Mandelbrot Rendering FPGA

Student name: Hoang Minh Le

Student number: 511907

Table of Contents

Introduction 1

Background 1

Requirements 4

Design architecture 5

HDMI driver system 5

Mandelbrot plotting system 7

Testing 8

Result 10

Conclusion 12

Reflection 12

Appendix 12

Introduction

The goal of the project is to rendering the Mandelbrot set by using FPGA. Rendering the Mandelbrot set is computationally intensive because it involves a large number of complex number operations for each pixel in the image. Traditional CPU-based computation can be slow, particularly for high-resolution images or real-time rendering applications. FPGA offers a promising solution due to their ability to perform parallel processing and high-speed arithmetic operations, which can significantly accelerate the rendering process. The HDL I choose for this project is Verilog.

Background

The Mandelbrot set

The Mandelbrot set is the fractal structure with the simple mathematic iterative definition:

$$z_{n+1} = z_n^2 + c$$

Where both z and c are complex numbers and n is the non-negative integer. The use of The initial value z_0 is typically set to 0. The complex number c is a parameter, and the Mandelbrot set is the set of all points in the complex plane for which the sequence does not tend to infinity.

A complex number c belongs to the Mandelbrot set if and only if $|z_n| \leq 2$. The pseudo code for the algorithm to render the Mandelbrot is shown in the Figure 1

```
for each pixel (Px, Py) on the screen do
    x0 := scaled x coordinate of pixel (scaled to lie in the Mandelbrot X scale (-2.00, 0.47))
    y0 := scaled y coordinate of pixel (scaled to lie in the Mandelbrot Y scale (-1.12, 1.12))
    x := 0.0
    y := 0.0
    iteration := 0
    max_iteration := 1000
    while (x^2 + y^2 ≤ 2^2 AND iteration < max_iteration) do
        xtemp := x^2 - y^2 + x0
        y := 2*x*y + y0
        x := xtemp
        iteration := iteration + 1
```

```
    color := palette[iteration]
    plot(Px, Py, color)
```

Figure 1 Pseudo code for the Mandelbrot rendering, source: Wikipedia

Following the algorithm, it will render a beautiful Mandelbrot set shown in the Figure 2.

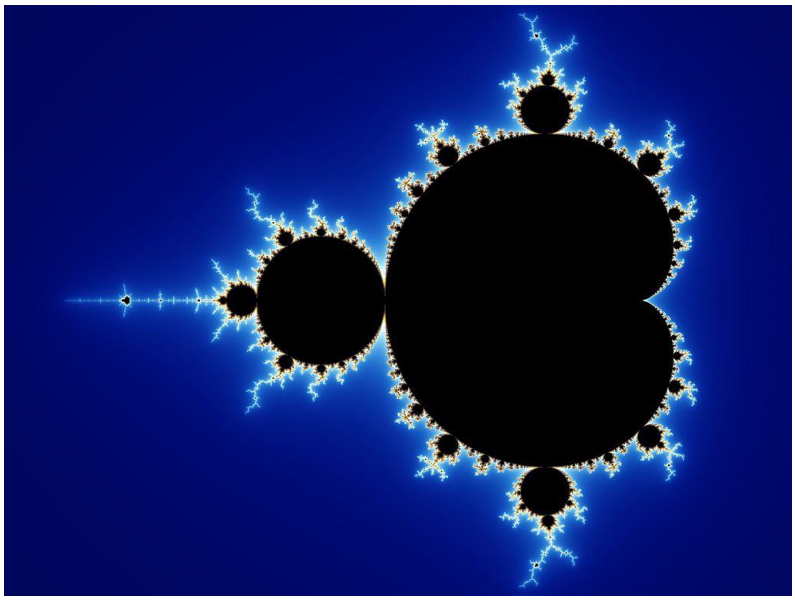

Figure 2 Mandelbrot set, source: Wikipedia

For this project, I set my Mandelbrot coordinate from -2.0 to 1.0 with the X-axis, and -1.0 to 1.0 with the Y-axis, and the max iteration is 255.

