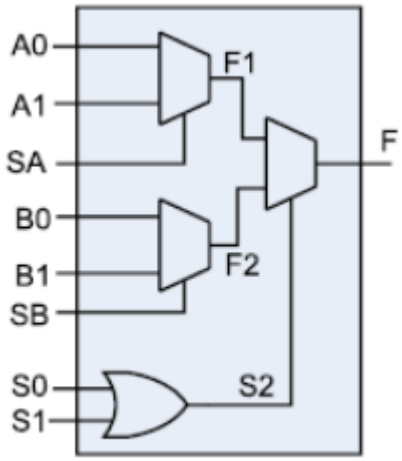
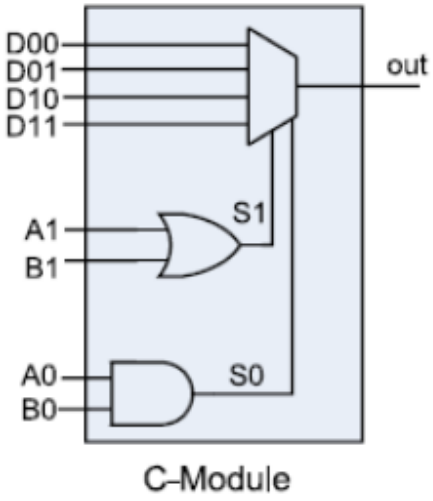
Module C1:



Logic Module

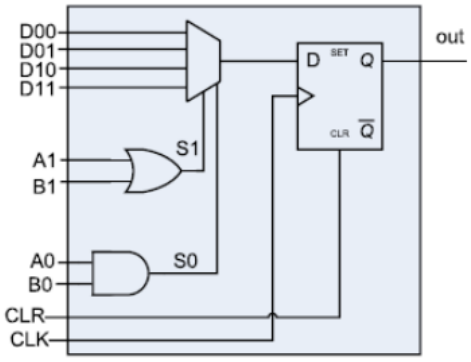
	XOR	AND	OR	NOT
A0	0	0	A	1
A1	1	0	1	1
SA	B	0	B	1
B0	1	0	1	0
B1	0	A	1	0
SB	B	B	1	0
S0	A	C	C	A
S1	0	0	0	0

Module C2:



	MUX2	MUX4
D00	A	A
D01	B	B
D10	-	C
D11	-	D
A1	0	Select[1]
B1	0	0
A0	Select	Select[0]
B0	1	1

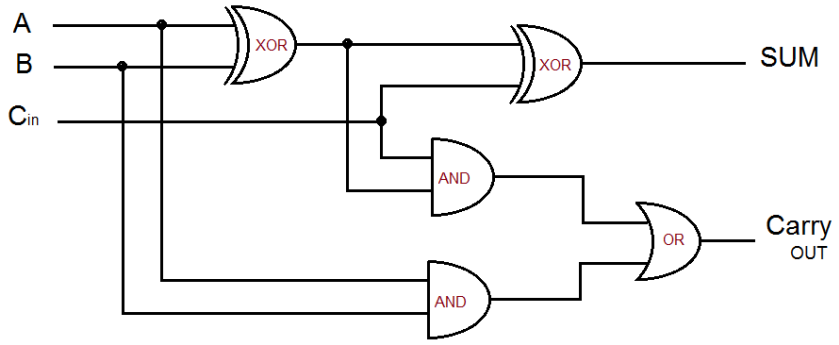
Module S2:



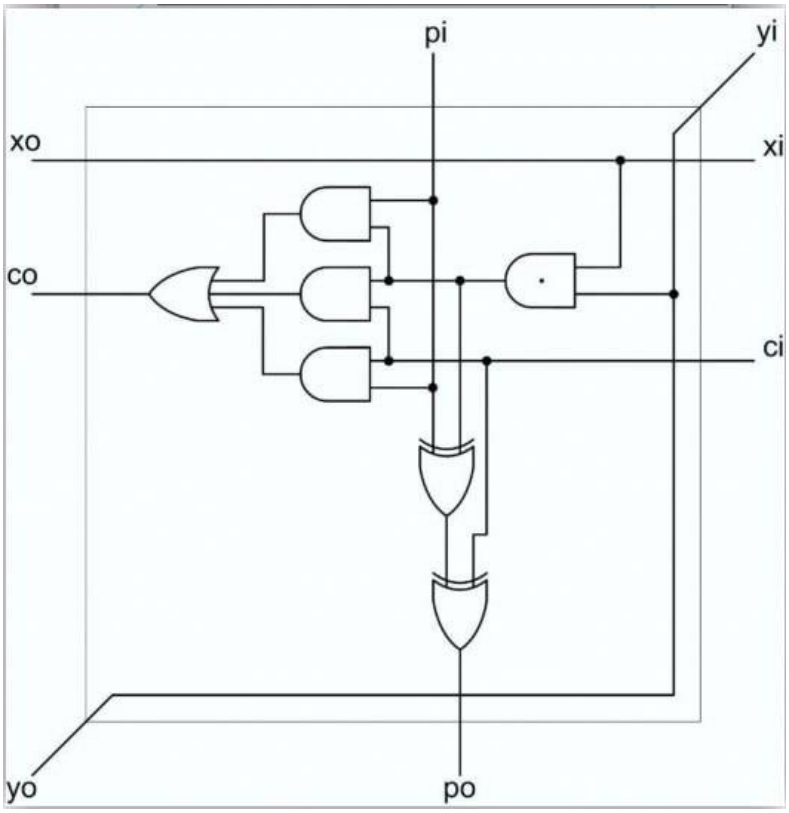
S-Module (Act 3)

	reg
D00	Out
D01	In
D10	-
D11	-
A0	0
A1	0
B0	Enable
B1	Out

Full Adder:

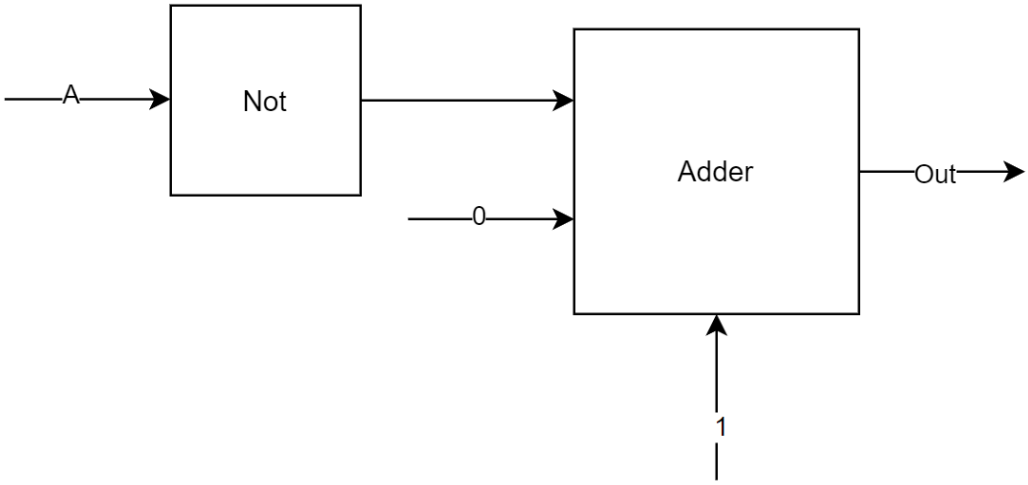


Bit Multiplier:

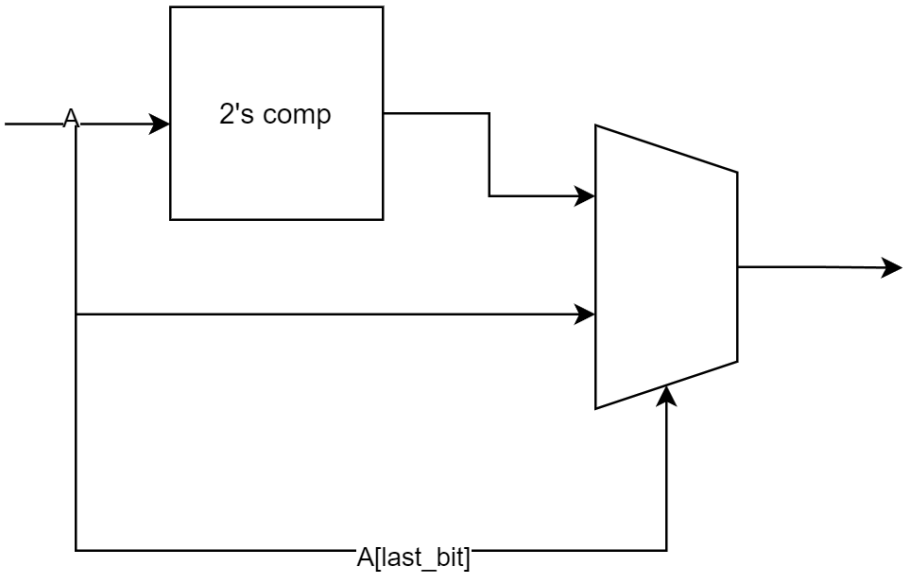


- To implement Adder and Multiplier, we just cascaded the Full adder and bit multiplier respectively.

2’s complement:



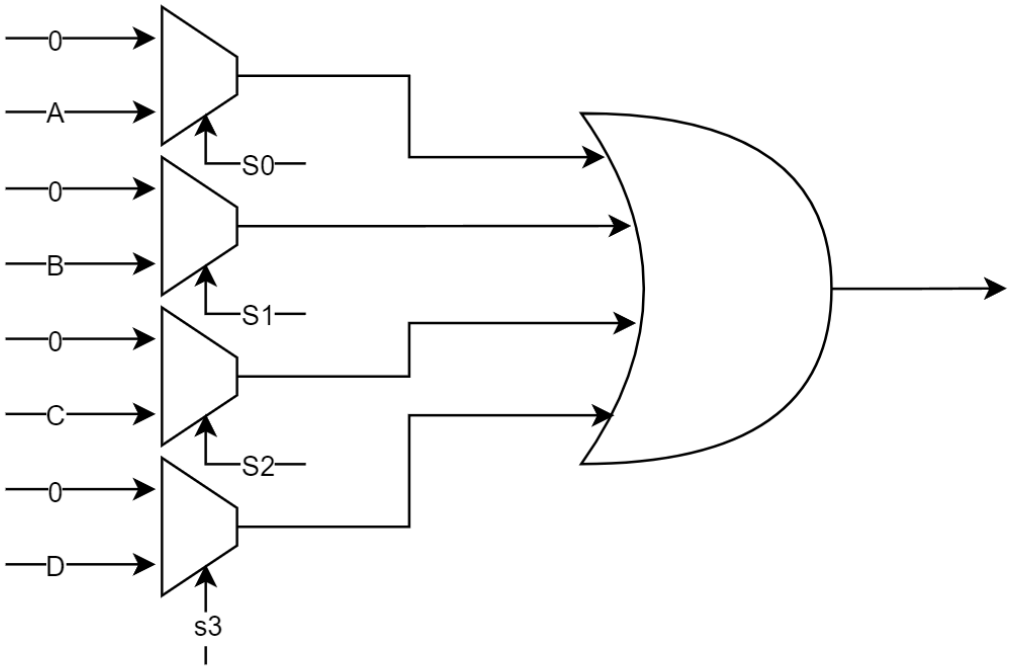
Absolute:



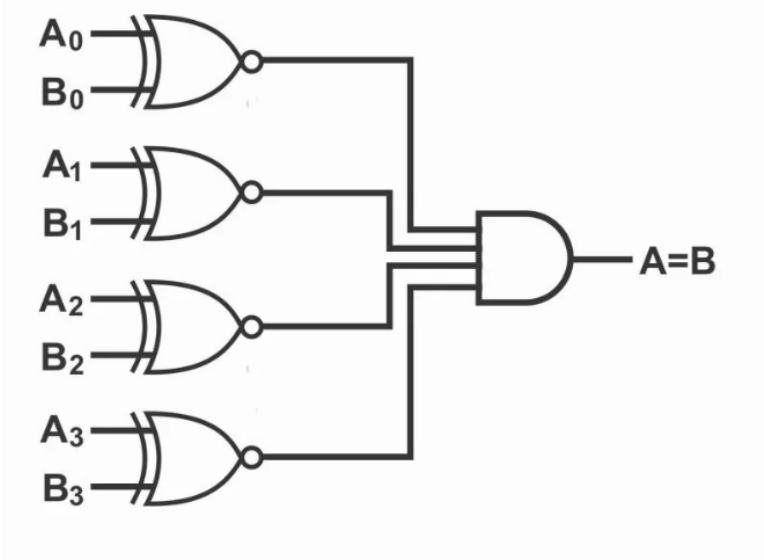
To do multiplication:

- First we multiply absolute of 2 numbers.
- Then according to bit sign, it may be needed to complement result.

One-Hot multiplexer:



Comparator:



Arithmetic:

- Numbers are 5-bit fixed-point floats. 1 sign bit, 1 integer bit and 3 fractional bits.

