

# ELE523E Computational Nanoelectronics FINAL PROJECT

Deadline: Monday, 07/01/2019, 16:30

*Grading: 1-a)20%, 1-b)20%, 1-c)10%, 2-a)20%, 2-b)10%, 2-c)20%*

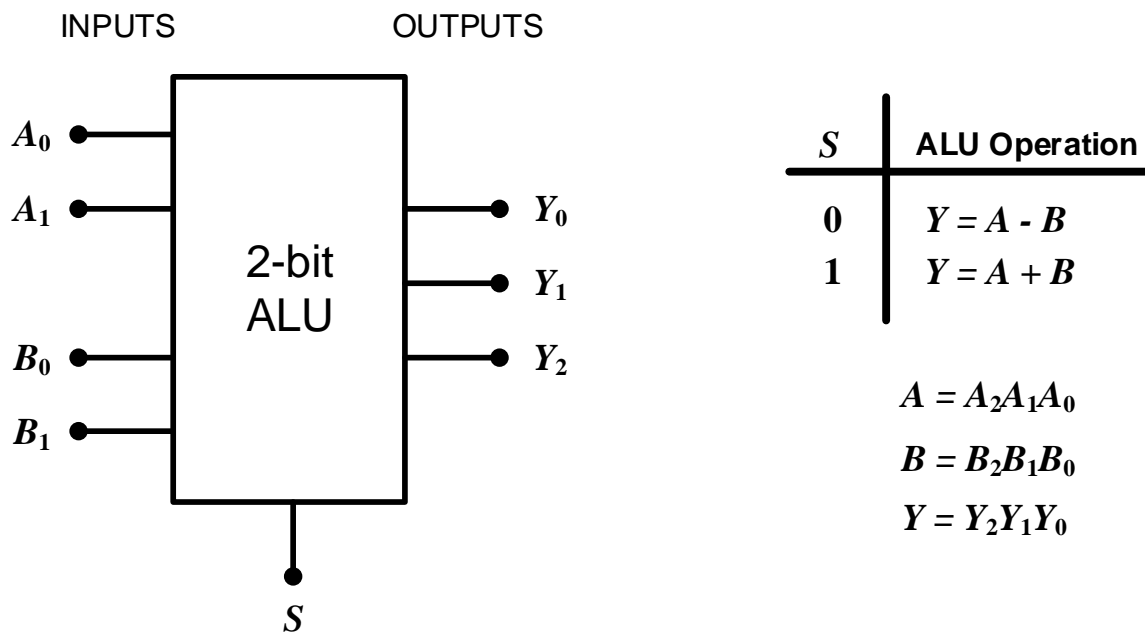
*Return a hard-copy of your final project report*

*Collaboration is not permitted*

*Late submissions will be downgraded*

## 1. DESIGNING A 2-BIT DETERMINISTIC ALU

Consider a 2-bit arithmetic logic unit (ALU) shown below. The ALU works as a 2-bit binary subtractor and adder if  $S=0$  and  $S=1$ , respectively. For subtractor operation  $B \geq A$  is a restricted case that should result in  $Y_2=1$ ; otherwise  $Y_2=0$ . As an example,  $S=0, A=11$  and  $B=01$  results in  $Y=010$ . For adder operation  $Y_2$  is the **carry-out** output. As an example,  $S=1, A=11$  and  $B=01$  results in  $Y=100$ .



*2-bit arithmetic logic unit (ALU)*

In this project, the above ALU is implemented with nano arrays. The ALU **circuit area cost function** is defined as follows: **cost function** =  $25 \times (\text{number of separate nano arrays} - 1) + (\text{number of total crosspoints})$ . For example, if you use 2 separate  $8 \times 5$  nanoarrays, each having 40 crosspoints, then the value of the cost function is  $25 + 80 = 105$ .

- a) Implement the ALU with **reconfigurable two-terminal FET based** nano arrays with **minimizing** the cost function value. You are allowed to directly use variables ( $x$ ) and their negations ( $\bar{x}$ ) as inputs. **Test your circuit implementation** for input assignments ( $S=1, A=11, B=10$ ) and ( $S=0, A=11, B=10$ ); show explicitly that your circuit gives the correct result.
  - Note that pull-up and pull-down networks should be implemented by separate nano arrays. Wires or lines connecting these networks/arrays should be neglected for the cost calculation.

Suppose that each probable FET in each crosspoint, both in pull-up and pull-down networks) has **stuck-at ON fault** with an independent **fault probability of  $\epsilon=0.2$** . Note that a faulty FET has **shorted** source and drain terminals. Suppose that wires or lines connecting the pull-up and pull-down networks are fault-free.

- b) Without increasing the area, develop a reconfiguration-based technique/algorithm to tolerate faults. Try to **maximize the success rate**. Find the accuracy or success rate of your algorithm by generating a fault map and running your algorithm at least 100 times. Also **minimize the runtime**; it should be under 5 minutes.
- c) Use **triple modular redundancy (TMR)** in order to improve your success rate calculated in b). You basically use 3 replicas of the circuit designed in a), and a voter circuit to be designed with a FET based nano array followed by running the algorithm designed in b). Is the improvement satisfactory and as expected? Justify your answer.

