
# CSE331 – Computer Organization

Nevzat Seferoglu
171044024

## Homework 3

# *Project Purpose:

In this homework, we are going to implement 31-bit unsigned multiplication operation in Logisim. There are different parts that are mandatory to implement the given operation. Those are datapath, control unit, main execution of the program. I also wrote six different test cases that linearly increasing as a problem size.

**Optimizations that I made:**

- Using **D-FLipFlop** for efficiency reason to keep overflow record in safe place.
- I did not use comparator, counter carry-out bit was enough.

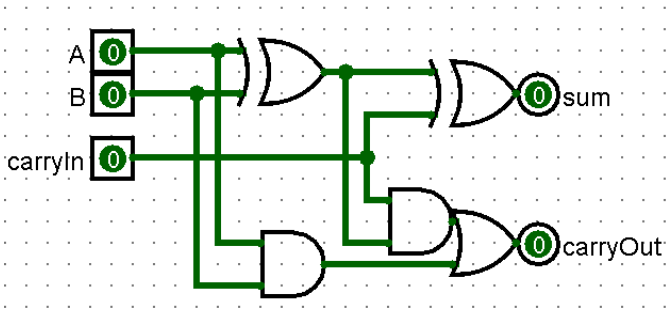
**Bonus Parts that I made:**

- 32Bit-Adder
- 64Bit-Shifter

# *Datapath Design:

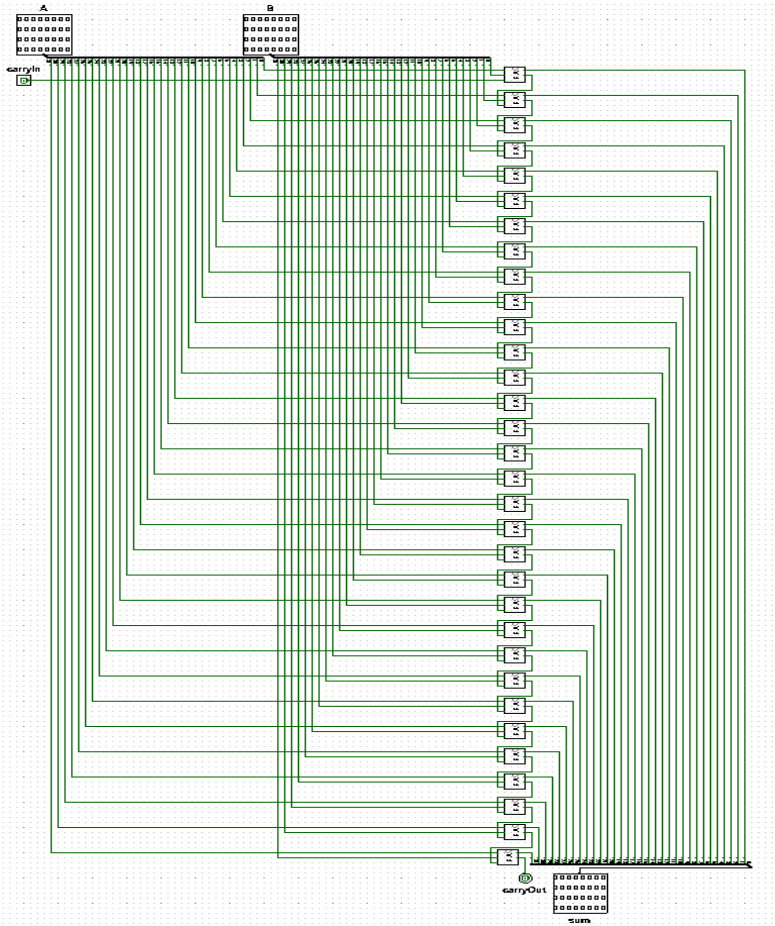
My datapath is consist of different combinational unit for some operation. These are 1bit-adder, 32bit-adder, 64-shifter and 64-bit register which works with other part of the datapath synchronize. I will explain each of them one by one.

## *1Bit-Adder:



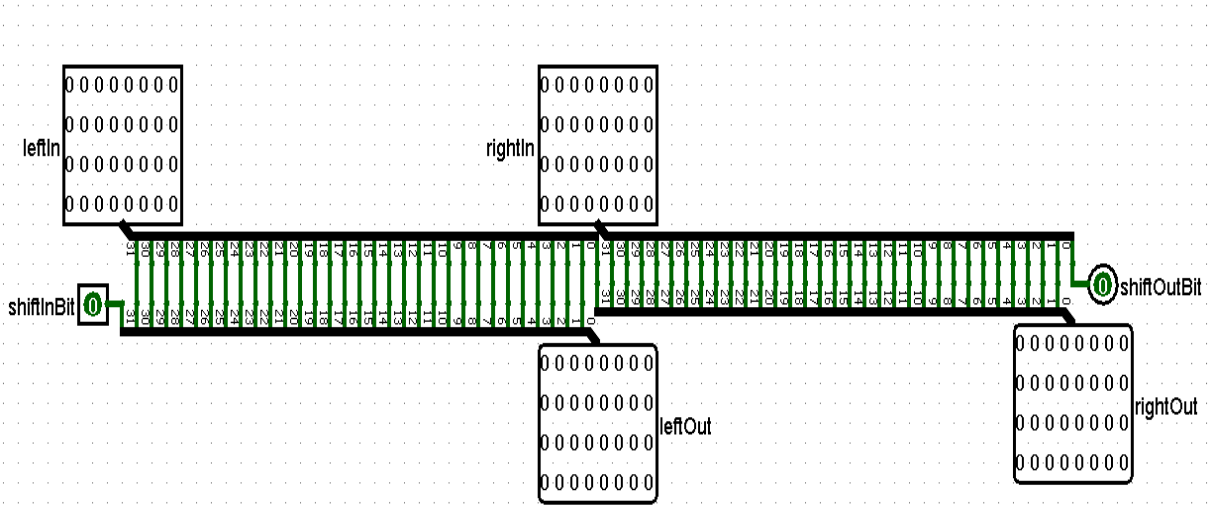

Typical 1-bit full adder implementation. There is nothing special. I benefitted from this to implement 32-bit adder.