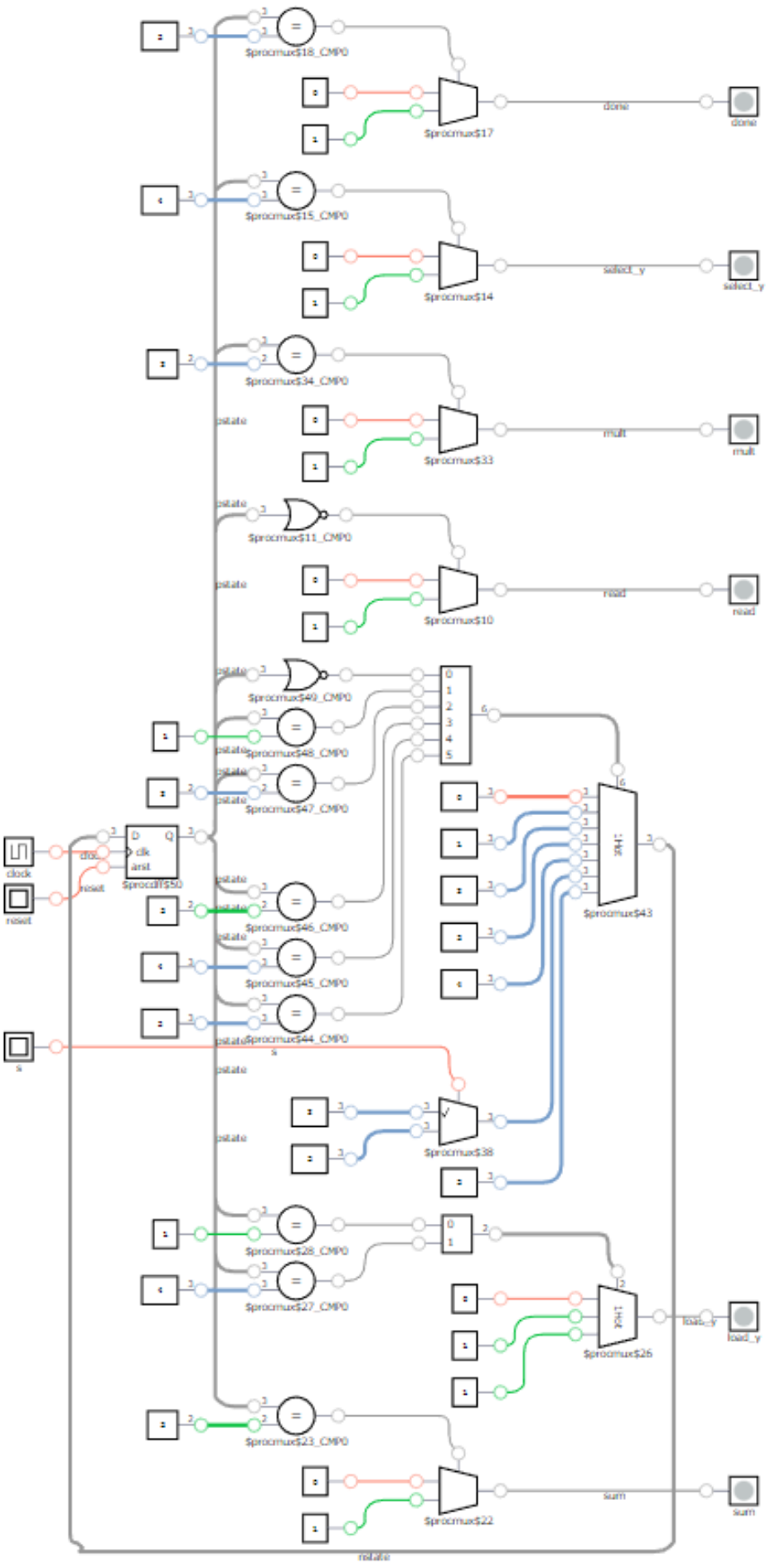
Datapath:

- Same as previous CA, the design is attached to ‘doc’.
- ‘W’ and ‘X’ which are weights and layer inputs respectively will be read from file to increase flexibility.

Controller:

- Controller logic is same as previous CA, but to re-implement using ACT modules, we used these modules:
    - Register
    - Comparator
    - One-Hot Multiplexer

  Where all implementations are mentioned before.
- The diagram can be found in the next page.


$procmux$18_CMP0
$procmux$17
done
$procmux$15_CMP0
$procmux$14
select_y
$procmux$34_CMP0
$procmux$33
mult
$procmux$11_CMP0
$procmux$10
read
$procmux$40_CMP0
$procmux$48_CMP0
$procmux$47_CMP0
$procmux$46_CMP0
$procmux$45_CMP0
$procmux$44_CMP0
$procmux$43
$procmux$38
$procmux$28_CMP0
$procmux$27_CMP0
$procmux$26
load_y
$procmux$23_CMP0
$procmux$22
sum
clock
reset
s
pstate
nstate