**VLSI Project Planning Stage**

PLANNING STAGE We’ve designed our ALU to accept 2 inputs A,B and output Y to fed into 4-bit registers, the ALU designed to implement the 4 operations in groups of : { (XOR) , (ADD, SUB) , (shift) }. Each group is selected by MUX/DEMUX using opcode bits as show in the block diagram. The purpose of splitting it into groups is to minimize the power consumption as far as possible so that modules which are not required for the operation being performed are disabled.

|  |  |
| --- | --- |
| opcode | operation |
| 10XXZ S.T Z IS FEEDBACK ENABLE | **add** |
| 11XXZ S.T Z IS FEEDBACK ENABLE | sub |
| 00SSZ s.t S is number of shifting bits | shift |
| 01XXZ | XOR |

Firstly, the opcode[0]-(Z)- indicates to choose between A or Y (feedback indicator). Opcode[4][3] indicates which operation to implement among (add, sub, shift, xor). When opcode[4][3] is 10 our command is to add between (A/Y and B) when opcode[4][3] is 11, we perform subtraction between (A/Y and B) when opcode[4][3] is 00 our command is to shift (A or Y) and opcode[1][2] (SS) is the number of bits we want to shift , and for opcode[4][3] = 01, our command is xor between (A/Y and B), Notice that for opcode[4][3] = 11, we implement subtraction. So, opcode[4] bit is (add, sub) function enable, and opcode[3] indicates which operation to choose, add or sub. XX bits aren't important.

תמונה שמכילה טקסט, תרשים, תוכנית, שרטוט טכני

התיאור נוצר באופן אוטומטי

As we can observe, A and Y serve as inputs to the upper right side of the diagram. As explained earlier, opcode[0] selects between them. For implementing feedback, opcode[0] is set to 1, and for another mode, it's set to 0. In the upper left side, B is stored in a register, then inserted as input to the DEMUX. The left and righ DEMUXes enables navigation of the signal B and (A/Y) to ADD/SUB/BSR/XOR.

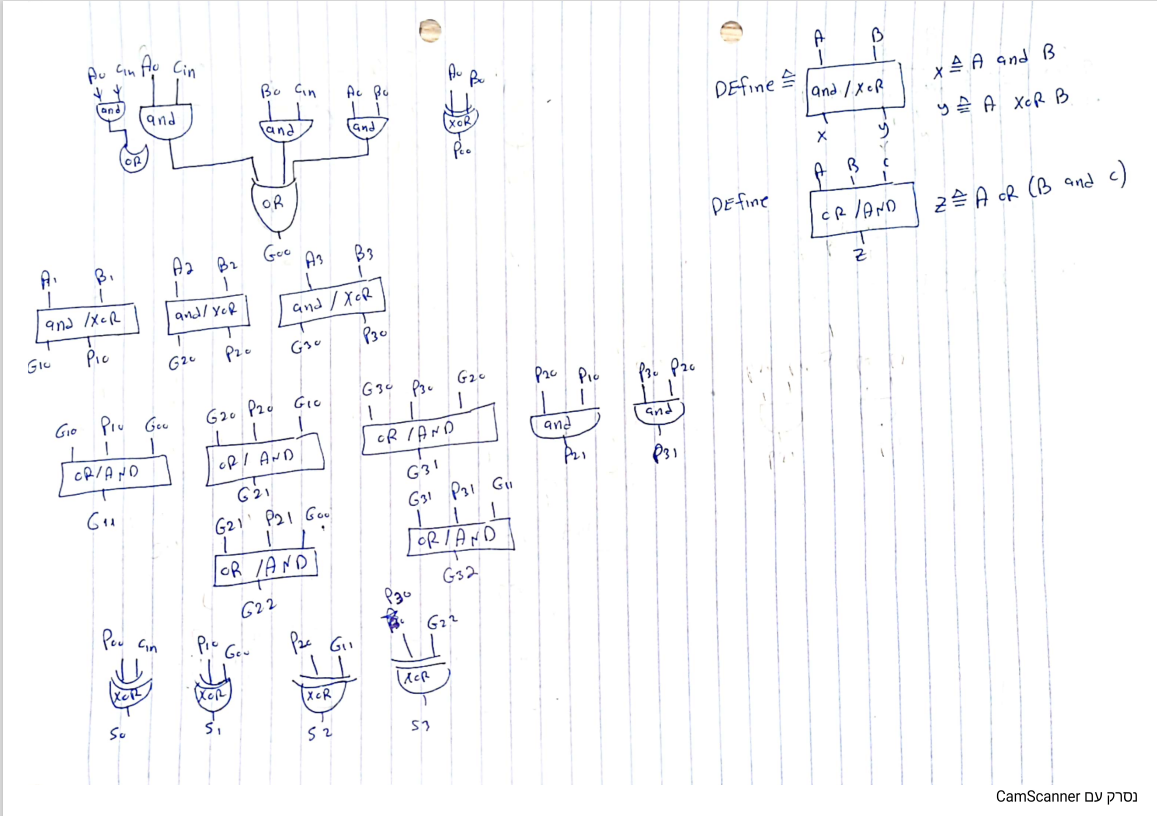
Let's begin with the ADD-SUB operation. Starting from the left DEMUX, op[4] indicates whether we're implementing ADD-SUB. If op[4] is high, B moves to the ADD/SUB block. On the other side, A/Y signal flows through the upper right DEMUX. If op[4] is high, then A moves to ADD/SUB. After that, the output of ADD/SUB goes as input to the mux. Note that opcode[4][3] is 10 for addition and 11 for subtraction. After constructing a block that implements addition or subtraction according to op[3], the output is either the sum of A and B or their subtraction. Inserting it into the 3rd and 2nd mux inputs doesn’t affect the functionality of the mux. Op[4:3] serve as the mux selector, and for 11 or 10, it selects the ADD-SUB gate.

