A Mandelbrot Set Generator Implemented on an Altera DE1 Board ECE 573, Winter 2008

Jesse Armagost and Eddie Yang*

March 21, 2008

*This report covers the work done by Jesse. Eddie will turn in his own project report.

1 The Mandelbrot Set and Fractals

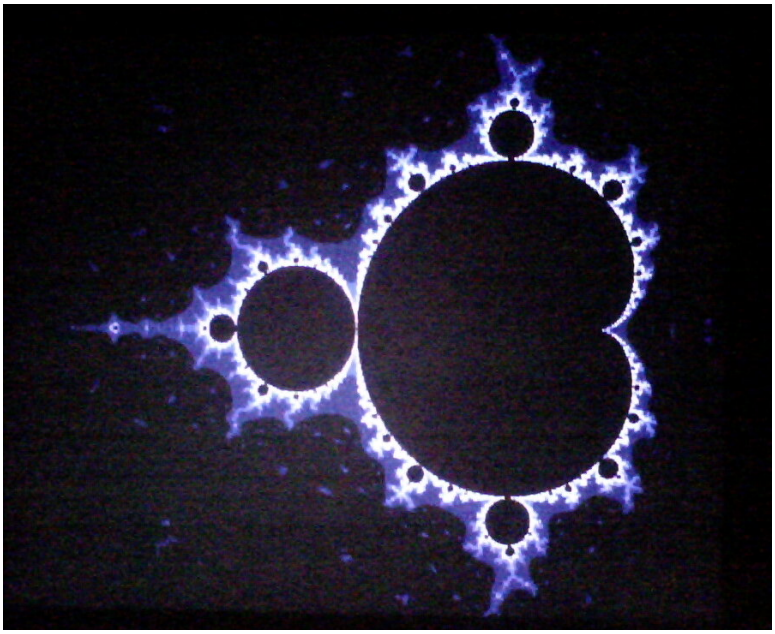

Figure 1: The Mandelbrot Set

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.

To generate the Mandelbrot Set, an iterative formula is applied to points in the complex plane. The formula is surprisingly simple:

$$z_1 = z_0^2 + c \tag{1}$$

where z_0 is the z_1 of the previous calculation and c is a fixed point in the complex plane.

During repeated applications of the formula on a point in the complex plane, the magnitude of z_1 will either remain finite or diverge to infinity. If the point diverges, the point is *not* in the Mandelbrot set—otherwise it is.

This discussion describes a rather boring binary image—either points are in the set or they're not. To add more artistic value to the image, false colors can be applied. The colors are applied according to how fast a point diverges to infinity. This is given by how many iterations are computed before the point diverges.

In practice, it is not reasonable to detect whether a point diverges to infinity, so a shortcut method can be used. The shortcut takes advantage of the fact that if the magnitude of z_1 reaches 2, then the point is

guaranteed to diverge.

2 Description of the DE1 Board

This project was implemented in an Altera Cyclone II FPGA (EP2C20), 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 board is controlled by a PC which is connected via the RS-232 connection, which can autobaud to several popular baud rates. The FPGA computes the Mandelbrot Set and stores the graphical data in the SRAM, which is used as a frame buffer. 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 1024 x 768 pixels.

3 Fixed Point Math

The Mandelbrot Set is typically computed using floating point math, which in today's computers means IEEE-754 floating point math. Implementing the IEEE-754 standard in a small FPGA, however, would use far too many resources and would severely reduce the operating frequency of the device. For this project it was decided to use fixed point math instead. Fixed point math is used in many microprocessors and embedded systems in which there is no floating point hardware.

In fixed point math the decimal point is implied to exist at a certain bit position, and bits to the left of the decimal take on standard binary values while bits to the right of the decimal take on values that are the reciprocal of the standard binary values. Fixed point math is represented using the "Q" notation. For example, using Q2.2 fixed point math means that we have 2 bits to the left of the decimal point and 2 bits to the right. We can then represent the following numbers exactly with this math:

$$
\begin{aligned}
0000 &= 0 + 0 + 0 + 0 = 0 \\
0001 &= 0 + 0 + 0 + \tfrac{1}{4} = 0.25 \\
0010 &= 0 + 0 + \tfrac{1}{2} + 0 = 0.50 \\
&\vdots \\
1110 &= 2 + 1 + \tfrac{1}{2} + 0 = 3.50 \\
1111 &= 2 + 1 + \tfrac{1}{2} + \tfrac{1}{4} = 3.75
\end{aligned}
$$

Standard two's complement can be formed by using the leftmost bit to represent sign. The final version of this project uses signed Q2.29 fixed point math, but the design is parameterized to use any desired values.