Fix point number

When it comes to the computation part, it uses the fix point number. This is to simply reduce the hardware logic gate because the implementation of fix point number in FPGA is simpler than floating point number. The trade off for this approach is the fix point number is not accuracy compared to the floating point number. Fix point number is generally presentation as integer number in hardware. Thus, normal mathematic operations such as addition, subtraction, multiplication, and division in the integer number work in the fix point number. The Figure 3 below show the 8 bits fix point number representation with one sign bit, three integer bits, and four fractional bits.

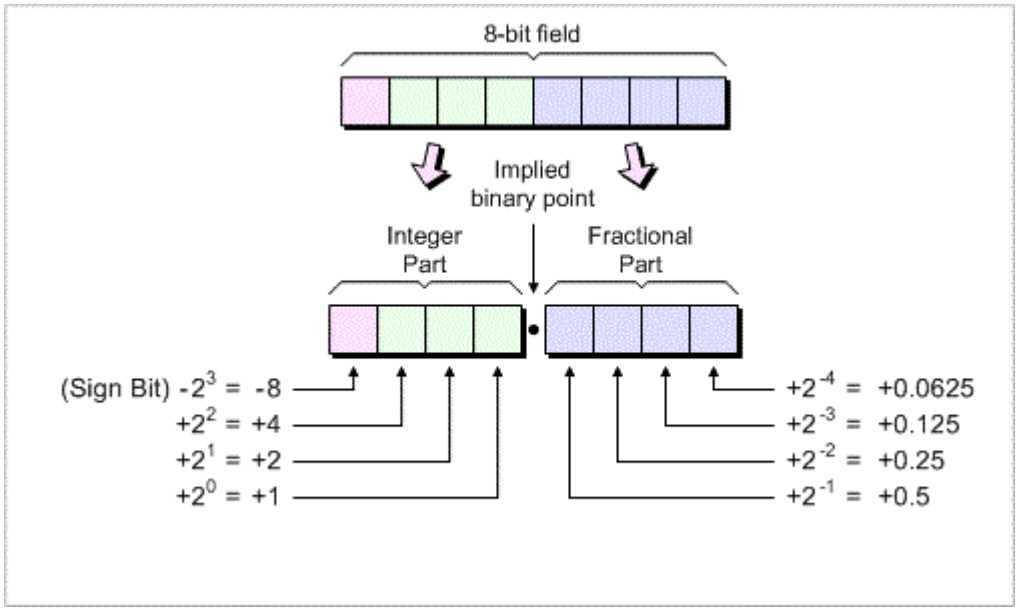

Figure 3 Fix point number, source: All about circuit

Although the multiplication operation works in the fix point number with the same principle as in the integer number, it requires the output result in the same format with the multiplication. Therefore, some parts of the result number needs to be removed. The Figure 4 illustrates about the result of the fix point number after multiplication.

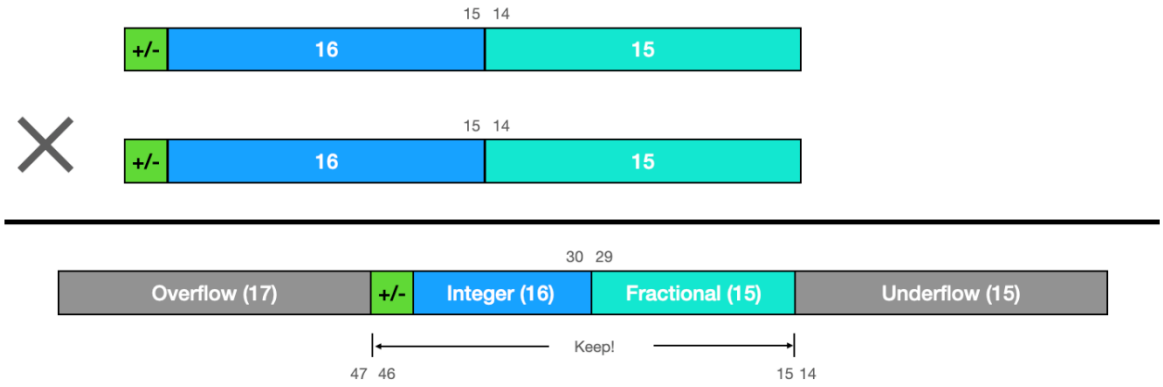

Figure 4: Fix point multiplication, source: vanhunteradam.com

The fix point number format I use is 4.23, which means 27 bits fix point number with 1 sign bit, 3 integer bits, and 23 fractional bits. The reason for this is that the DSP block in the Cyclone V FPGA of the DE10-Nano broad works at precision 27 bits. This requires only 1 DSP block for the multiplication operation, and the multiplication occurs in a single clock cycle. Moreover, the precision is up to 2^{-23} fractional bit, which is good enough for the detail Mandelbrot set rendering.

