ECE1718 S1: Special Topics in Computer Hardware - Embedded Systems

Project Report

Mandelbrot Viewer

Implemented on an Altera DE1SoC Board

ECE 1718, Winter 2020

Prepared by – Station no - 73

Diego Defaz – 1004905132 and **Yogesh Iyer -** 1005603438

April 13, 2020

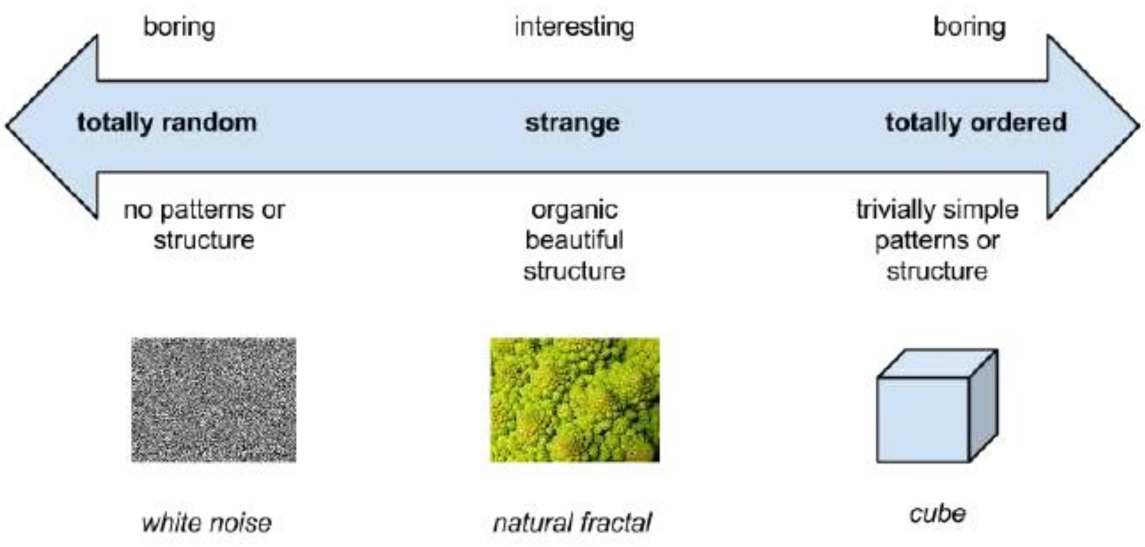
Contents

CHAPTER 1: Introduction

The Mandelbrot Set is a product of IBM research scientist Benoit Mandelbrot's work in the 1970's and 1980's. Mandelbrot was studying the work done previously by both Gaston Julia and Pierre Fatou, who were rival scientists in the 1920's. The advent of modern computer graphics and processing power around the time of Mandelbrot's work allowed him to visualize the Mandelbrot Set in more detail than ever before.

In fact, the Mandelbrot set had been described previously in connection with the related Julia Sets, but it was not possible to appreciate its full beauty and shape before computer technology had matured enough. The term "fractal" was coined by Mandelbrot in his publications describing the Mandelbrot set. The word is a contraction of the term "fractional dimension", which is used to describe how fully a geometric figure fills space.

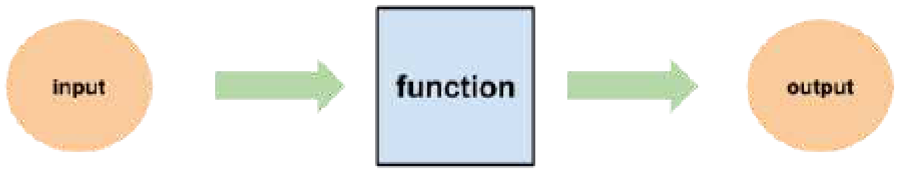
The following summarises the spectrum of objects, from totally random to totally ordered, with the interesting objects between the two. It's as if the tension, the constant battle between total order and total randomness produce the most natural, organic, intricate, beautiful objects.



Lets get to few concepts that we should be fimiliar with so as to get a proper understanding of what Mandel brot set actually is.

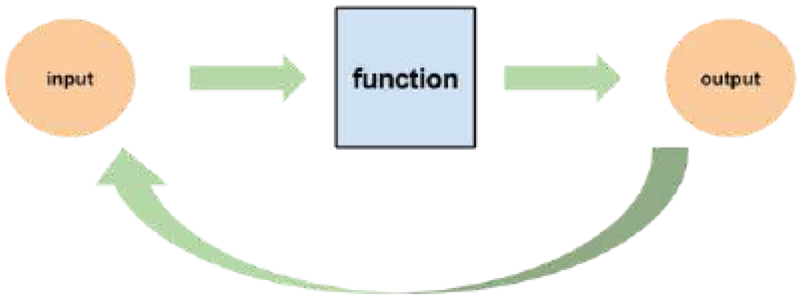
Function

So here we see that the number thrown into this function is called the **input** , and the number that pops out is called the **output.**



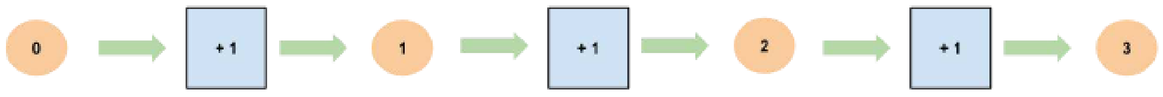
Iteration

The following diagram shows the idea of iterating a function. The input and output are the same as before but this time we take the output and put it back into the function as input for the next iteration.