4 Description of Verilog Modules

Figure 2 shows a block diagram of the Mandelbrot set generator. The figure illustrates which modules operate in which of the three clock domains that were used in the design. The following sections describe the modules in more detail.

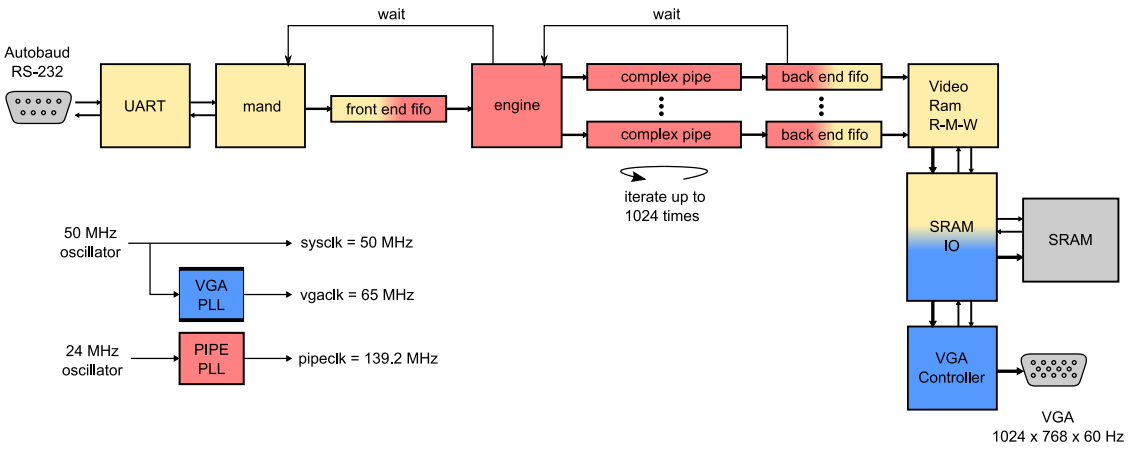

Figure 2: Block Diagram of the Mandelbrot Set Generator

4.1 mand_defs

The mand_defs module defines various values that are used in the rest of the design.

4.2 top

The Top module is the top level of the design. It contains all of the FPGA pins and instantiates all of the lower level modules.

4.3 pipe_pll and vga_pll

The design uses three separate clock domains, with two of them being generated by PLLs. The pipe_pll module contains the pipe PLL, which generates the 139.2 MHz complex pipe clock from the 24 MHz oscillator input. The vga_pll module contains the vga PLL, which generates the 65 MHz VGA pixel clock from the 50 MHz oscillator input.

4.4 reset_filter

The reset_filter module provides a way to produce a proper reset signal in each clock domain. An asynchronous reset signal can be asserted at any time relative to the clock, but must be deasserted *synchonously* with the clock. The reset filter module provides this functionality. There is one reset filter per clock domain, for a total of three.

4.5 mand

The mand module is responsible for responding to commands from the uart and uart_decode modules, and generating a series of complex values for the complex pipes to operate on. When panning or zooming, the mand module adjusts the current point in the complex plane appropriately, and also adjusts the various pan and zoom factors in the module. When commanded to start, the mand module calculates the value of the upper left corner of the image in the complex plane, and then iterates through all 1024 x 768 values. For each of these values, it loads the complex plane coordinate into the front end pipe, where it will be removed

by the complex pipe engine and operated on. The mand module will continue to load complex coordinates into the front end FIFO until all 1024 x 768 pixels are processed. If the front end FIFO temporarily fills, the mand module will wait until free space appears in the FIFO.

4.6 engine

The engine module instantiates the front end FIFO, the complex pipes and the back end FIFOs. When the engine module detects that complex coordinates are waiting in the front end FIFO, it removes them and loads them into an empty slot in a complex pipe. If there are no free slots in any complex pipe, the engine module will wait until a slot becomes available. It continues this process until it completely empties the front end FIFO.

