Group 54

Eva Suska 260618967
Marcel Morin 260605670

g54_randu - Lab 2

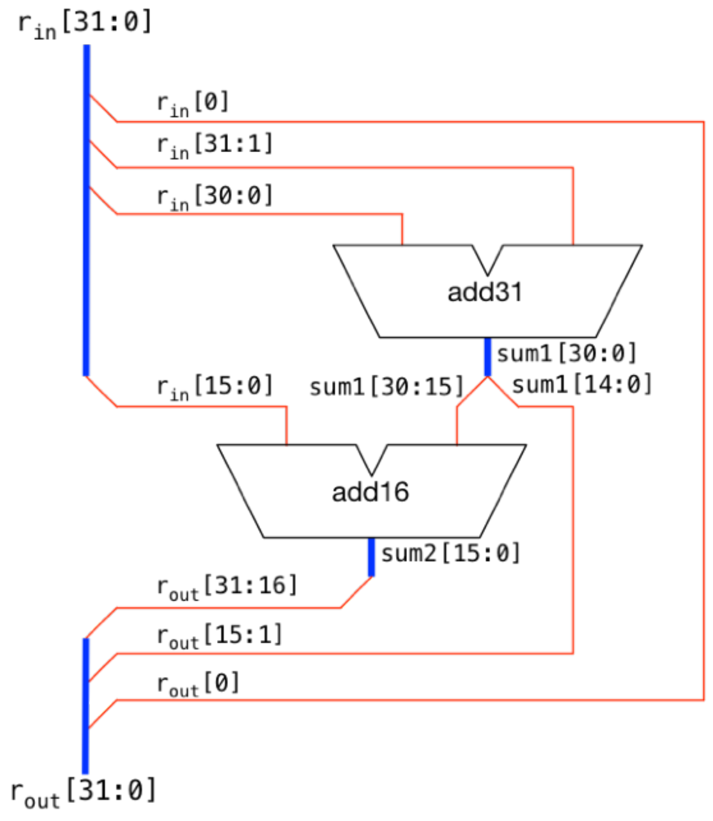
Description

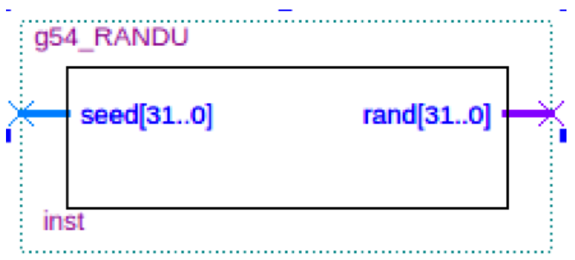
Input: 32 bit seed[31..0]
Output: 32 bit rand[31..0]

This circuit implements the IBM RANDU function used to generate random numbers using the Linear Congruential Generator algorithm. It takes in a 32 bit input seed as some initial value. It then computes R = mod(a * SEED + b, c), with a=65539, b=0, c=2^31. Since the number 65539 is 10000000000000011 in binary we can use additions and two shifts to preform the multiplication, and computing the modulo of a number which is a power of 2, can be done simply by setting all bits of a beyond the (N-1)th bit to zero.

To be the most efficient this is implemented with wiring the least significant bit of the input to the least significant bit of output since it doesn't change. Also using a 31 bit adder with the rest of the bits of the input (bits 31-1) and the seed shifted left by one (bits 30-0). Then wiring the first 15 bits (14-0) of that adder output to the next bits of the rand output (bits 15-1). With the rest of the output of the 31 bit adder (bits 30-15) and a 16 bit left shift of the input seed (bits 15-0) we add these together using a 16 bit adder circuit. The corresponding output of that adder then becomes the remaining bits of the output rand (bits 31-16).

To generate multiple numbers you would feed the previous rand generated into the input seed.





Eva Suska 260618967
Marcel Morin 260605670

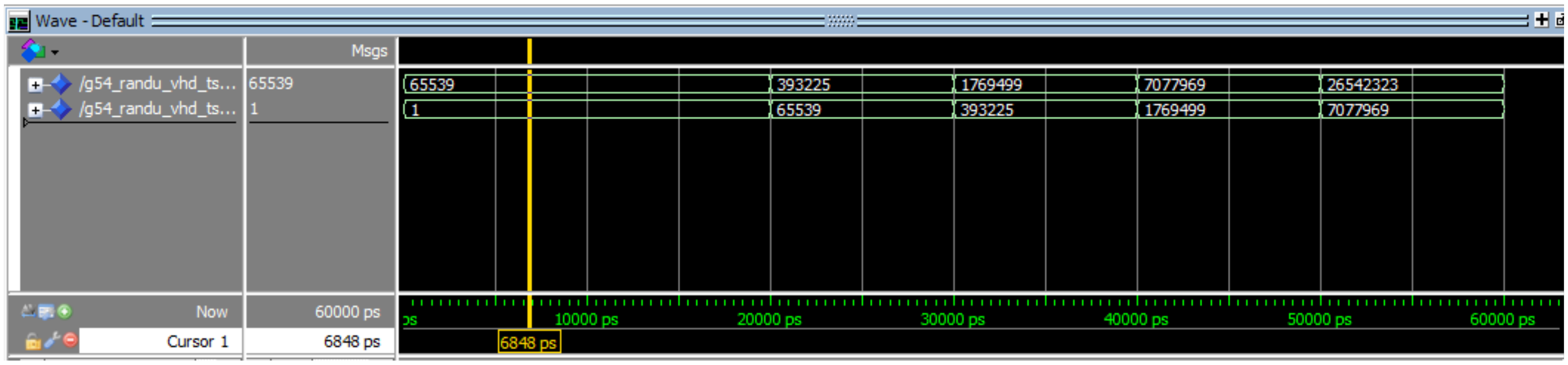
Testing

To test we created a Test Bench file to simulate results. Since we couldn't practically test all the (2^32) cases, we choose an initial seed value equal to 1 and waited 10 ns then fed that output as the input seed and repeated the same process 4 more times.

```
seed <= "00000000000000000000000000000001";
                WAIT FOR 10 ns;

                for i in 0 to 4 loop
                        WAIT FOR 10 ns;
                        seed <= rand;
                end loop;
```

This wave output was as follows:



To check if these results were correct we calculated what the expected results would be and made sure they matched up.

```
a = 65539
b = 0
r0 = 1
r1 = (65539 ∗ r0 + b) mod 2^31 => 65,539
r2 = (65539 ∗ r1 + b) mod 2^31 => 393,225
r3 = (65539 ∗ r2 + b) mod 2^31 => 1,769,499
r4 = (65539 ∗ r3 + b) mod 2^31 => 7,077,969
r5 = (65539 ∗ r4 + b) mod 2^31 => 26,542,323
```

We also tested the same pattern but with seed set to 5 initially and compared the results to calculated results.