Requirements

The requires is based on the MosCow method

1. Must have
    - Fix point arithmetic module for addition and multiplication
    - Display module to display the Mandelbrot set on the screen.
    - Controller module for generating complex number for Mandelbrot calculation.
2. Should have
    - All modules should connect via bus interface.
    - Mechanism to turn on the Mandelbrot rendering.
3. Could have
    - Parallel distributor module to calculate the Mandelbrot set in parallel.
4. Will not have
    - Zoom in or Zoom out module

Design architecture

The design architecture contains 2 important parts: HDMI driver system and the Mandelbrot plotting system. All of them are connected together via Avalon bus memory map interface. It is important to point out the using of Avalon-MM Pipeline Bridge module. The use of Avalon-MM Pipeline Bridge module is necessary for the Quartus Program to synthesize width of the address correctly. The Figure 5 shows how the whole system connecting in the Platform designer:

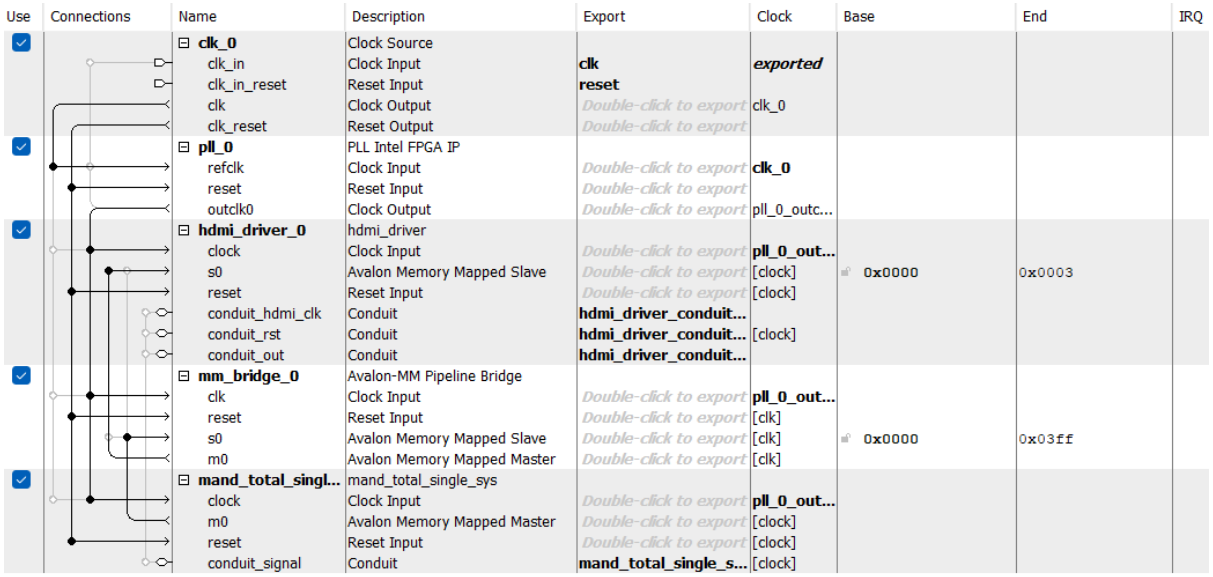

Figure 5 Platform designer system layout

HDMI driver system

The HDMI driver system contains 3 parts: Dual-ports RAMs, Framebuffer, and HDMI transmission system as shown in the Figure 6.