Let's try it with a starting number of zero, and the function "+ 1". The first time we apply this function we get 0 turned into 1, just as we saw before. Now we take this output 1 and pop it back into the function as input and we get 2 popping out. Again, and we get 3.

Then 4, 5, 6, 7,… you get the idea. The numbers keep getting bigger in steps of 1.



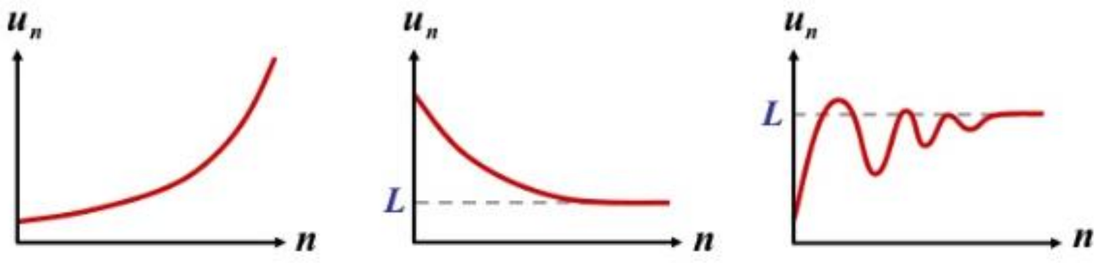
Iteration simply means doing the same thing again and again to produce a series of outputs, just as we did above.

Divergence, Convergence

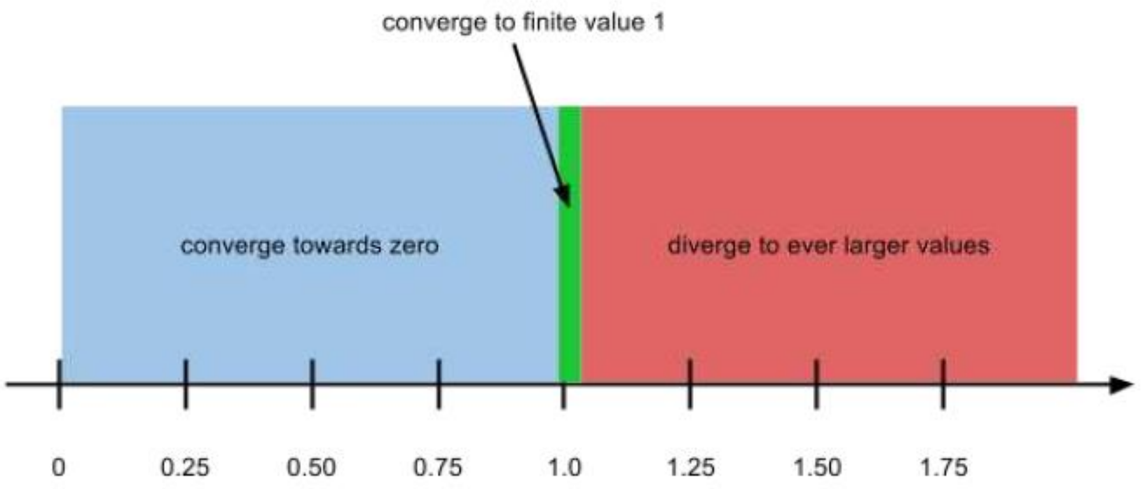
Some sequences never stop increasing, while others eventually settle at a particular number.

If the numbers in a sequence continue to get further and further apart the sequence diverges.

If a sequence tends towards a limit, it is described as convergent.



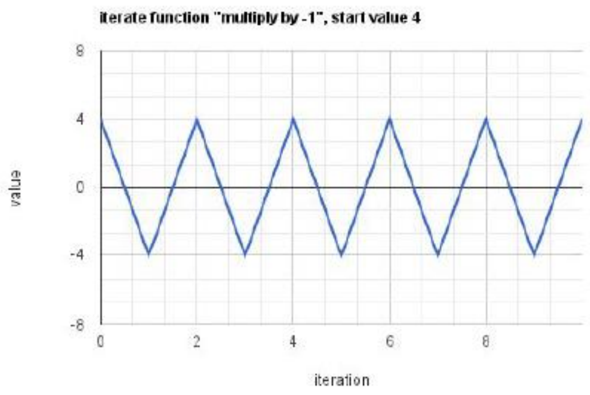
We can start to think about **regions** of the input space and categorising them according to how they impact the evolution of values from an iterating function. So, for the "square the input" function, we have 3 regions; convergence to zero where the seed x is less than 1, convergence to a finite value when the seed x is exactly 1, and divergence when x is more than 1. The following shows these three regions for the "square the input" function.



