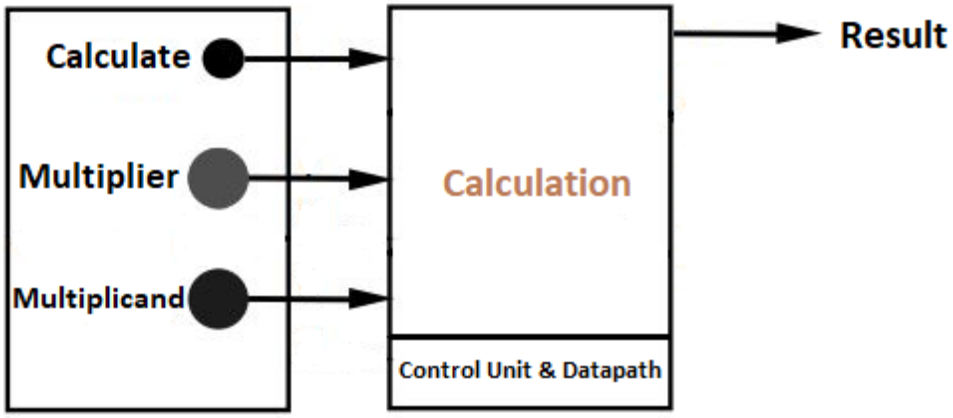
GTU Department of Computer Engineering CSE 331/503 - Fall 2020 Homework 3 Report

Akif KARTAL
171044098

Problem Definition

The problem is to compute the multiplication of two 32-bit numbers using shifter and adder.
A basic diagram to show the problem is;



Inputs: Calculate (Button), Multiplier and Multiplicand (32-bit unsinged numbers)

Outputs: 64-bit unsinged number

Solution Steps

Since we are solving this problem with FSM(Finite State Machine) we need to apply following steps;

*These steps are taken from last year CSE 232(logic) lectures.

1. Decide states and draw the state diagram for your FSM controller.
2. Draw datapath.
3. Draw truth table.
4. Derive Boolean expressions from the truth table.
5. Draw the circuit on Logisim.
6. Simulate and see whether it works. If it does not turn back to previous stages and check each carefully.

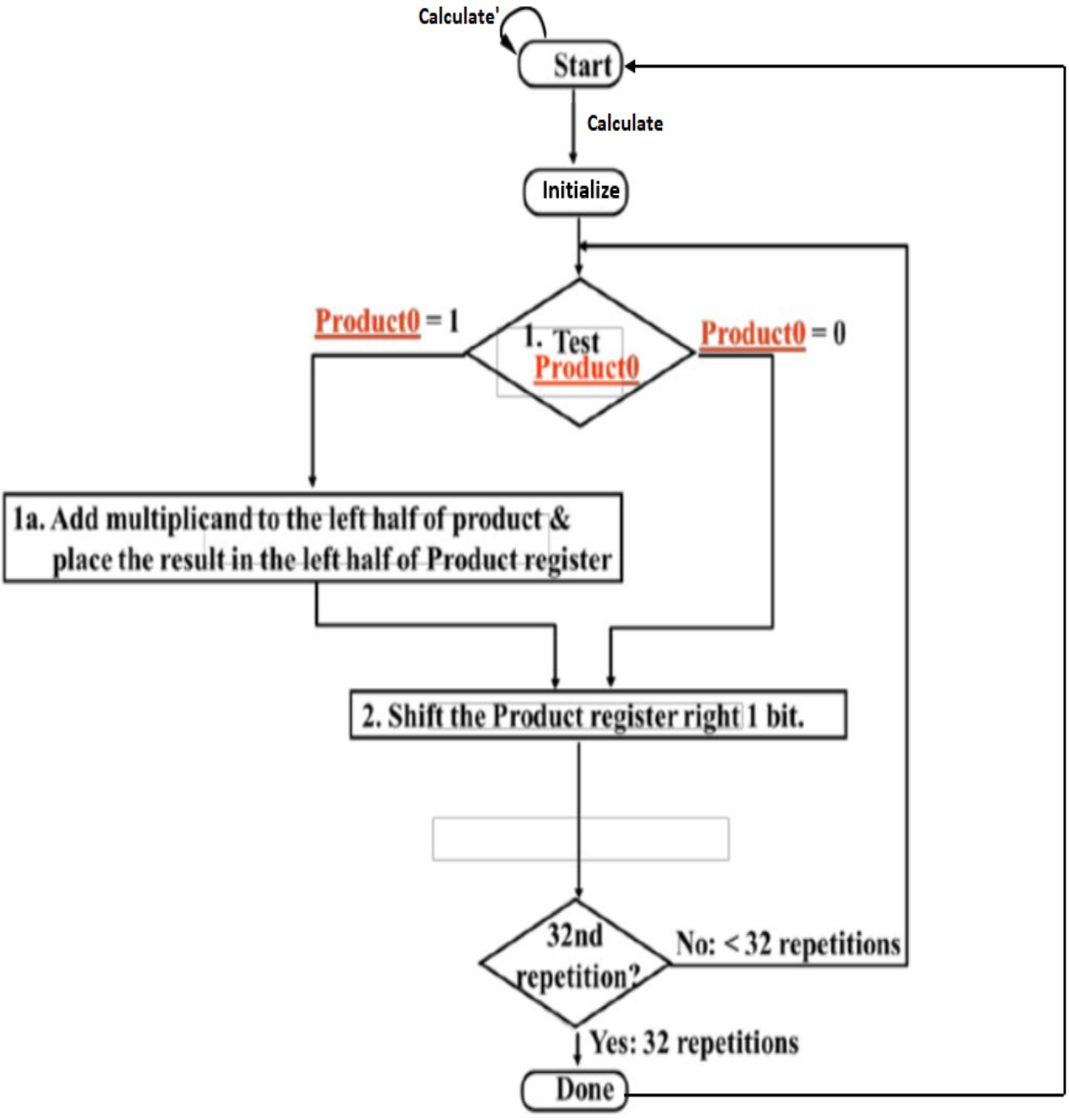
I will follow these steps one by one.

My Solution Step by Step

1) Decide states and draw the state diagram for your FSM controller.

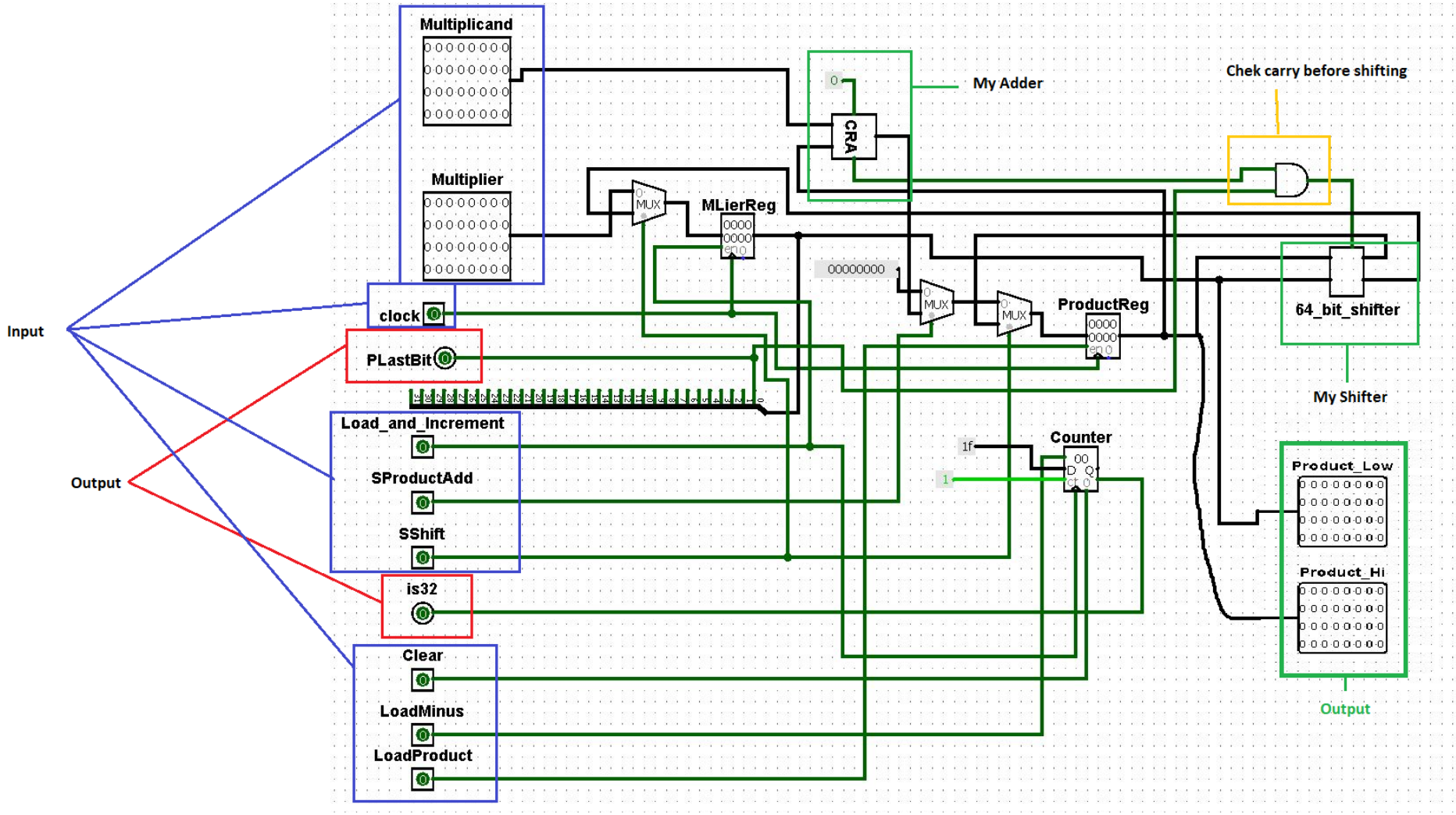
State diagram is already given in homework file, but I did some small changes on it.