The front end FIFO data takes the following form:

[zz:yy] = sgn_current_coord_im
[xx:20] = sgn_current_coord_re
[19:10] = v_pixel_count
[9:0] = h_pixel_count
where sgn_current_coord_im is the complex coordinate imaginary part,
sgn_current_coord_re is the complex coordinate real part,
v_pixel_count is the vertical screen pixel for this complex coordinate,
h_pixel_count is the horizontal screen pixel for this complex coordinate

(xx, yy and zz are variables because the size of the signed complex coordinates can vary depending on the fixed point math that is used.)

When a complex coordinate finishes in a complex pipe, the result is placed in the complex pipe's back end FIFO. If there is no room in the back end FIFO, the complex pipe is prohibited from being loaded. The load prohibit signal is generated when there are still enough slots left in the FIFO to accommodate any coordinates that still may be in the complex pipe (up to 9 coordinates in flight at once), because there is no way to stop them from being loaded into the back end FIFO once they start in the complex pipe.

The data in each back end FIFO takes the following form:

[36] = in_set
[35:20] = iterations
[19:10] = v_pixel
[9:0] = h_pixel
where in_set indicates whether the complex point is in the set,
iterations is the number of iterations before exiting the complex pipe,
v_pixel is the vertical screen pixel for this complex coordinate,
h_pixel is the horizontal screen pixel for this complex coordinate

4.7 complex_pipe

The complex pipe is the heart of the design. It is a pipelined module that is responsible for computing iterations of the Mandelbrot formula. The complex pipe was optimized as much as possible to achieve high

speed. This was fairly difficult to do, however, because the free version of the Altera software (Quartus II) has the manual floorplanning features disabled. With floorplanning enabled, each complex pipe could be constrained to a specific area of the FPGA, close to the dedicated multipliers that are used inside it. Without this ability, the best that could be done was to pipeline the complex pipe to 9 stages and let the place-and-route software do the best it could. The result was that the complex pipe could run at approximately 144 MHz. The closest value that could be generated with the PLL and 24 MHz input clock was 139.2 MHz.

The major blocks in the complex pipe were generated with the Quartus Megawizard tool, which performs parameterized generation of IP blocks. The blocks that are generated in this way are presumably tuned to achieve the highest performance possible for a given FPGA. There is no known way to look inside these blocks and see what optimizations are being made, however, so there must be an element of trust that the synthesis tool (and the tool developers) are making smart decisions.

Complex coordinates enter the front of the pipe with the load_en signal when there is a free slot, and then begin iterating. The pipe can hold nine complex coordinates at a time. Each time a complex coordinate passes through the pipe, its magnitude is computed and compared to 2. The number of iterations is also kept track of. If the number of iterations reaches 1024, or the magnitude of the result reaches 2, the complex coordinate is finished in the pipe. In the first case the coordinate is in the set, and in the second case it is not. Actually, instead of finding the magnitude of the result, the magnitude *squared* is compared to two *squared* (or four). This, of course, saves some logic resources while accomplishing the same effect.