This idea of marking regions according to the kind of function behaviour that results from them is a core idea in making our own Mandelbrot set. This is because the Mandelbrot set is just that, a kind of atlas marking out the regions according to their behaviour when a function is applied iteratively.

Now let's ask ourselves a question that Are there really no other kinds of function behaviour beyond diverging or converging towards zero or a nonzero value?

For this lets take an example multiply by 1, with starting value 4" which resulted in values +4, 4,+4, 4, … and these don't grow ever larger, nor do they get ever small towards zero. Let's plot it, so we can visualise it.

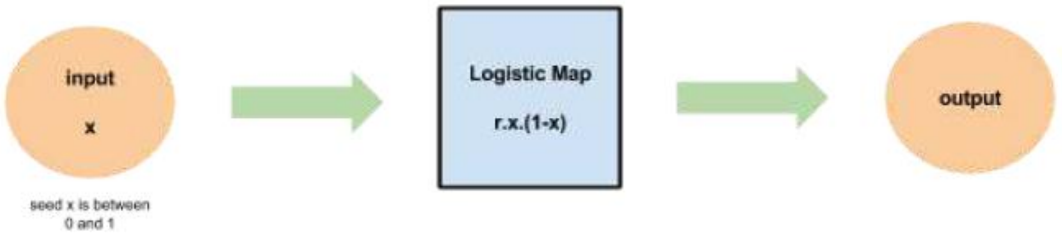


We have a new interesting class of function here, whose values flop between two values. A more scientific way to say this is that the values **cycle** between two values. We can say the function is **periodic.**

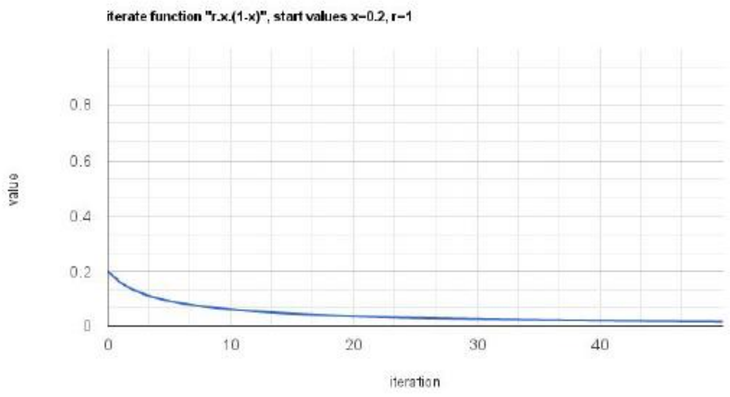
Chaos

Lets try understanding the term chaos by taking a simple example using graphs.

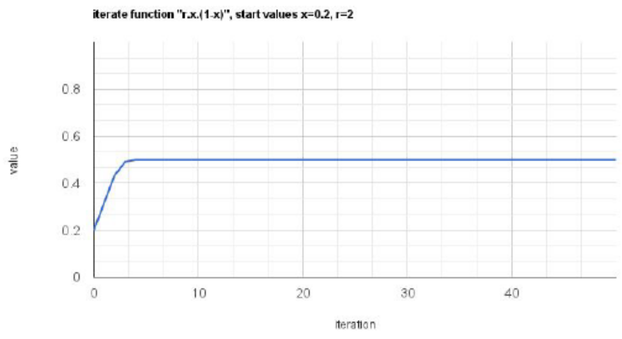
Let's look at an iterative function which has was developed by scientists trying to model population growth. Its called the Logistic Map. The following diagram shows this function.



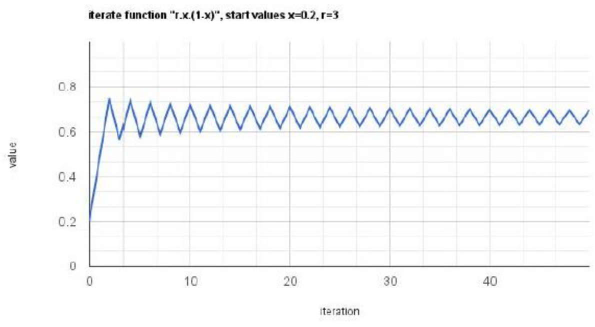
Let's plot a graph to visualise how the function behaves