Input: Calculate



Step 1: Capture the FSM

2) Draw datapath.



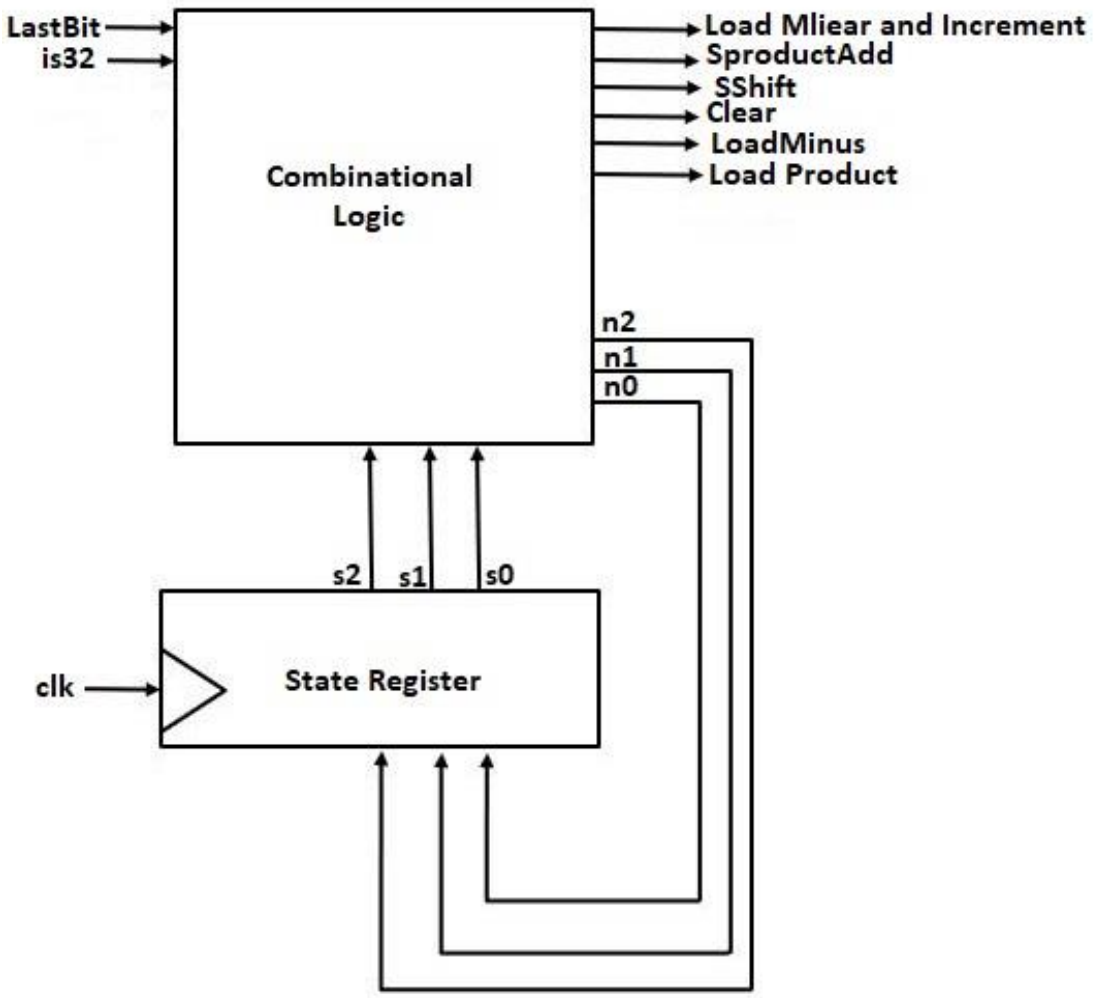
Optimization on Datapath

I changed my datapath desing so it look like ugly you can check it from datapath.circ file.

Note that in Logisim we don't have 64 bit thus, product and multiplier stored in two different register. Also result can be shown 64 bit as 32-32(high,low).

Also, I didn't use a comparator in datapath instead of I used counter overflow bit. Lastly carry can only be 1 in shifting when last bit is 1 and carry is 1.

Control Unit Architecture(FSM)



Inputs	Description of the Inputs
Last Bit	0th bit of product register
is32	Repetation input , if counter is 32 then, this input is 1
clk	Clock Input
s2 s1 s0	Present State

Outputs	Description of the Outputs
LMandInc	Register Multipler Load and increment input in Datapath
Sproadd	Select input of Add in datapath
SShift	Select input of Shift in datapath
Clear	Clear input of Counter in datapath
LoadMinus	Load -1 in Counter if needed in datapath
LProduc	Register Product Load input in Datapath
n2 n1 n0	Next State

Step 2A: Set up architecture

Step 2B: Encode states

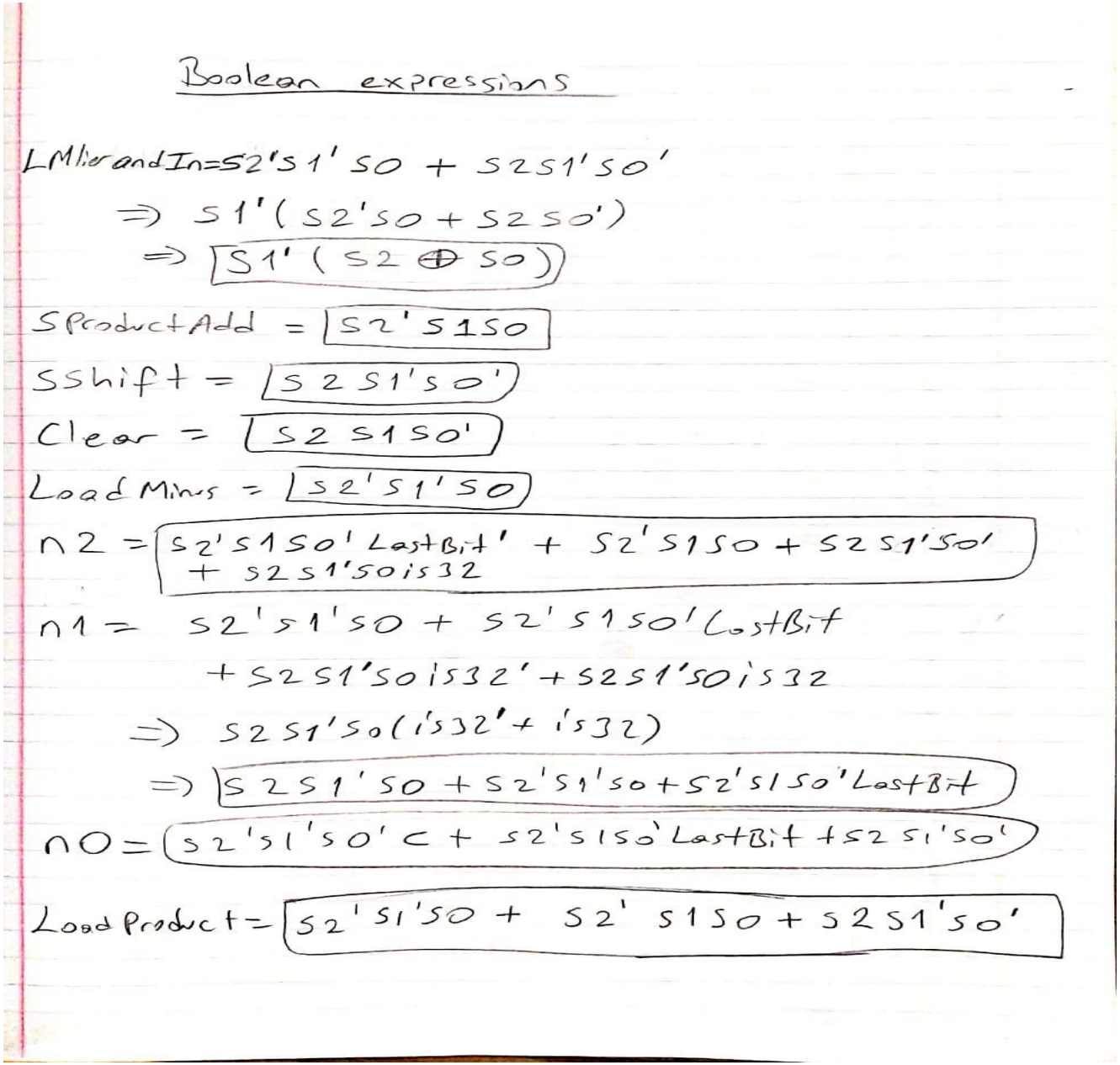
Encoding ➔ Start=000, Initialize=001, Test=010, Addition=011,

Shift = 100, Check32= 101, Done= 110

3) Draw truth table.

Inputs						Outputs									
S2	S1	S0	C	LastBit	is32	LMlierandIInc	SproductAdd	Sshift	Clear	LoadMinus	Load Product	n2	n1	n0	**Operation**
0	0	0	0	x	x	0	0	0	0	0	0	0	0	0	Start
0	0	0	1	x	x	0	0	0	0	0	0	0	0	1	Start
0	0	1	x	x	x	1	0	0	0	1	1	0	1	0	Initialize
0	1	0	x	0	x	0	0	0	0	0	0	1	0	0	Test
0	1	0	x	1	x	0	0	0	0	0	0	0	1	1	Test
0	1	1	x	x	x	0	1	0	0	0	1	1	0	0	add
1	0	0	x	x	x	1	0	1	0	0	1	1	0	1	shift
1	0	1	x	x	0	0	0	0	0	0	0	0	1	0	Check Rep.
1	0	1	x	x	1	0	0	0	0	0	0	1	1	0	Check Rep.
1	1	0	x	x	x	0	0	0	1	0	0	0	0	0	Done

4) Derive Boolean expressions from the truth table.



Boolean expressions

LMlierandIn = S2'S1'S0 + S2S1'S0'

⇒ S1'(S2'S0 + S2S0')

⇒ S1'(S2 ⊕ S0)

SProductAdd = S2'S1S0

Sshift = S2S1'S0'

Clear = S2S1S0'

LoadMinus = S2'S1'S0

n2 = S2'S1S0'LastBit' + S2'S1S0 + S2S1'S0' + S2S1'S0is32

n1 = S2'S1'S0 + S2'S1S0'LastBit + S2S1'S0is32' + S2S1'S0is32

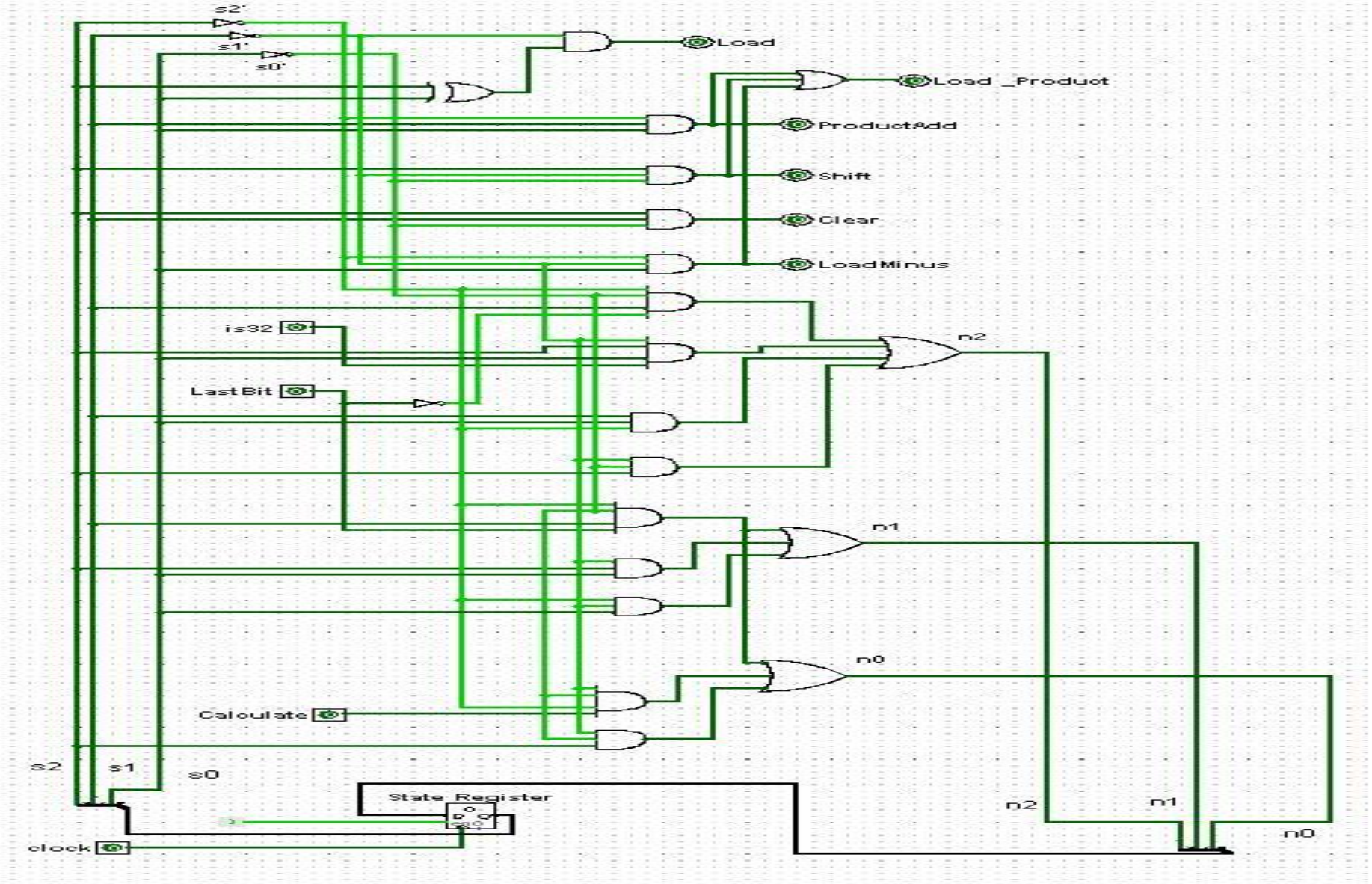
⇒ S2S1'S0(is32' + is32)

⇒ S2S1'S0 + S2'S1'S0 + S2'S1S0'LastBit

n0 = S2'S1'S0'C + S2'S1S0'LastBit + S2S1'S0'

LoadProduct = S2'S1'S0 + S2'S1S0 + S2S1'S0'

Control Unit as in Architecture from Boolean expressions



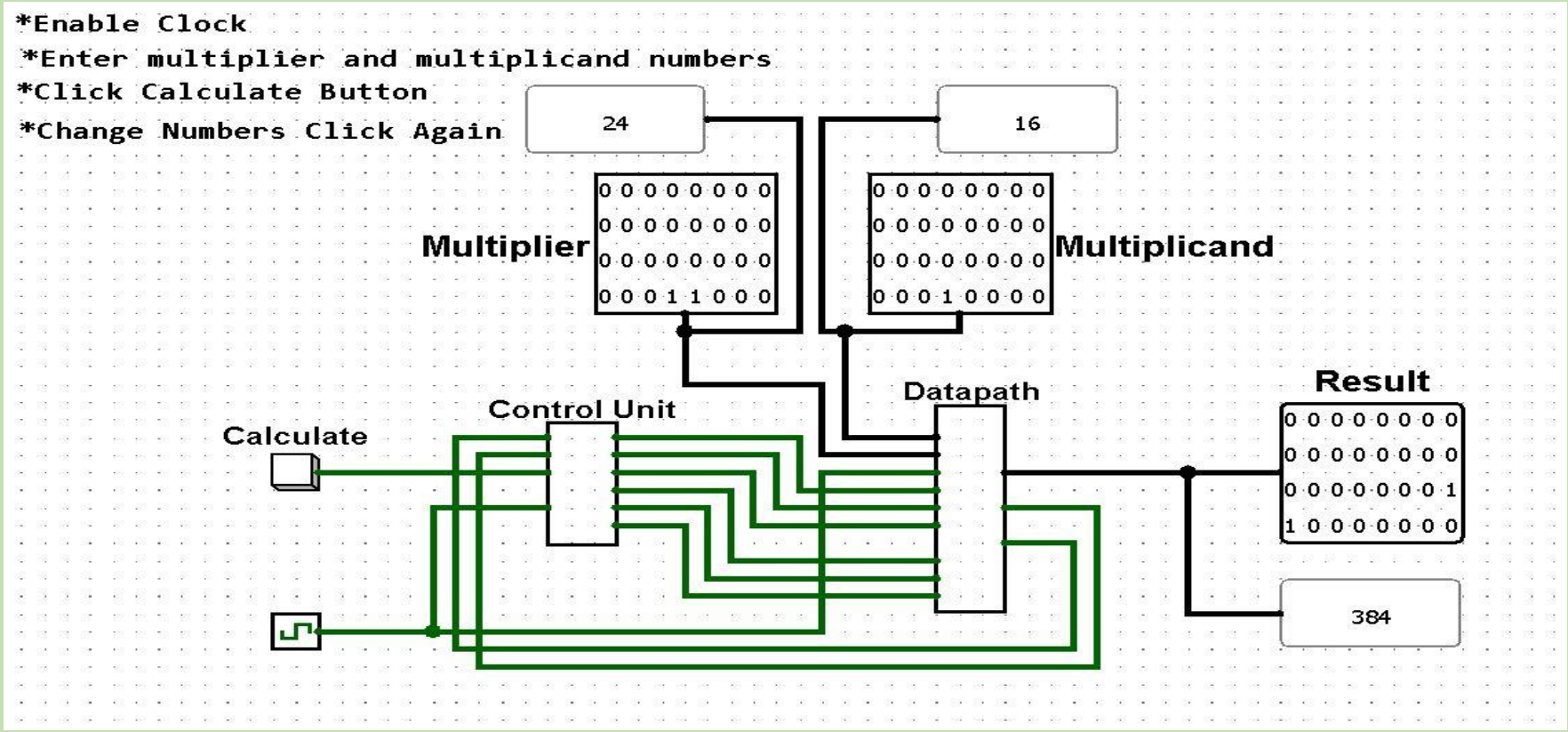
Control unit has only signals according to current state therefore, All other components must be in datapath.

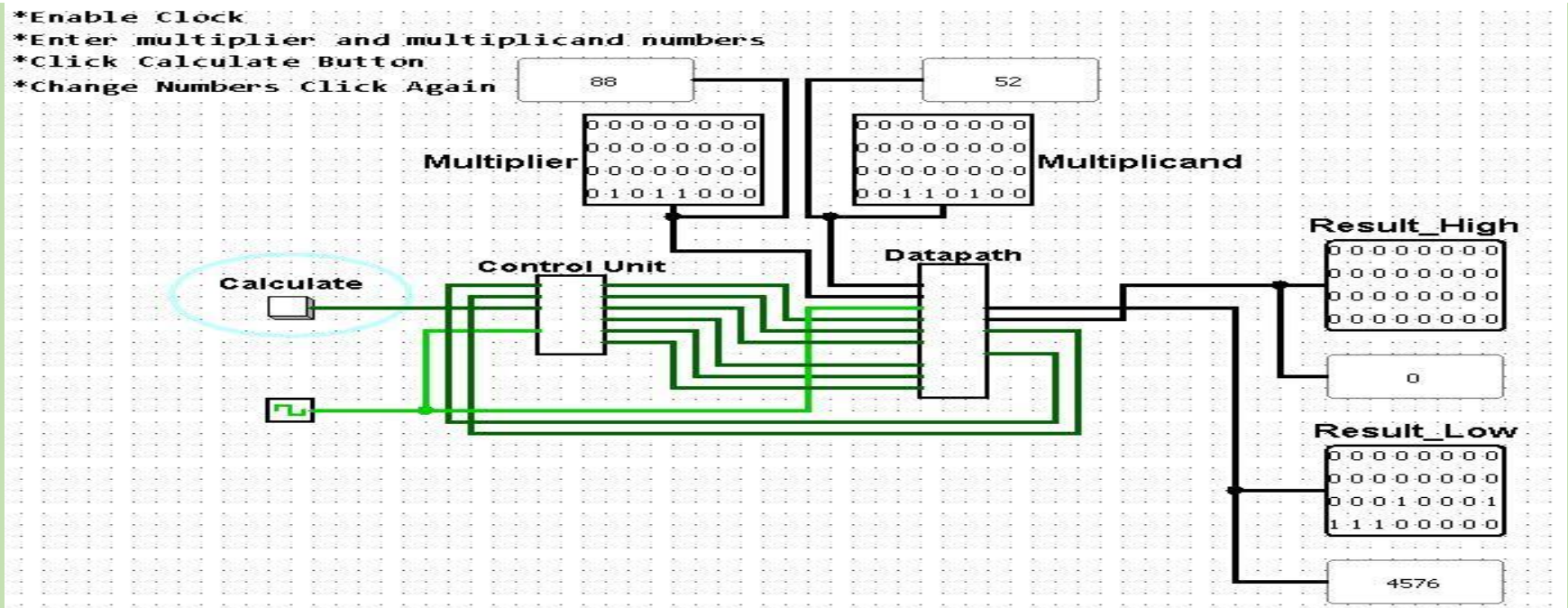
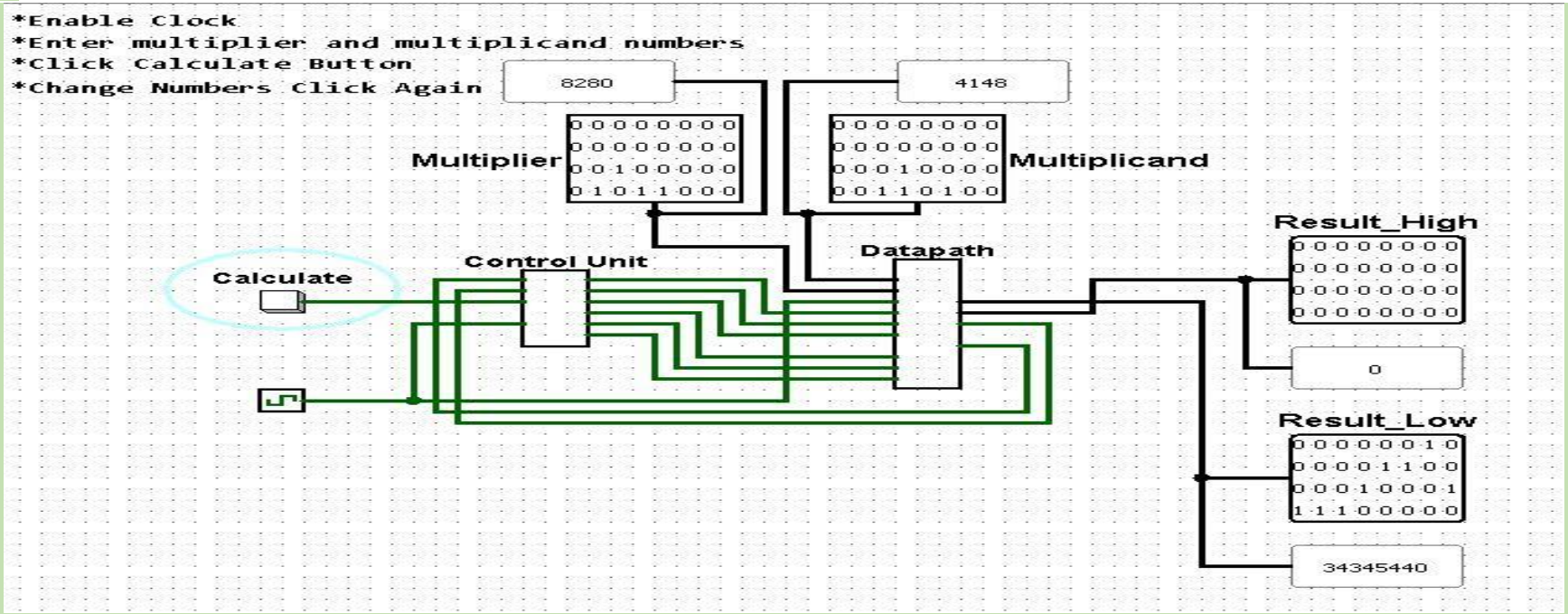
5) Draw the circuit on Logisim.

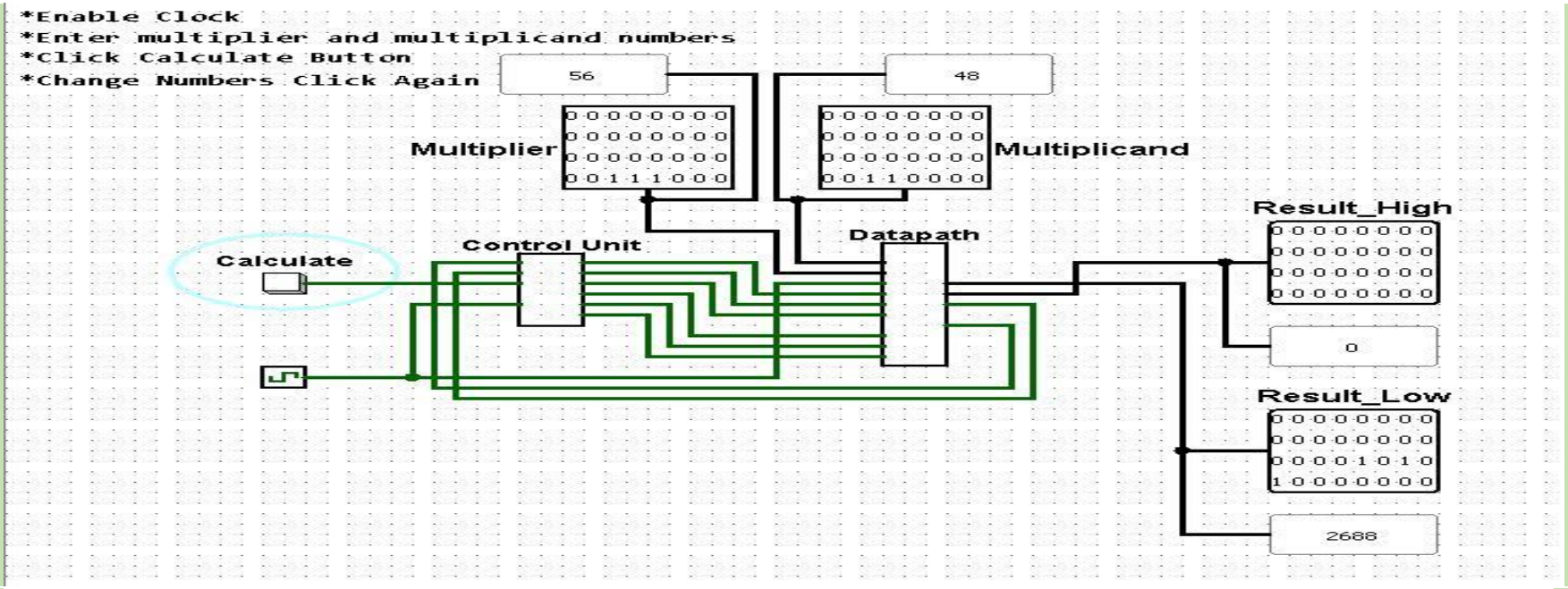
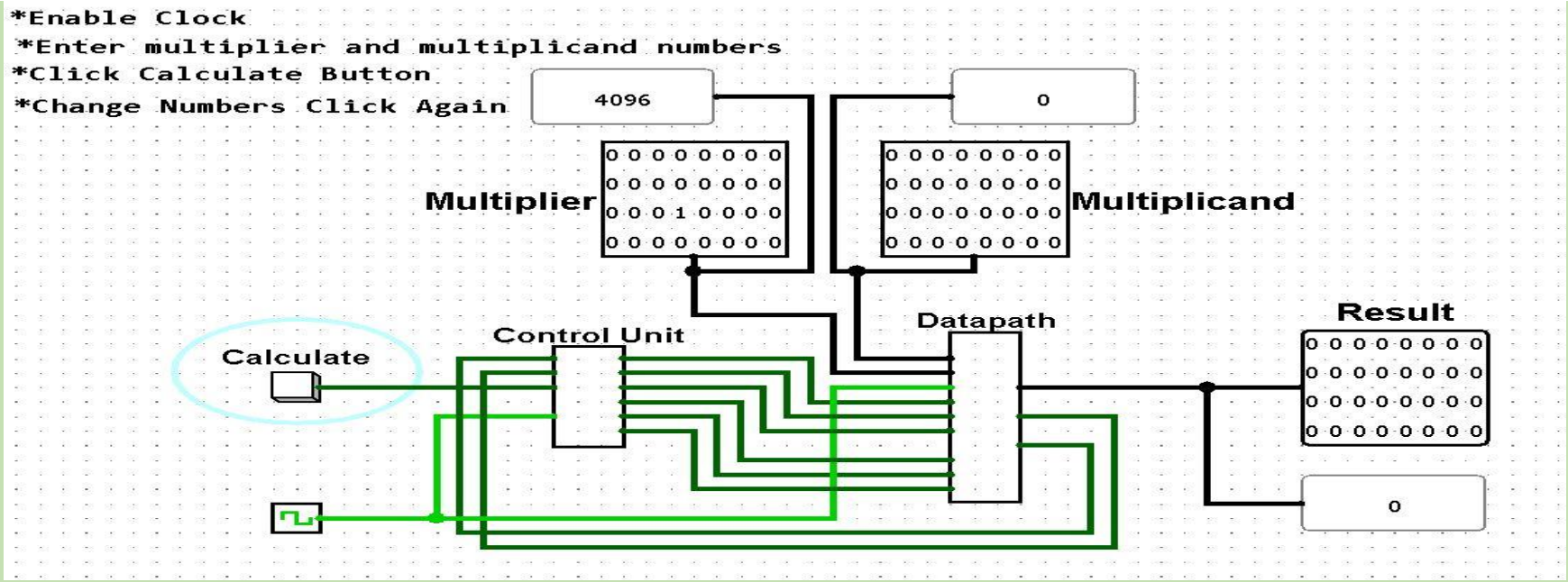
- Check **main** circuit in **mult32.circ** file to see circuit and for other parts **control and datapath.circ** files.

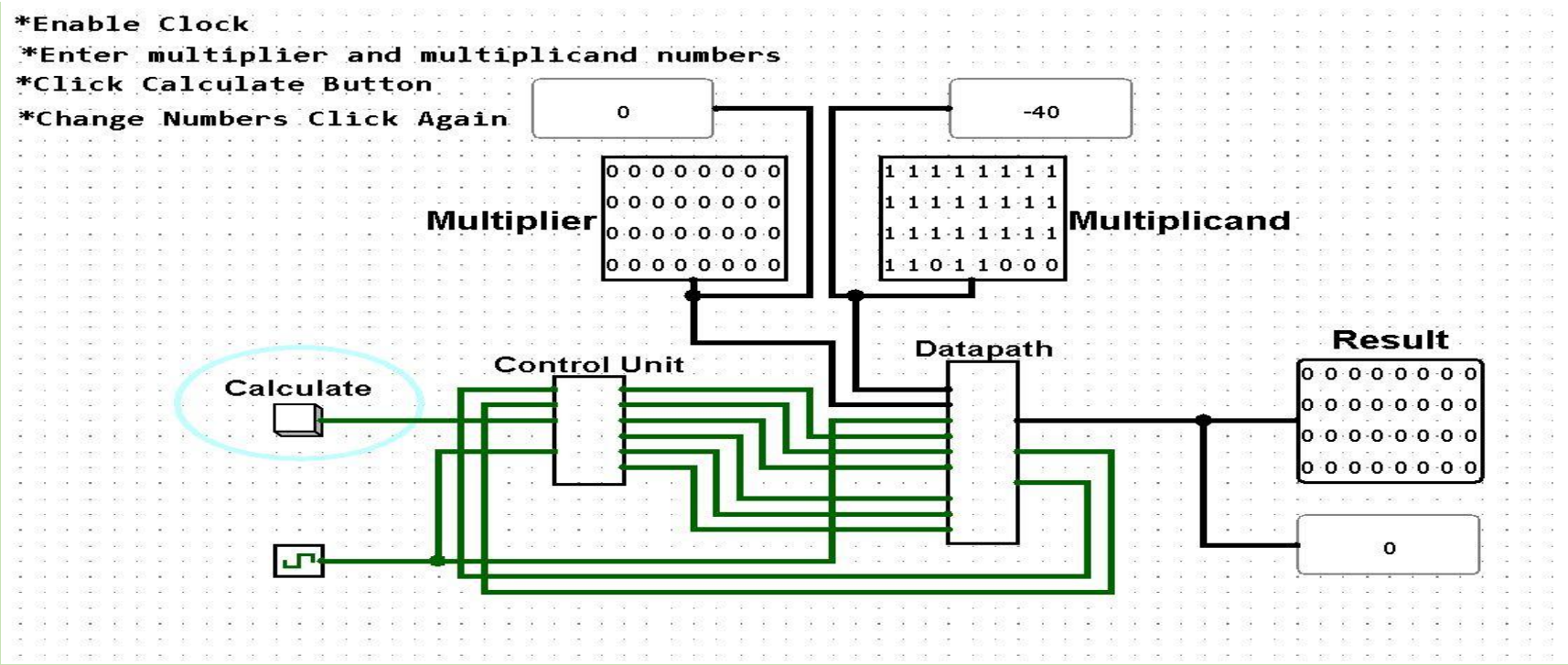
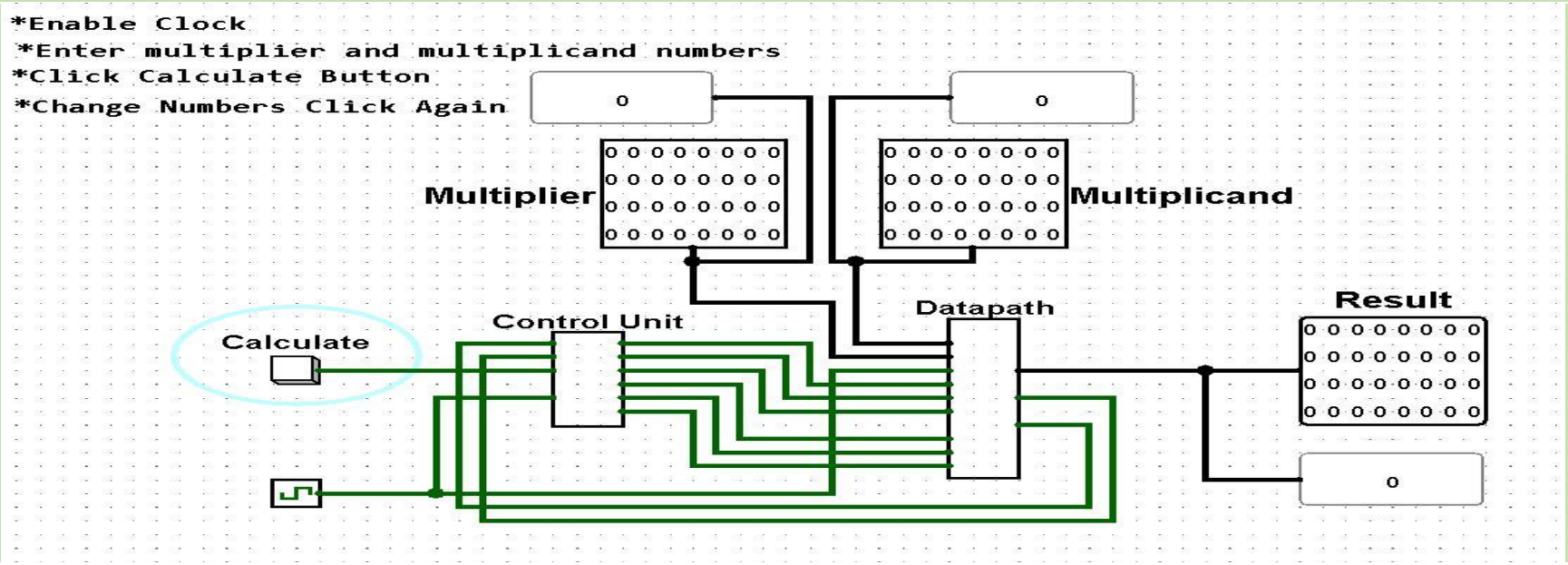
6) Simulate and see whether it works.

- Check following test results.
- First, I will show you 32 bit results last test will be 64 bit result. (I did test for signed number you can ignore them if you want.)
- I changed design for 64 bit results. Therefore, some picture is different but last design contain 64 bit result register.

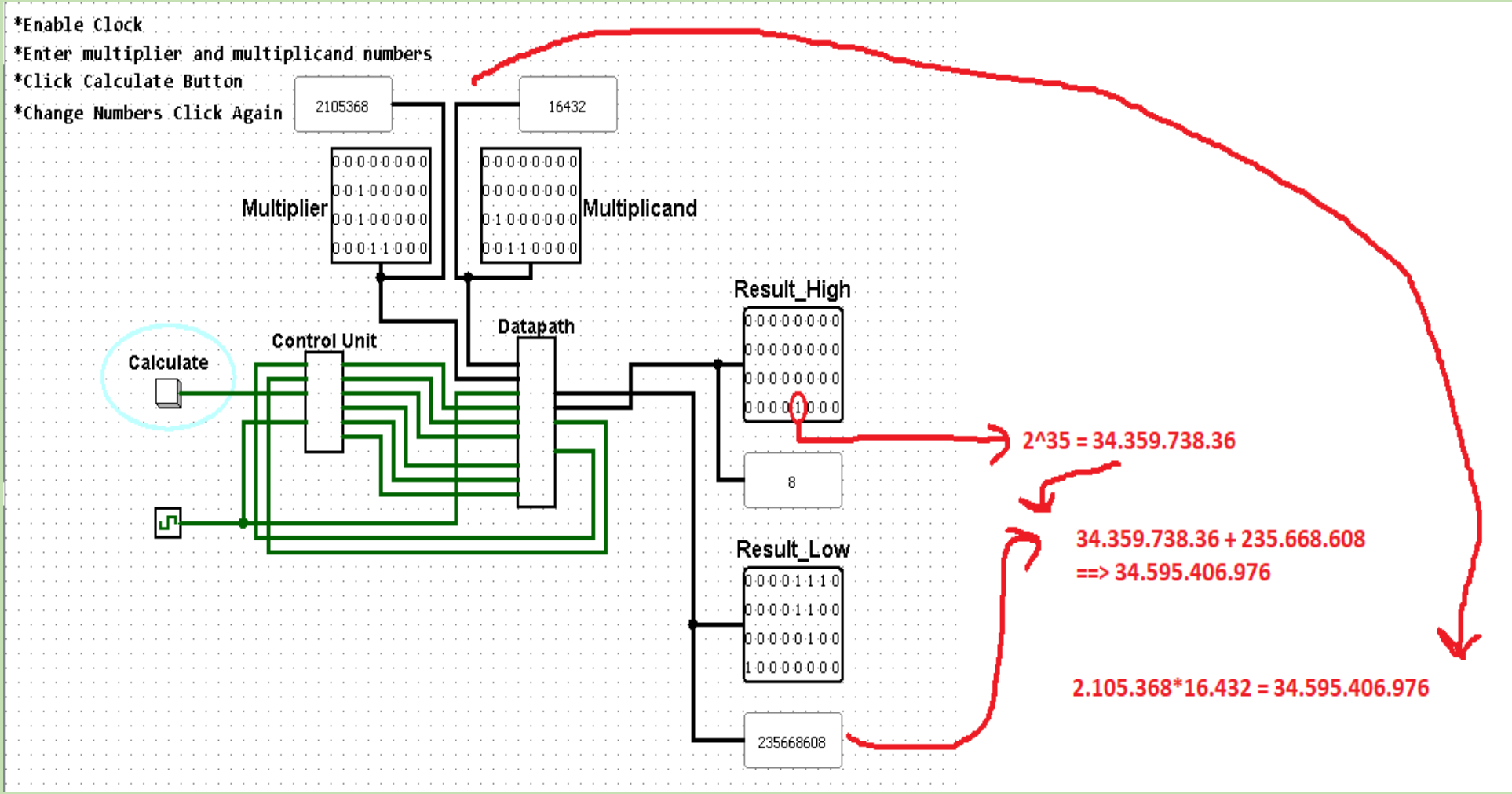
Description	Test Result
Two Positive Number	

Two Positive Number	
Two Positive Number	

Two Positive Number	
One zero number	

One zero number(vice versa)	
Two Zero Number	

➢ 64 bit test result.

Description	Test Result
64 bit result test	

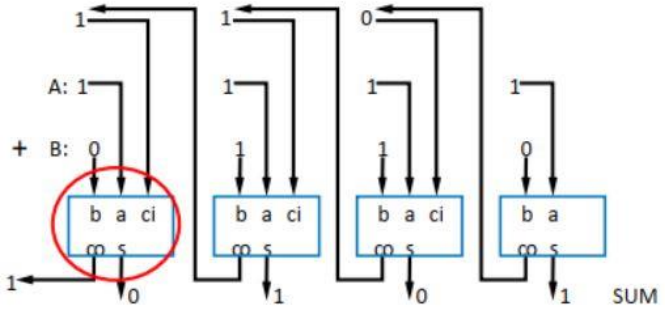
My helper Components(Bonus)

- **Adder**
  - ✓ **Full Adder(1 bit)**

    Full adder design is taken from last year CSE 232 lecture slides(Same design with this year). It could be from this year:)

Full-Adder

- ***Full-adder:*** Adds 3 bits, generates sum and carry
- Design using combinational design process from Ch 2



Step 1: Capture the function

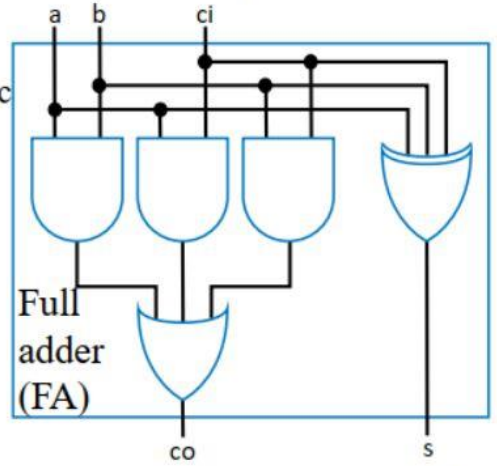
Inputs			Outputs	
a	b	ci	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2A: Create equations

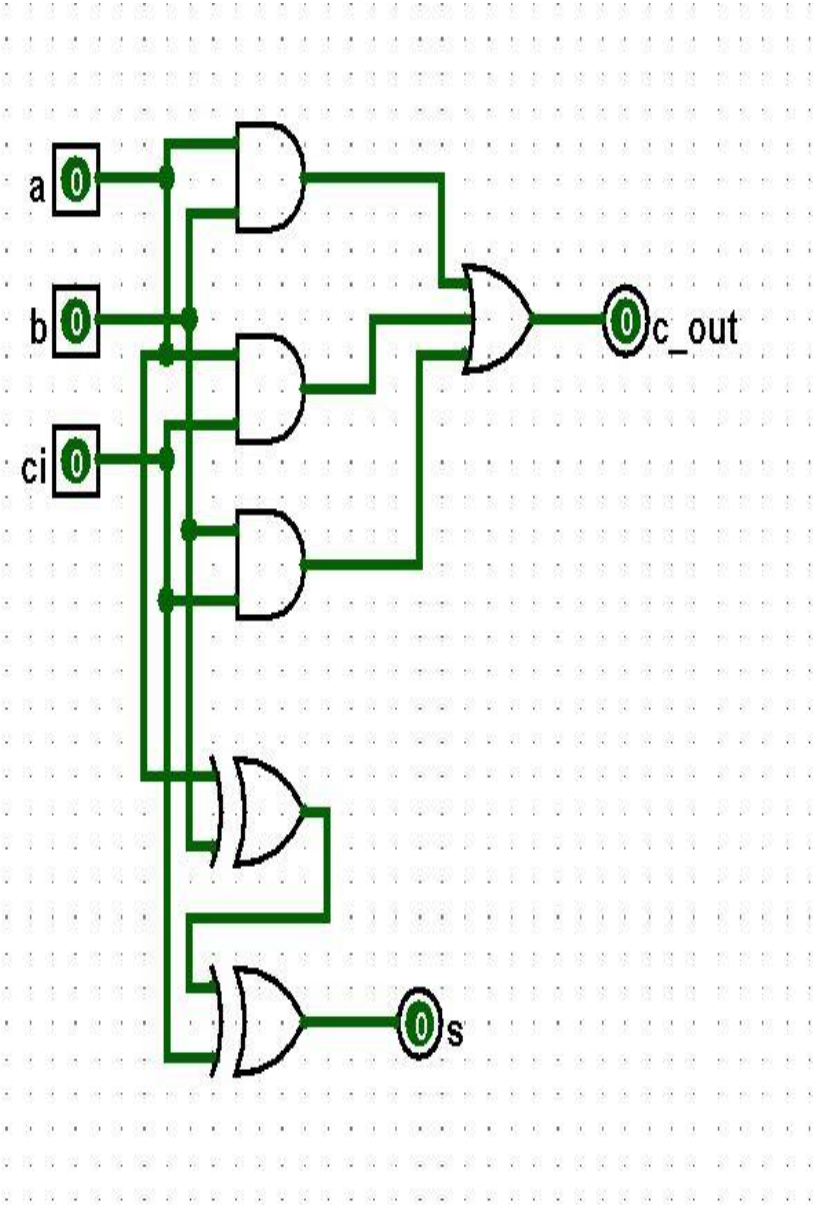
co = a'bc + ab'c + abc' + abc
co = a'bc +abc +ab'c +abc +abc' +abc
co = (a'+a)bc + (b'+b)ac + (c'+c)ab
co = bc + ac + ab

s = a'b'c + a'bc' + ab'c' + abc
s = a'(b'c + bc') + a(b'c' + bc)
s = a'(b xor c) + a(b xor c)'
s = a xor b xor c