# *32Bit-Adder:




I exploited 32 amounts of 1bit full-adder to implement 32-bit adder. I had to use splitter for connecting different cable coming from each 1bit full adder.

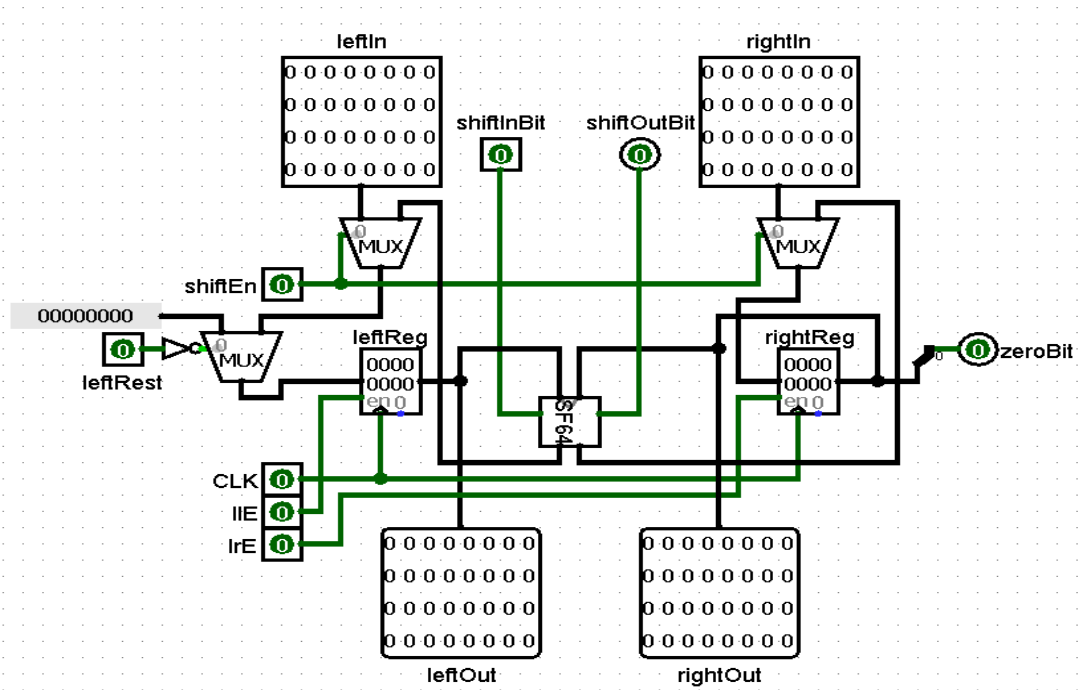
# *64Bit-Shifter:




In this part, shift bit was a tricky part. Because there was a situation that occurred overflow while calculating summation of left side of the product. Other bits must be shifted one bit throughout right. I also give an output shift out as carry.

Outputs are separated into two parts left and right. These are handled by 64bit-register at the product side.

# *64Bit-Register:

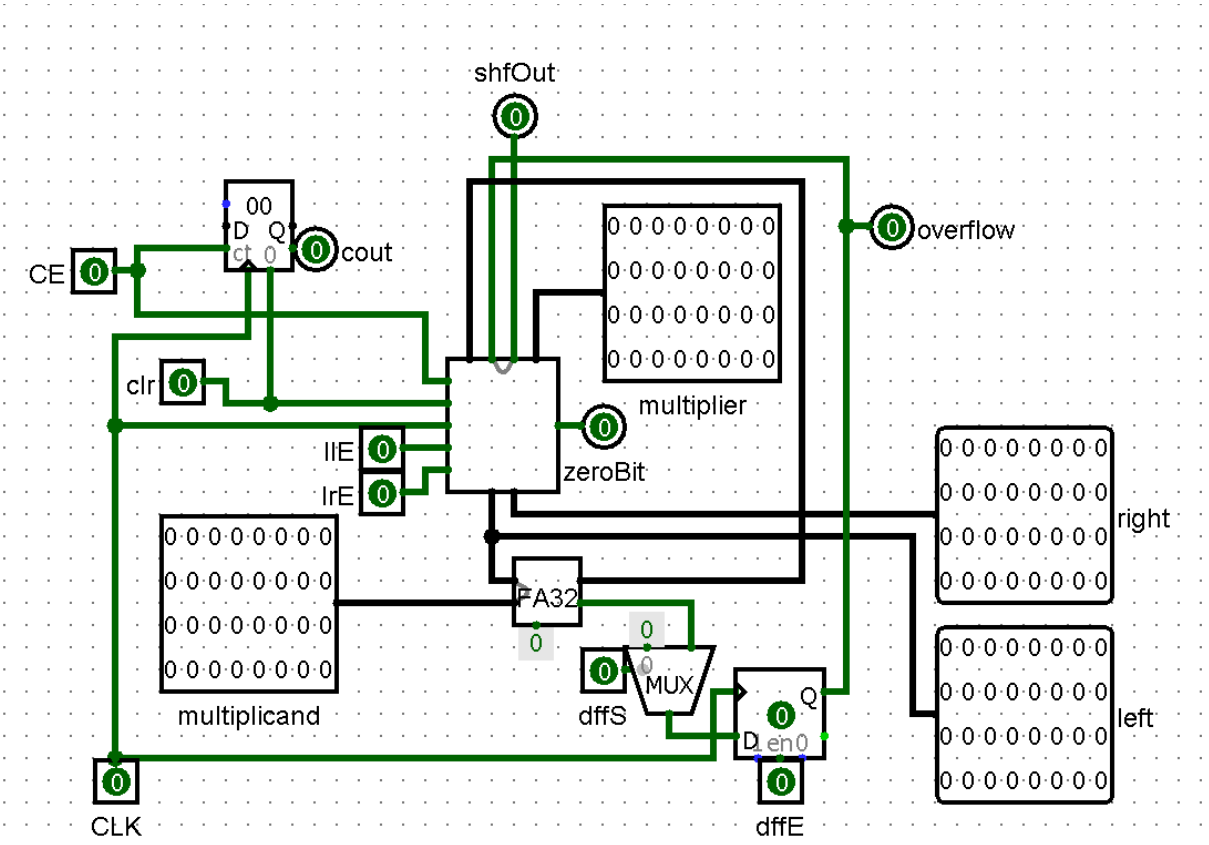



It is combined 64bit-register unit. There are several tasks that is achieved by this unit.

- Shifter, to shift the product
- Zero bit determination output
- 3 different muxes to manage the datapath decision.

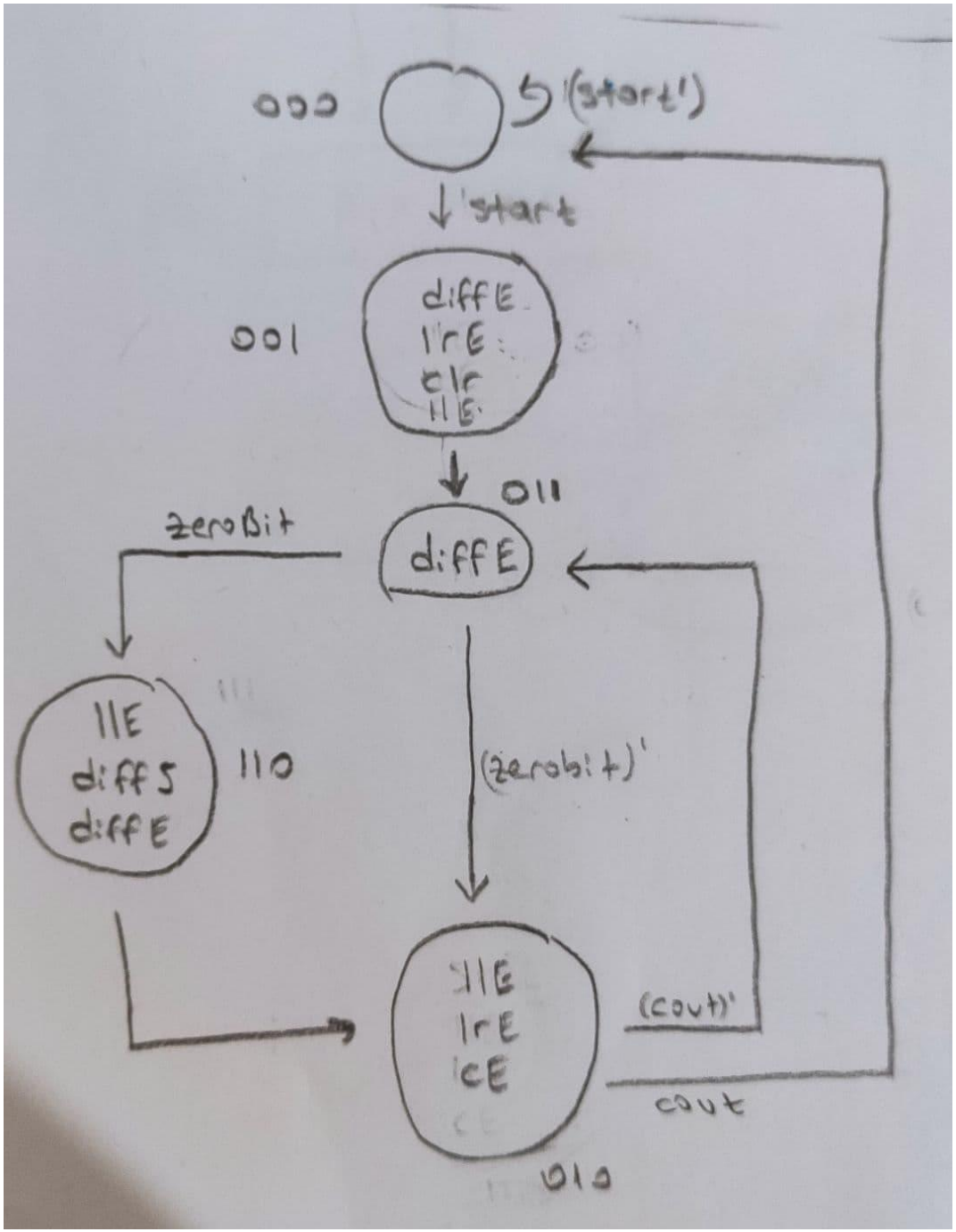
There are also some enable bits as an input to chose right decision for datapath calculation.

## *Entire Datapath Design:




I use counter for looping over the given FSM (it will be showed later part). The tricky part was that we do not need to use any comparator indeed. We can simply exploit **the carry out** bit of the counter to exit the loop. Another tricky one is using **clr** and **CE** for two different situations as a signal really reduce the complexity of the datapath. I also use **D-FlipFlop** for efficiency to keep overflow in safe place.