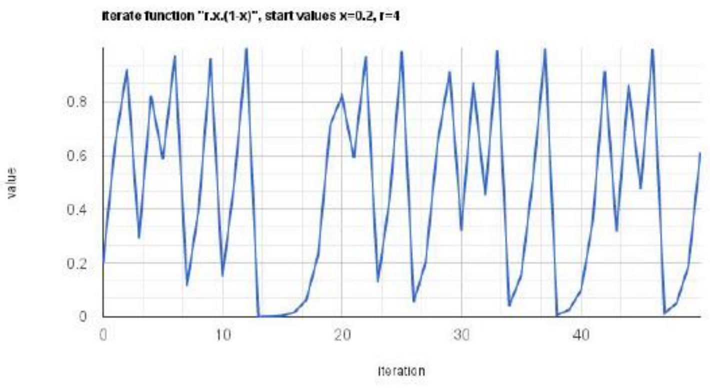
Function "r.x.(1x)",initial value x=0.2, r=1



Function "r.x.(1x)",initial value x=0.2, r=2



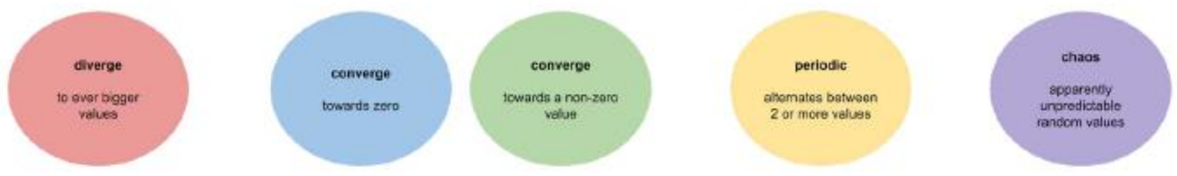
Function "r.x.(1x)",initial value x=0.2, r=3



Function “r.x.(1x)”,initial value x=0.2, r=4

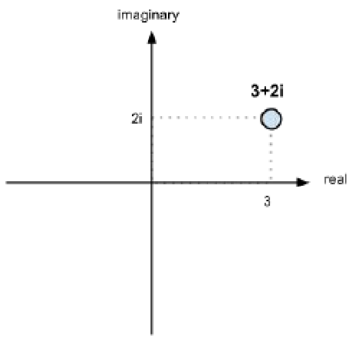
The behaviour we’re seeing for Function “r.x.(1x)”,initial value x=0.2, r=4 appears unpredictable, unruly, random. There is no discernible pattern like a gentle convergence, or even a rapid divergence, not even something we can pick out as a regular periodic oscillation. What we have found, is **chaos**.

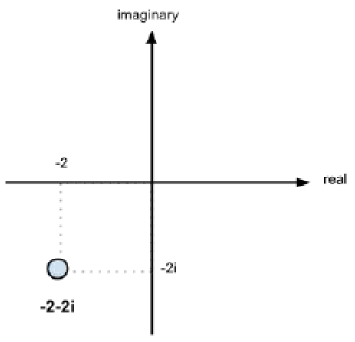
Now we have the following things and are familiar with all of them as shown below.



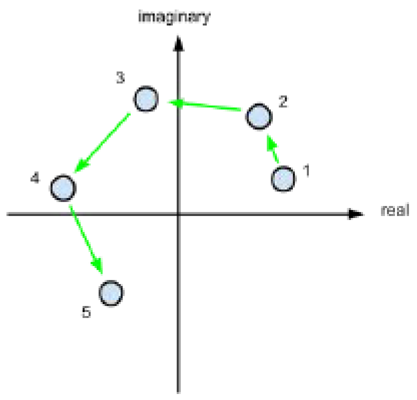
Lets now consider the complex numbers and its behavior

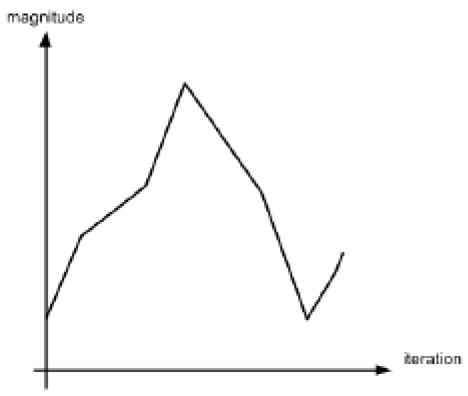
First let’s plot the complex numbers and check how they actually look like. We take an example of numbers (3 + 2i) and (-2-2i).





Visualising how a function changes a complex value over many iterations could be a bit difficult, but we can try it with this way shown as below.

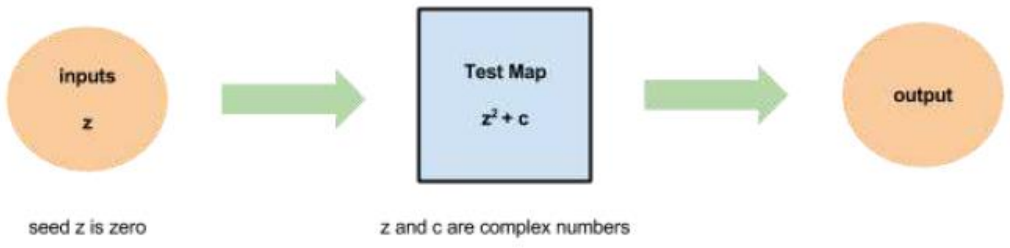




The first way we can visualise the evolution of complex numbers is simply to plot them on the grid as we did above and see if we can follow successive iterations. If the values move around gently, we can see the progression, but if they are densely packed or jump around wildly then we can't easily follow their evolution. Adding connecting lines helps.