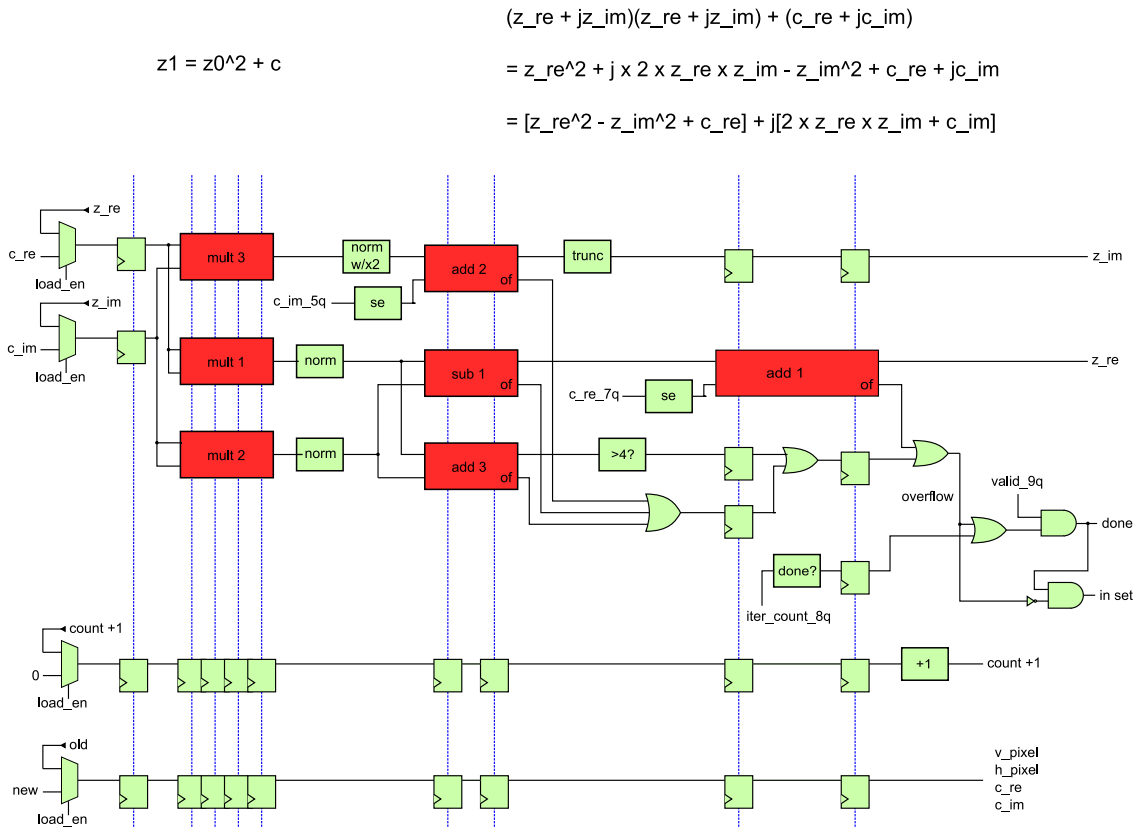

Figure 3: The Complex Pipe

4.7.1 multiplier block

The multiplier block is used three times in the complex pipe: one instantiation generates z_{re}^2, one instantiation generates z_{im}^2, and the third instantiation generates 2 * z_{re} * z_{im}. Although the design is fully parameterized, the final version of the project used 32 x 32 bit signed multipliers. To increase Fmax, each of the multipliers was specified to be internally registered four times.

4.7.2 adder block

The adder block is used three times in the complex pipe: twice to implement parts of the iterative formula, and once to check the magnitude of the result for the escape test. After the first multipliers, the internal precision is 2 x 32 bits = 64 bits, so the adders were specified to use signed 64 bit addition. The adders were specified to be internally registered twice in an effort to increase Fmax.

4.7.3 subtractor block

The subtractor block is used once in the complex pipe to implement the z_{re}^2 - z_{im}^2 term. As with the adders, it is specified to implement signed 64 bit math, and it is registered twice to increase Fmax.

4.8 SRAM IO

Once a complex coordinate has finished in the complex pipe, it is written to the pipe's back end FIFO, where it is transferred into the slower system clock domain and written into the SRAM video buffer. Normally this would be a simple operation, but it is complicated by the fact that the SRAM is only 256K x 16. 256K addresses are not enough to allocate one pixel per address. The solution is to store four pixels at each SRAM address and perform a read/modify/write operation when we want to add new data. In order to properly store a 4 bit pixel value at a particular SRAM address, we must read what was already at the SRAM address and include it in the write so that it does not get lost.

The SRAM IO module controls access to the SRAM by the video RAM read/modify/write module and the VGA controller module. Since the SRAM is only a single-ported device, the VGA controller must have exclusive access to it during a VGA frame. If this were not the case, we would see corrupted data on the screen when the RAM writing module was performing a write. As described below, the writes to the SRAM are only allowed during vertical and horizontal retrace periods.

4.9 VGA Controller

The VGA controller is responsible for generating the VGA signaling. Figure 4 shows the timing involved in creating VGA video. As shown in the figure, the lines of a frame are denoted with a horizontal sync signal and the frames are denoted with a vertical sync signal. During these sync (or retrace) periods, and just before and after, the monitor requires no visual information. This is because the raster is off the edge (or top or bottom) of the screen during these periods. Since the vga controller does not need to access the SRAM for visual data, we can allow the new pixel data to be written it during these periods. In this way, we can give two separate modules access to the same single-ported SRAM device and still maintain a pristine VGA image.