***FSM with datapath signals:**



## *Datapath table and some equations:

| Present State | diffE | llE | lrE | clr | cE | diffs |
|---|---|---|---|---|---|---|
| 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 0 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 1 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 1 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 1 0 | 0 | 1 | 1 | 0 | 1 | 0 |

$diffE = S2'S1'S0 + S2'S1S0 + S2S1S0'$

$= S2'S0 + S2S1S0'$

$llE = S1S0 + S2'S1'S0$

$lrE = S2'S1'S0 + S2'S1S0'$

$= S2'(S1 \oplus S0)$

$clr = S2'S1'S0$

$cE = S2'S1S0'$

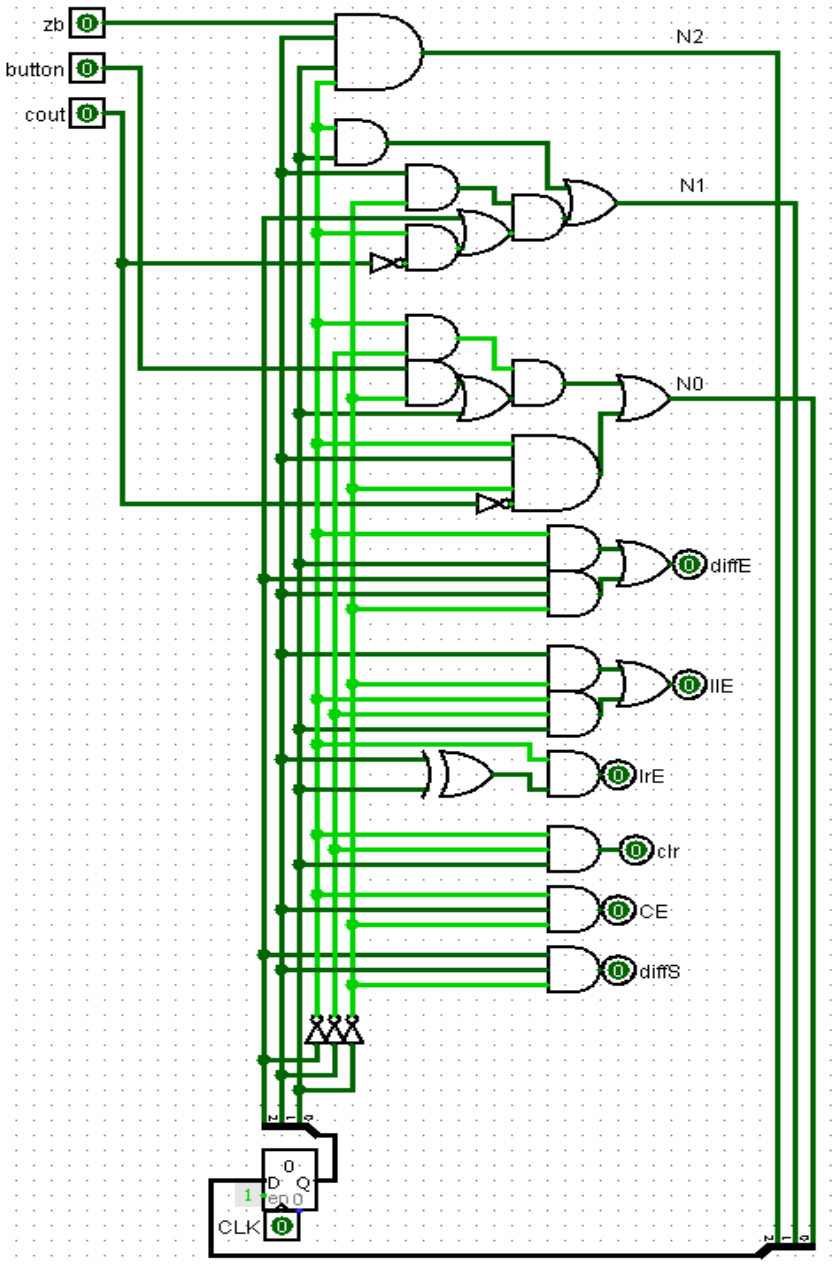
$diffs = S2S1S0'$

- Signals which are right side of the table are inputs for datapath for each step. I made some simplification on equation of each signal while calculating.

- Some signals in datapath does not exist here. Because these signals are occurring at the same states of the FSM. We do not need to include them as another column. We can use signals which are already indicated in the datapath table.

# *Control Unit Design:

In control unit, outputs of the datapath will be an inputs of our control unit. These are **zb**, **button**, **cout**. For determination of next state and input signals of datapath, those signals are needed. PC also compose of register which is 3 bits because we have 5 states.



## *Control-Unit table and some equations:

| Present State | | | Control Unit Inputs | | | Next State | | |
|---|---|---|---|---|---|---|---|---|
| S2 | S1 | S0 | zb | Cout | b | N2 | N1 | N0 |
| 0 | 0 | 0 | X | X | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | X | X | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | X | X | X | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | X | X | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | X | X | 1 | 1 | 0 |
| 1 | 1 | 0 | X | X | X | 0 | 1 | 0 |
| 0 | 1 | 0 | X | 0 | X | 0 | 1 | 1 |
| 0 | 1 | 0 | X | 1 | X | 0 | 0 | 0 |