The second way discards the real and imaginary components and instead combines them into the magnitude and these are plotted against the iteration number

Now we'll take the function as shown below.



We will now take different values and see on the plots as how it looks like.

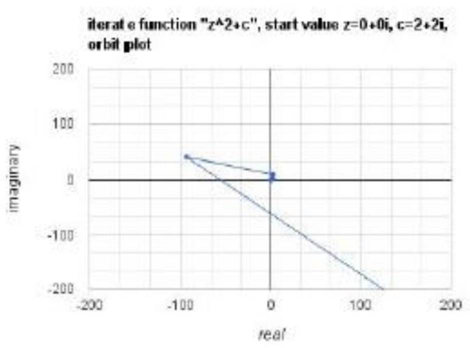
Any complex number z consists of two parts (a + bi), where a is the real part and bi is the imaginary part. The function squares this complex number, which means the multiplying it by itself. So Z^2 is $(\mathrm{a} + \mathrm{bi})(\mathrm{a} + \mathrm{bi}) = a^2 - b^2 + 2(ab)i$

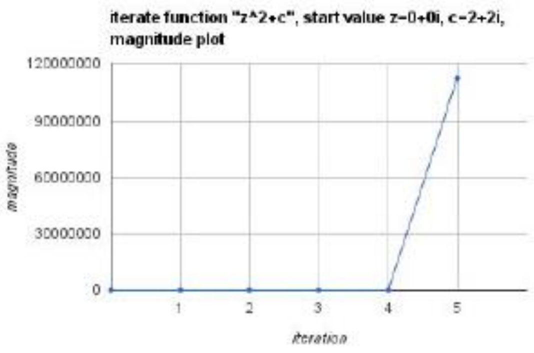
Example 1: c = (2 + 2i)
The following table sets out the iterations of the Test Map with c set to (2+2i)

Function: z 2 + c, for complex z, c, seed z=(0+0i), c set to (2 + 2i)			
iteration	real	imaginary	magnitude
0 (initial value)	0	0	0
1	2	2	2.82842712474619
2	2	10	10.1980390271856

3	-94	42	102.95630140987
4	7074	-7894	10599.8449045257
5	-12273758	-111684310	112356709.793491

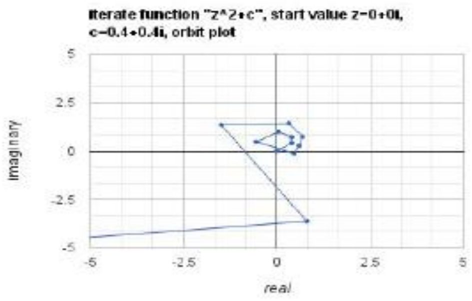
We can see the by looking at the table that the real and imaginary parts of the Test Map output grow in size quickly. The column of magnitudes for each output shows that the complex numbers are getting further and further away from the origin point (0+0i) at an ever-quickening pace. Let's plot an orbit plot and a magnitude plot to confirm it.

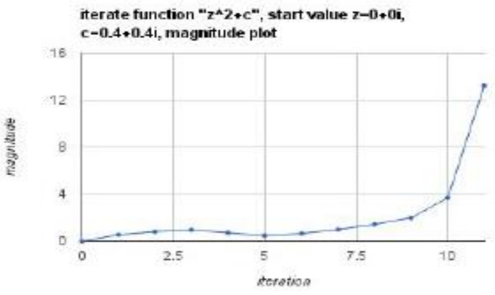




Example 2: c = (0.4 + 0.4i)

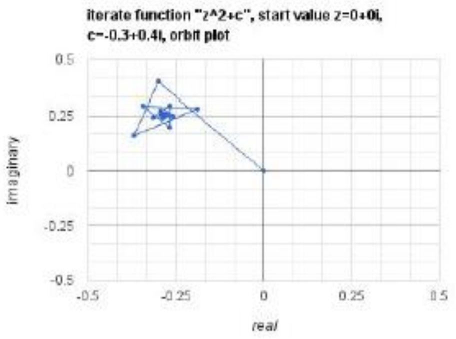
Lets just directly see the plot how it shows.

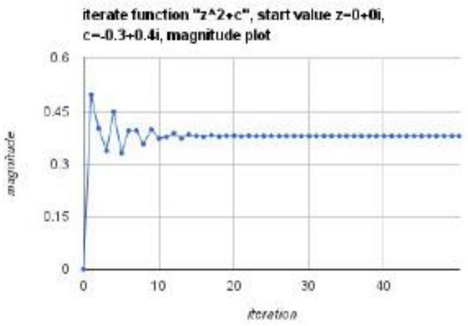




The orbit plot is interesting. It shows the outputs apparently orbiting not far from the origin before finally relenting to some external force and diverging. The magnitude plot shows this too, with the magnitude falling after the third iteration, before relenting and diverging after about the eighth iteration.