It is interesting to note that in areas of the Mandelbrot set that do not require a lot of iterations in the complex pipe, that the system bottleneck is actually the write of the pixel data to the SRAM. In areas of the

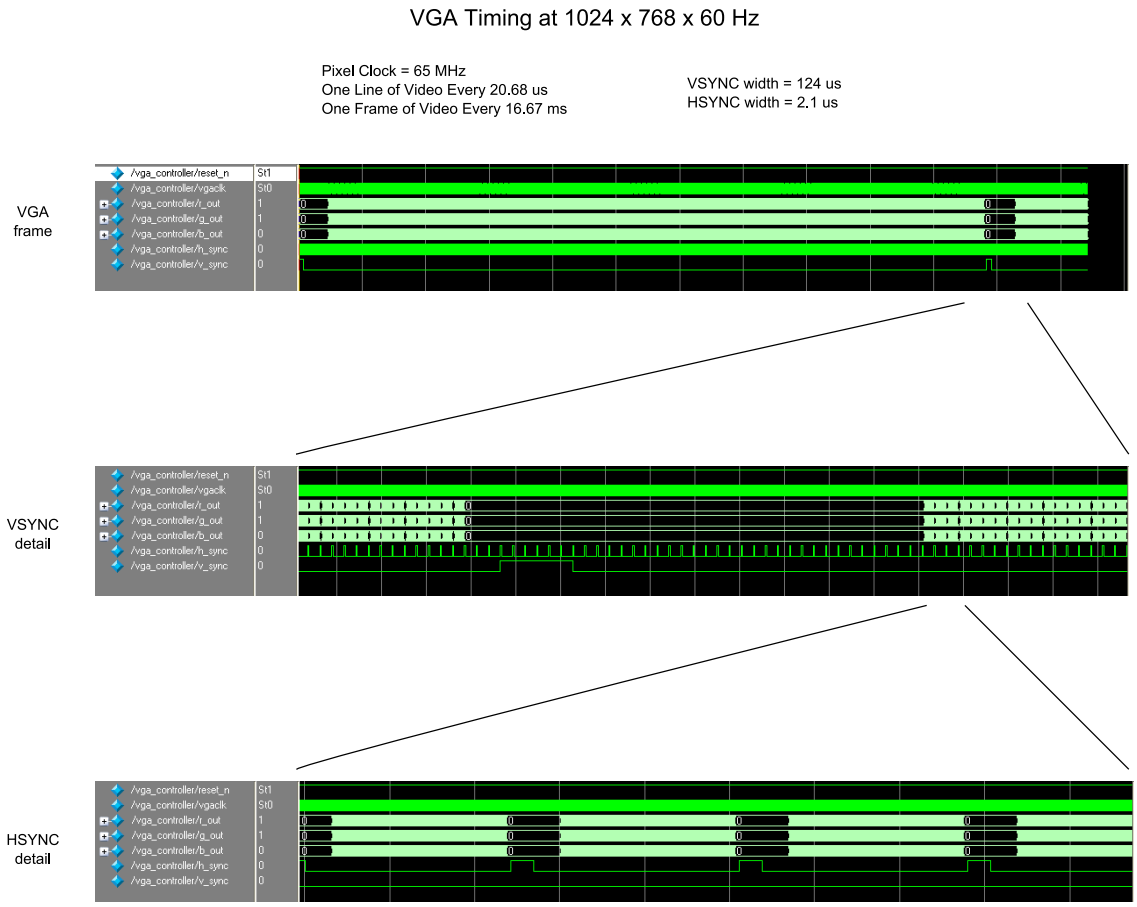

Figure 4: VGA Timing for 1024 x 768 x 60 Hz

set that require a large number of complex pipe iterations, the system bottleneck is the speed of computation in the complex pipes.

5 Navigating in the Set

While the image of the set shown in figure 1 is fascinating, it is much more fun to pan and zoom within the set. By using the appropriate keys on the computer connected to the FPGA board, we can explore the set more fully. Cursor mode allows us to place a cursor on the screen which represents the center of drawing and zooming. The cursor is shown in figure 5. The cursor can be moved around the screen, and then we can zoom into the area under the cursor. Figure 6 shows a possible path that can be taken by panning and zooming.

The control keys will undoubtedly be described in Eddie's paper (because he worked on this part of the design), but I will show them in table 1 for reference.