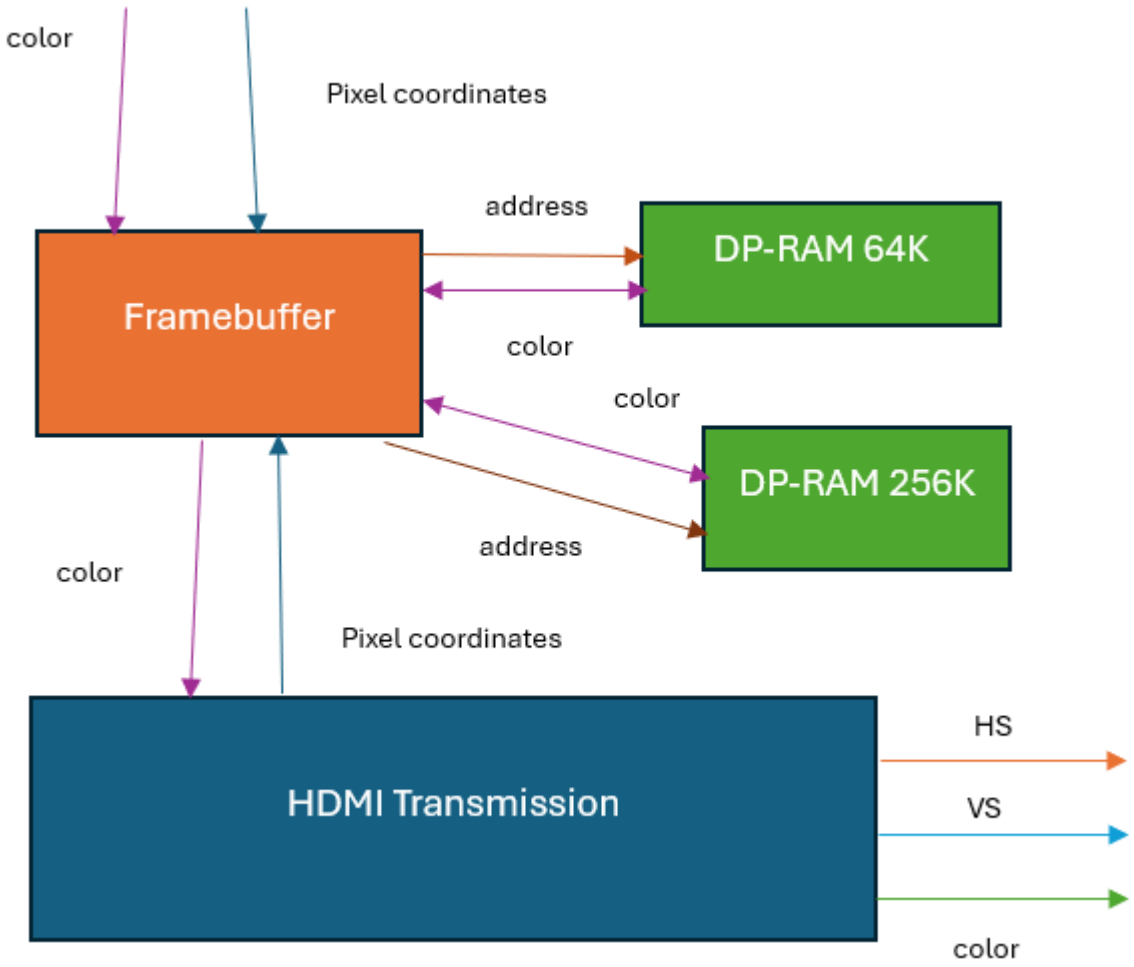

Figure 6 HDMI driver system architecture

1. Dual-ports RAM(DP-RAM) modules: They are 64K and 256K RAM modules. The reason for the 2 separated RAM modules is that the Quartus program is not good at implementing block RAM for the synthesis. This leads error in hardware synthesis which Quartus program cannot implement enough memory block due to memory waste. Fortunately, from the video about Frame Buffer Pixel Generation Circuit which is in Appendix, I understand the problem and solve it by using dividing the RAM into 2 block RAM modules.
2. Framebuffer module: It is responsible for translating pixel coordinates coming from the HDMI Transmission module and from Avalon bus data into an unique address for the Dual-ports RAM. In addition, it controls and decides which DP-RAM module will be used for storing color.
3. HDMI Transmission module: This module is responsible for creating Vertical sync, Horizontal sync and 24 bits color RGB for the HDMI display. I design it to use only 640x480 resolution for the simplicity. The code of this module is the modified code from the HDMI_TX example from the Terrasic company for the DE10-Nano broad.