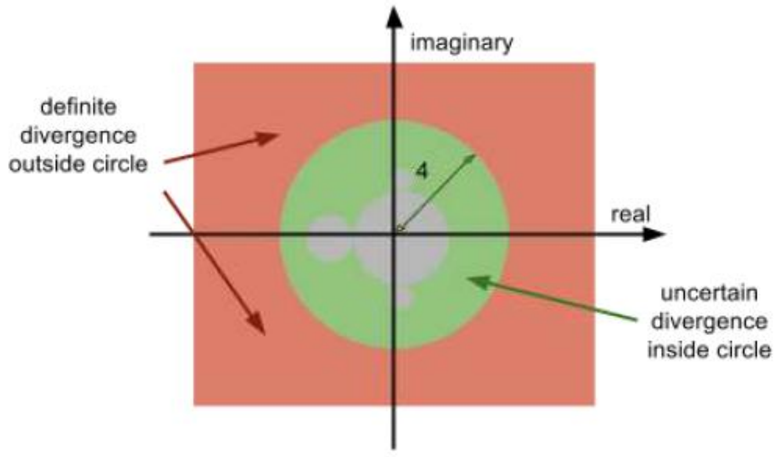
Example 3: c = (0.3 + 0.4i)





The orbit plot now shows the values first circulating a point and then falling into it, almost like an asteroid getting trapped by the gravity of a planet, circulating it before falling into it. The Test Map for c=(0.3+0.4i) converges to a nonzero value.

The mandelbrot set is the set of complex numbers, for which the iterated function described above doesn’t diverge.


imaginary
definite divergence outside circle
4
real
uncertain divergence inside circle

The unusual behaviour here is that we would have expected a circular border between these two regions. What we see instead is a strange shape! It kind of looks like a beetle with arms and legs.

This is amazing. An extremely simple function “$z^2 + c$” when iterated over complex numbers seems to result in a strange insect like shape.

We went through couple of interesting videos on YouTube and there is a video by Numberphile and the concept has been explained by Dr Holly Krieger who is a pure mathematician and lecturer at University of Cambridge.

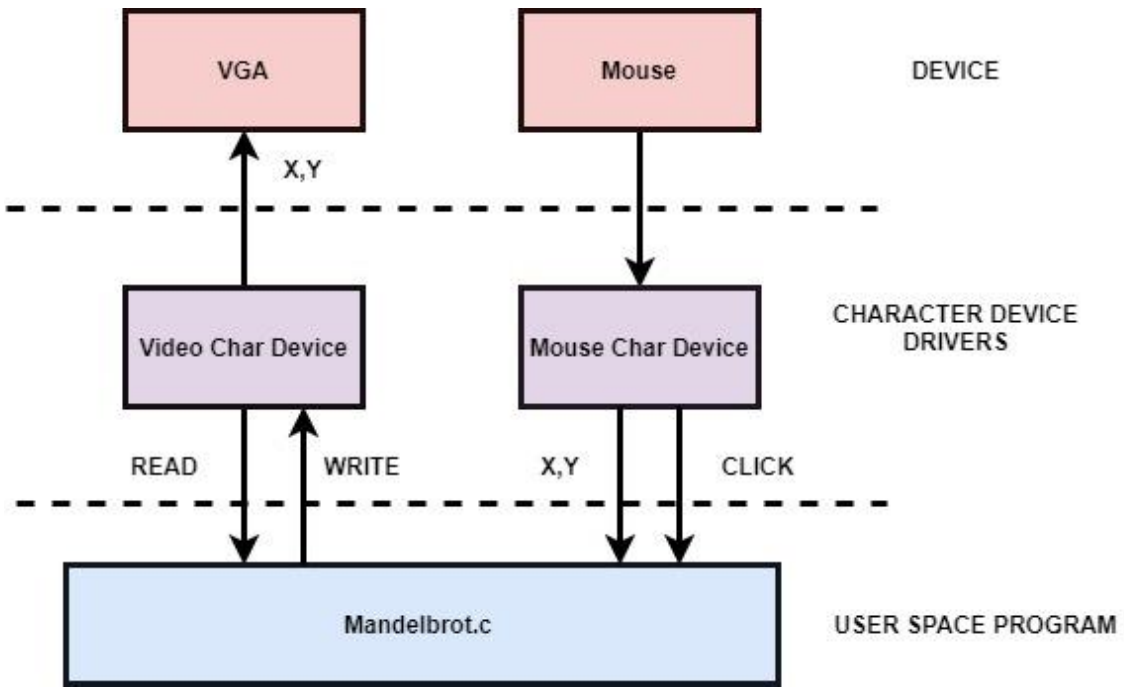
CHAPTER 2 - Hardware

DE1SoC Board - This project has been implemented on DE1SoC which is included on the DE1 development board from TerASIC Technologies. The board includes a full complement of peripherals including FLASH memory, SDRAM, SRAM, RS-232 transceiver, VGA port, LEDs and switches. The Mandelbrot Set image is displayed on a VGA monitor which is connected to the VGA port. In order to provide the highest quality image, the VGA resolution is 640 x 480 pixels

VGA Monitor - VGA monitor is useful in showing the output of the program. It will show the output generated of the Mandelbrot set that has been calculated. The VGA cable would be attached to the board and the monitor. The Intel video driver can be used for better portability.