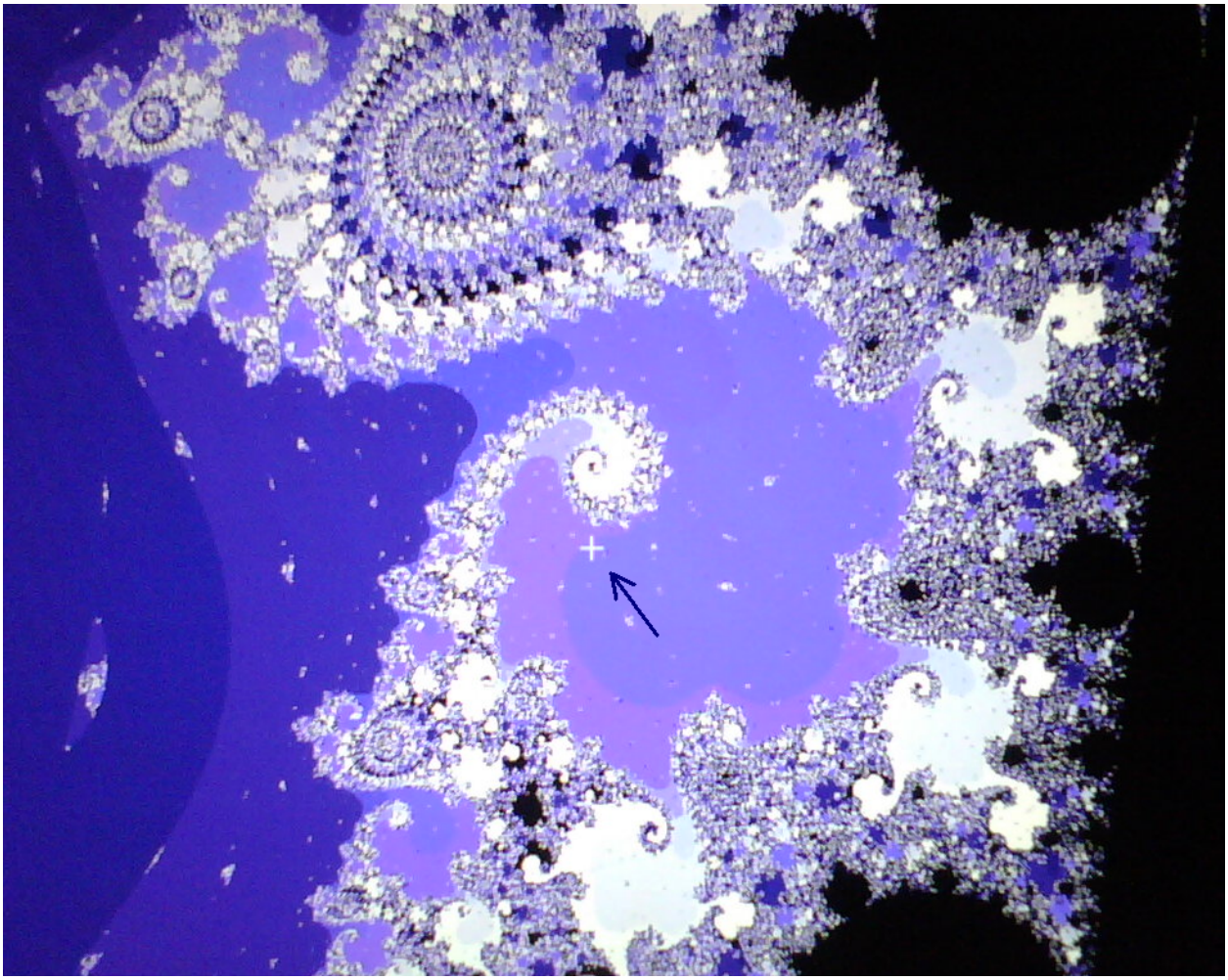

Figure 5: The Navigation Cursor

6 Summary and Estimate of Development Time

This project was a joy to work on. It was my most involved FPGA project to date and I appreciated working on something that was very challenging, a lot of fun, and that also earned class credit. Table 2 lists all the modules in the project, and their authors. The total estimated time working on this project is between 80 and 100 hours.

Key	Function
t	Pan Up
g	Pan Down
f	Pan Left
h	Pan Right
i	Zoom In
o	Zoom Out
c	Cursor On/Off
s	Start Drawing
a	Animate
r	Restart