Mandelbrot plotting system

The Mandelbrot plotting system contains 2 important parts: the calculation part and the controller part. The calculation part involves in the calculation of the iteration based on the complex number, and the controller part is to provide the complex numbers for the calculation part to calculate the iteration. The output data of the system is the 32-bits data in the format pixel coordinates and iteration concatenates together and it is transmitted to the HDMI driver system via Avalon bus memory-map interface. The Figure 7 shows the architecture of the Mandelbrot plotting system.

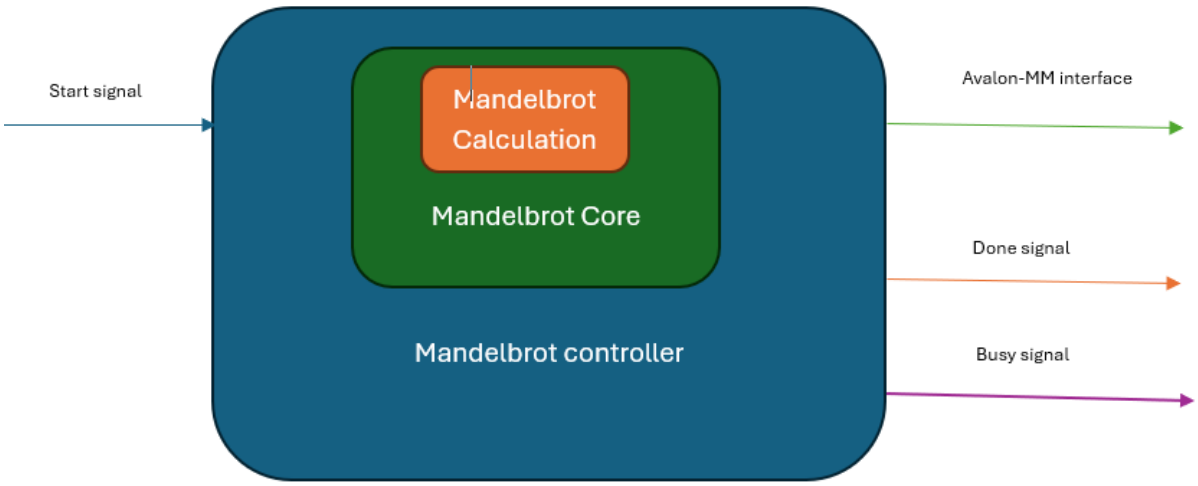

Figure 7 Mandelbrot plotting system architecture

The calculation module: the calculation module is the piece of code that calculates the iteration based on the real number and imaginary number it receives from the controller part. The Figure 8 shows the hardware state machine diagram which is based on the algorithm from the Figure 1.

