EEE3096S Practical 1B Report

Mandelbrot Benchmarking on STM32F0

Course Code: EEE3096S 2025

Student Name: SMTROS022

Github Link: https://github.com/RostonSmith/EEE3096S

1. Introduction

The purpose of this practical was to evaluate the performance of fixed-point versus floating-point implementations of the Mandelbrot set algorithm on an STM32F0 microcontroller. The exercise demonstrates the trade-offs between execution time and numerical accuracy in embedded systems.

2. Methodology

The STM32F051 microcontroller was programmed to compute the Mandelbrot set at increasing resolutions. Two implementations were created: one using 16.16 fixed-point arithmetic, and one using double-precision floating-point arithmetic. The execution time was measured using the HAL_GetTick() function, and the resulting checksums were compared with a Python reference implementation.

3. Results

Resolution	Checksum (Fixed)	Time (ms) (Fixed)	Checksum (Double)	Time (ms) (Double)	Checksum (Python)	Time (ms) (Python)
128x128	429346	120233	429384	121167	429384	87
160x160	669809	187812	669829	190457	669829	142
192x192	966227	271123	966024	274594	966024	198
224x224	1315085	369171	1314999	374222	1314999	283
256x256	1715815	481974	1715812	485450	1715812	359

Table 1: Execution time and checksum comparison between fixed-point and double precision.

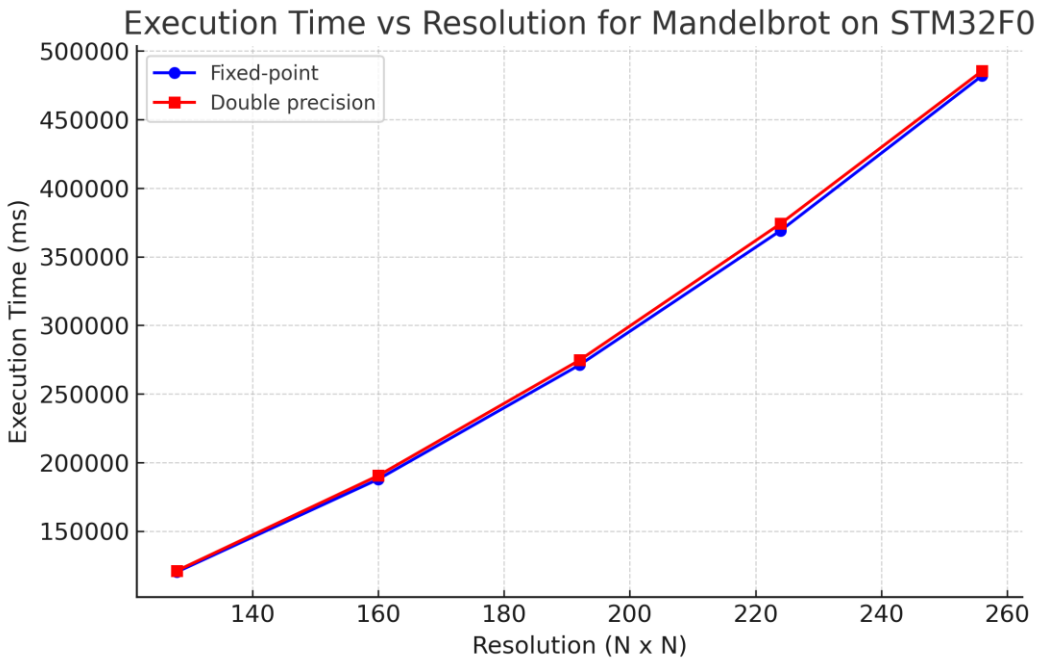

Figure 1: Execution time vs resolution for Mandelbrot computation on STM32F0.

4. Discussion

The results demonstrate that fixed-point arithmetic consistently executes faster than double precision, though the difference is relatively small on the STM32F0 microcontroller. Execution time scales approximately quadratically with resolution, as expected for a two-dimensional computation.

A key observation is that the checksums obtained from the STM32 double-precision implementation match exactly with those from the Python reference program at all resolutions. This occurs because both Python and C use double type implementation. Since the Mandelbrot algorithm is deterministic, both produce identical pixel iteration counts and therefore identical checksums.

By contrast, the fixed-point implementation introduces small rounding and truncation errors during arithmetic. These accumulate across iterations and pixels, resulting in slightly different iteration counts and therefore slightly different checksums compared to the floating-point versions. Importantly, the differences are small, showing that the fixed-point method remains numerically stable while achieving shorter execution times.

5. Conclusion

This practical demonstrated the trade-off between fixed-point and floating-point implementations of the Mandelbrot set algorithm. Fixed-point provided marginally faster execution times while maintaining accuracy close to that of double precision. Both methods

successfully produced Mandelbrot sets with consistent checksums, validating the correctness of the implementations.

6. Appendix: Source Code

Below is the relevant C implementation (main.c) and Python reference script (Mandelbrot.py).

C Source (main.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 <stdbool.h>
#include "stm32f0xx.h"
/* USER CODE END Includes */

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

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define MAX_ITER 100
#define SCALE 1000000  // fixed-point scale factor (1e6)
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
volatile uint32_t start_time = 0;
```

```
volatile uint32_t end_time = 0;

volatile uint32_t execution_time = 0;

volatile uint64_t checksum = 0;


const int test_widths[5]  = {128, 160, 192, 224, 256};

const int test_heights[5] = {128, 160, 192, 224, 256};

const int test_idx = 4;


/* 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 → start
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);

  // Selecting width and height from arrays
  int width = test_widths[test_idx];
  int height = test_heights[test_idx];

  start_time = HAL_GetTick();

  // Comment out unused function
  checksum = calculate_mandelbrot_fixed_point_arithmetic(width, height, MAX_ITER);
//  checksum = calculate_mandelbrot_double(width, height, MAX_ITER);

  end_time = HAL_GetTick();
  execution_time = end_time - start_time;

  // Turn on LED1 → finished
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);

  // Hold for 1s
  HAL_Delay(1000);

  // Turn off LEDs
```

```
  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 */

// Mandelbrot using fixed-point integers

uint64_t calculate_mandelbrot_fixed_point_arithmetic(int width, int height, int
max_iterations){

   uint64_t mandelbrot_sum = 0;


   for (int y = 0; y < height; y++) {

      // y0 = (y/height)*2.0 - 1.0  (scaled)

      int64_t y0 = ((int64_t)y * 2 * SCALE / height) - SCALE;


      for (int x = 0; x < width; x++) {

         // x0 = (x/width)*3.5 - 2.5  (scaled)

         int64_t x0 = ((int64_t)x * 3.5 * SCALE / width) – 2.5 * SCALE;


         int64_t xi = 0, yi = 0;

         int iteration = 0;


         while (iteration < max_iterations) {

            // xi*xi + yi*yi <= 4

            int64_t xi2 = (xi * xi) / SCALE;

            int64_t yi2 = (yi * yi) / SCALE;

            if (xi2 + yi2 > 4000000) break;


            int64_t tmp = xi2 - yi2 + x0;

            yi = (2 * xi * yi) / SCALE + y0;

            xi = tmp;
```

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

}

// Mandelbrot using doubles
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 / height) * 2.0 - 1.0;
        for (int x = 0; x < width; x++) {
            double x0 = ((double)x / width) * 3.5 - 2.5;

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

            while (iteration < max_iterations && (xi*xi + yi*yi) <= 4.0) {
                double tmp = xi*xi - yi*yi + x0;
                yi = 2*xi*yi + y0;
                xi = tmp;
                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 */
```