
100   /* USER CODE BEGIN Init */
101
102   /* USER CODE END Init */
103
104   /* Configure the system clock */
105   SystemClock_Config();
106
107   /* USER CODE BEGIN SysInit */
108
109   /* USER CODE END SysInit */
110
111   /* Initialize all configured peripherals */
112   MX_GPIO_Init();
113   /* USER CODE BEGIN 2 */
114   //TODO: Turn on LED 0 to signify the start of the operation
115   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
116
117   //TODO: Record the start time
118   start_time = HAL_GetTick();
119
120   //TODO: Call the Mandelbrot Function and store the output in the checksum variable defined
121  // checksum = calculate_mandelbrot_fixed_point_arithmetic(IMAGE_WIDTH, IMAGE_HEIGHT, MAX_ITER);
122   checksum = calculate_mandelbrot_double(IMAGE_WIDTH, IMAGE_HEIGHT, MAX_ITER);
```

```c
  //TODO: Record the end time
  end_time = HAL_GetTick();

  //TODO: Calculate the execution time
  execution_time = end_time - start_time;

  //TODO: Turn on LED 1 to signify the end of the operation
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);

  //TODO: Hold the LEDs on for a 1s delay
  HAL_Delay(1000);

  //TODO: Turn off the LEDs
  // turn off LED 0 and LED 1
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0 | GPIO_PIN_1, GPIO_PIN_RESET);


  /* 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 */
//TODO: Mandelbrot using variable type integers and fixed point arithmetic
uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations){
  uint64_t mandelbrot_sum = 0;
    //TODO: Complete the function implementation

    const int64_t SCALE = 1000000; // Scale factor for fixed-point arithmetic

    const int64_t LIMIT = 4 * SCALE * SCALE; // Limit for the escape condition (|z|^2 < 4)

    for (int y = 0; y < height; y++){
      for (int x = 0; x < width; x++){
        // Map pixel coordinate to complex plane (c = c_real + i*c_imag)
            // c_real = (x / width) * 3.5 - 2.5
            // c_imag = (y / height) * 2.0 - 1.0
            // Using 64-bit integers to prevent overflow during intermediate multiplication.
            int64_t c_real = ((int64_t)x * 3500000) / width - 2500000;
            int64_t c_imag = ((int64_t)y * 2000000) / height - 1000000;

            int64_t z_real = 0;
            int64_t z_imag = 0;
            int iteration = 0;

            while (iteration < max_iterations) {
                int64_t z_real_sq = z_real * z_real;
                int64_t z_imag_sq = z_imag * z_imag;

```

```
                // Check for divergence
                if ((z_real_sq + z_imag_sq) > LIMIT) {
                    break;
                }

                // Iterate z_new = z^2 + c
                // z_imag_new = 2 * z_real * z_imag + c_imag
                // The term 2*z_real*z_imag is scaled by SCALE^2, so we divide by SCALE
                // to bring it back to a number scaled by SCALE.
                int64_t z_imag_new = (2 * z_real * z_imag) / SCALE + c_imag;

                // z_real_new = z_real^2 - z_imag^2 + c_real
                // The term (z_real^2 - z_imag^2) is also scaled by SCALE^2, divide by SCALE.
                int64_t z_real_new = (z_real_sq - z_imag_sq) / SCALE + c_real;

                z_real = z_real_new;
                z_imag = z_imag_new;

                iteration++;
            }
            mandelbrot_sum += iteration;
        }
    }
    return mandelbrot_sum;

}

//TODO: Mandelbroat using variable type double
uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations){
    uint64_t mandelbrot_sum = 0;
    //TODO: Complete the function implementation
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            // Map pixel coordinate to complex plane (c = c_real + i*c_imag)
            double c_real = ((double)x / width) * 3.5 - 2.5;
            double c_imag = ((double)y / height) * 2.0 - 1.0;

            double z_real = 0.0;
            double z_imag = 0.0;
            int iteration = 0;

            // Iterate z_new = z^2 + c until |z| > 2 or max_iterations is reached.
            while (iteration < max_iterations && (z_real * z_real + z_imag * z_imag) <= 4.0) {
                // We use a temporary variable for the new real part to ensure the new
                // imaginary part is calculated using the old real part.
                double z_real_new = z_real * z_real - z_imag * z_imag + c_real;
                z_imag = 2 * z_real * z_imag + c_imag;
                z_real = z_real_new;

                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 */

```