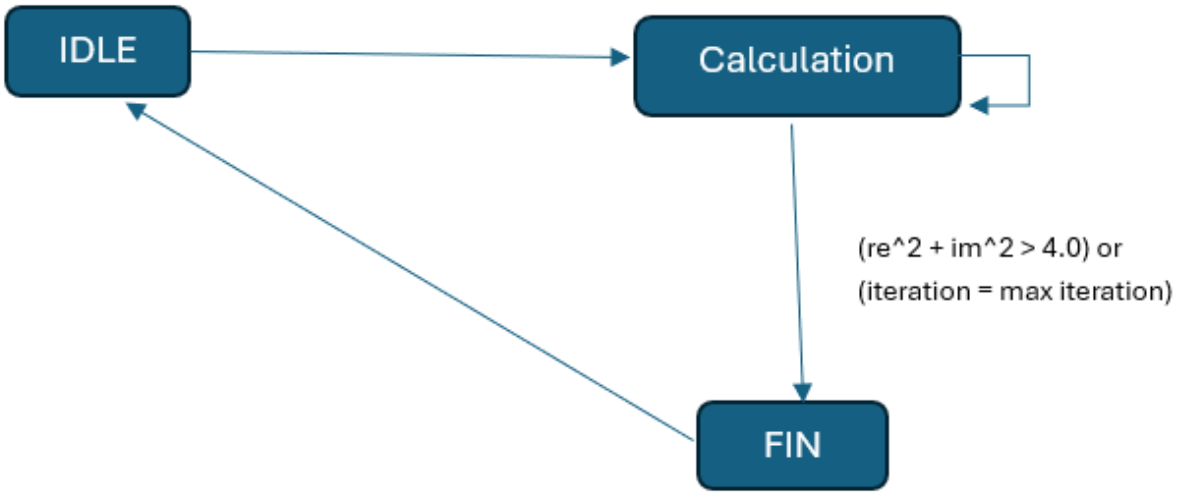

Figure 8 Mandelbrot calculation module state machine diagram

The controller module: The controller module is the separated module which generates the complex number of the Mandelbrot coordinates corresponding to the pixel coordinates. This module converts the result of the iteration and pixel coordinates into a single 32 bits data and uses the Avalon memory-map interface to transmit it to the HDMI driver system.

Testing

The testing is created with the testbenches for both HDMI driver system and the Mandelbrot plotting system. I use the Questasim for the simulation and wave form view. The figure below are the wave form view of the HDMI driver system and the Mandelbrot plotting system.