Table 1: Navigation Keys

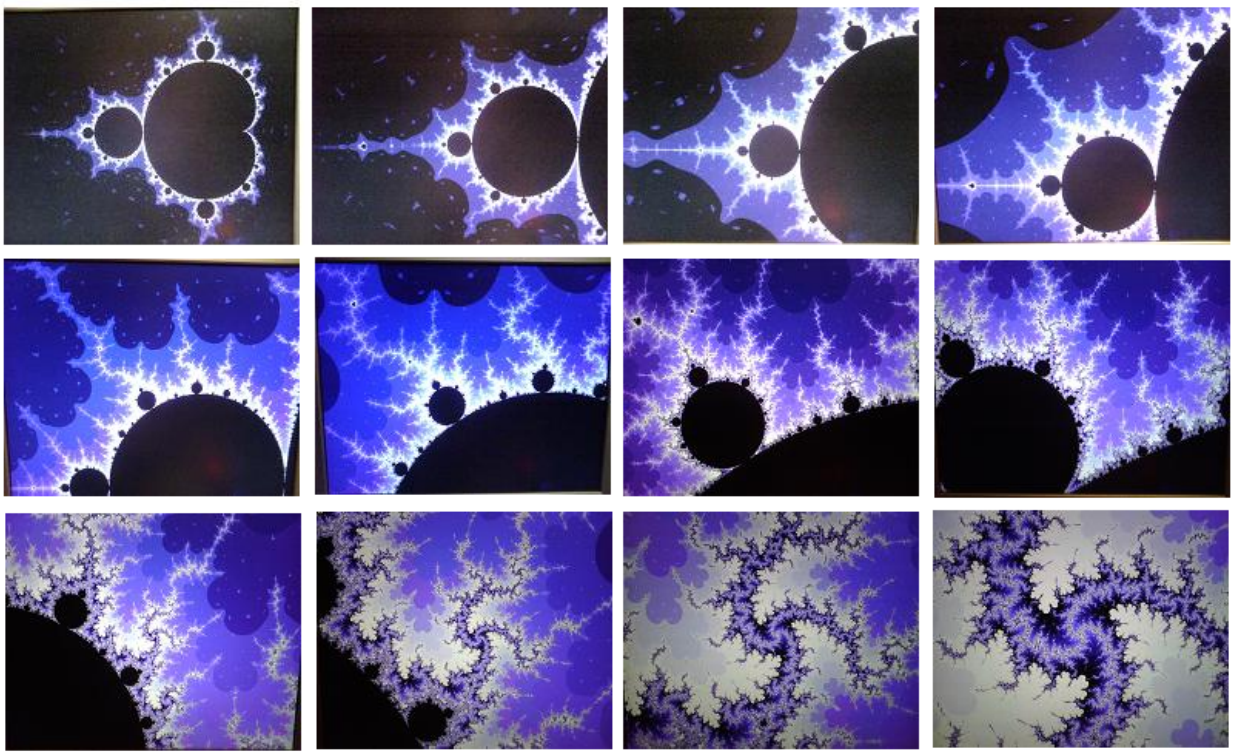

Figure 6: One Possible Path Through the Set

Module Name	Function	Author
add.v	64 Bit Signed Adder	Altera MegaWizard Generator
be_fifo.v	Complex Pipe Back End FIFO	Altera MegaWizard Generator
complex_pipe.v	Complex Math Pipeline	Jesse Armagost
cursor.v	Provides Cursor Navigation Support	Jesse Armagost
engine.v	Feeds Complex Coordinates to Pipes	Jesse Armagost
fe_fifo.v	Complex Pipe Front End FIFO	Altera MegaWizard Generator
mand.v	Navigation and Coordinate Generation	Jesse Armagost
mand_defs.v	Project-wide Definitions	Jesse Armagost
mult.v	32 Bit Signed Multiplier	Altera MegaWizard Generator
pipe_pll.v	Complex Pipe Clock PLL	Altera MegaWizard Generator
reset_filter.v	Reset Synchronization and Filtering	Jesse Armagost
sram_io.v	Provides Off-FPGA SRAM Access	Jesse Armagost
ssd.v	Seven Segment Display Logic	Jesse Armagost
ssd_bank.v	Seven Segment Display Logic	Jesse Armagost
sub.v	64 Bit Signed Subtractor	Altera MegaWizard Generator
top.v	Top Level of the Design	Jesse Armagost
uart.v	Autobauding UART	Jesse Armagost and Eddie Yang
uart_decode.v	Keystroke Decoding	Eddie Yang
vga_controller.v	Provides VGA Timing	Jesse Armagost
vga_defs_1024_768.v	VGA Definitions for 1024 x 768	Jesse Armagost
vga_palette.v	Provides Different Color Palettes	Jesse Armagost
vga_pll.v	VGA Pixel Clock PLL	Altera MegaWizard Generator

Table 2: Description of Verilog Modules