# Mandelbrot Set in Chisel

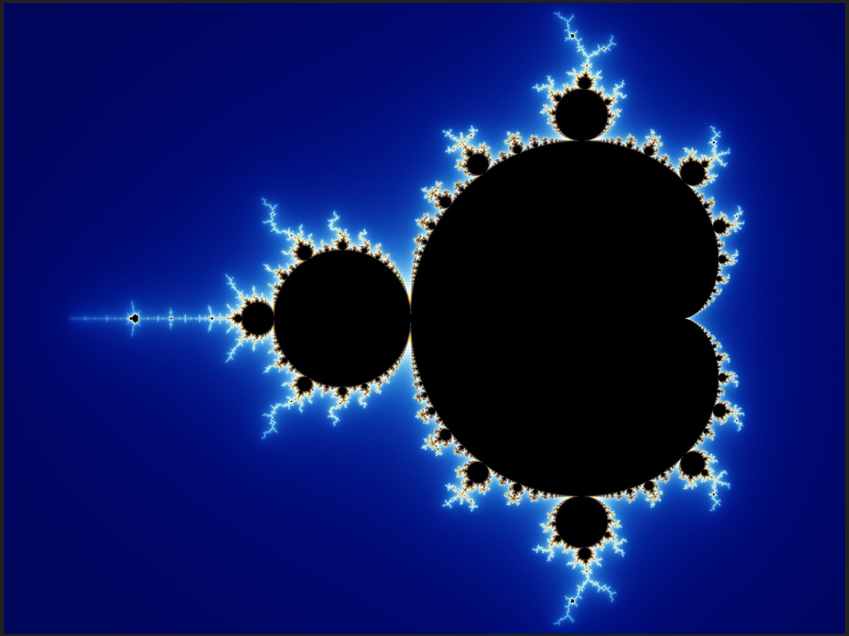
Ben Grant

Repository: https://github.com/190n/chisel-mandelbrot/

# The Mandelbrot set

- Iterating a function on complex numbers
- Numbers in the set are those that do not diverge ($|z| > 2$) after some number of iterations

$$f_c(z) = z^2 + c$$



# Mandelbrot in Chisel

- Use (experimental) fixed-point support
- Similar to matrix multiplication: iterate on multiple numbers at once, and send results for multiple numbers at once (but not the whole area)
- Parameters: `precision`, `iters`, `parallelism`, `cyclesPerTransfer`
  - Range is always from -2 - 1.25i to 0.5 + 1.25i.
  - Fixed-point numbers use 4.precision bits.
- Output: `Valid(Vec(Bool()))`
  - Currently powers on, computes the whole grid, outputs the results, and then idles.
  - Each output block is a portion of a row.
  - For simplicity, each row must contain an integer number of output blocks and parallelism blocks (one block being a bunch of `MandelbrotIters` that work in parallel). `ManelbrotParams` enforces these and many other requirements.

# Complex numbers

- `Complex` is a simple bundle of two fixed-point numbers with the same width and binary point.
- Companion object with conversion utilities:
  - `def apply(width: Int, binaryPoint: Int, re: Double, im: Double)`
  - `def apply(precision: Int)`
  - `def apply(re: FixedPoint, im: FixedPoint)`
- Bundle literals are helpful, but verbose.

# The Mandelbrot function

- `MandelbrotFn` is a combinatorial module to calculate $z^2 + c$.
- Complex arithmetic is written out (no `ComplexALU`) since this is the only place in hardware where I perform complex arithmetic.
- `ComplexModel` is a complex number in Scala which does support arbitrary arithmetic and multiplication, in addition to the Mandelbrot function.

# Iterating the function

- `MandelbrotIter` is a module parameterized by the precision and the number of iterations.
- A complex number `c` is input, and it tracks the function’s iterations in an internal register and returns whether or not the value diverged.
- Test case models this iteration in Scala using `ComplexModel`.

# The main module

- `Mandelbrot` constructs many `MandelbrotIter` modules and mostly waits for them to finish.
- Results stored in `SyncReadMem` (1600-bit register is slow).
- `map` and `reduce` are helpful to check whether all the modules are ready or finished; `foreach` to initialize their I/O.

  -
    ```
    when(iterators.map{ _.io.c.ready }.reduce{ _ && _ }) {
        // all iterators are ready
    }
    ```

# Runner

- `MandelbrotRunner` uses Chiseltest to create and simulate the module.
- Changes from `MandelbrotTester` test case that already prints a grid:
  - Prints each character twice (i.e. `##` for a single grid point in the set) to account for most terminal fonts that are taller than they are wide (Iosevka's aspect ratio is exactly 1:2!)
  - Works around a bug that causes some values to be rearranged in the output (see comment for details).

# Results

```
                              ####
                              ##
                          ########
                        ############
                      ################
              ####    ################
              ######  ##################
######################################
              ######  ################
              ####    ################
                      ##################
                        ##############
                          ############
                              ######
                              ####

                                ##
```

```
                            ##
                            ##
                     ##  ######
                     ############
                     ################
                     ################
              ############################
########################################
              ##########################
              ##  ####################
                     ##################
                     ################
                     ##  ############
                            ########  ##
                            ##
                            ##
                            ####
```

Both use step size 1/8, but the left uses 32 bits of precision while the right only has 3 (minimum for that step size). You can see that the left side has a better-looking shape.

# Future possibilities

- Select a custom window instead of the entire range
  - `MandelbrotParams` already has `xMin`, `xMax`, `yMin`, and `yMax`; they're just assigned constants. They could be provided as inputs so that the module can run multiple times. I would just have to make sure that there are no subtle assumptions about their values baked into the code.
- Implement a different fractal
  - Fairly easy or fractals that also use an iterated function that may or may not diverge on complex numbers (for instance, Multibrot sets are Mandelbrot sets with z raised to a different power in the main function).
  - In other cases, the difficulty varies by how different the fractal in question differs from the Mandelbrot set. A lot of the I/O and parallelism in the `Mandelbrot` module itself could probably be used, but the different core operations might require very different helper modules (for instance, lookup tables for fractals that use trigonometric functions).