## *Next-State determination equations:

$$N2 = S2'S1S0\,zeroBit$$

$$N1 = S2'S1'S0 + S2'S1S0(zb)' + S2'S1S0(zb) + S2S1S0' + S2'S1S0'(Cout)'$$

$$= S2'S1S0 + S2'S1'S0 + S1S0(S2 + S2'(Cout)')$$

$$= S2'S0 + S1S0'(S2 + S2'(Cout)')$$

$$N0 = S2'S1'S0'b + S2'S1'S0 + S2'S1S0'(Cout)'$$

$$= S2'S1'(S0'b + S0) + S2'S1S0'(Cout)'$$

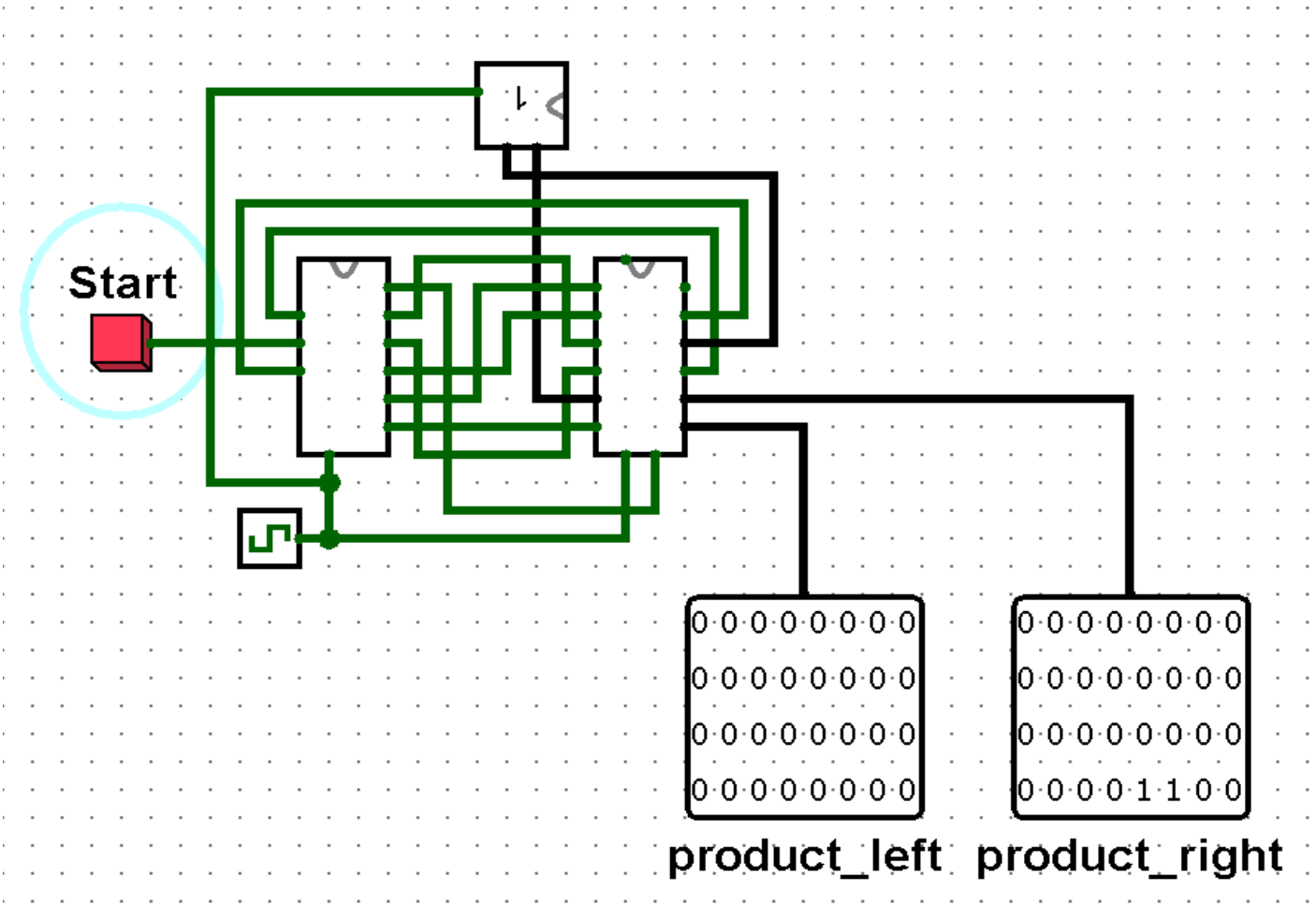
## *Test Cases:

### Test-1:

Multiplicand: 2

Multiplier: 6

Result: 12

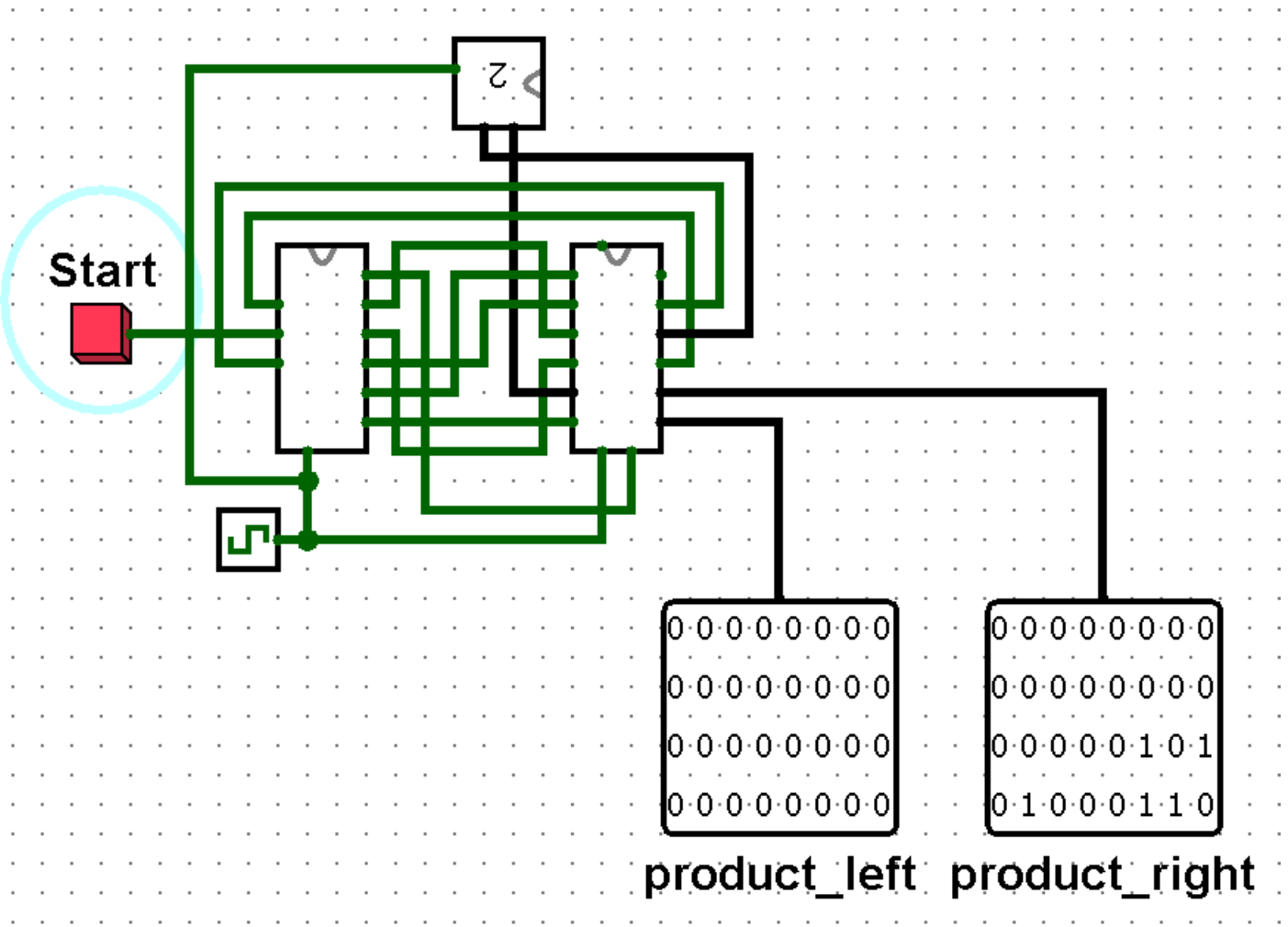


**Test-2:**

Multiplicand: 54

Multiplier: 25

Result: 1350

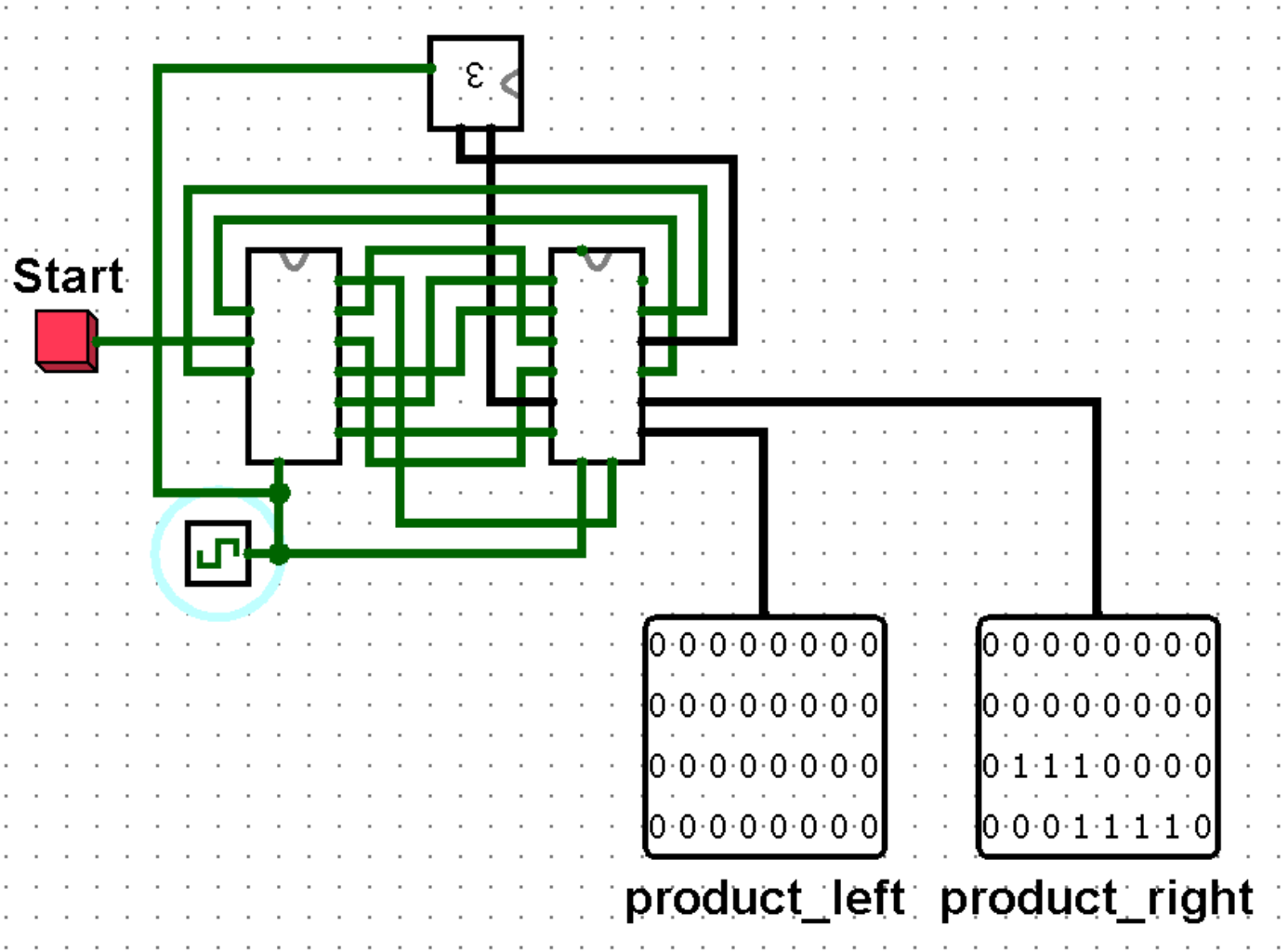


**Test-3:**

Multiplicand: 254

Multiplier: 113

Result: 28.702