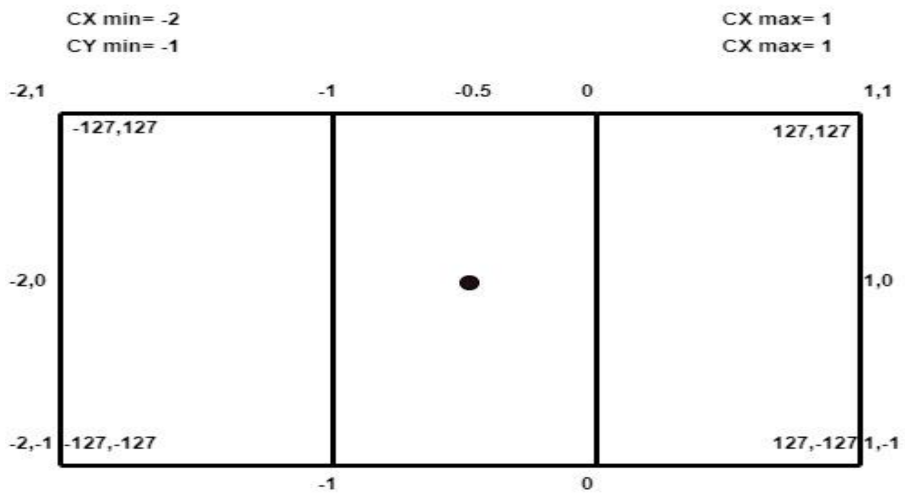
Mouse – Mouse would be useful in taking the zoom in and zoom out of the the Mandelbrot set. Each of the button will have some functions to perform. The mouse would be read at a known rate and the speed scaled, then (usually) numerically integrated to give position on the screen. The USB based Mice will be used in the same manner that we used the Keyboard driver.

Flowchart



We are following the above given flow for the project.

The hardware device consists of Mouse and the VGA monitor. Character device drivers are for video and mouse. Finally, the User space program is the Mandelbrot.c, the VGA monitor showing the output would be controlled using the video character device driver and the read and write of the pixels is being handled using the user space program. Now coming to the mouse part, after plugging in the USB mouse we can see that on terminal window the data comes that is for each button: Left – 1 , Middle – 4, Right -2. x and y would show the movement coordinates. These things have been defined in the user space program.



As we see from the image given above that the cx and cy are the two floating point arrays required to convert the screen space to the image space. The given cx min, cy min, cx max, cy max are the variables useful for initial optimal resolution for the view. The mouse coordinates in the complex space is shown as cx_m, cy_m.

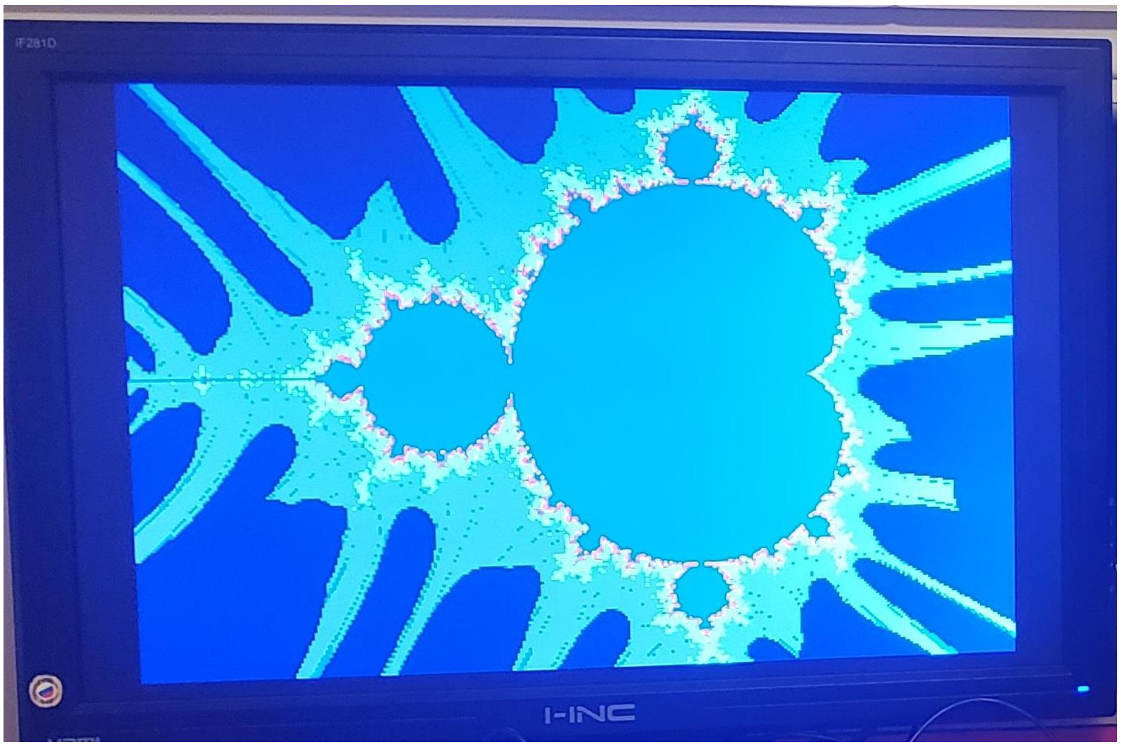
Note:

The program Mandelbrot.c prints the complex coordinates where the mouse is in although it doesn't show the zoom function at all in the VGA monitor. The program Mandelbrot_old.c has the zoom with a factor of 2 but it doesn't show the mouse coordinate in the complex space.

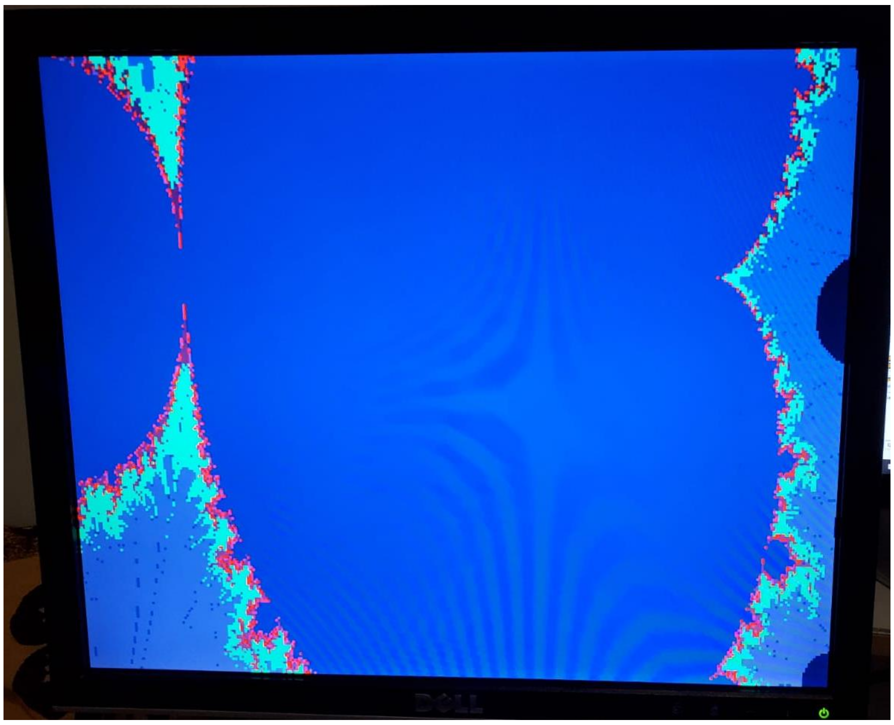
Regarding the mouse part, what we are doing is that we defined left button to do the zoom in part, middle button would take back to the original screen and the right button will help in getting the zoom out. Currently the zoom in is with just a factor of 2, we tried couple of things to get more zoom and make it a function rather than just a number but we ended up getting very weird result hence we removed it and kept only the part that is working.

CHAPTER 3 – Results

So here we can see the output of the Mandelbrot set.



And here is the image of the zoomed in version using mandelbrot_old.c program



CHAPTER 4 – Conclusion & Future work

In conclusion, this project was somewhat successful. We couldn't achieve exactly what we wanted to do with our mouse driver as we wanted to have more zoom in and zoom out. Specially we didn't get a chance to meet in person making the situation more difficult to convey and communicate the messages for the work in the project.

We also researched ways to improve our algorithm. One potential way was writing in a way that would have taken advantage of perturbation theory and series approximation to improve performance. The idea is that once the iteration series for one pixel is computed in expensive high precision arithmetic, it can be used to solve nearby pixels with much faster single-precision arithmetic. According to Wikipedia and other sources, this method can provide anywhere from a 10x to 100x performance increase at high zoom levels. Unfortunately, we didn't have time to implement this optimization.

References

[1] En.wikipedia.org. 2020. *Mandelbrot Set*. [online] Available at: <https://en.wikipedia.org/wiki/Mandelbrot_set> [Accessed 13 March 2020].

[2] "Coding Games and Programming Challenges to Code Better", *CodinGame*, 2020. [Online]. Available: https://www.codingame.com/playgrounds/2358/how-to-plot-the-mandelbrot-set/mandelbrot-set. [Accessed: 15- Mar- 2020].