```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body (EEE3096S Practical 1B)
  ******************************************************************************
  * @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 */
/* Per brief: do not change max iterations; keep at 100. */
#define MAX_ITER 100
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* Fixed-point Q16.16 configuration. */
#define FIXED_POINT_SHIFT   16
#define FIXED_ONE           (1 << FIXED_POINT_SHIFT)      /* 1.0 in Q16.16 */
#define FLOAT_TO_FIXED(f)   ((int32_t)((f) * (double)FIXED_ONE))

/* Image sizes required by the brief. */
#define NUM_SIZES 5
static const int IMAGE_SIZES[NUM_SIZES] = {128, 160, 192, 224, 256};
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
static inline int32_t fixed_mul(int32_t a, int32_t b)
{
  /* 64-bit intermediate to avoid overflow, then back to Q16.16 */
  return (int32_t)(((int64_t)a * (int64_t)b) >> FIXED_POINT_SHIFT);
}
/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
/* ---- Globals to be visible in Live Expressions (per brief) ---- */
volatile uint64_t checksum;                     /* Holds last run's checksum */
volatile uint32_t start_time, end_time;         /* HAL_GetTick() timestamps (ms) */
volatile uint32_t execution_time;               /* end - start (ms) */
```

```c
/* Arrays to capture all results in one debug session (watch/expand in Live Expressions) */
volatile uint64_t checksums_fixed[NUM_SIZES];   /* checksum for fixed-point runs   */
volatile uint64_t checksums_double[NUM_SIZES];  /* checksum for double runs        */
volatile uint32_t exec_ms_fixed[NUM_SIZES];     /* execution time (ms), fixed      */
volatile uint32_t exec_ms_double[NUM_SIZES];    /* execution time (ms), double     */
/* 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_fixed_point_arithmetic(int width, int height, int max_iterations);
uint64_t calculate_mandelbrot_double               (int width, int height, int max_iterations);
/* 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 */
  /* Turn on LED0 to signify start of operations. PB0 configured as output in MX_GPIO_Init. */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);

  /* -------- Fixed-point runs for all required sizes -------- */
  for (int i = 0; i < NUM_SIZES; ++i)
  {
    int N = IMAGE_SIZES[i];

    start_time    = HAL_GetTick();
    checksum      = calculate_mandelbrot_fixed_point_arithmetic(N, N, MAX_ITER);
    end_time      = HAL_GetTick();
    execution_time = end_time - start_time;

    /* Save per-run results into arrays (watch these in Live Expressions) */
```

```
123     checksums_fixed[i] = checksum;
124     exec_ms_fixed[i]   = execution_time;
125
126     /* Blink LED0 briefly between runs so you can see progress on the board. */
127     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0);
128     HAL_Delay(150);
129   }
130
131   /* Small pause to visually separate phases. */
132   HAL_Delay(600);
133
134   /* Turn on LED1 to signify start of double-precision phase. */
135   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
136
137   /* -------- Double-precision runs for all required sizes -------- */
138   for (int i = 0; i < NUM_SIZES; ++i)
139   {
140     int N = IMAGE_SIZES[i];
141
142     start_time    = HAL_GetTick();
143     checksum      = calculate_mandelbrot_double(N, N, MAX_ITER);
144     end_time      = HAL_GetTick();
145     execution_time = end_time - start_time;
146
147     /* Save per-run results into arrays (watch these in Live Expressions) */
148     checksums_double[i] = checksum;
149     exec_ms_double[i]   = execution_time;
150
151     /* Blink LED1 briefly between runs. */
152     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_1);
153     HAL_Delay(150);
154   }
155
156   /* Hold LEDs on for a 1s delay, then turn both off. */
157   HAL_Delay(1000);
158   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
159   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_RESET);
160   /* USER CODE END 2 */
161
162   /* Infinite loop */
163   /* USER CODE BEGIN WHILE */
164   while (1)
165   {
166     /* USER CODE END WHILE */
167
168     /* USER CODE BEGIN 3 */
169     /* Idle forever; results are available in Live Expressions. */
170   }
171   /* USER CODE END 3 */
172 }
173
174 /**
175   * @brief System Clock Configuration
176   * @retval None
177   */
178 void SystemClock_Config(void)
179 {
180   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
181   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
182
183   /** 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 */
/* ---------------------------------------------------------------------------
 * Mandelbrot using fixed-point arithmetic (Q16.16 integers only).
