SW 4 - Hailstone Sequence

Jonathan Schwartz

Section 003L

TA: Conner Rickermann

Date submitted: 3/25/2021

**Theory**

The Hailstone sequence is a sequence which has two options, depending on whether the previous number is even or odd. In the even case, the following number is simply dividing the number by two, while in the odd case, the following number is multiplying by three and adding one. When applied to binary numbers, we need to look at both cases separately, and handle each when necessary with a multiplexer.

In the case of an even number, in binary this will always store a 0 in the least significant bit, allowing us to divide by two by simply eliminating this bit and allowing the second least significant bit to become the least significant bit: EDCBA → 0EDCB, weighted 16, 8, 4, 2, 1 respectively in each *5-bit* number. This case is very easy to do.

In the case of an odd number, in binary, we can multiply by two by adding a 0 as the least significant bit: 0EDCBA → EDCBA0, weighted 32, 16, 8, 4, 2, 1 respectively for each *6-bit* number, although we can do this with 5-bit inputs because the 6th bit, whether it is the least or most significant bit is a 0 which can be supplied with a “Ground”. If we add a 5-bit number to its doubled 6-bit form, this is the same as multiplying by three. Now all that is necessary is to add 1 to get 3N+1.

The only thing to worry about now to generate a single subsequent term based on the previous one in the hailstone sequence is to determine which of these two functions to utilize. This depends on whether the input is even or odd, so we can use the least significant bit, which is weighted as 1, as the control bit in a multiplexer, because an odd number will have A=1, and an even number will have A=0. If we utilize two 4-bit adders with carries and 2 multiplexers with 1-bit controls and 4-bit inputs, we can achieve the 8-bit output desired for a 5-bit input with the hailstone sequence, accounting for even and odd options.

**Table of Values**

| E | D | C | B | A |  | S | T | U | V | W | X | Y | Z |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |  | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |  | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |  | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

**Question**

What circuit/s would you add to your design to enable the Hailstone sequence to be continuously generated?

The first thing I would add would be for the output to connect to the input of the original circuit, and this would (assumingly) never exceed the 7-bit output of 11111, so 2 4-bit adders and MUX’s will suffice.

Next, I would implement a clock with adequate delay and use AND gates with the inputs of the hailstone sequence to allow for a delay in the computation of each bit so there are no incorrect numbers if the gates in the chips have different speeds.

Finally, I would connect the output of the circuit to its own input with OR gates and a separate place for the 5-bit original input to pulse once so that if it starts, 00000000 and the 5-bit input will result in the 5-bit input, then, after, the 5-bit original input will disconnect and the OR will simply be wired to the output of the circuit. Now that the initial input and the subsequent inputs are accounted for, I would have a sequence of binary probes listing off the values of the outputs of the MUX’s at the end and this will change as the hailstone sequence continually evaluates the next number with the delay from the clock.