Design of Digital Circuits Arithmetic Logic Unit - ALU

Yew Heng Ngoh
Vulcu Horia - Rares
Michiliia Gureva
Sandeep Kumar
Stephen Nnamani

1. Adders

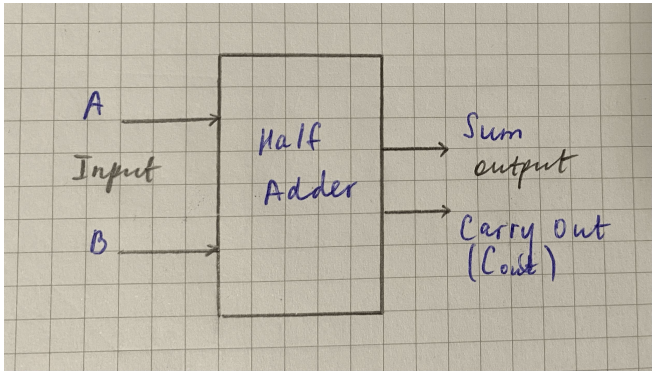
1.1) Half adder

1.2) Half adder to implement full adder

1.3) Full adder to implement Carry-Rippler adder

1.1. Implement a half adder:

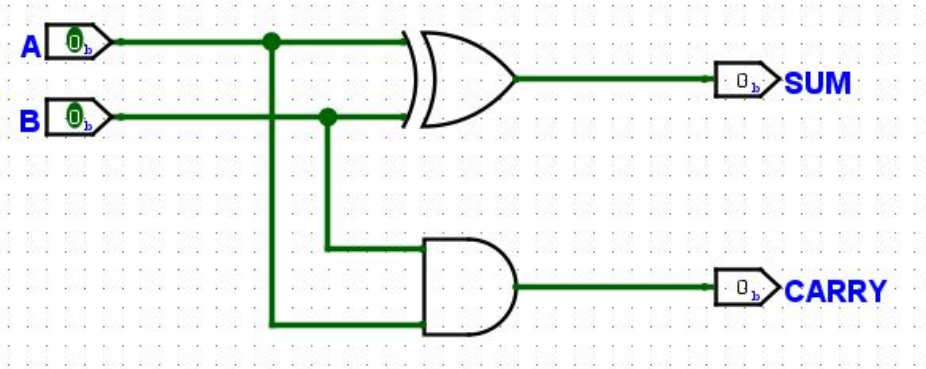
a) Design the circuit as a block diagram:



This is used for the purpose of adding two single bit numbers.

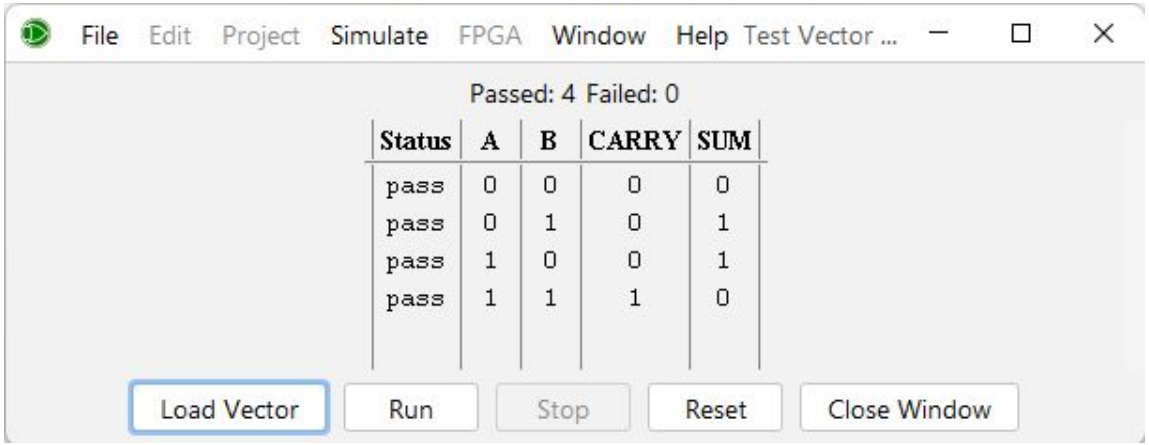
It contains 2 inputs (A and B) and 2 outputs (sum and carry).

b) Implement the circuit in Logisim:



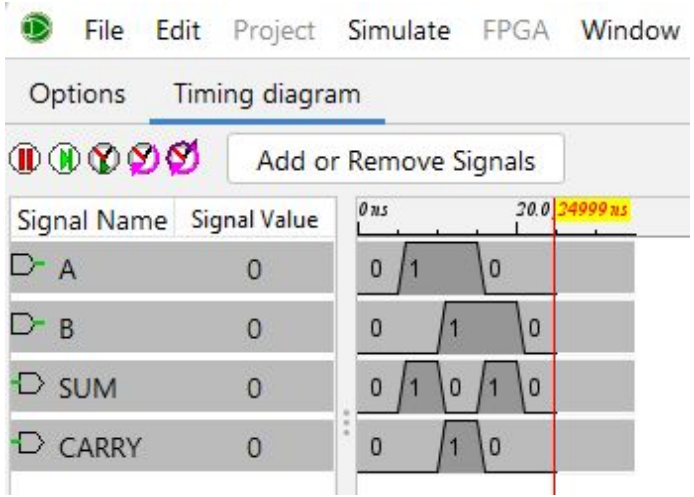
Implemented using 1 XOR gate and 1 AND gate

c) Test the circuit by using Test-Vector:

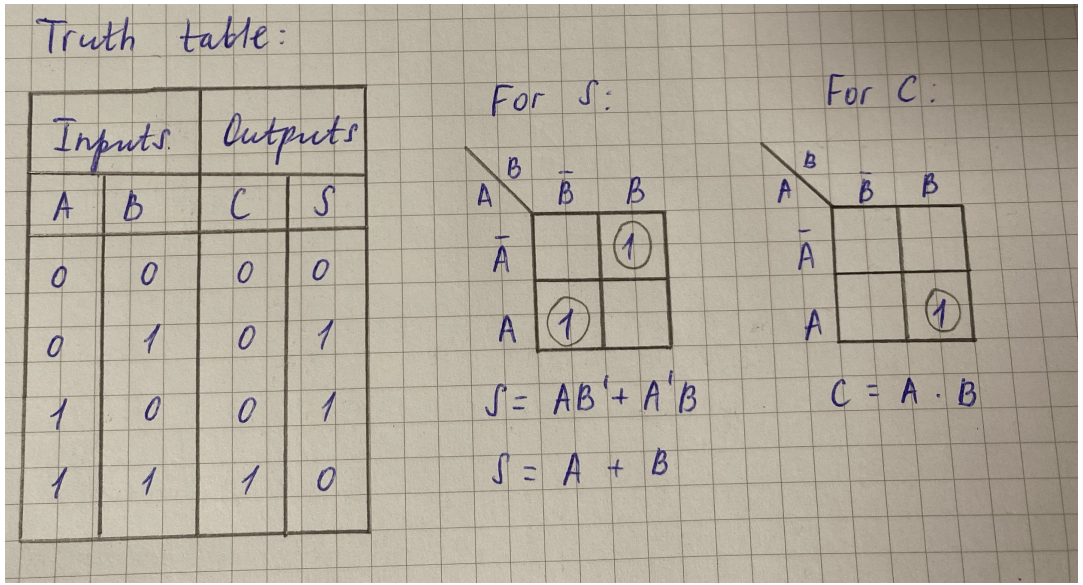


Status	A	B	CARRY	SUM
pass	0	0	0	0
pass	0	1	0	1
pass	1	0	0	1
pass	1	1	1	0

d) Provide meaningful chronogram to visualize the behavior of the circuit:

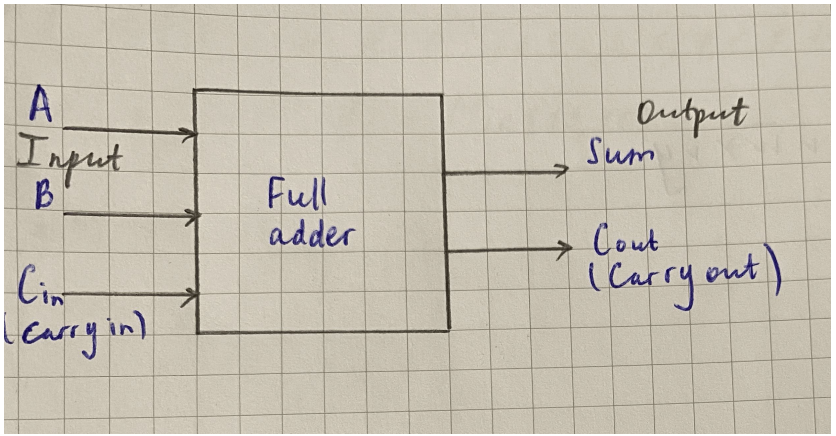


Truth table, K-map and resulting Boolean expressions for the sum S and the carry output bit:



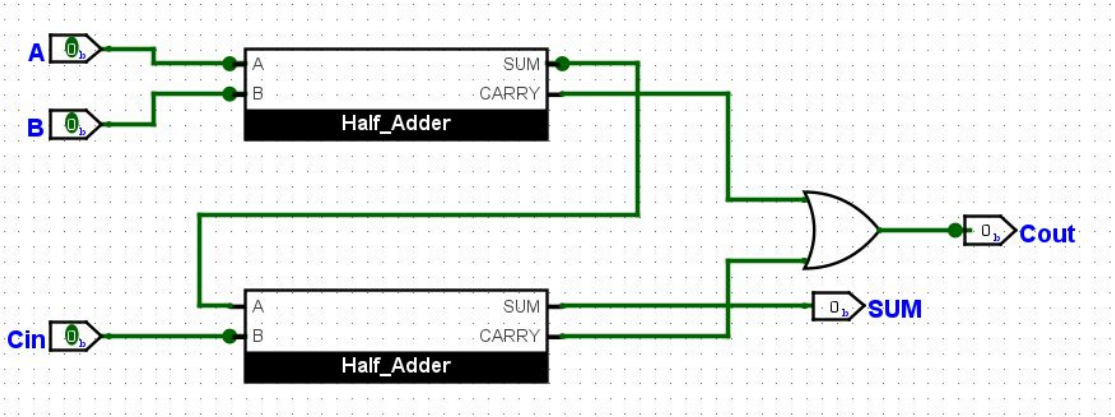
1.2. Use the half adder to implement a full adder:

a) Design the circuit as a block diagram:



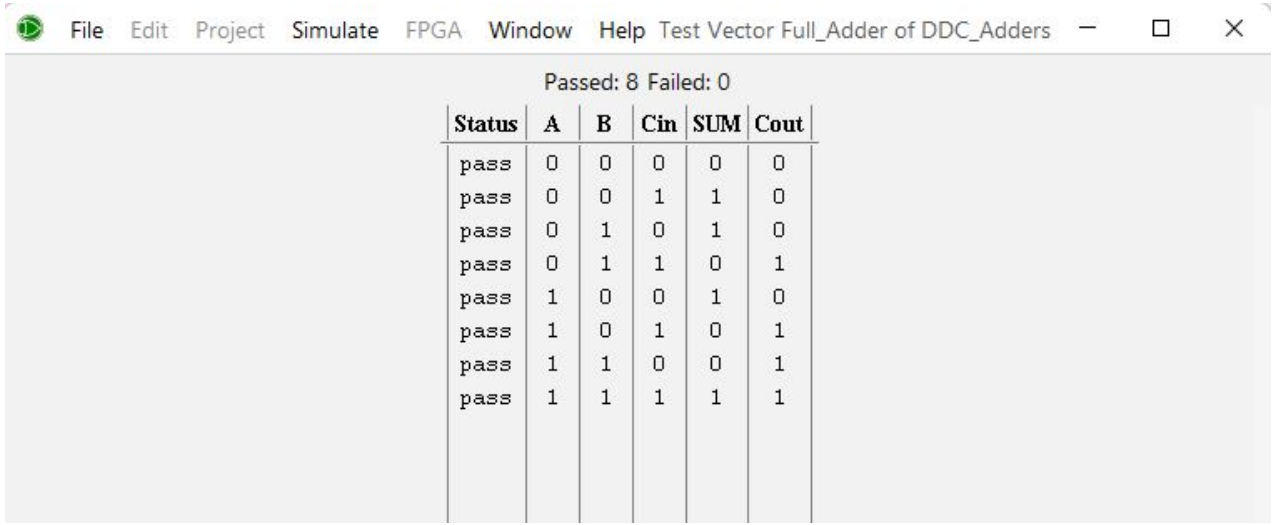
The full adder has three input states (A, B, Cin) and two output states (Sum and Cout)

b) Implement the circuit in Logisim:



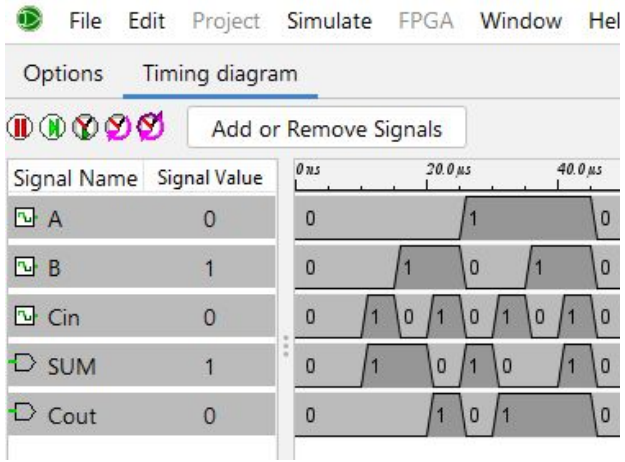
A full adder can be implemented using two half adders and one OR gate

c) Test the circuit by using Test-Vector:

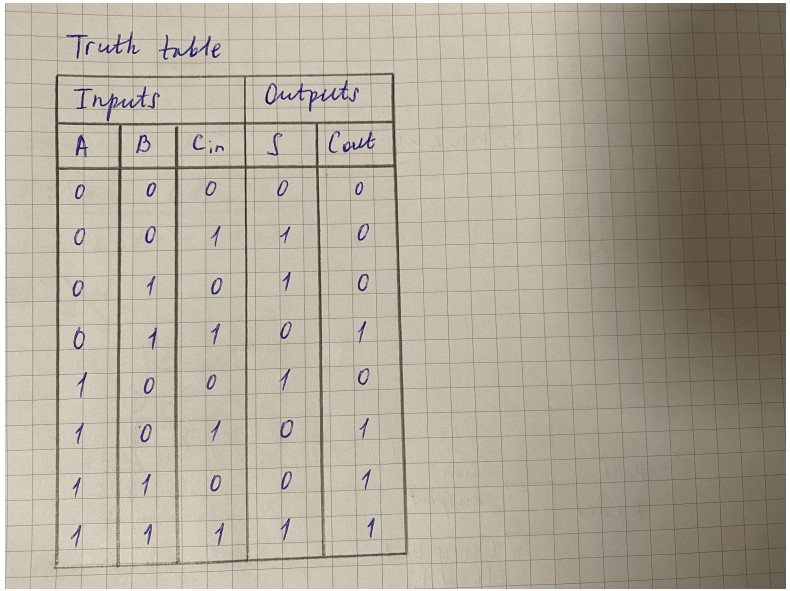


Status	A	B	Cin	SUM	Cout
pass	0	0	0	0	0
pass	0	0	1	1	0
pass	0	1	0	1	0
pass	0	1	1	0	1
pass	1	0	0	1	0
pass	1	0	1	0	1
pass	1	1	0	0	1
pass	1	1	1	1	1

d) Provide meaningful chronogram to visualize the behavior of the circuit:



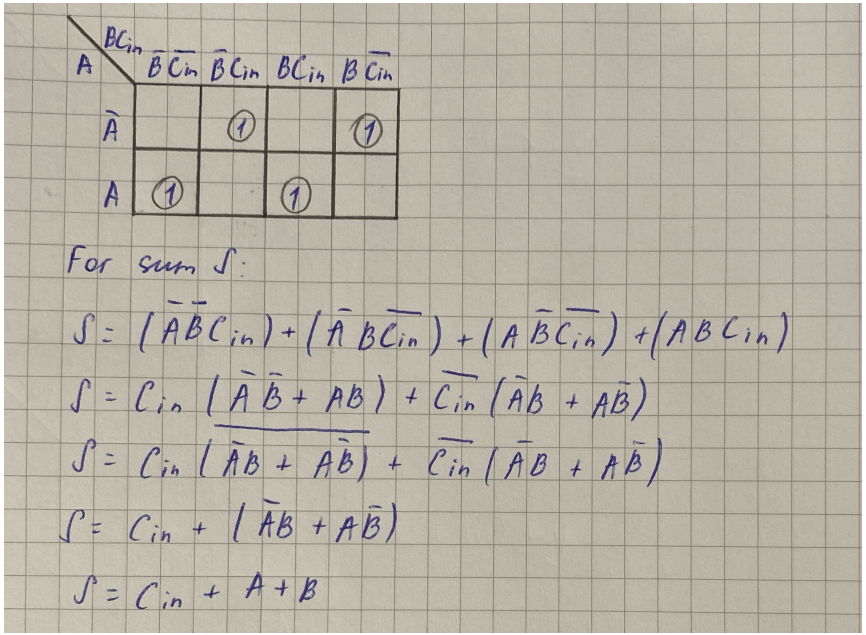
Truth table, K-map and resulting Boolean expressions for the sum S and the carry output bit:



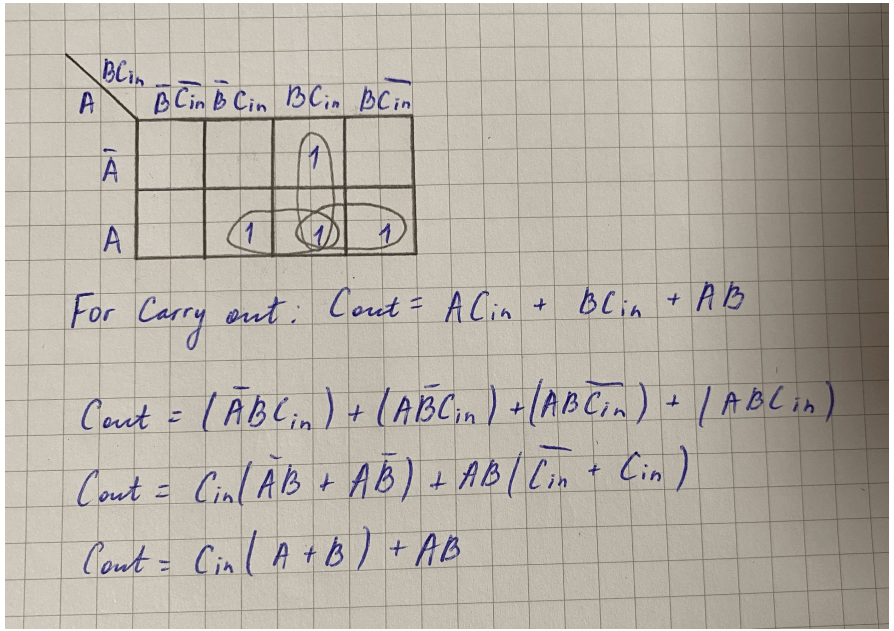
Truth table

Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table, K-map and resulting Boolean expressions for the sum S and the carry output bit:

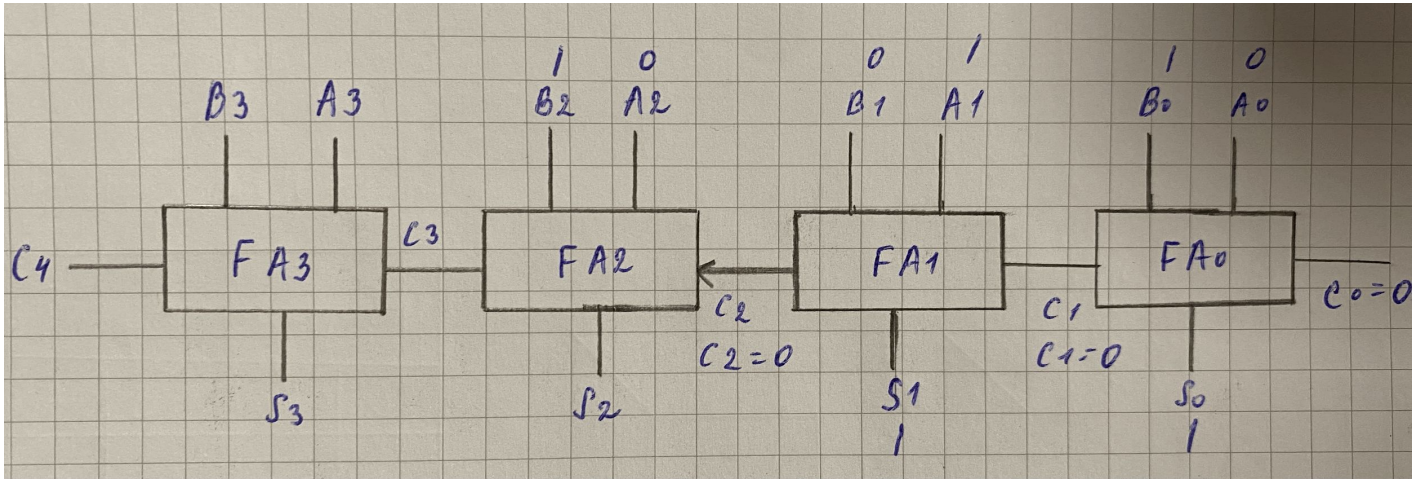


Truth table, K-map and resulting Boolean expressions for the sum S and the carry output bit:

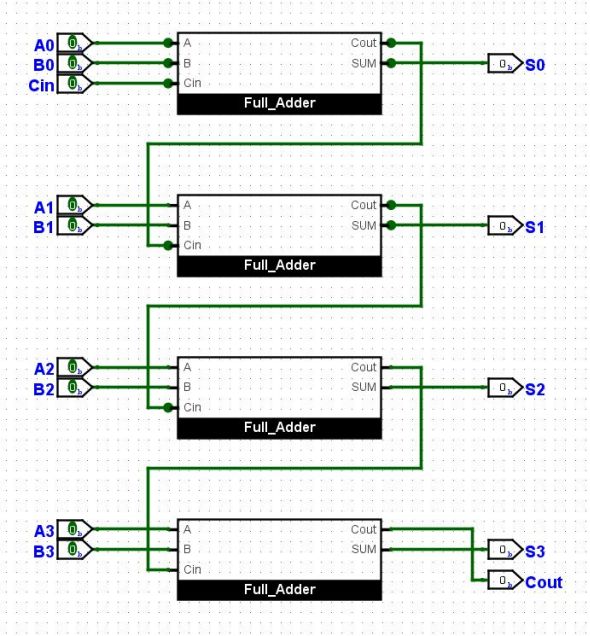


1.3. Use the full adder to implement a Carry-Ripple adder:

a) Design the circuit as a block diagram:



b) Implement the circuit in Logisim:



2. Arithmetic Unit

2.1) B-input-logic component for 1 bit

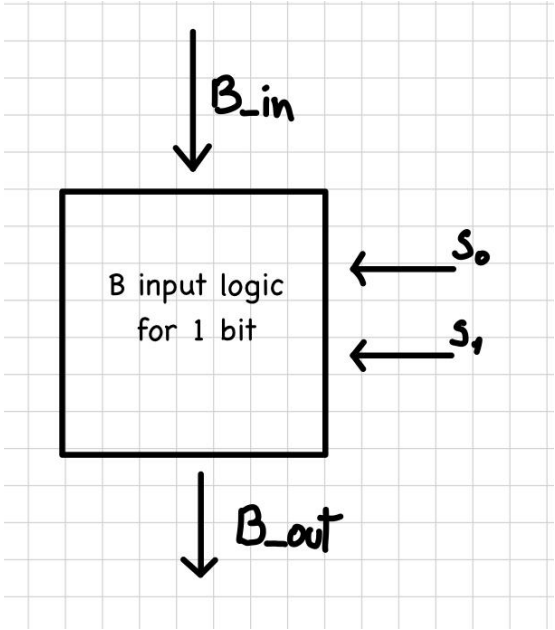
2.2) 1-bit B-input-logic to construct an n-bit version of it (n=4)

2.3) Implement the arithmetic unit for 1 bit.