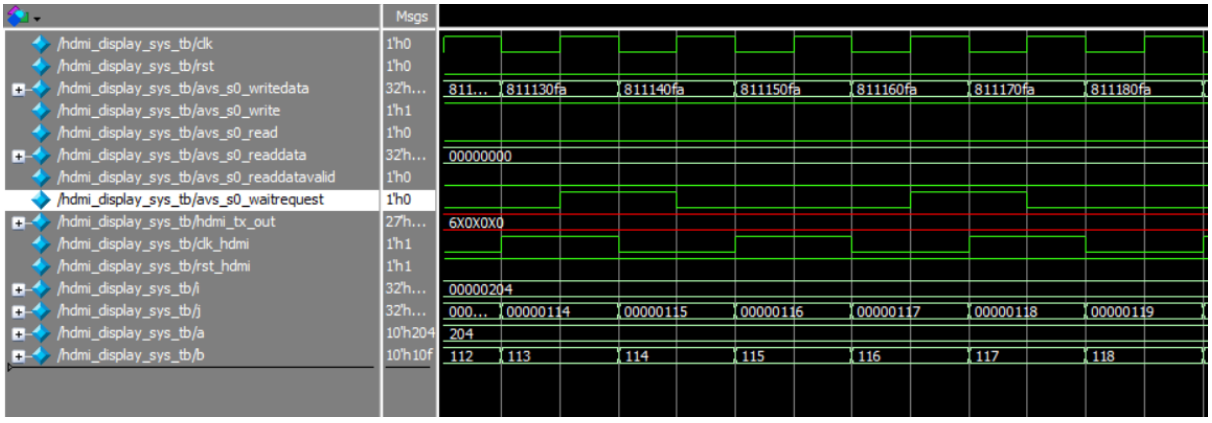

Figure 9 HDMI driver system waveform simulation

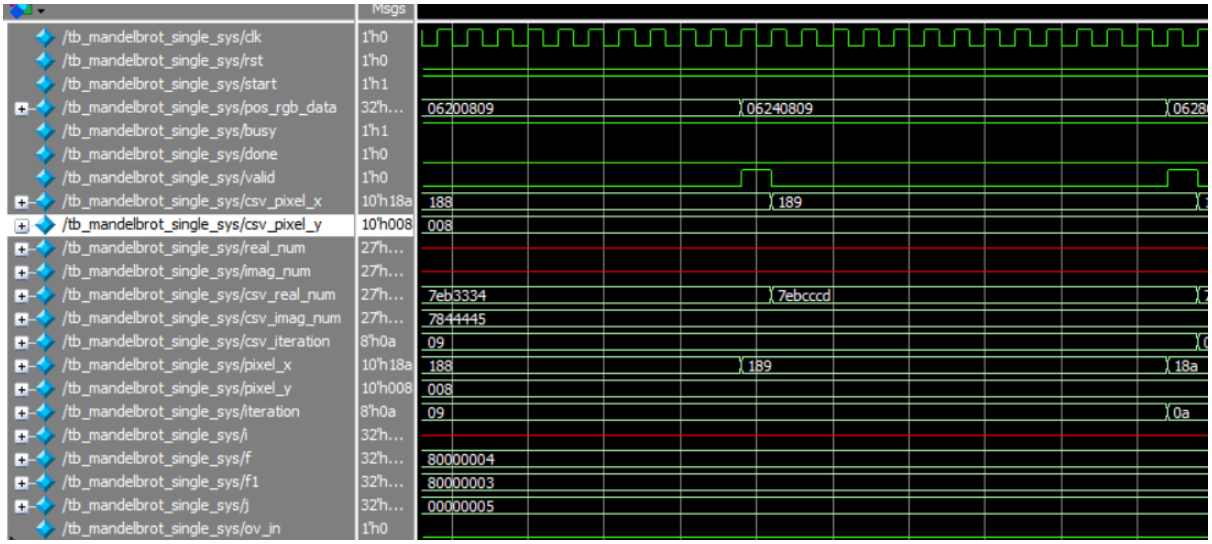

Figure 10 Mandelbrot system waveform simulation

For the testing with the Mandelbrot plotting system, I had to generate another CSV file which contains pixel coordinates, real and imaginary number corresponding to the pixel coordinates and the iteration using a Python script. I write the separate testbench for the Mandelbrot system which prints out the pixel coordinates and the iteration of the module in along with the pixel coordinates and the iteration it reads from the CSV to another txt file shown in the Figure. With this way, I can check the accuracy of the Mandelbrot plotting system in the simulation.

ot > mandelbrot_data.csv > data

```
0,0,-16777216,-8388608,1
1,0,-16737894,-8388608,1
2,0,-16698572,-8388608,1
3,0,-16659251,-8388608,1
4,0,-16619929,-8388608,1
5,0,-16580608,-8388608,1
6,0,-16541286,-8388608,1
7,0,-16501964,-8388608,1
8,0,-16462643,-8388608,1
9,0,-16423321,-8388608,1
10,0,-16384000,-8388608,1
11,0,-16344678,-8388608,1
12,0,-16305356,-8388608,1
13,0,-16266035,-8388608,1
14,0,-16226713,-8388608,1
15,0,-16187392,-8388608,1
16,0,-16148070,-8388608,1
17,0,-16108748,-8388608,1
18,0,-16069427,-8388608,1
```

Figure 11 Data in the CSV file for the testbench

```
csv_x:   0, csv_y:   0, csv_iter:  1, x:   0, y:   0, iter:  1
csv_x:   1, csv_y:   0, csv_iter:  1, x:   1, y:   0, iter:  1
csv_x:   2, csv_y:   0, csv_iter:  1, x:   2, y:   0, iter:  1
csv_x:   3, csv_y:   0, csv_iter:  1, x:   3, y:   0, iter:  1
csv_x:   4, csv_y:   0, csv_iter:  1, x:   4, y:   0, iter:  1
csv_x:   5, csv_y:   0, csv_iter:  1, x:   5, y:   0, iter:  1
csv_x:   6, csv_y:   0, csv_iter:  1, x:   6, y:   0, iter:  1
csv_x:   7, csv_y:   0, csv_iter:  1, x:   7, y:   0, iter:  1
csv_x:   8, csv_y:   0, csv_iter:  1, x:   8, y:   0, iter:  1
csv_x:   9, csv_y:   0, csv_iter:  1, x:   9, y:   0, iter:  1
csv_x:  10, csv_y:   0, csv_iter:  1, x:  10, y:   0, iter:  1
csv_x:  11, csv_y:   0, csv_iter:  1, x:  11, y:   0, iter:  1
csv_x:  12, csv_y:   0, csv_iter:  1, x:  12, y:   0, iter:  1
csv_x:  13, csv_y:   0, csv_iter:  1, x:  13, y:   0, iter:  1
csv_x:  14, csv_y:   0, csv_iter:  1, x:  14, y:   0, iter:  1
csv_x:  15, csv_y:   0, csv_iter:  1, x:  15, y:   0, iter:  1
csv_x:  16, csv_y:   0, csv_iter:  1, x:  16, y:   0, iter:  1
csv x:  17, csv y:   0, csv iter:  1, x:  17, y:   0, iter:  1
```

Figure 12 Data of the txt file generated from the Mandelbrot testbench

Result

The result is a beautiful Mandelbrot set appearing in the screen. The rendering takes less than 1 second to finish. The Mandelbrot is rendered in with 400x300 pixel in the 640x480 standard VGA resolution in the 800x600 HDMI screen which is show in the Figure 13. The hardware usage for the synthesis is shown in the Figure 14.

