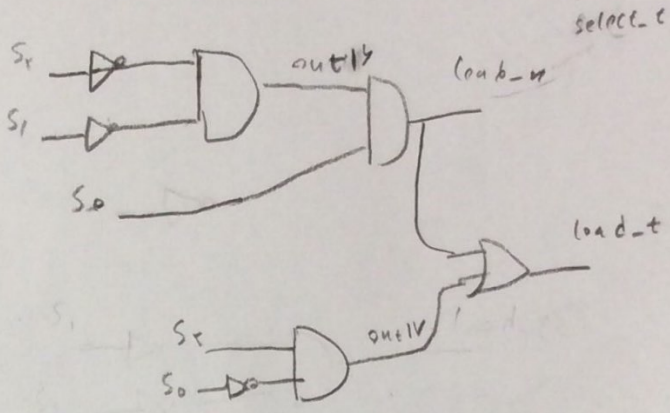
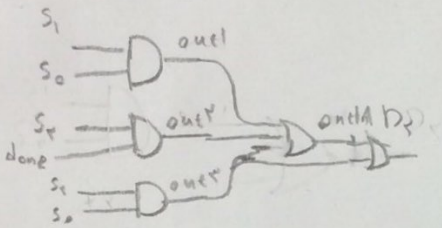
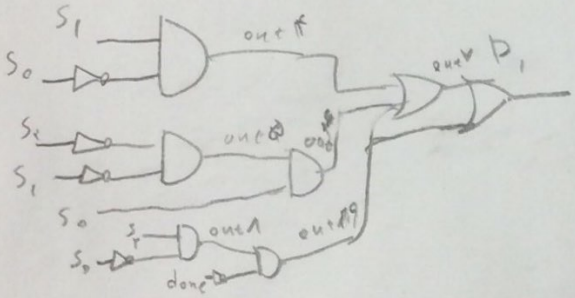
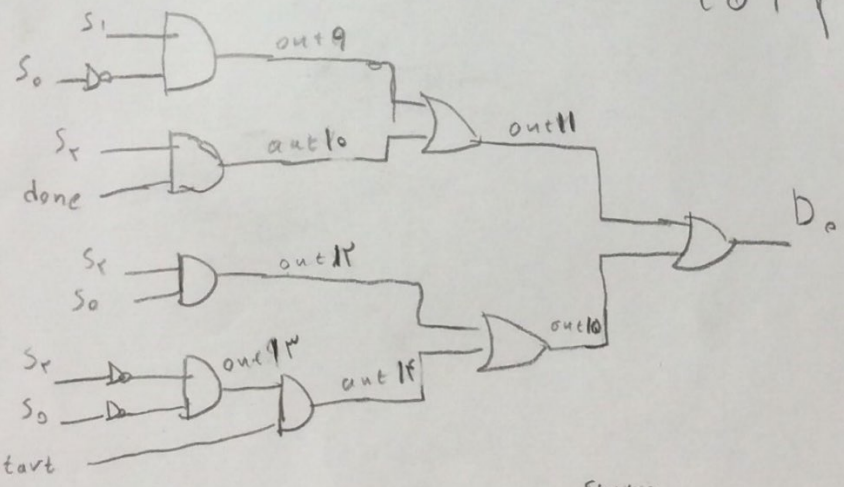
S_2 S_1 S_0	load_w	load_t	select_t
000	0	0	0
001	1	1	1
010	0	0	0
011	0	0	0
100	0	1	0
101	0	0	0

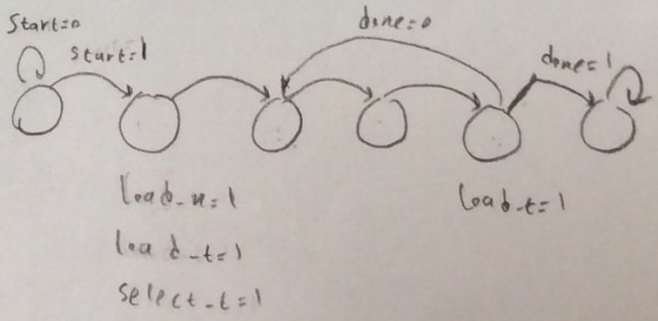






A B C S_2 S_1 S_0	D start	E done	D_2 D_1 D_0
000	0	–	000
000	1	–	001
001	–	–	010
010	–	–	011
011	–	–	100
100	–	0	010
100	–	1	101
101	–	–	101