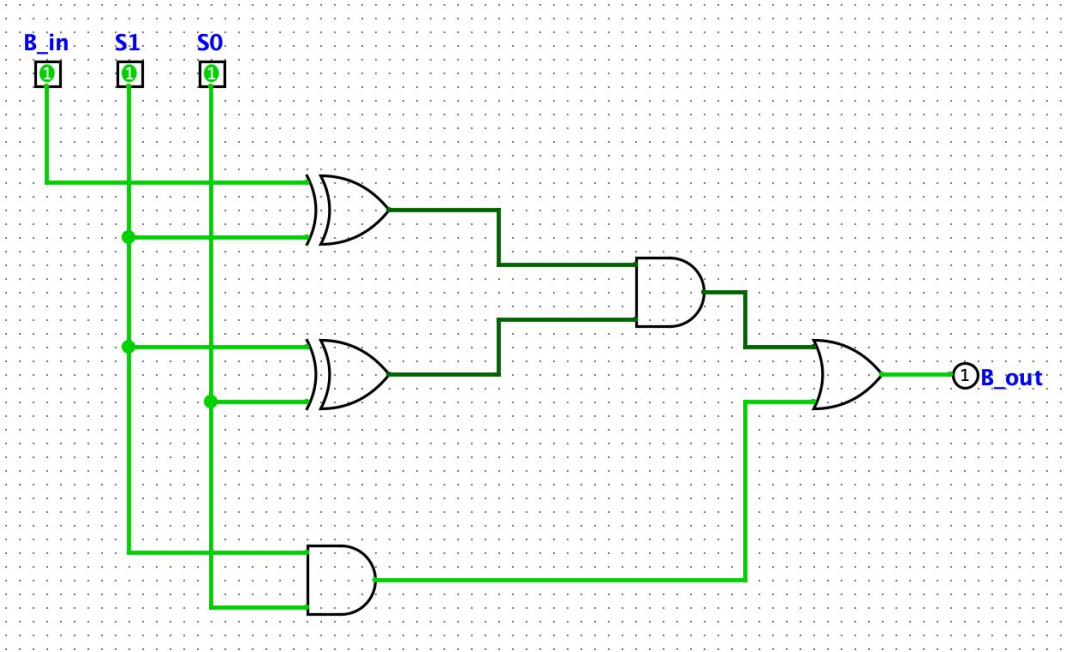
2.4) 1-bit arithmetic unit to construct an n-bit version of it.

2.1 B-input-logic component for 1 bit

Design the circuit as a block diagram



2.1 B-input-logic component for 1 bit



Implement the circuit in Logisim

2.1 B-input-logic component for 1 bit

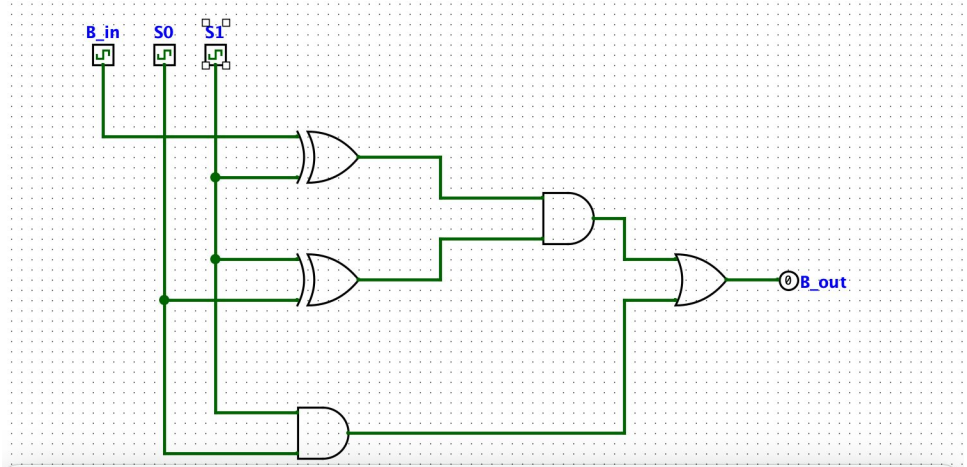
Test Vectors + Timing Diagram

Test Vector input_logic_1bit of B_inputlogic

Passed: 8 Failed: 0

Status	B_in	S0	S1	B_out
pass	0	0	0	0
pass	0	0	1	1
pass	0	1	0	0
pass	0	1	1	1
pass	1	0	0	0
pass	1	0	1	0
pass	1	1	0	1
pass	1	1	1	1

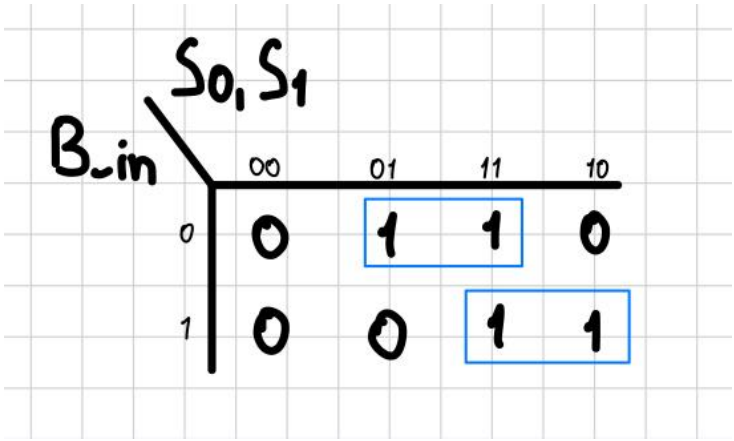
Load Vector | Run | Stop | Reset | Close Window



2.1 B-input-logic component for 1 bit

Truth table and K-Map

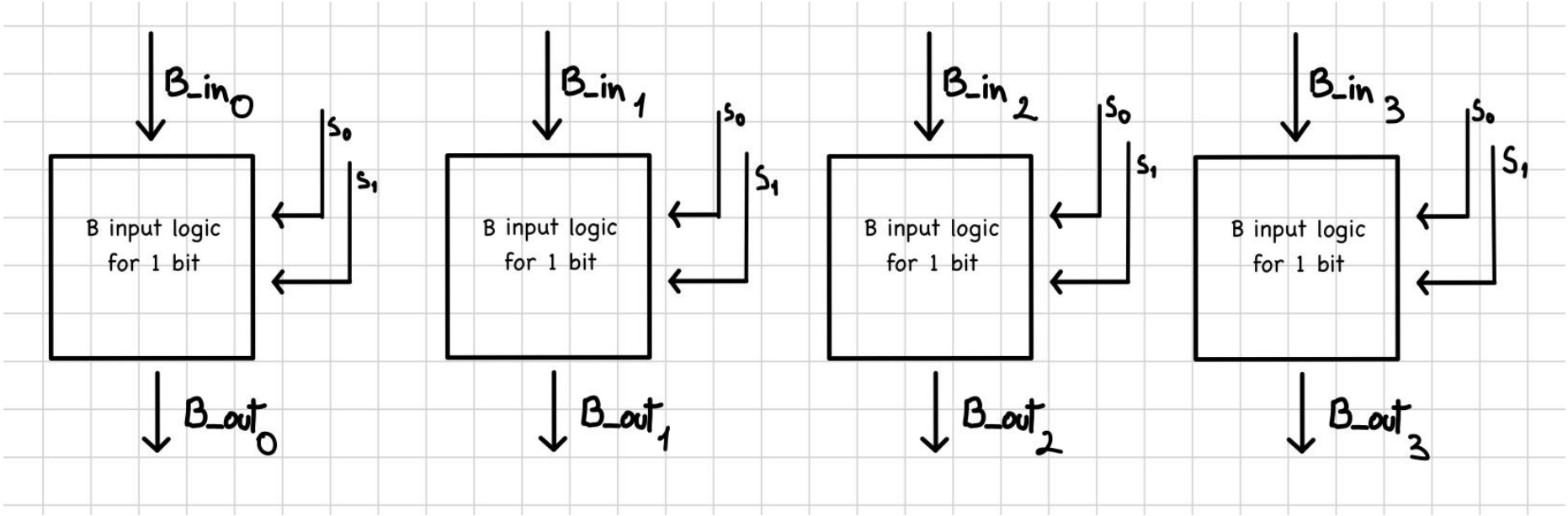
B_in	S_0	S_1	B_out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$(\overline{B\_in} \times S1) + (B\_in \times S0)$

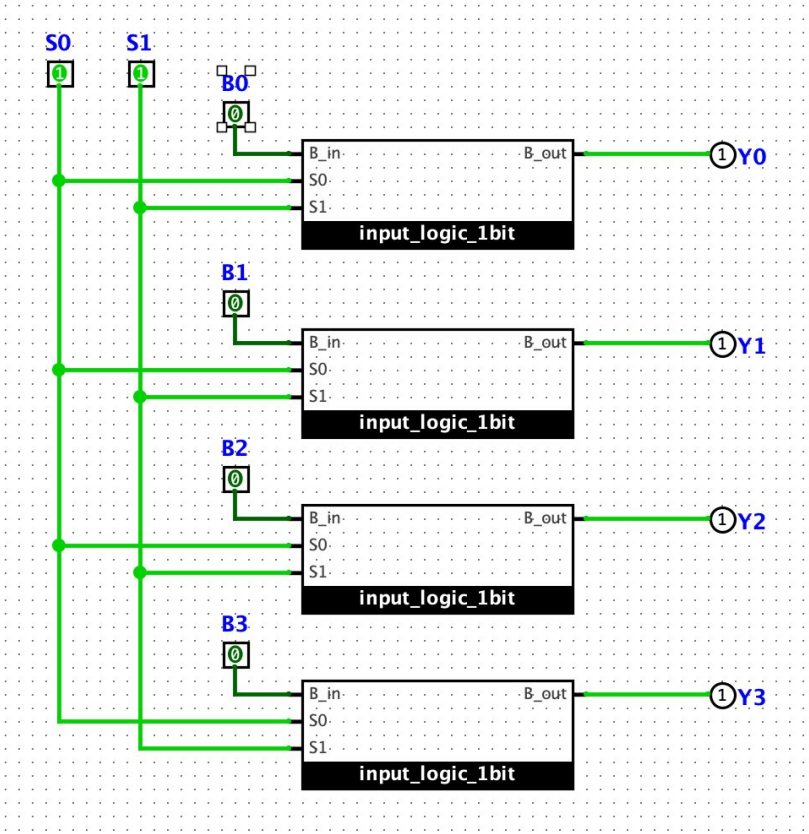
2.2) 1-bit B-input-logic to construct an n-bit version of it (n=4)

Design the circuit as a block diagram



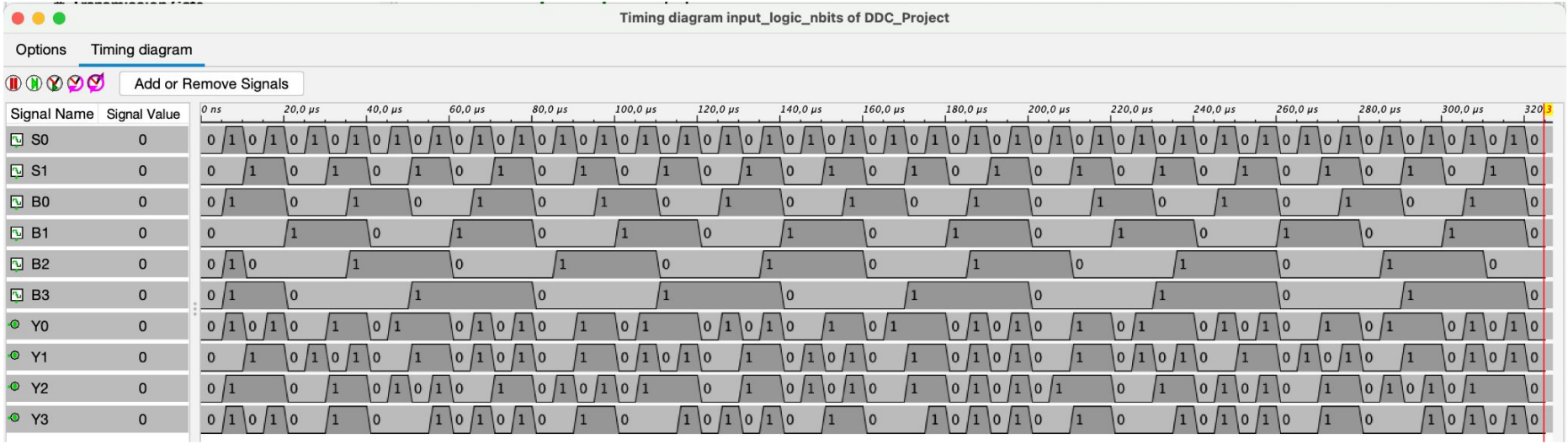
2.2) 1-bit B-input-logic to construct an n-bit version of it (n=4)

Implement the circuit in Logisim



2.2) 1-bit B-input-logic to construct an n-bit version of it (n=4)

Timing Diagram



2.2) 1-bit B-input-logic to construct an n-bit version of it (n=4)

Test the circuit using Test-Vectors

64/64 Passed

Test Vector implement of B_inputlogic

Passed: 64 Failed: 0

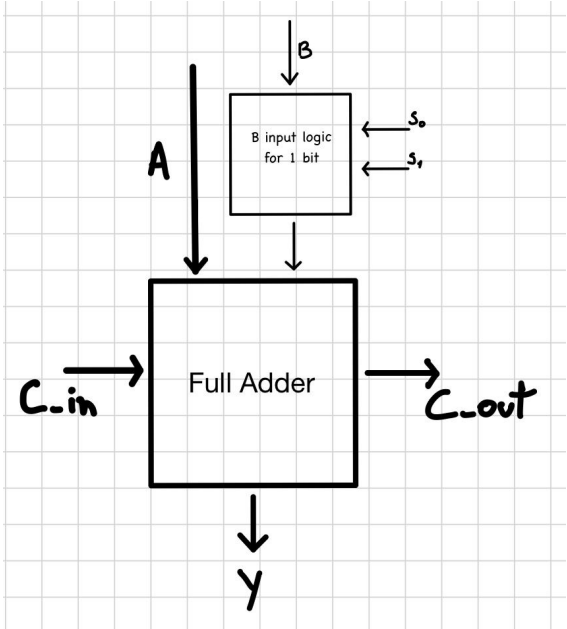
Status	S0	S1	B0	B1	B2	B3	Y0	Y1	Y2	Y3
pass	0	0	0	0	0	0	0	0	0	0
pass	0	0	0	0	0	1	0	0	0	0
pass	0	0	0	0	1	0	0	0	0	0
pass	0	0	0	0	1	1	0	0	0	0
pass	0	0	0	1	0	0	0	0	0	0
pass	0	0	0	1	0	1	0	0	0	0
pass	0	0	0	1	1	0	0	0	0	0
pass	0	0	0	1	1	1	0	0	0	0
pass	0	0	1	0	0	0	0	0	0	0
pass	0	0	1	0	0	1	0	0	0	0
pass	0	0	1	0	1	0	0	0	0	0
pass	0	0	1	0	1	1	0	0	0	0
pass	0	0	1	1	0	0	0	0	0	0
pass	0	0	1	1	0	1	0	0	0	0
pass	0	0	1	1	1	0	0	0	0	0
pass	0	0	1	1	1	1	0	0	0	0
pass	0	1	0	0	0	0	1	1	1	1
pass	0	1	0	0	0	1	1	1	1	0
pass	0	1	0	0	1	0	1	1	0	1
pass	0	1	0	0	1	1	1	1	0	0
pass	0	1	0	1	0	0	1	0	1	1
pass	0	1	0	1	0	1	1	0	1	0
pass	0	1	0	1	1	0	1	0	0	1
pass	0	1	0	1	1	1	1	0	0	0
pass	0	1	1	0	0	0	1	1	1	1
pass	0	1	1	0	0	1	1	1	1	0
pass	0	1	1	0	1	0	1	1	0	1
pass	0	1	1	0	1	1	1	1	0	0
pass	0	1	1	1	0	0	1	0	1	1
pass	0	1	1	1	0	1	1	0	1	0
pass	0	1	1	1	1	0	1	0	0	1
pass	0	1	1	1	1	1	1	0	0	0
pass	1	0	0	0	0	0	0	0	0	0
pass	1	0	0	0	0	1	0	0	0	1
pass	1	0	0	0	1	0	0	0	1	0
pass	1	0	0	0	1	1	0	0	1	1
pass	1	0	0	1	0	0	0	1	0	0
pass	1	0	0	1	0	1	0	1	0	1
pass	1	0	0	1	1	0	0	1	1	0
pass	1	0	0	1	1	1	0	1	1	1
pass	1	0	1	0	0	0	1	0	0	0
pass	1	0	1	0	0	1	1	0	0	1
pass	1	0	1	0	1	0	1	0	1	0
pass	1	0	1	0	1	1	1	0	1	1
pass	1	0	1	1	0	0	1	1	0	0
pass	1	0	1	1	0	1	1	1	0	1
pass	1	0	1	1	1	0	1	1	1	0
pass	1	0	1	1	1	1	1	1	1	1
pass	1	1	0	0	0	0	1	1	1	1

Load Vector | Run | Stop | Reset | Close Window

S0	S1	B0	B1	B2	B3	Y0	Y1	Y2	Y3
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0
0	1	0	0	0	0	1	1	1	1
0	1	0	0	0	1	1	1	1	0
0	1	0	0	1	0	1	1	0	1
0	1	0	0	1	1	1	1	0	0
0	1	0	1	0	0	1	0	1	1
0	1	0	1	0	1	1	0	1	0
0	1	0	1	1	0	1	0	0	1
0	1	0	1	1	1	1	0	0	0
0	1	1	0	0	0	0	1	1	1
0	1	1	0	0	1	0	1	1	0
0	1	1	0	1	0	0	1	0	1
0	1	1	0	1	1	0	1	0	0
0	1	1	1	0	0	0	0	1	1
0	1	1	1	0	1	0	0	1	0
0	1	1	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0	1	0
1	0	0	0	1	1	0	0	1	1
1	0	0	1	0	0	0	1	0	0
1	0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1	0
1	0	0	1	1	1	0	1	1	1
1	0	1	0	0	0	1	0	0	0
1	0	1	0	0	1	1	0	0	1
1	0	1	0	1	0	1	0	1	0
1	0	1	0	1	1	1	0	1	1
1	0	1	1	0	0	1	1	0	0
1	0	1	1	0	1	1	1	0	1
1	0	1	1	1	0	1	1	1	0
1	0	1	1	1	1	1	1	1	1
1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	1	1	1	1	1
1	1	0	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1	1	1
1	1	0	1	0	0	1	1	1	1
1	1	0	1	0	1	1	1	1	1
1	1	0	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	0	0	1	1	1	1
1	1	1	0	0	1	1	1	1	1
1	1	1	0	1	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1
1	1	1	1	0	0	1	1	1	1
1	1	1	1	0	1	1	1	1	1
1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1

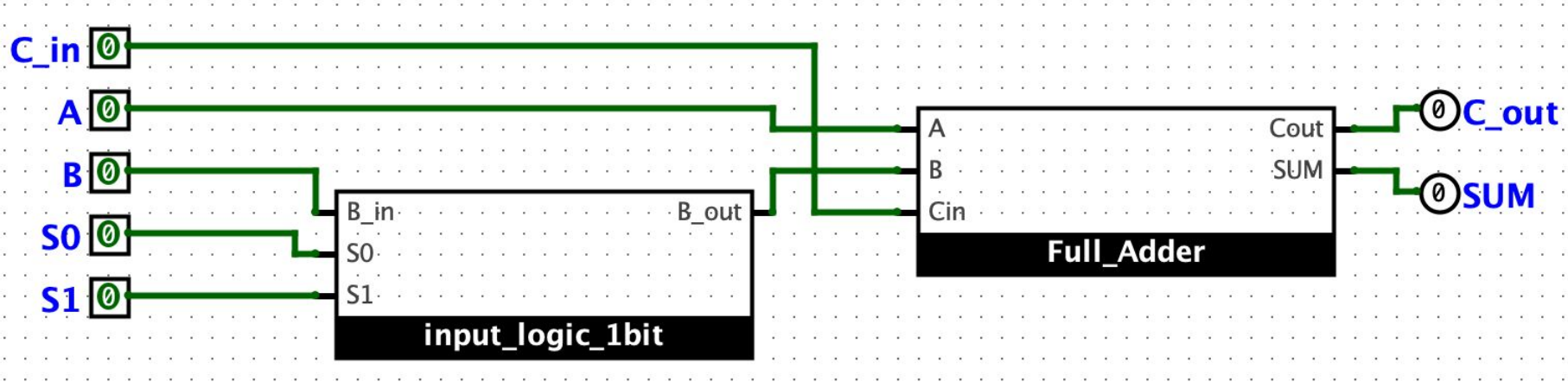
2.3) Implement the arithmetic unit for 1 bit.

Design the circuit as a block diagram



2.3) Implement the arithmetic unit for 1 bit.

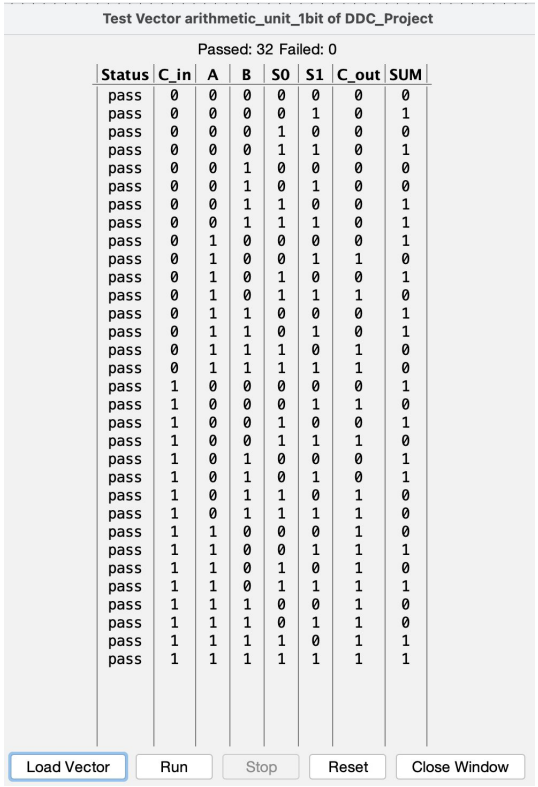
Implement the circuit in Logisim



2.3) Implement the arithmetic unit for 1 bit.

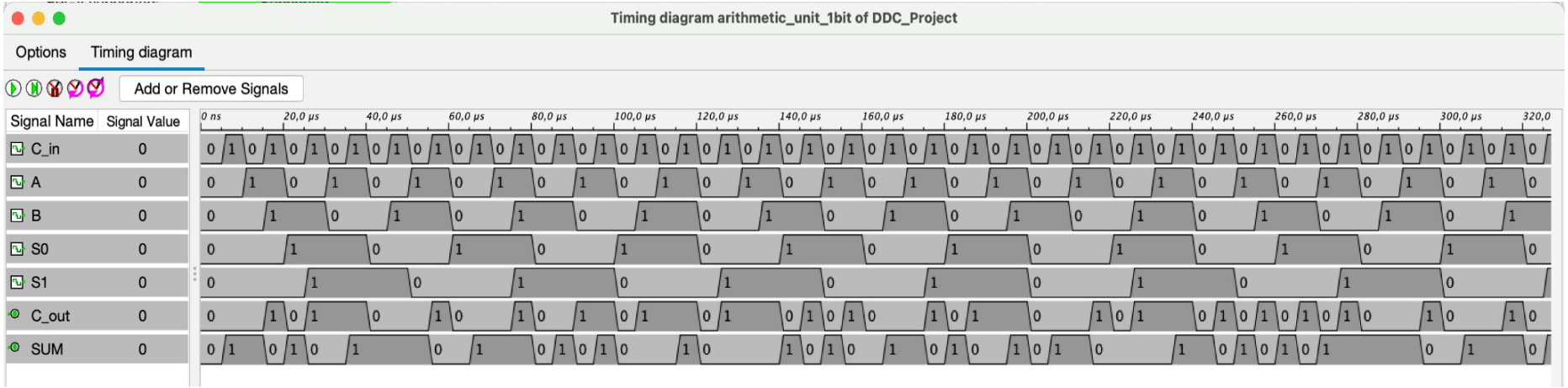
Test Vector

C_in	A	B	S0	S1	C_out	SUM
0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	0	0
0	0	0	1	1	0	1
0	0	1	0	0	0	0
0	0	1	0	1	0	0
0	0	1	1	0	0	1
0	0	1	1	1	0	1
0	1	0	0	0	0	1
0	1	0	0	1	1	0
0	1	0	1	0	0	1
0	1	0	1	1	1	0
0	1	1	0	0	0	1
0	1	1	0	1	0	1
0	1	1	1	0	1	0
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	0	1	1	0
1	0	0	1	0	0	1
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	0	1	0	1
1	0	1	1	0	1	0
1	0	1	1	1	1	0
1	1	0	0	0	1	0
1	1	0	0	1	1	1
1	1	0	1	0	1	0
1	1	0	1	1	1	1
1	1	1	0	0	1	0
1	1	1	0	1	1	0
1	1	1	1	0	1	1
1	1	1	1	1	1	1



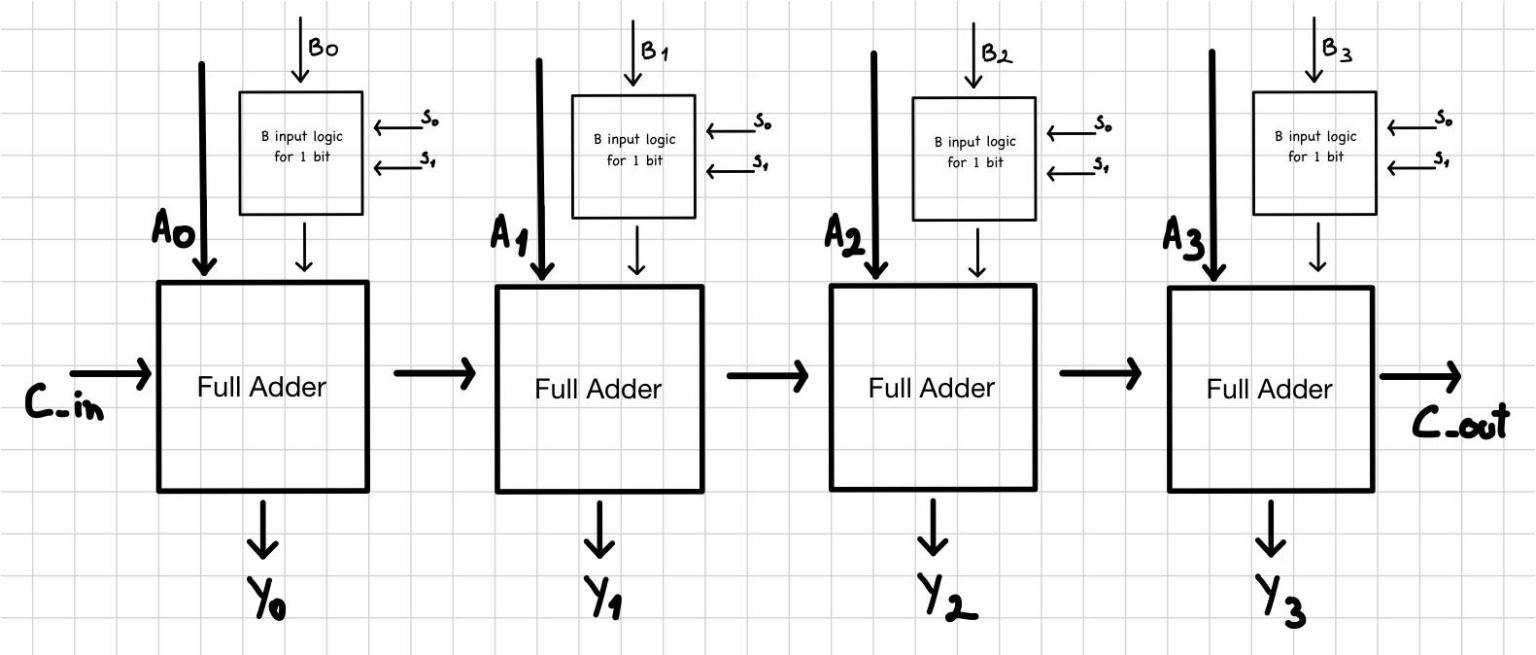
2.3) Implement the arithmetic unit for 1 bit.

Time Diagram



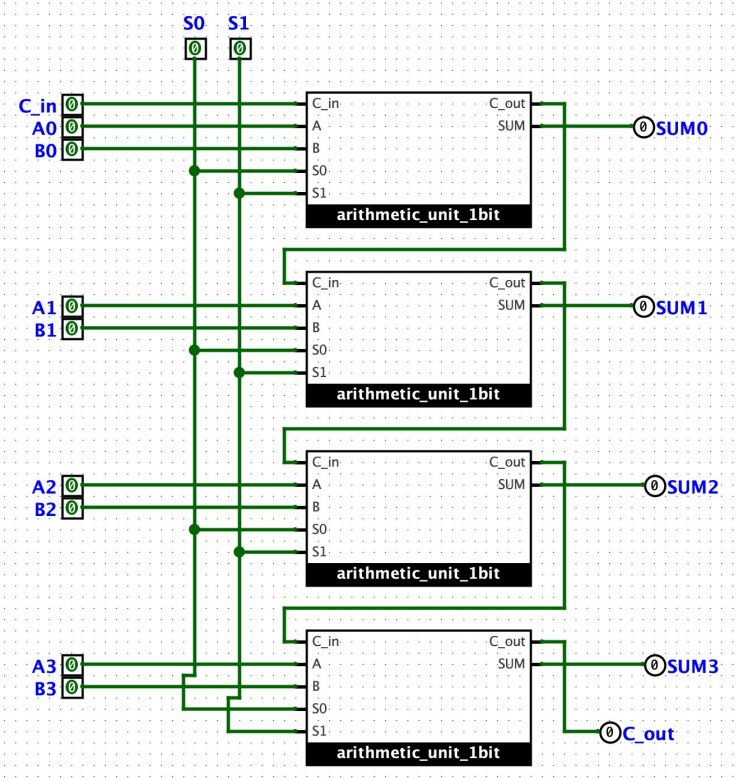
2.4) 1-bit B-input-logic to construct a 4-bit version of it

Design the circuit as a block diagram



2.4) 1-bit B-input-logic to construct a 4-bit version of it

Implement the circuit in Logisim



2.4) 1-bit B-input-logic to construct a 4-bit version of it

Test Vector

2048/2048 Passed

S0	S1	C_in	A0	B0	A1	B1	A2	B2	A3	B3	SUM0	SUM1	SUM2	SUM3	C_out
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	0
0	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0
0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0
0	0	0	0	0	0	1	1	0	1	1	0	0	1	1	0
0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0
0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	1	0	1	0
0	0	0	0	0	0	1	1	1	1	1	0	1	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0
0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0
0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	1	0	1	0	1	0	0	0	1	1	0
0	0	0	0	0	1	0	1	0	1	1	0	0	1	1	0
0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	0
0	0	0	0	0	1	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	1	0	1	1	1	0	0	1	0	1	0
0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0
0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	0
0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	0
0	0	0	0	0	1	1	0	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	0	1	1	1	0	0	0	1	0
0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0
0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	0
0	0	0	0	0	1	1	1	0	1	0	0	0	1	1	0
0	0	0	0	0	1	1	1	0	1	1	0	0	1	1	0
0	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0
0	0	0	0	0	1	1	1	1	0	1	0	1	0	0	0
0	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0
0	0	0	0	0	1	1	1	1	1	1	0	1	0	1	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	0

Test Vector arithmetic_unit_nbits of B_inputlogic

Passed: 2048 Failed: 0

Status	S0	S1	C_in	A0	B0	A1	B1	A2	B2	A3	B3	SUM0	SUM1	SUM2	SUM3	C_out
pass	1	1	1	1	1	0	1	0	0	0	0	1	0	0	0	1
pass	1	1	1	1	1	0	1	0	0	0	1	1	0	0	0	1
pass	1	1	1	1	1	0	1	0	0	1	0	1	0	0	1	1
pass	1	1	1	1	1	0	1	0	0	1	1	1	0	0	1	1
pass	1	1	1	1	1	0	1	0	1	0	0	1	0	0	0	1
pass	1	1	1	1	1	0	1	0	1	0	1	1	0	0	0	1
pass	1	1	1	1	1	0	1	0	1	1	0	1	0	0	1	1
pass	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1
pass	1	1	1	1	1	0	1	1	0	0	0	1	0	1	0	1
pass	1	1	1	1	1	0	1	1	0	0	1	1	0	1	0	1
pass	1	1	1	1	1	0	1	1	0	1	0	1	0	1	1	1
pass	1	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1
pass	1	1	1	1	1	0	1	1	1	0	0	1	0	1	0	1
pass	1	1	1	1	1	0	1	1	1	0	1	1	0	1	0	1
pass	1	1	1	1	1	0	1	1	1	1	0	1	0	1	1	1
pass	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1
pass	1	1	1	1	1	1	0	0	0	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	0	0	0	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	0	0	0	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	0	0	0	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	0	0	1	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	0	0	1	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	0	0	1	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	0	1	0	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	0	1	0	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	0	1	1	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	0	1	1	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	1	0	0	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	1	0	0	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	1	0	0	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	1	0	1	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	1	0	1	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	1	0	1	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	1	1	0	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1
pass	1	1	1	1	1	1	1	1	1	0	0	1	1	0	0	1
pass	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	1
pass	1	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1
pass	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1

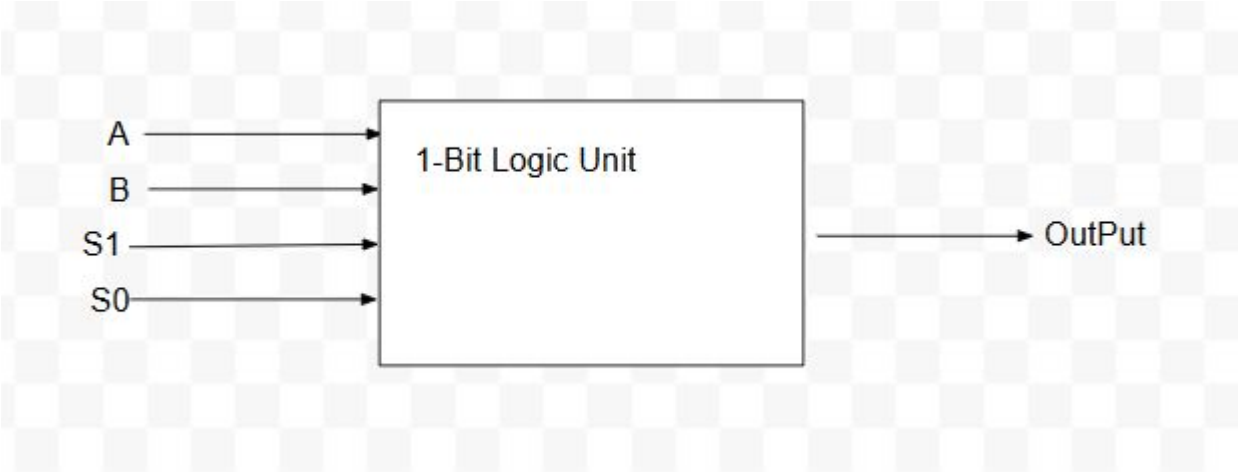
Load Vector | Run | Stop | Reset | Close Window

3. Logical Unit

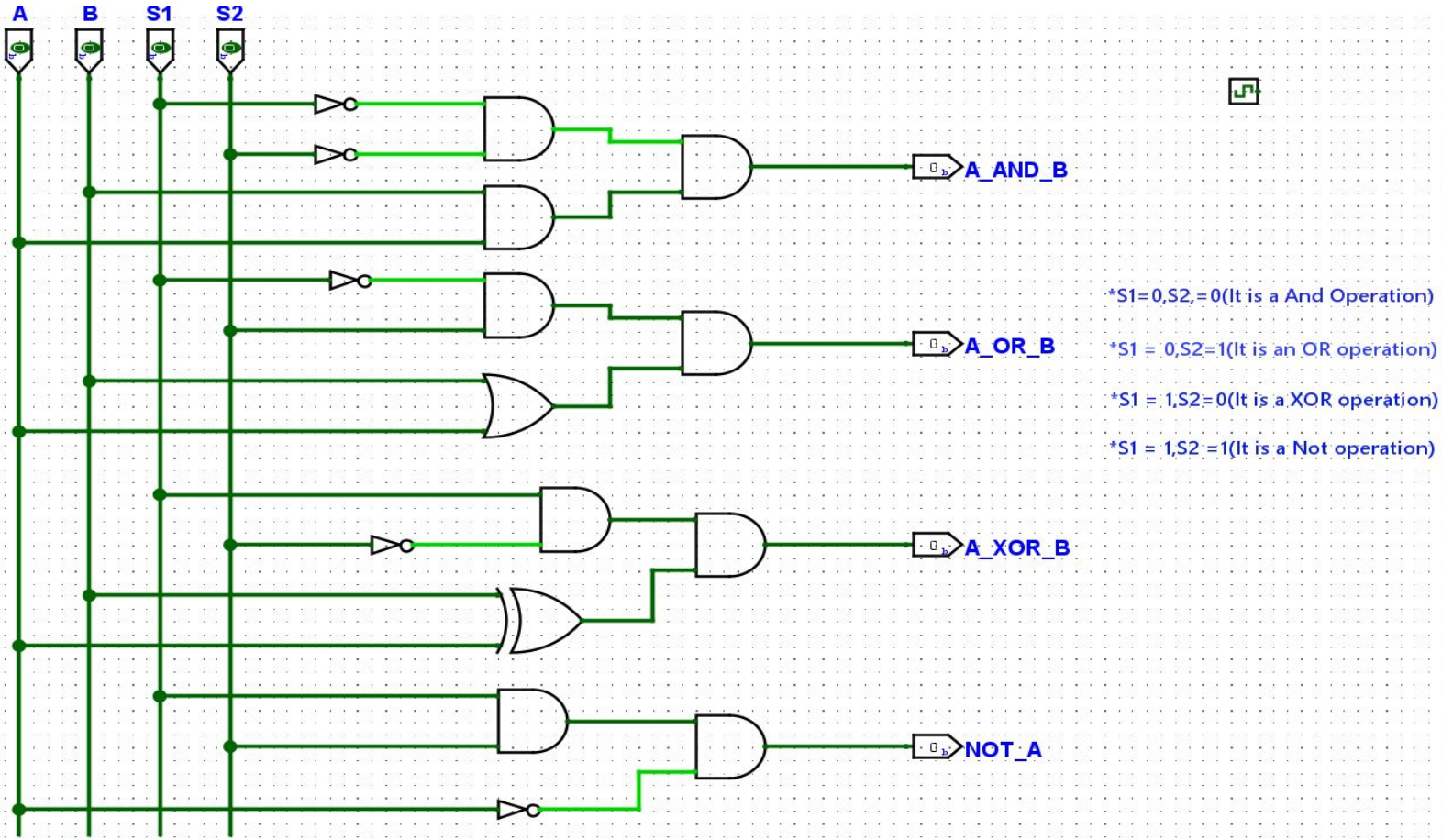
3.1) Implement logical unit for 1-bit

3.2) Use the 1-bit logical unit to construct an n-bit version of it.

3.1: Logic Unit 1-bit Block Diagram

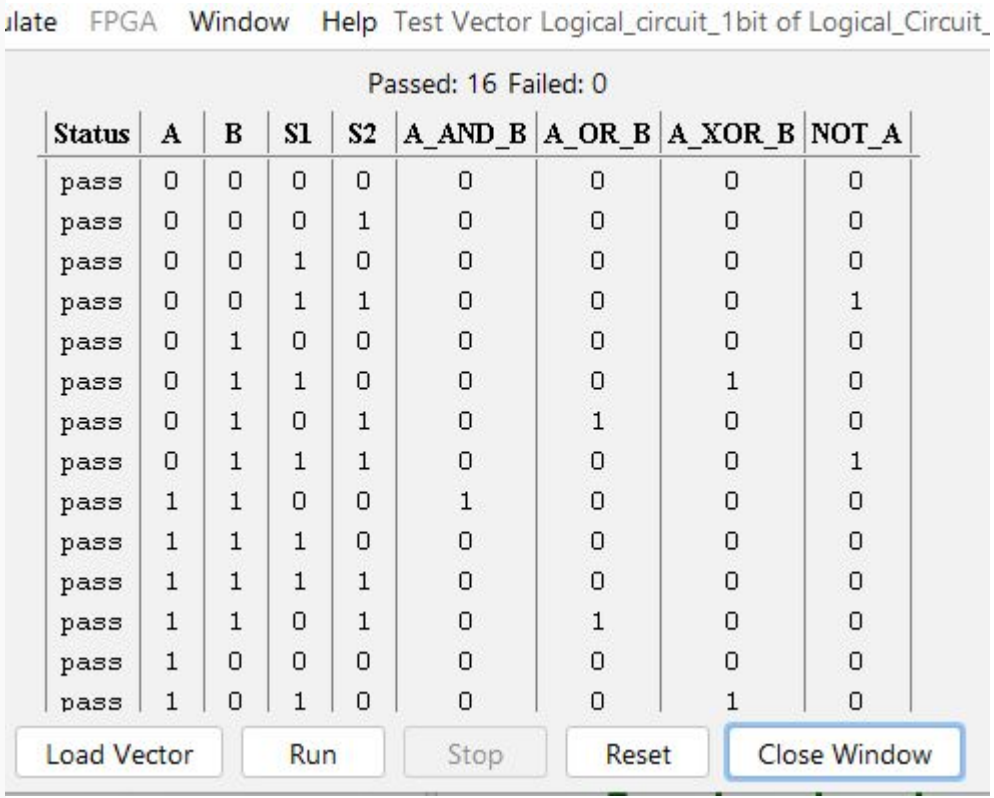


3.1:1-bit Logic Unit

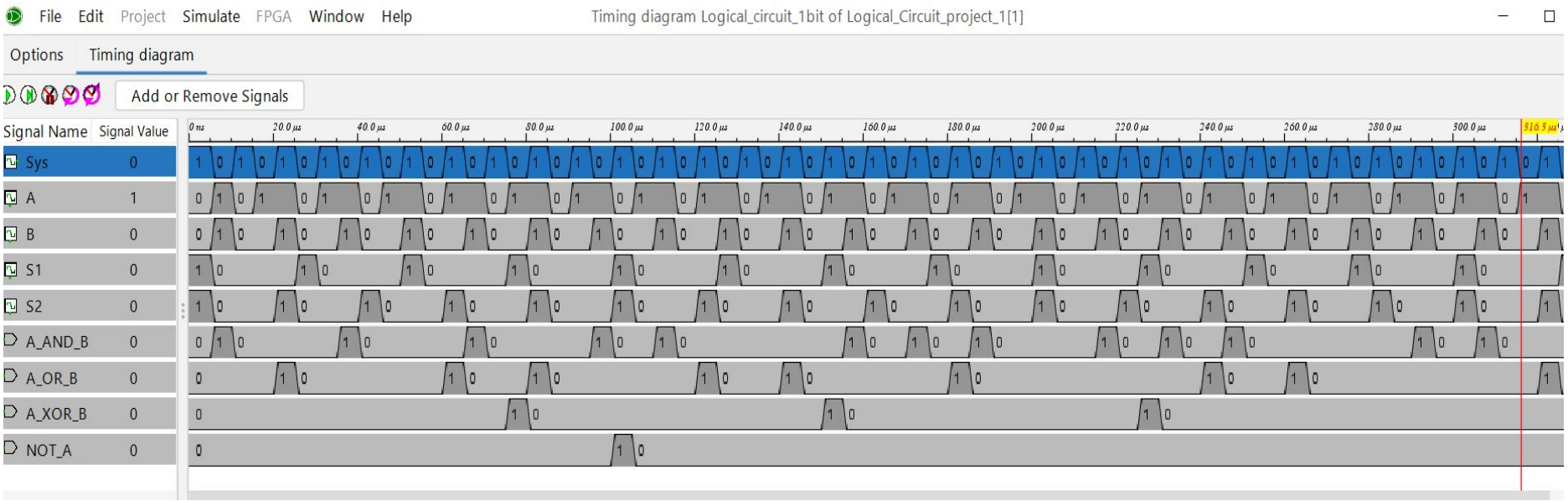


3.1:1-Bit Logic Unit Test Vector

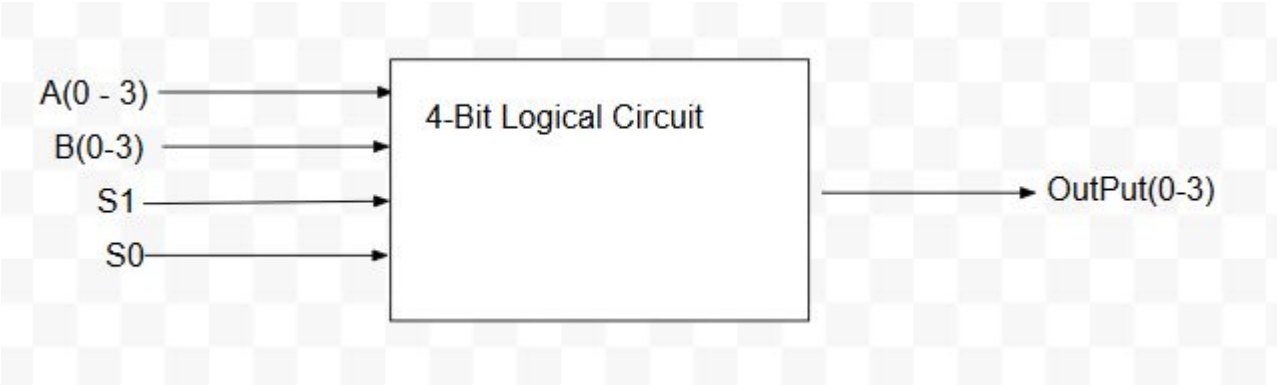
A	B	S1	S2	A-AND-B	A-OR-B	A-XOR-B	NOT-A
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	1	0	0	0	0
1	1	0	0	1	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0
1	1	0	1	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0
1	0	0	1	0	1	0	0
1	0	1	1	0	0	0	0



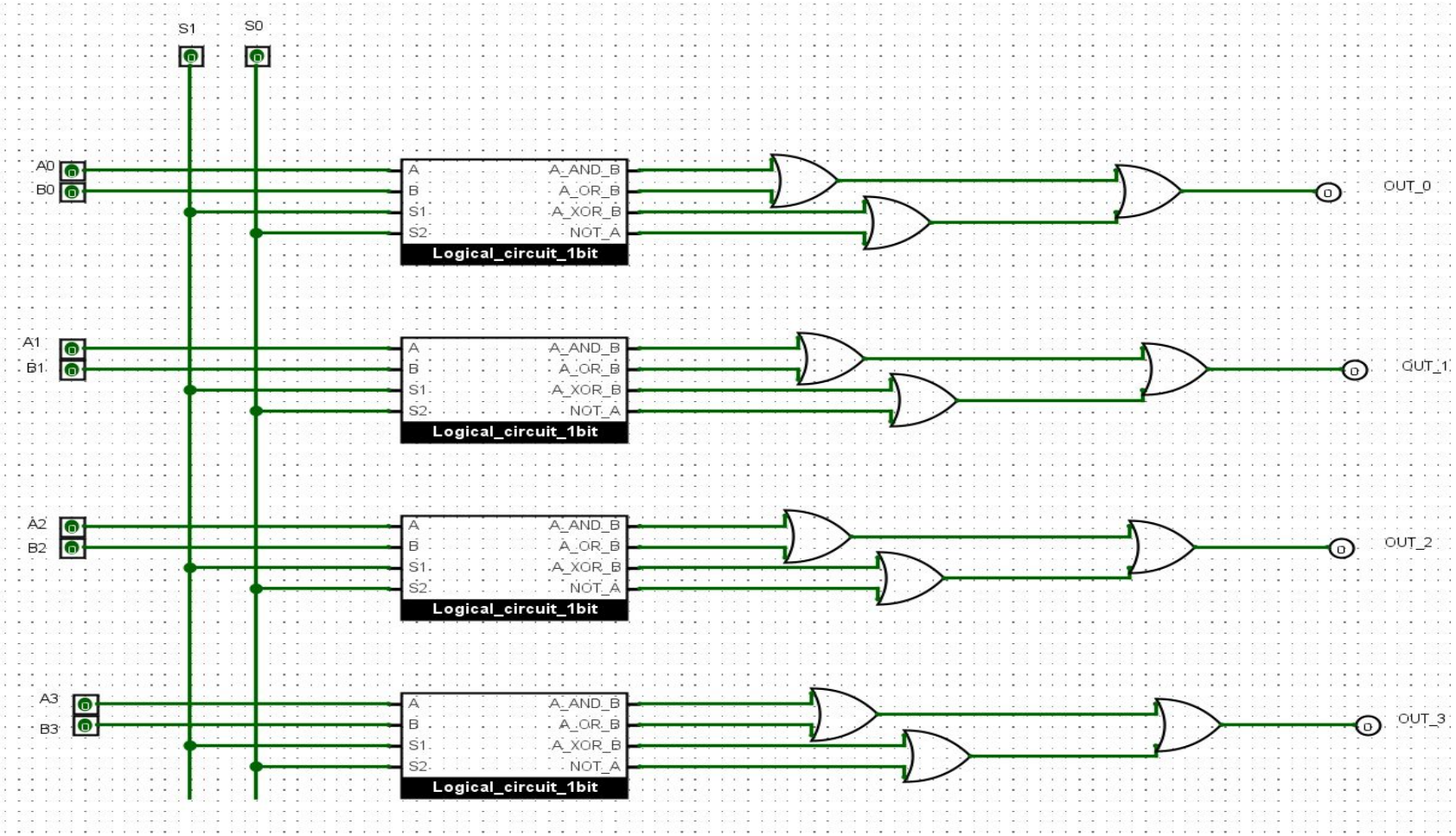
3.2:Chronograms for 1-bit Logic unit



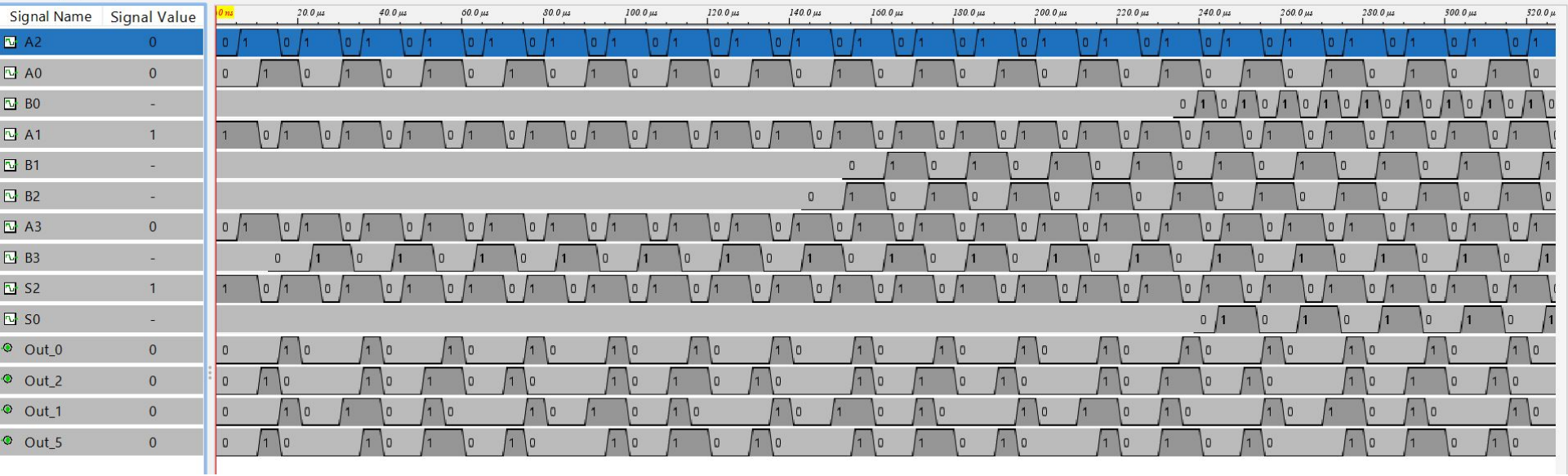
3.2:n-bit Logic Unit :Block Diagram



3.2:n-bit Logic Unit



3-2:Chronograms for n-bit logic unit

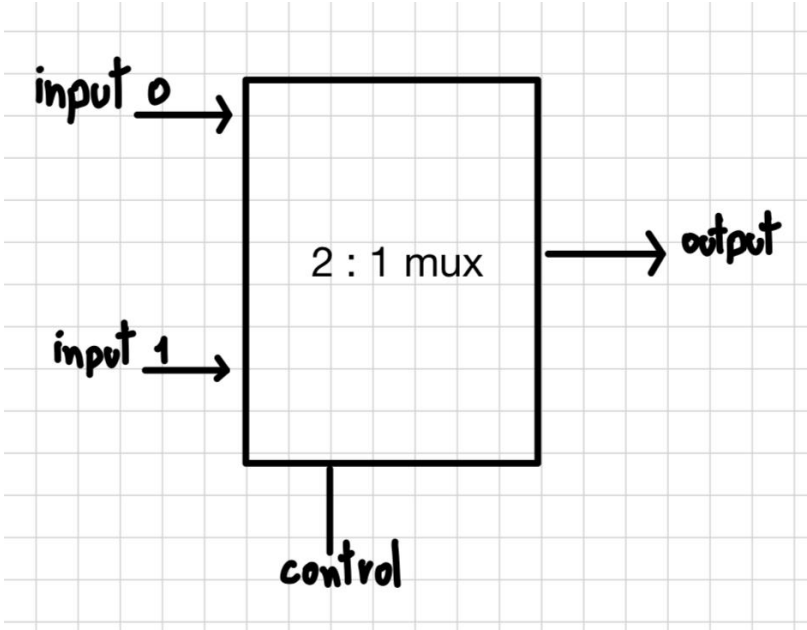


4. Multiplexing

Implement a 2:1 multiplexer. Provide truth table, K-map and the resulting Boolean expressions for output of the multiplexer.