Figure 13 Mandelbrot set final result

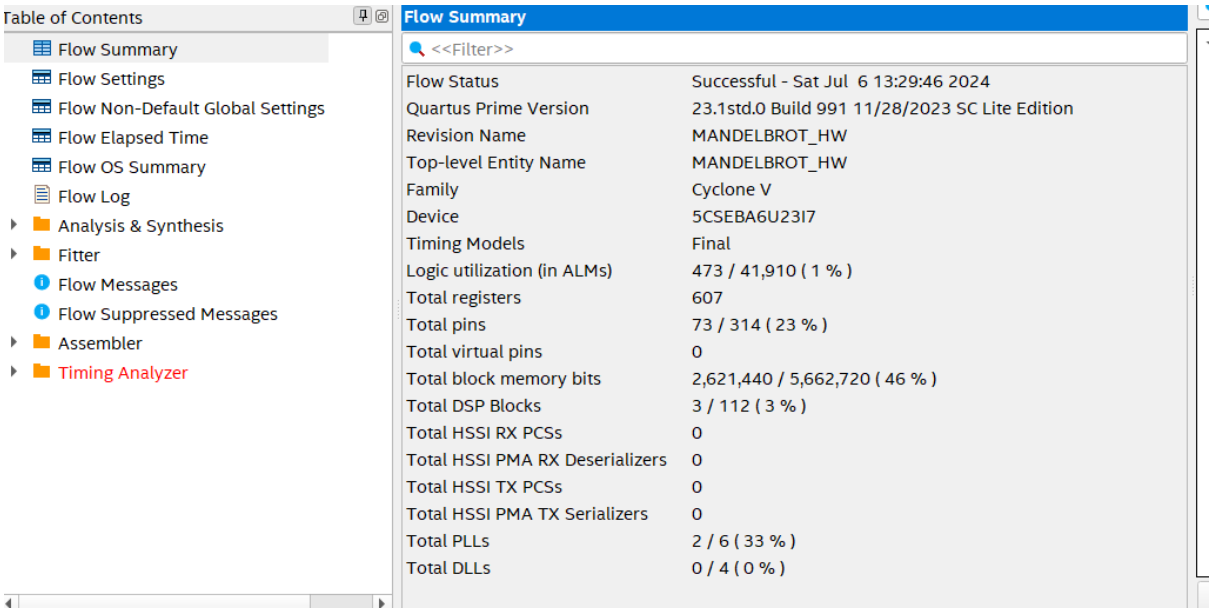

Figure 14 Synthesis hardware usage in Quartus

Conclusion

In this project, I successfully implemented an FPGA-based hardware design for rendering the Mandelbrot set. The primary objectives were to develop a processing architecture capable of performing the necessary iterative calculations for each pixel and to leverage the FPGA's high-speed arithmetic capabilities to improve rendering performance. The implementation demonstrated significant improvements in processing speed and efficiency by using FPGA.

Reflection

Working on the Mandelbrot set rendering project using an FPGA has been an enriching and challenging experience that significantly enhanced my understanding of both fractal mathematics and hardware design. One of the main challenges was implementing efficient arithmetic operations for complex numbers, which required a thorough grasp of fixed-point representation and optimization techniques specific to FPGA architectures. This project not only improved my technical skills in hardware description languages like Verilog but also deepened my knowledge of FPGA programming and configuration.

Appendix

Frame Buffer Pixel Generation Circuit

Mandelbrot FPGA explorer from Mark Browers

Fix point number arithmetic from Van Hunter Adams

Link to the Github code: https://github.com/yourcomrade/MANDELBROT_HW