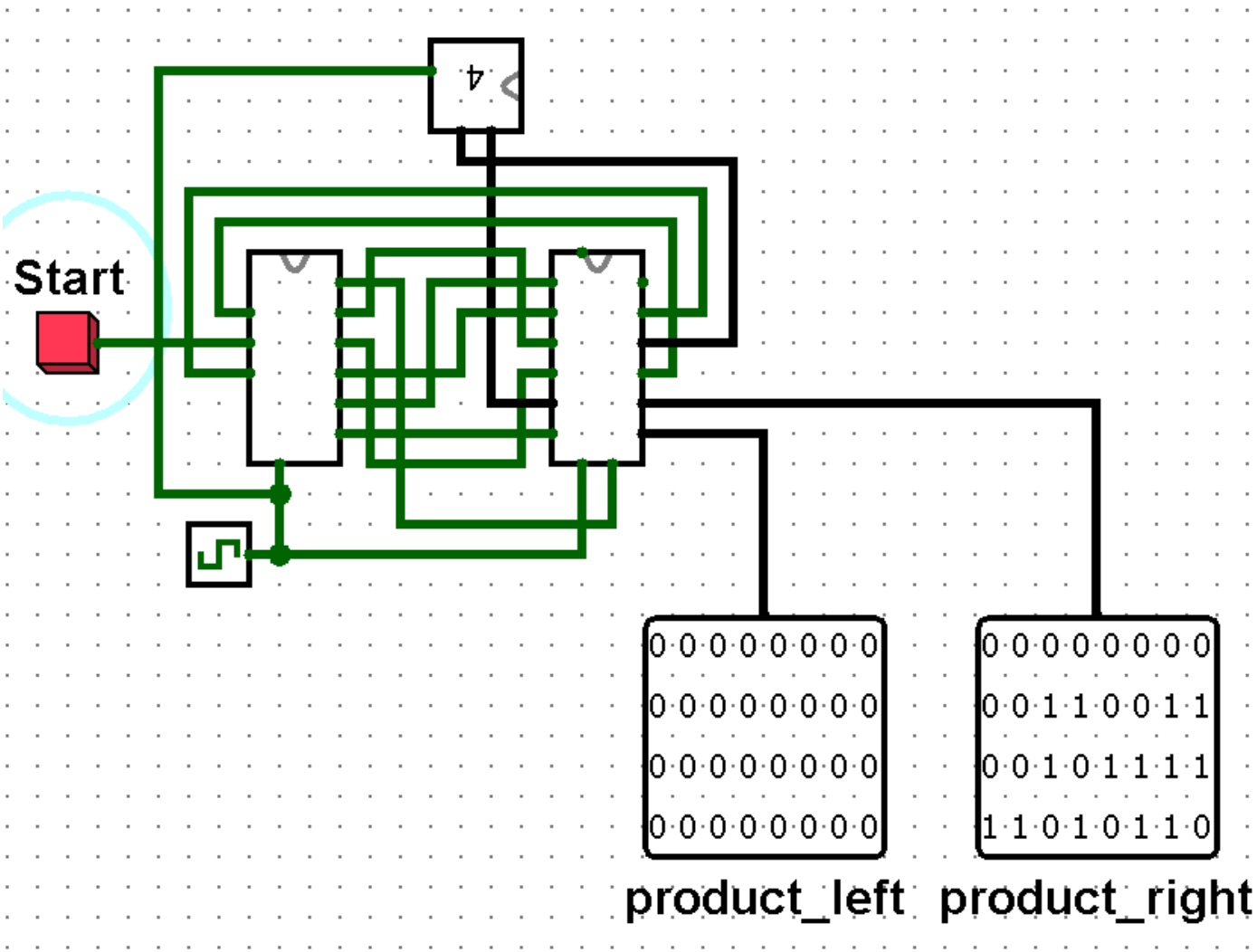


**Test-4:**

Multiplicand: 1113

Multiplier: 3014

Result: 3.354.582

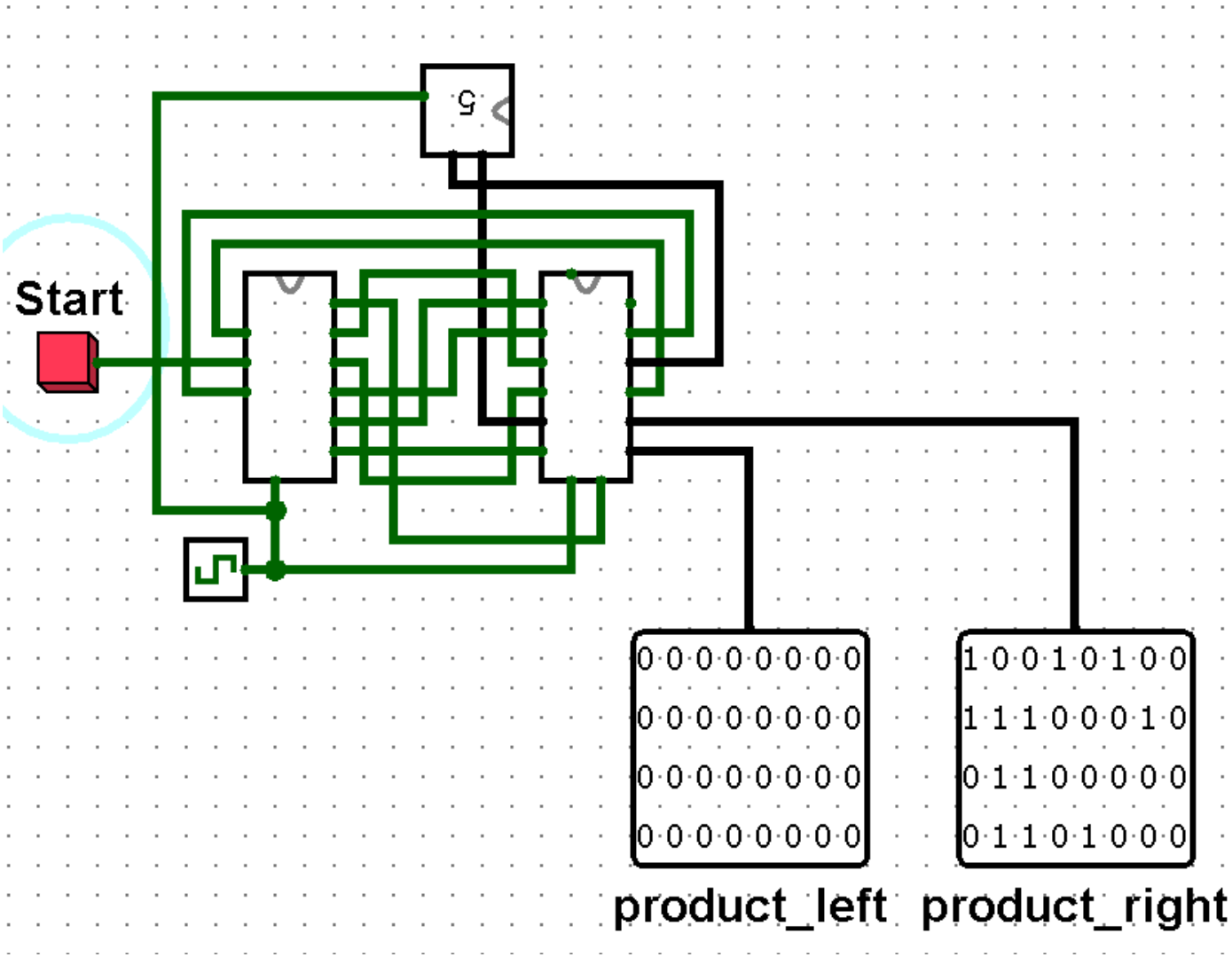


**Test-5:**

Multiplicand: 44356

Multiplier: 56314

Result: 2.497.863.784

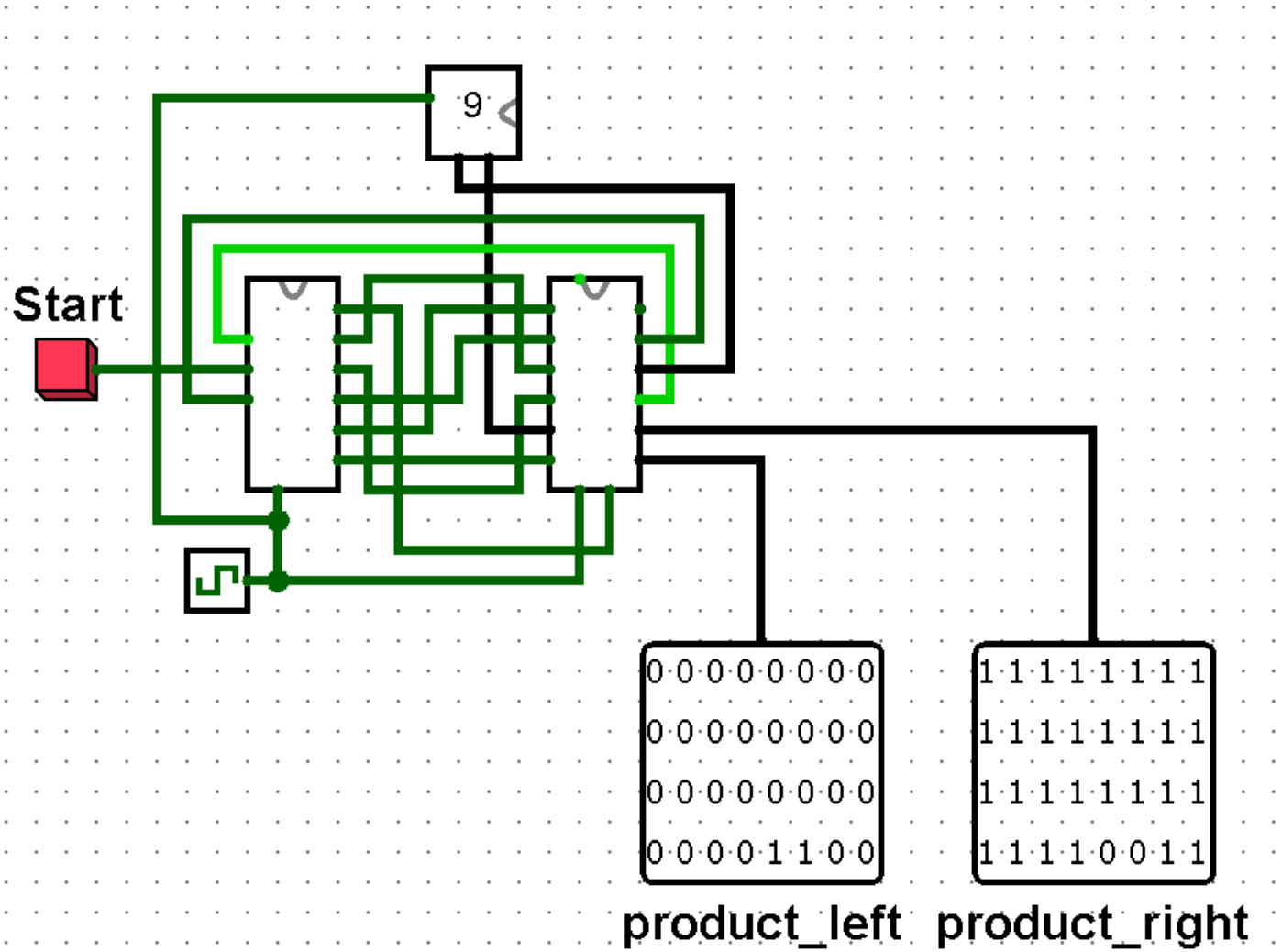


**Test-6:**

Multiplicand: 4294967295

Multiplier: 13

Result: 55.834.574.835



# *Notes and Result:

- Every single given task should work fine.
- I remarked extra optimization and bonuses at the beginning of the file.

**End of the homework report.**