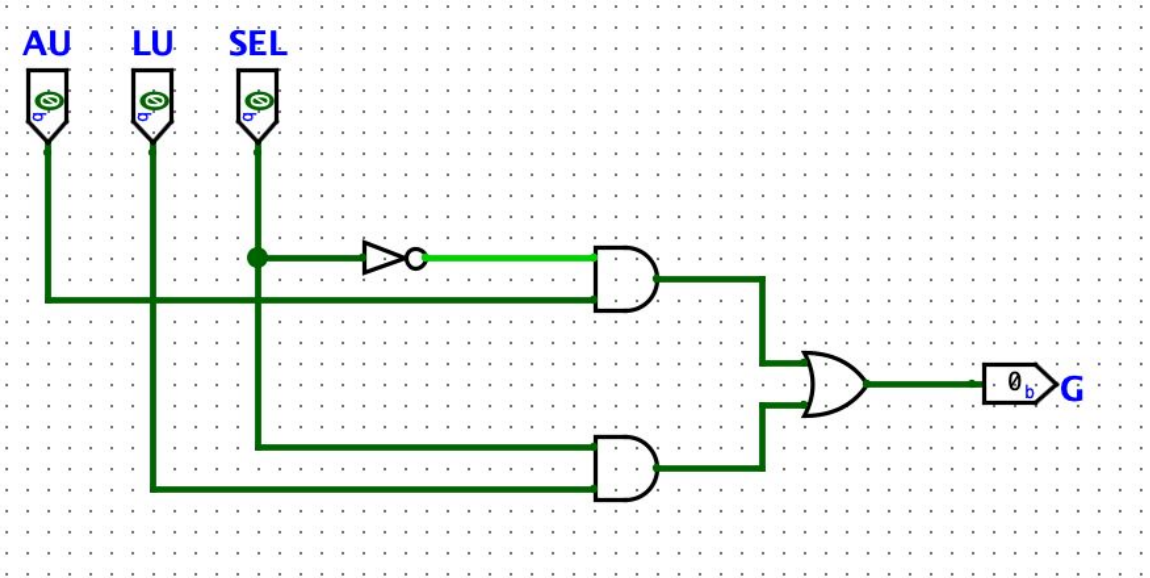
4. Multiplexing

Design a block diagram



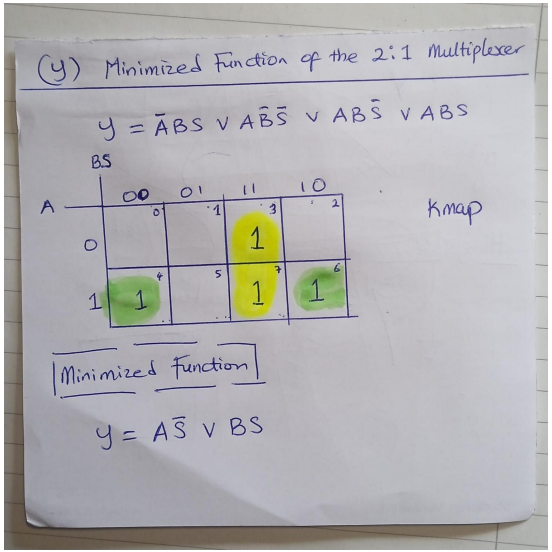
4. Multiplexing

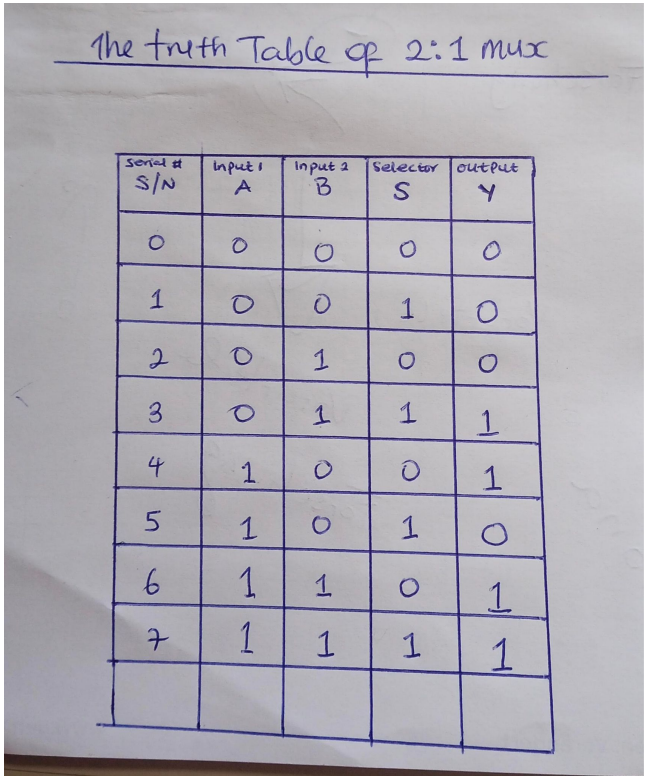
Implement the circuit in Logisim



4. Multiplexing

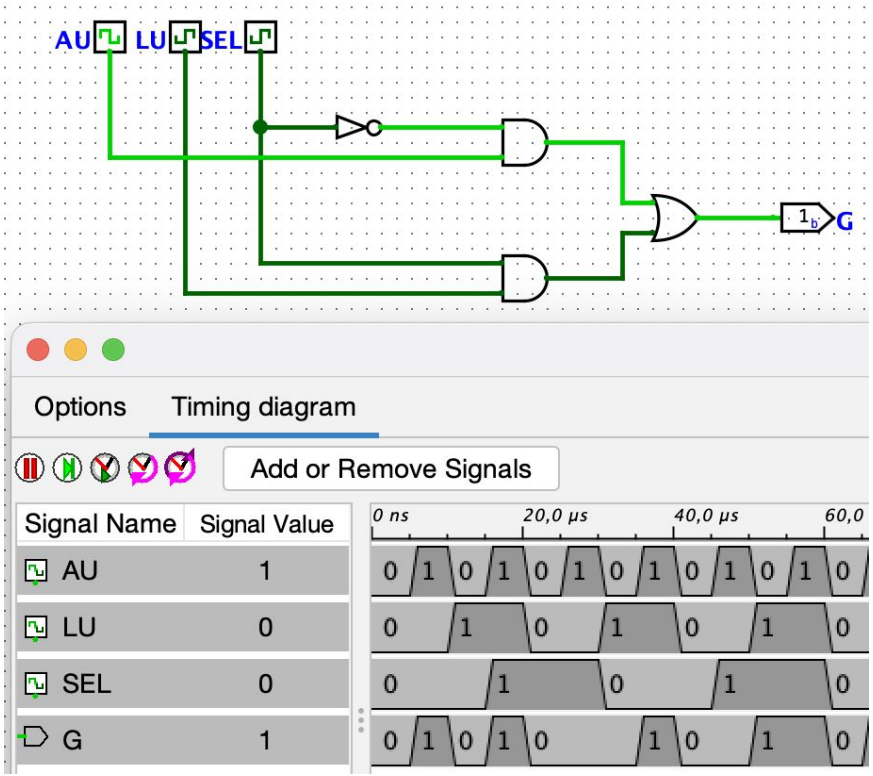
Truth Table + K-Map





4. Multiplexing

Timing Diagram

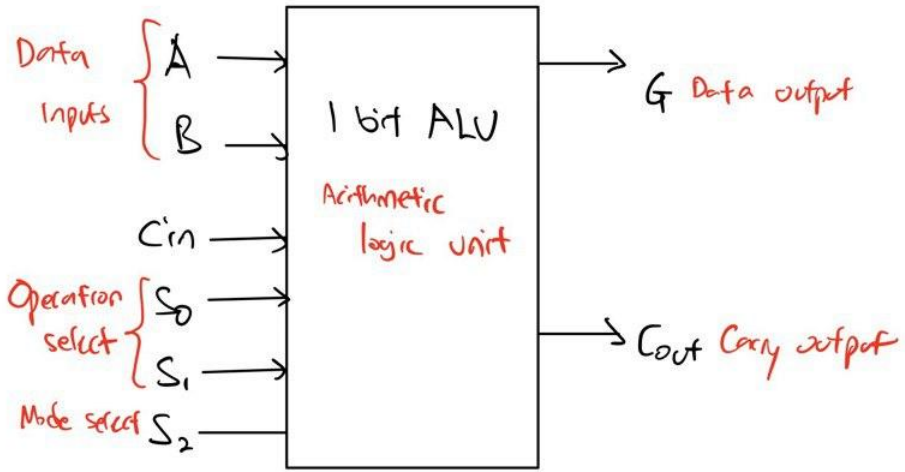


5. ALU

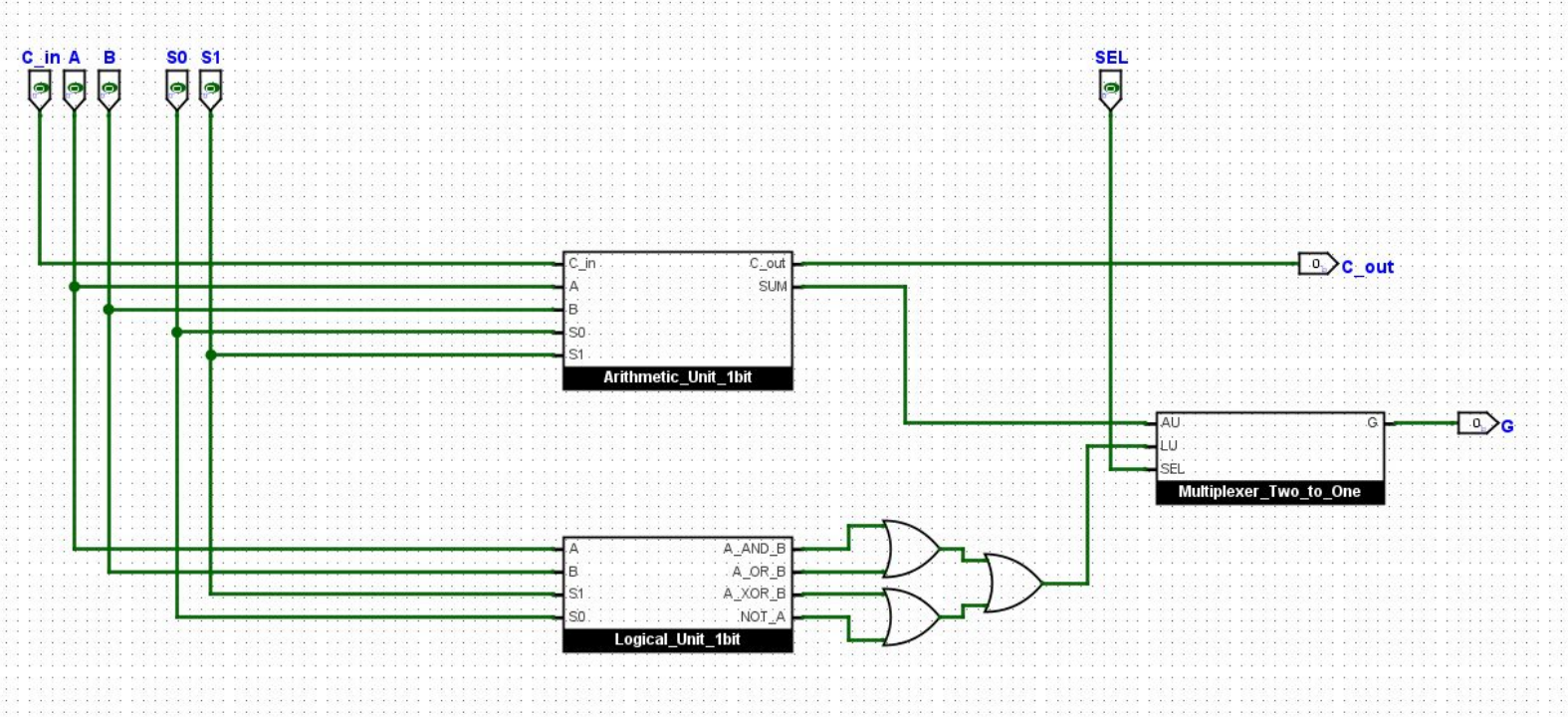
5.1) Implement the ALU for 1-bit.

5.2) Use the 1-bit ALU to construct an n-bit version of it.

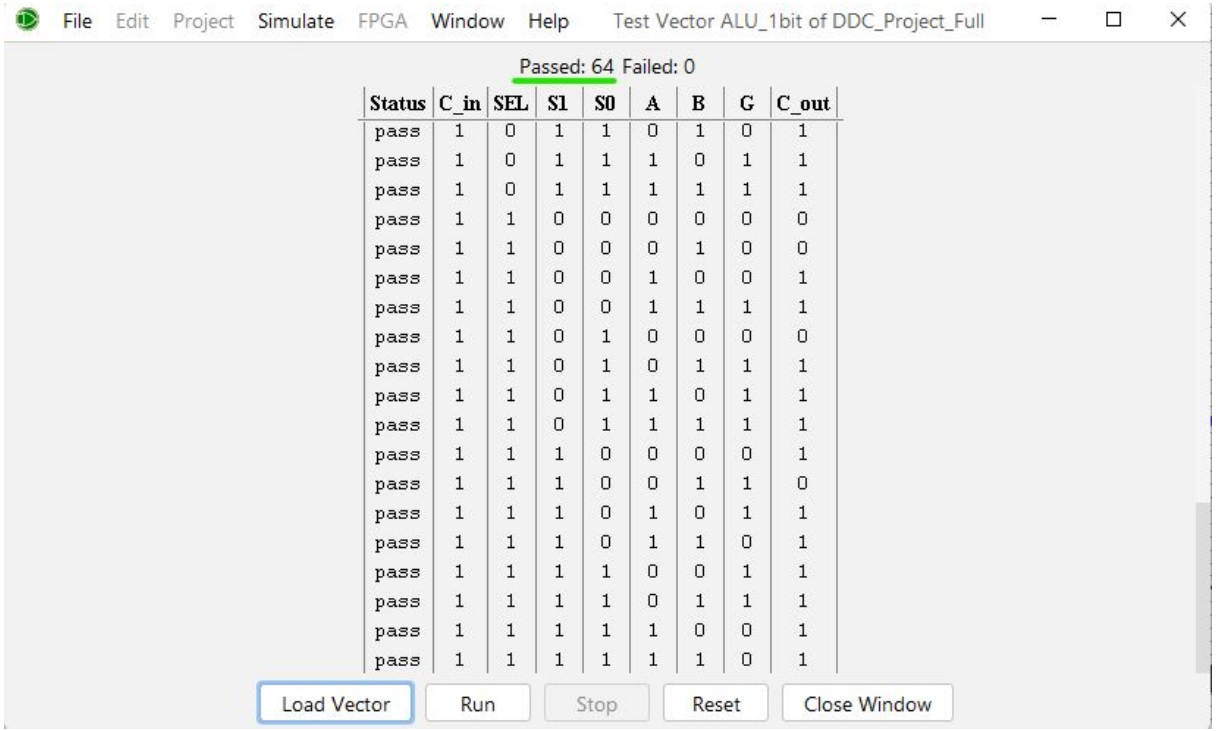
ALU - Block diagram (1 bit)



ALU - Logisim (1 bit)

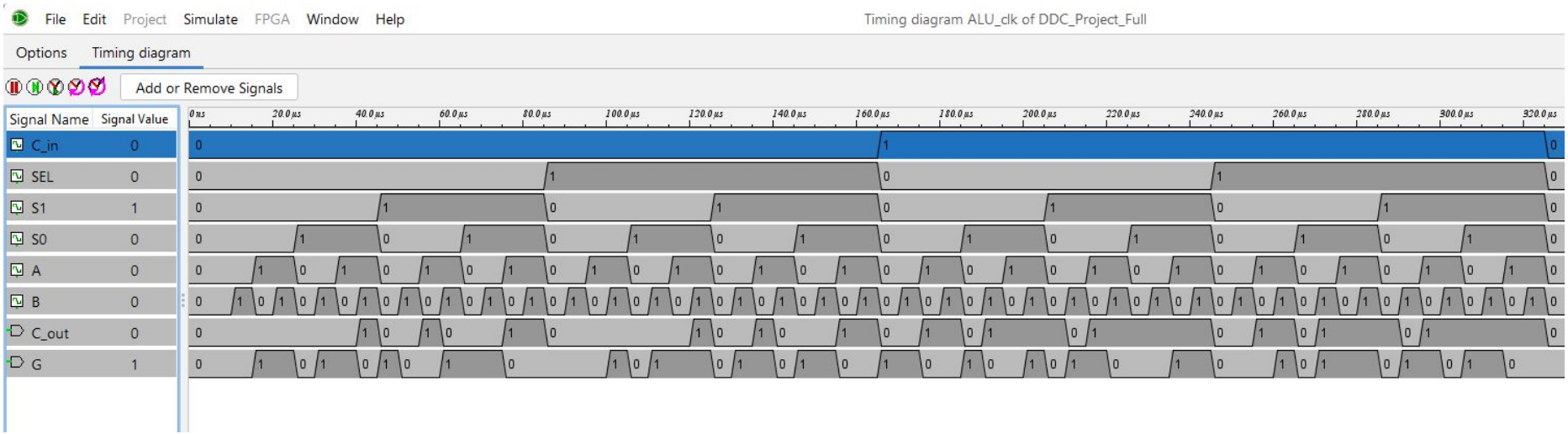


ALU - Test vectors (1 bit)

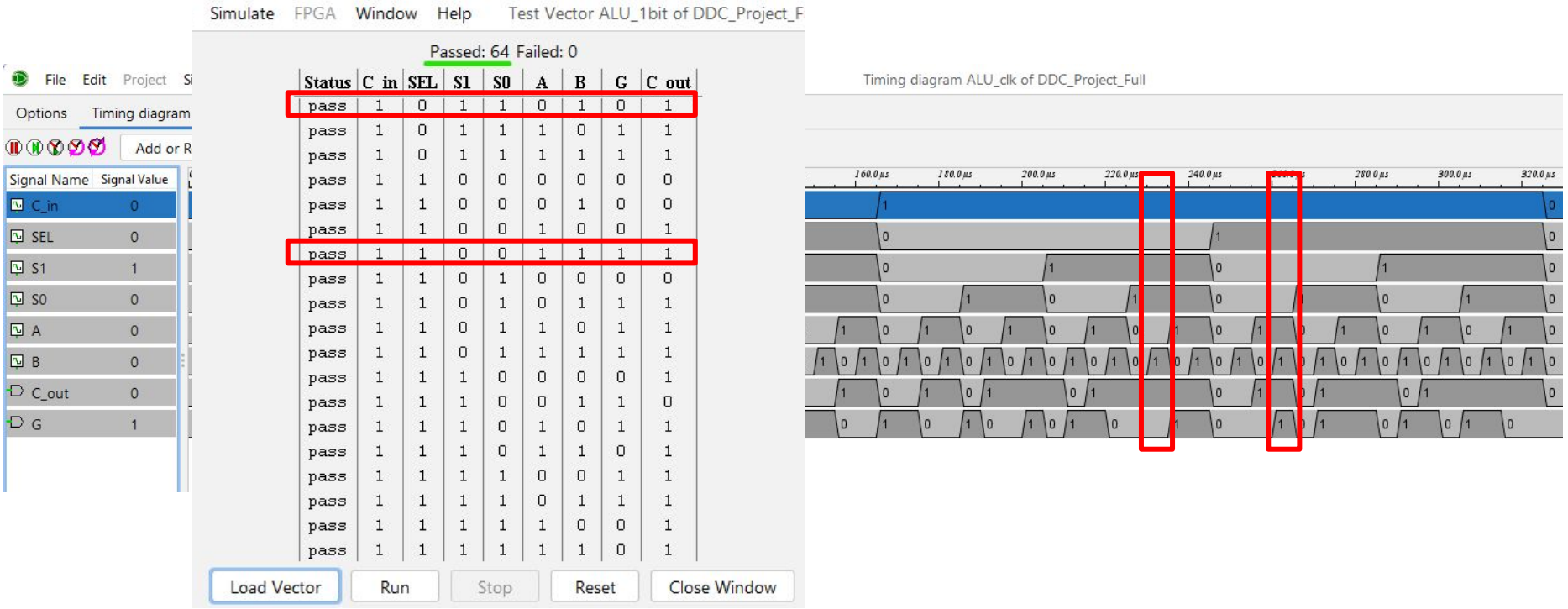


Status	C_in	SEL	S1	S0	A	B	G	C_out
pass	1	0	1	1	0	1	0	1
pass	1	0	1	1	1	0	1	1
pass	1	0	1	1	1	1	1	1
pass	1	1	0	0	0	0	0	0
pass	1	1	0	0	0	1	0	0
pass	1	1	0	0	1	0	0	1
pass	1	1	0	0	1	1	1	1
pass	1	1	0	1	0	0	0	0
pass	1	1	0	1	0	1	1	1
pass	1	1	0	1	1	0	1	1
pass	1	1	0	1	1	1	1	1
pass	1	1	1	0	0	0	0	1
pass	1	1	1	0	0	1	1	0
pass	1	1	1	0	1	0	1	1
pass	1	1	1	0	1	1	0	1
pass	1	1	1	1	0	0	1	1
pass	1	1	1	1	0	1	1	1
pass	1	1	1	1	1	0	0	1
pass	1	1	1	1	1	1	0	1

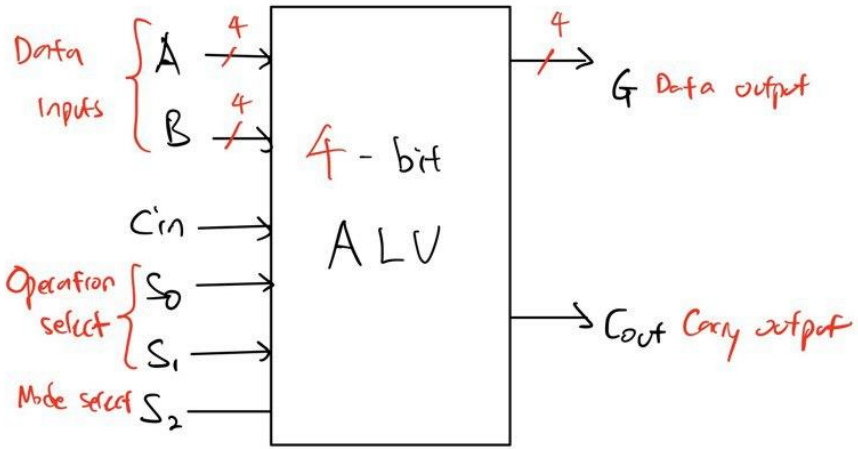
ALU - Chronogram (1 bit)



ALU - Chronogram (1 bit)



ALU - Block diagram (4 bits)



ALU - Logisim (4 bits)