 *
 * Algorithm follows the pseudocode in the brief; the complex plane mapping:
 *   x0 = (x/width) * 3.5 - 2.5
 *   y0 = (y/height) * 2.0 - 1.0
```

```
 * Iteration:
 *   while (iter < MAX_ITER) and (xi^2 + yi^2 <= 4.0)
 *     temp = xi^2 - yi^2
 *     yi = 2*xi*yi + y0
 *     xi = temp + x0
 * checksum += iter
 * ---------------------------------------------------------------------------- */
uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int max_iterations)
{
  uint64_t sum = 0;

  /* Pre-compute scaled constants in Q16.16 */
  const int32_t scale_x  = FLOAT_TO_FIXED(3.5) / width;   /* 3.5/width  */
  const int32_t scale_y  = FLOAT_TO_FIXED(2.0) / height;  /* 2.0/height */
  const int32_t offset_x = FLOAT_TO_FIXED(-2.5);
  const int32_t offset_y = FLOAT_TO_FIXED(-1.0);
  const int32_t threshold = FLOAT_TO_FIXED(4.0);          /* compare against xi^2 + yi^2 */

  for (int y = 0; y < height; ++y)
  {
    /* y0 = (y/height)*2.0 - 1.0 */
    const int32_t y0 = fixed_mul((int32_t)y * FIXED_ONE, scale_y) + offset_y;

    for (int x = 0; x < width; ++x)
    {
      /* x0 = (x/width)*3.5 - 2.5 */
      const int32_t x0 = fixed_mul((int32_t)x * FIXED_ONE, scale_x) + offset_x;

      int32_t xi = 0;    /* real(z)   in Q16.16 */
      int32_t yi = 0;    /* imag(z)   in Q16.16 */
      int      iteration = 0;

      while (iteration < max_iterations)
      {
        /* xi^2 and yi^2 are still Q16.16 after fixed_mul */
        const int32_t xi2 = fixed_mul(xi, xi);
        const int32_t yi2 = fixed_mul(yi, yi);

        /* If |z|^2 > 4.0, escape (threshold is 4.0 in Q16.16). */
        if ((int64_t)xi2 + (int64_t)yi2 > (int64_t)threshold)
          break;

        /* temp = xi^2 - yi^2 (Q16.16) */
        const int32_t temp = xi2 - yi2;

        /* yi = 2*xi*yi + y0:
           fixed_mul(xi, yi) is Q16.16; multiply by 2 by left-shifting one or
           multiplying by FIXED representation of 2. */
        yi = ((fixed_mul(xi, yi) << 1)) + y0;

        /* xi = temp + x0 */
        xi = temp + x0;

        ++iteration;
      }

      sum += (uint64_t)iteration;
    }
  }

  return sum;
```

```c
}

/* ---------------------------------------------------------------------------
 * Mandelbrot using double-precision floating point.
 * (Same algorithm; uses doubles instead of fixed-point.)
 * ------------------------------------------------------------------------- */
uint64_t calculate_mandelbrot_double(int width, int height, int max_iterations)
{
  uint64_t sum = 0;

  for (int y = 0; y < height; ++y)
  {
    const double y0 = ((double)y / (double)height) * 2.0 - 1.0;

    for (int x = 0; x < width; ++x)
    {
      const double x0 = ((double)x / (double)width)  * 3.5 - 2.5;

      double xi = 0.0;
      double yi = 0.0;
      int    iteration = 0;

      while (iteration < max_iterations && (xi*xi + yi*yi) <= 4.0)
      {
        const double temp = xi*xi - yi*yi;
        yi = 2.0*xi*yi + y0;
        xi = temp + x0;
        ++iteration;
      }

      sum += (uint64_t)iteration;
    }
  }

  return 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 */
  __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)
```

```
367 {
368   /* USER CODE BEGIN 6 */
369   (void)file; (void)line;
370   /* USER CODE END 6 */
371 }
372 #endif /* USE_FULL_ASSERT */
373
```