For XOR operation, when opcode[4:3] = 01, XOR is implemented. Starting from the upper DEMUX, when it's a XOR operation, op[4] is zero, then B flows until it reaches the buffer. On the other side, (A/Y) flows to the lower right DEMUX. If opcode[3] = 1 and opcode[4] = 0 indicating the XOR command, then the lower right DEMUX enables A/Y to flow through the XOR gate. Since opcode[3] is 1, B flows through the buffer, and both signals A/Y and B are inserted as input to the XOR gate. The output of the XOR gate is inserted as input to the mux, and for opcode 01, the mux selector is 01, and Y equals B XOR (A/Y).

For the shift operation, opcode[3:4] should be 00, and the shift input is (A/Y). When opcode[4] and opcode[3] are both zeros, then A/Y flows through the right DEMUXes and reaches BSR (block for implementing shift operation). As explained earlier, opcode[1:2] represent the number of bits to shift, serving as input to BSR. After that, the output of BSR is inserted into the mux as input, and for opcode[00], the output Y represents the shift result. Note that Y is connected in a feedback loop to the input A.

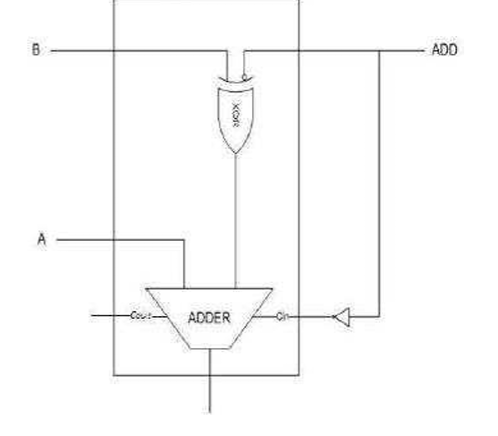
Adder\_Sub Designing

First of all, we designed an ADDER gate where A[3:0], B[3:0], and cin are its inputs, and S[3:0] is the output. After completing this section, we will proceed to implement the SUB operation based on this adder.





To implement the SUBB gate, we first have to calculate the 1's complement of B and then add 1 to it. After that, we will add A to (-B) using the attached adder gate. Therefore, our approach to implementing the SUBB gate is as follows:



Notice ADD is the bit which indicates us to implement addition so if add is 1 then our implementation is adding and when its 0 our implement is sub , assume add is 1 then the input of xor gate is (B,0) and the output of the xor gate is B the cin in this case is zero and the adder gate implement sum A+B (we don’t care about Cout) , if ADD is 0 then input of xor gate is (B,1) ) and the output of the xor gate is B\* the cin in this case is 1 because of the inverter and the adder gate implement sum A+(\*B+1) which is A + (-B) , notice thar cout in our adder design is G32