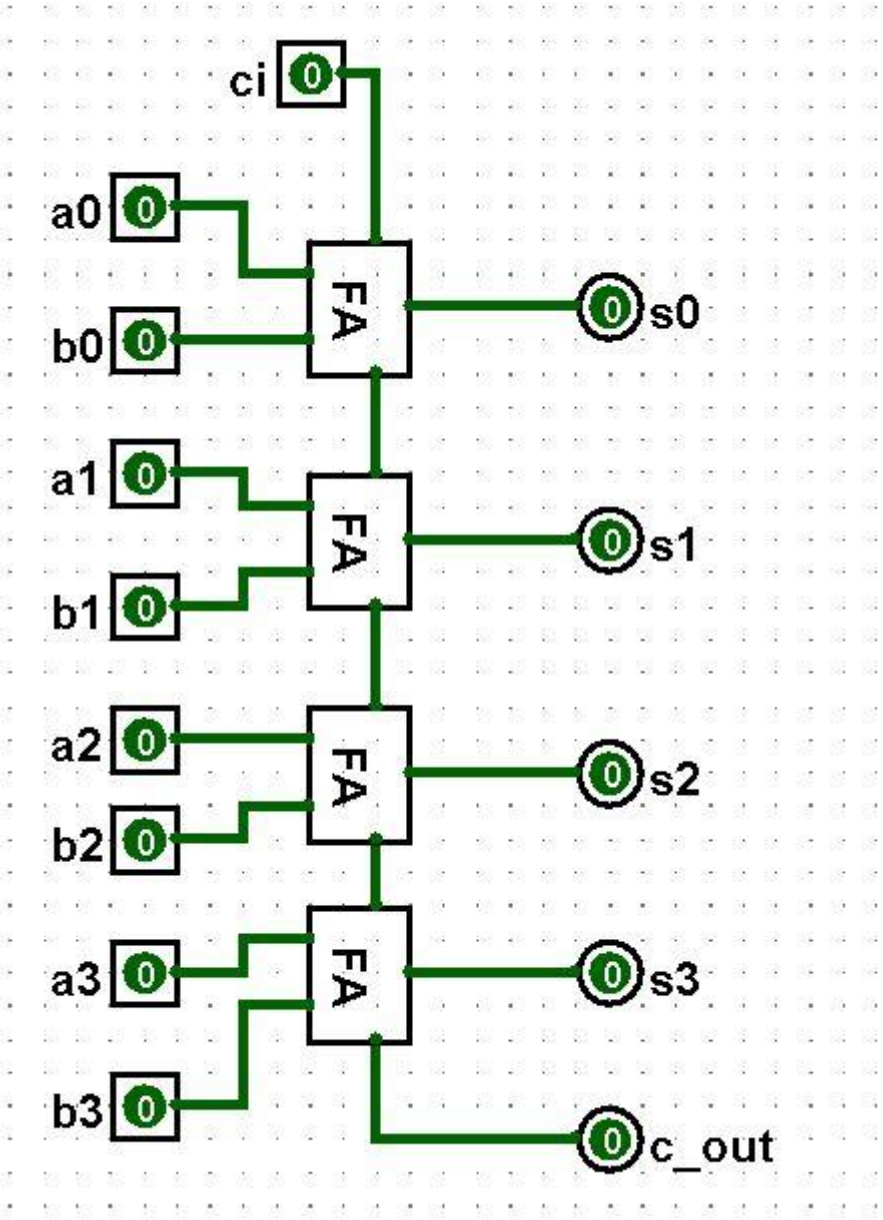
Step 2B: Implement as circuit



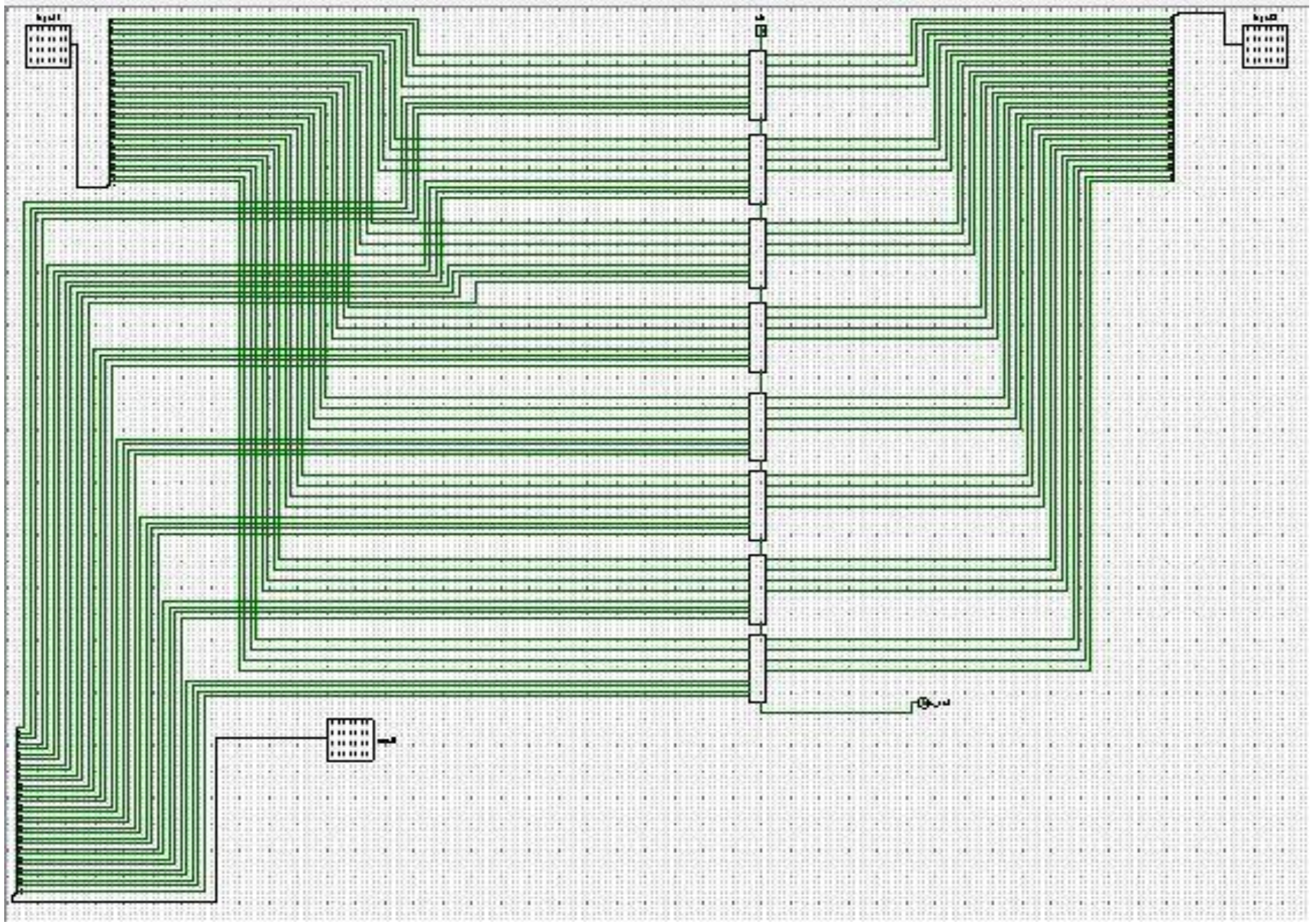
1 bit Full adder



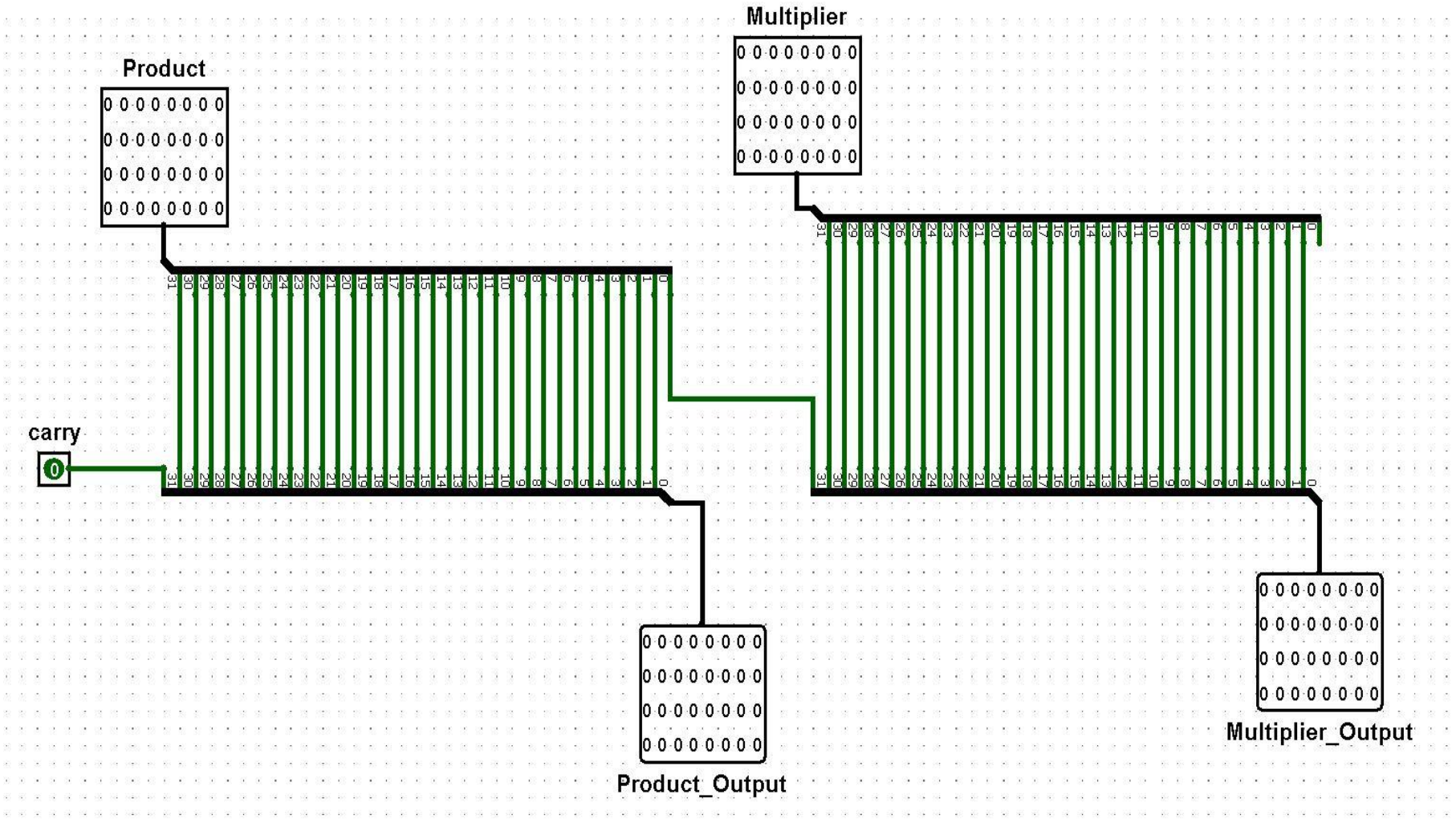
4 bit carry ripple adder



32 bit Carry ripple adder from 4 bit CRA (2 input(32 bit) and 1 output(32 bit), with cin and cout)



- **1 bit right Shifter**



Note that since Logisim has maximum 32 bit we need to make separate shift as in picture.
Design belongs to me. It is very simple.
Product means high part of product, Multiplier means low part of product(64 bit register).
Note: Both adder and shifter have test files in datapath.circ file. You can test them.