## 2. DESIGNING A 4-BIT STOCHASTIC ALU

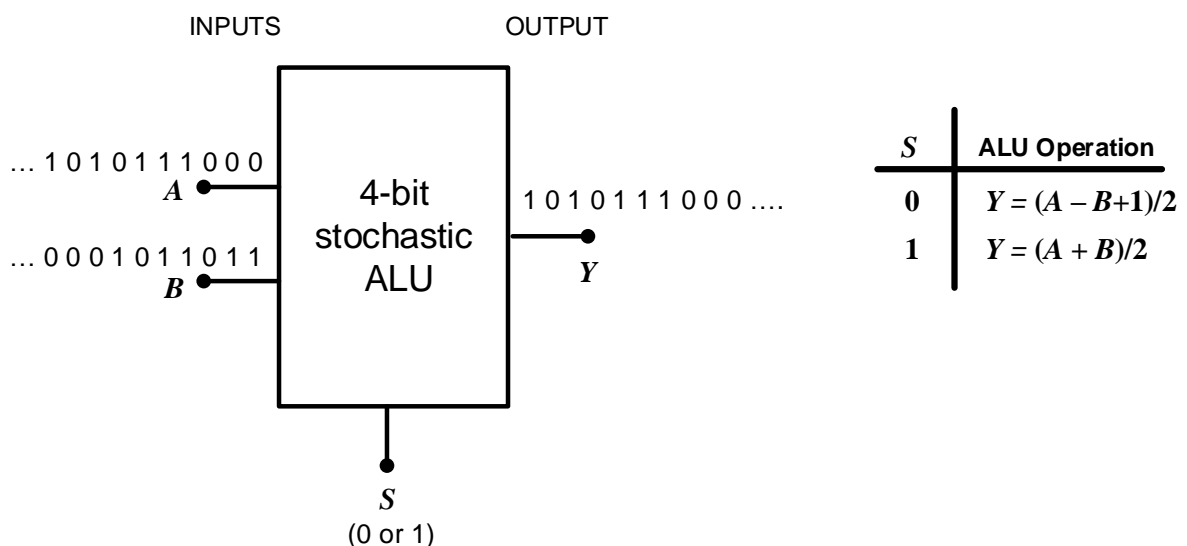
Implement a stochastic 4-bit arithmetic logic unit (ALU). The ALU should have two input bit streams **A** and **B** representing two binary fractions from  $(0.0000)_2 = 0/16$ ,  $(0.0001)_2 = 1/16$ ,  $(0.0010)_2 = 2/16$ , ...,  $(0.1110)_2 = 14/16$ ,  $(0.1111)_2 = 15/16$  such that  $A > B$ . The ALU should have a select input **S** that is 0 or 1, and a output bit stream **Y** representing a binary fraction from  $(0.00000)_2 = 0/32$ ,  $(0.00001)_2 = 1/32$ , ...,  $(0.11101)_2 = 29/32$ ,  $(0.11110)_2 = 30/32$ . ALU's output evaluates **the inputs' average** sum or difference. As an example,  $S=0$ ,  $A=0.1010$  and  $B=0.1001$  results in  $Y=0.00001 = (0.1010 - 0.1001)/2$  for the output. For each of the input assignments, error at the output is defined as follows:

$$\mathbf{Error} = \frac{\mathit{round}(|32 \times (z - z_{exp})|)}{32 \times z_{exp}}$$

where **z** and **z<sub>exp</sub>** represent real and ideal (expected) output probabilities, respectively – “**round**” operation rounds a decimal number to the nearest integer.

For example, if you apply inputs  $0.1011=11/16$ ,  $0.0011=3/16$ , and  $S=1$  to your circuit then it gives you the output probability  $0.45=14.4/32$ . For this case  $z=14.4/32$  and  $z_{exp}=14/32$ , so  $\mathbf{Error} = \frac{\mathit{round}(0.4)}{14} = 0$ .

For the ALU, **the computing time cost function** is defined as follows: cost function = (number of bits in input bit streams) × (number of logic gates). For example, if you use 6 gates and bit streams having 32 bits then the value of the cost function is 192.



Stochastic ALU

- a) Implement the above stochastic ALU with using only **NAND-2**, **NOR-2**, and **NOT** logic gates. You are also **allowed** to use a stochastic input with  $p=1/2$ . For the input assignment  $S=0$ ,  $A=0.1110$ , and  $B=0.0110$ , **minimize the cost function** and find its value while meeting the average error value of **1%** (or less). Note that bit streams are generated using **random shuffling**, so the desired input stream values are always met.

Evaluate your ALU in an image processing application of Sobel edge detector. A renowned image “LENA” is used for evaluations; [click here](#) to download. The image has a size of  $350 \times 275$  and its each pixel has a value between 0 and 255 (binary  $11111111_2$ ).

- b) Apply a **conventional Sobel filter** to the image by directly using the technique [given here](#). Round the final pixel values to the nearest integers. Show the image with edges.
- c) Apply a **stochastic Sobel filter** to the image by using the ALU designed in **a)** for all adding and subtracting operations. For all (+) and (-) operations in the implementation of the filter, you are supposed to the ALU. Suppose that bit streams have  $n=510$  bits. Each pixel has an independent Binomial distribution with a mean as the pixel’s value over 255. Note that  $n$  is the number of trials in binomial distribution. For example,  $n=510$  and a pixel value of 124 correspond to a bit stream such that each of its 510 bits is 1 with a probability  $p$  of  $124/255$ . Note that bit streams are generated using **random assigning**, so the desired input stream values can not be met. Obtain the image and calculate peak signal-to-noise ratio (**PSNR**) in comparison with the image obtained in **b)**. Justify your answer.