CAD Project 2

Mohammad Reza Alavi
9810100253

Kasra Noorbakhs
9810100230

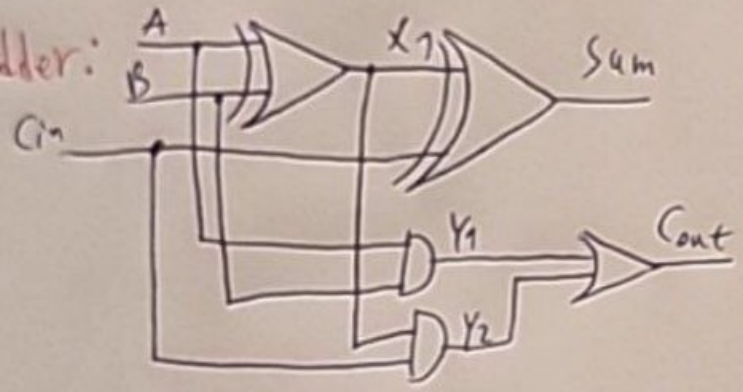
Lets break throw the parts of datapath:

we have 8 5_bit registers with enable that 4 of them are used to be the inputs of PU and 4 of them for final answer. first using Actel modules we wrote the code fo 1-bit register with enable and then casraded it.

we have 4, 5_bit 2to1 MUX's and 1, 5_bit 4to1 MUX.

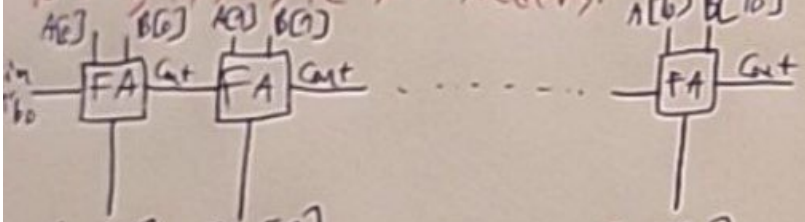
for checking that if a Neuron has berone 0 we have 4, 5_bit Or gates.

Adder:



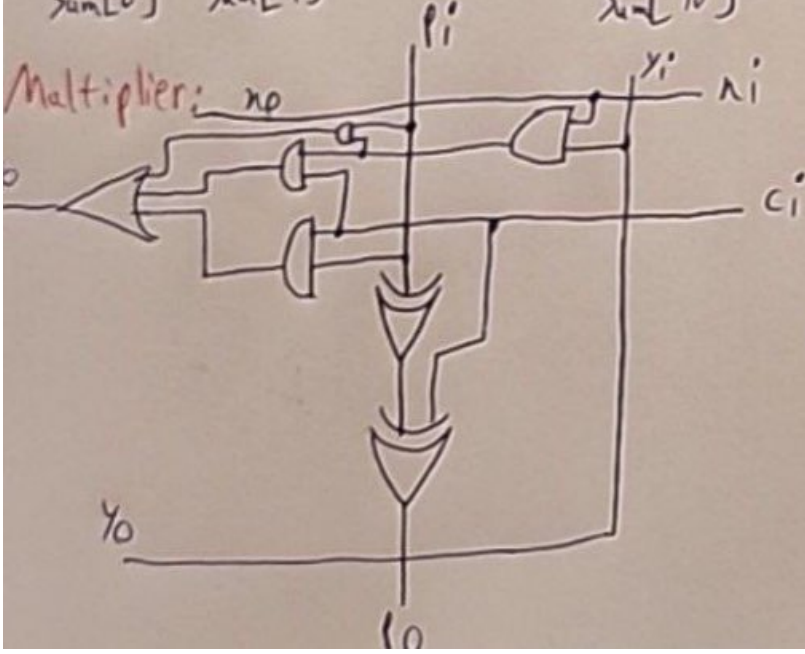
using this logical gates, we implemented a 1-bit full_Adder. we wrote the code of and, or gates with Actel modules and for Xor we used 4 NAND gates.

10 bit, 11 bit, 12 bit Adders:



whenever we needed a 11 or 12 bit adder we casraded our 1 bit full adder to make it!

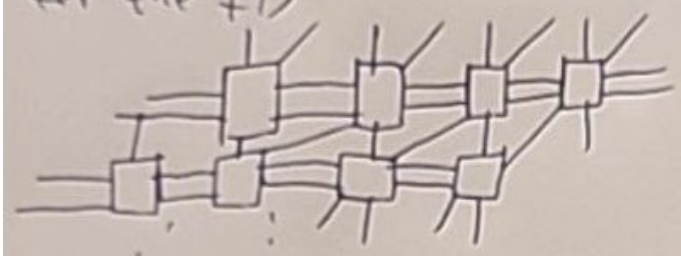
Multiplier:



using this logic for bit by bit Multiplication we wrote down the code for an 5-bit array multiplier.

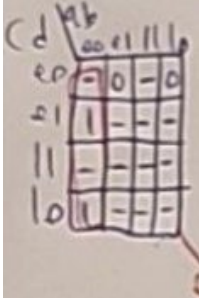
in multiplication we first calculate the 2's complements of our 2 inputs and then we have a Max that decides to be the original input or 2's complement of it

After multiplication, if both inputs were + or −, we just output our result, but if the result were −, we output the 2's complement of the result. (2's complementing is offered by a not gate and an adder for the +1)



the schematic of array multiplier is shown.

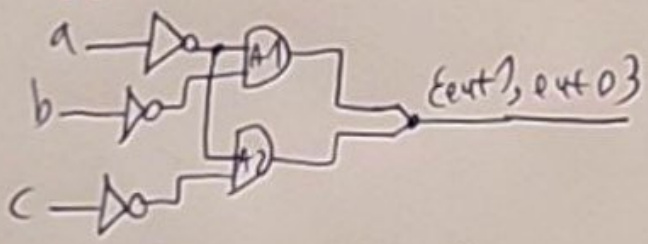
Encoder: this module takes the result of 4-5-bit or gates and calculates the select of our 4 to 1 MUX.



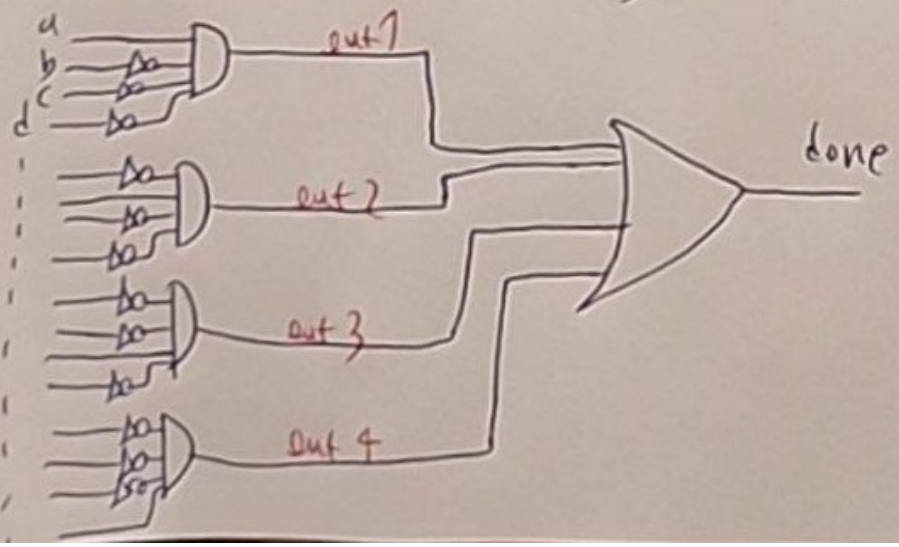
$out1 = \bar{a}\,b$

$out0 = \bar{a}\,\bar{c}$

(we wrote the code for all logical gates with Actel modules.)

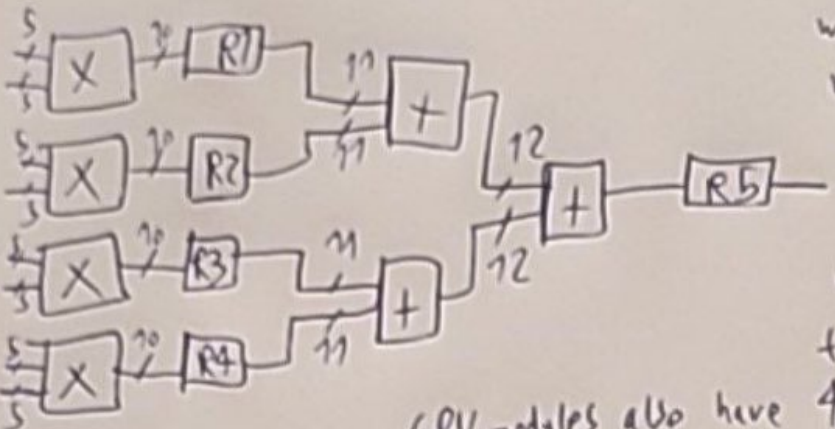


done checker: this module determines that whether the maximum number is find or not using this logic:



Activation: this module uses 5, 1-bit 2to1 MuXs. it works just fine as the project description.

PU:



when we multiply two 5_bit numbers the result is 10-bit. when we want to add these two 10-bit numbers, we fist must sign extend them beraus the Carry_out bit!

(PU modules also have 4 weights input)

data path: