THE FLOOR PLAN

Block Level Logical Diagram:

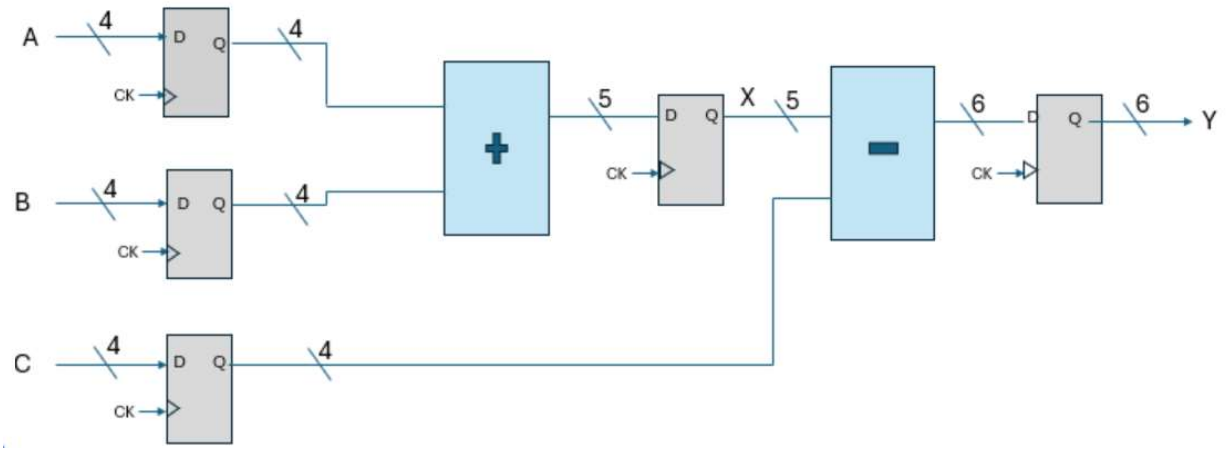


Figure 1

ALU Implementation:

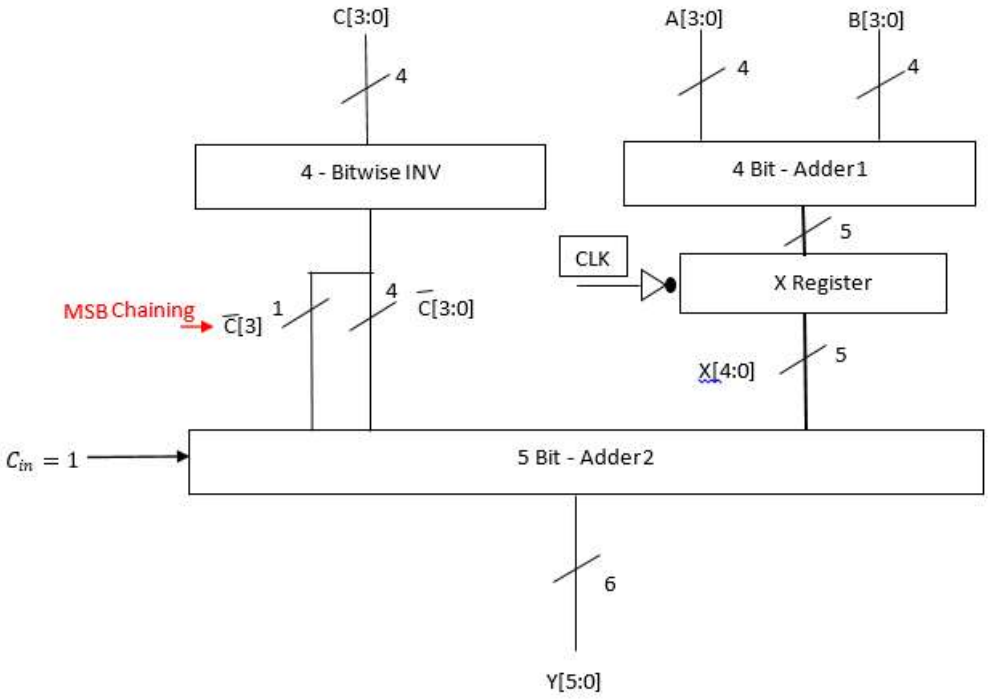


Figure 2

Bitwise 4 - bit inversion implementation:

We used the bitwise inverter to invert the C input bits:

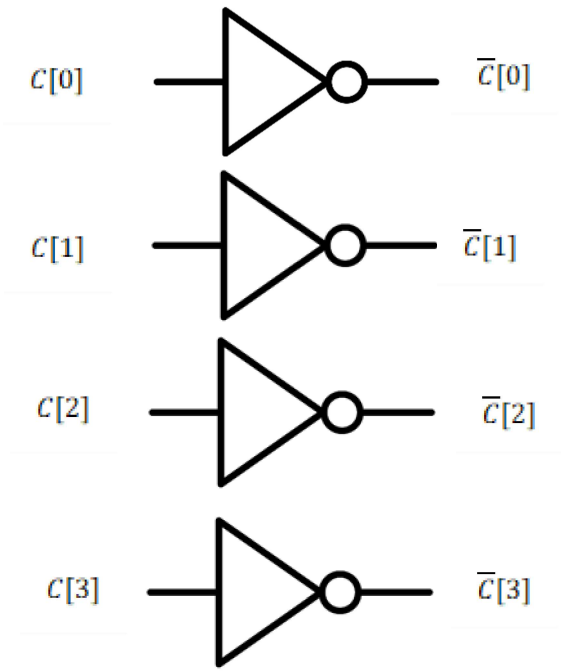
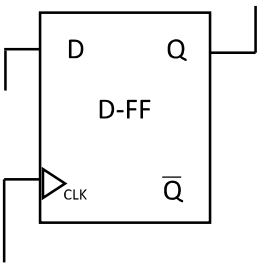


Figure 3

The Registers Implementations:

For register implementation we will use the D-FF cell



- 4 bit register implementation for inputs A ,B&C:

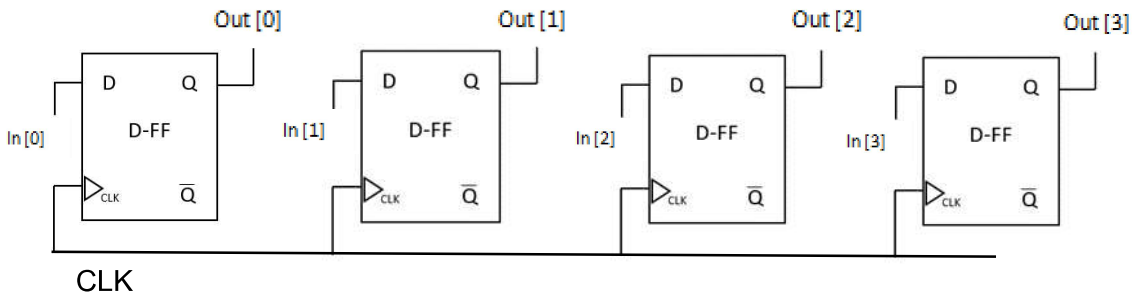


Figure 4

- 5-bit register implementation for X (the output of Adder 1)

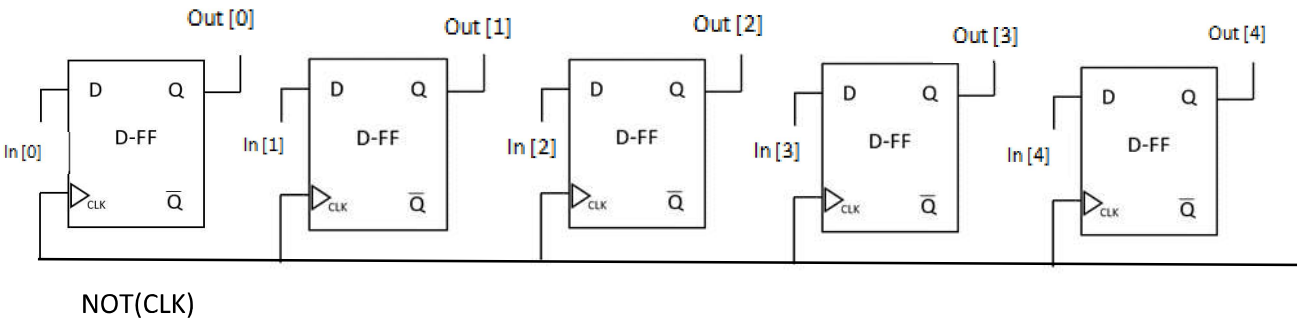


Figure 5

❖ 6-bit register implementation for the output Y

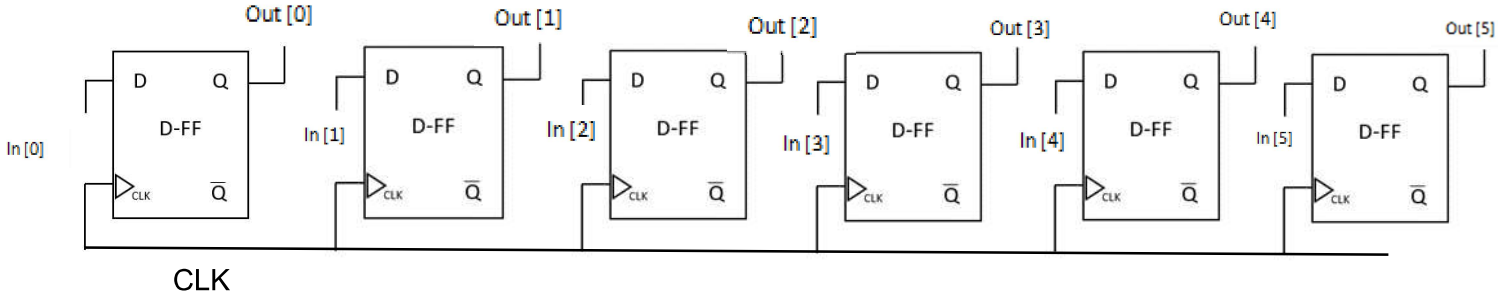
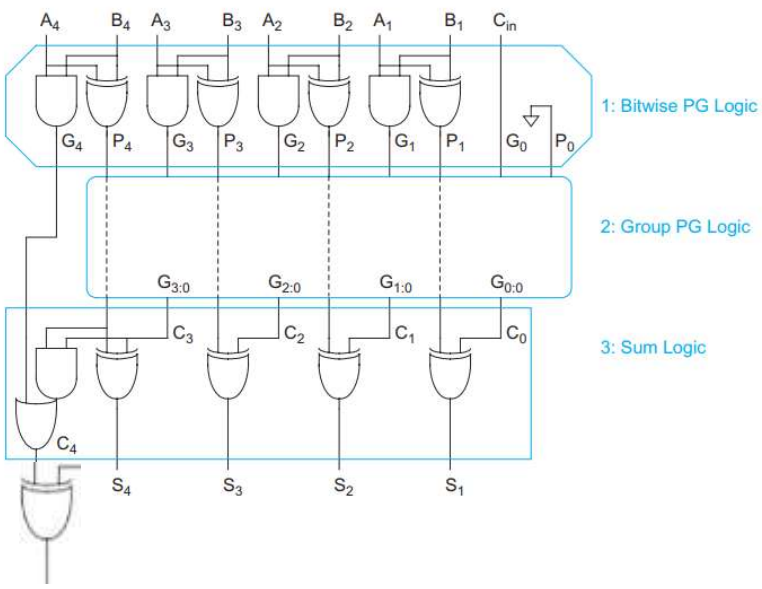


Figure 6

Adders Implementation: Kogge-Stone adder

❖ Full logical diagram of the 4- bit adder (Adder 1) & 5- bit adder (Adder 2)



Cout

Figure 7 – 4- bit adder

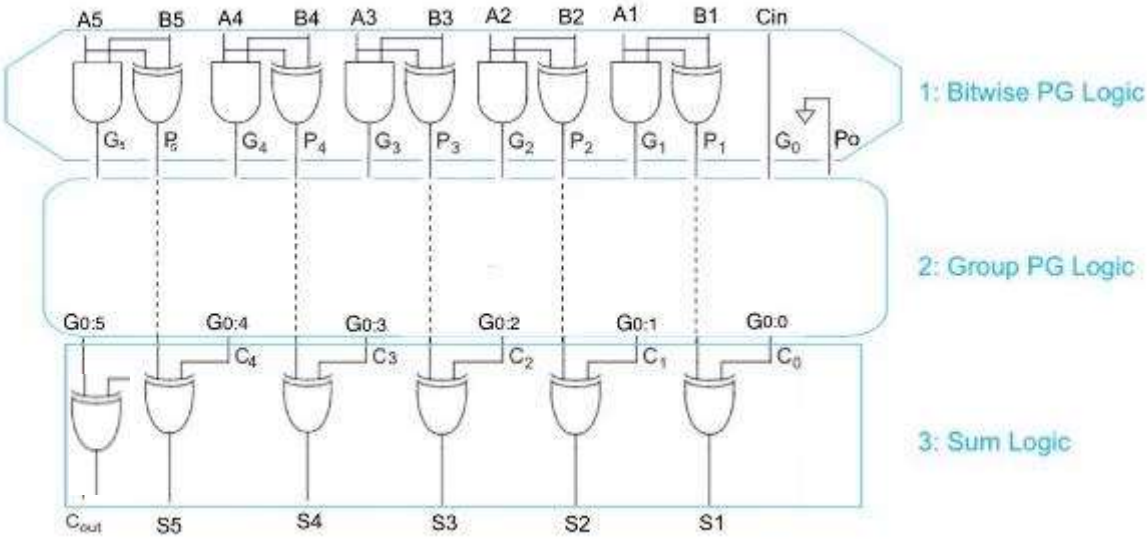


Figure 8 - 5-bit adder

❖ The tree adder PG network for 4-bit (PG logic)

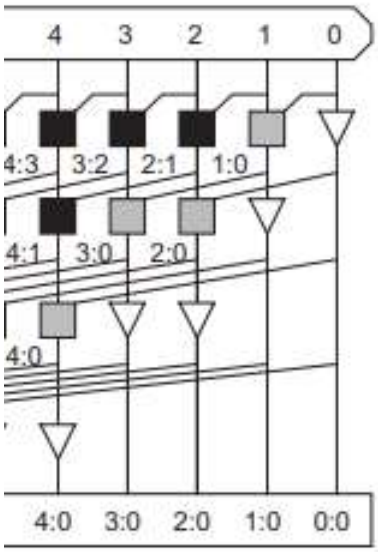


Figure 9

❖ The tree adder PG network for 5-bit (PG logic)

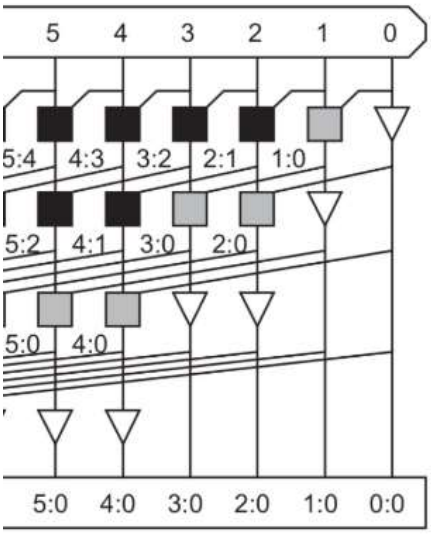


Figure 10

The black, Grey cells and the Buffer implementation:

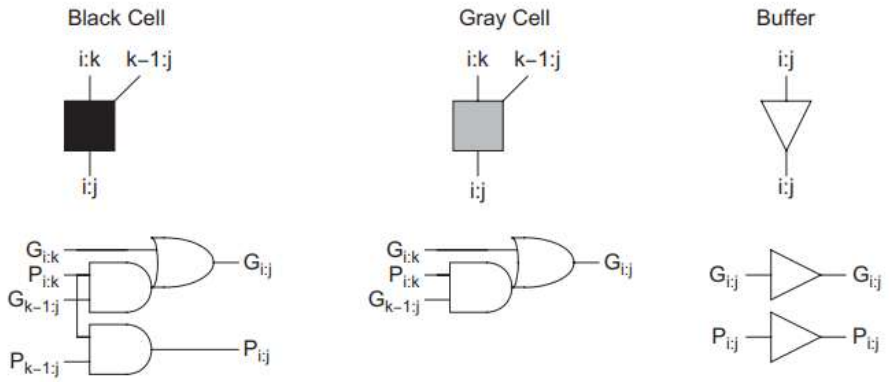


Figure 11

As we can see the inputs of the kogge stone adder are P and G (propagate, generate), so we are getting P and G from the bitwise PG logic stage as shown in figures

The Workflow of the ALU:

The inputs of the ALU are fed from three registers that store 4 bit inputs A, B and C.

Our goal is to compute ⟨Y⟩ = ⟨A⟩+⟨B⟩ −⟨C⟩, while ⟨Y⟩, ⟨A⟩, ⟨B⟩ and ⟨C⟩ are the two's complement representation of Y, A, B, and C respectfully.

The first adder (Adder 1) computes the sum of A [3:0] and B [3:0] strings, resulting in a 5-bit output X [4:0] which will be stored in a 5-bit low edge sensitive register. In the next stage we want to compute subtraction between X and C, thus we want to represent –C in two's complement representation and then apply ⟨X⟩ +⟨-C⟩ using 5-bit adder (Adder2).

In order to get ⟨-C⟩ we need to invert the bits of C and add 1 to the result, we will do that by applying a bitwise inversion on C [3:0], and then appending the MSB of the inversion result to the result in order to achieve a 5-bit length string ($\bar{C}[3]°\bar{C}\ [3:0]$) while saving the strings sign, so now we can fed ($\bar{C}[3]°\bar{C}\ [3:0]$) and X[4:0] into a 5-bit adder (Adder 2). The carry-in of Adder 2 is set to 1 to account for the +1 required when computing the two's complement to represent ⟨-C⟩. The result of the second adder is a 6-bit output Y [5:0] in two's complement representation (⟨Y⟩ =⟨X⟩ -⟨C⟩).

- We used low edge sensitive register for X in order to get a correct timing, in other words, to get the correct Y in the next high edge of the clock.

Block Level Floor Plan:

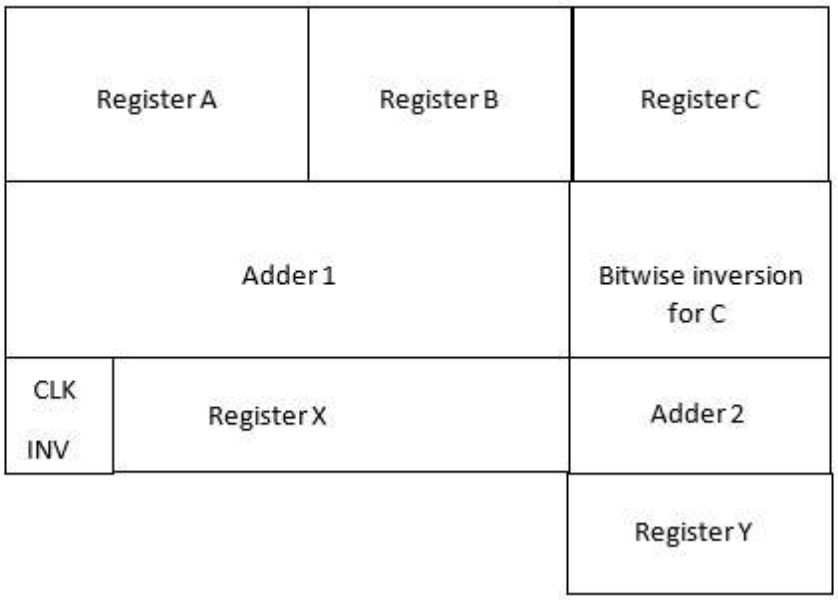


Figure 12

The number of cells:

- 3 x 4-bit register = 12 cells.
- 1 x 5-bit register =10 cells.
- 1 x 6-bit register = 6 cells.
- 1 x bitwise inverter = 4 cells.
- 1 x inverter = 1 cell.
- Adder 1 = 5 x buffer + 4 x gray cell + 4 x black cell = 5 + 4 x 2 + 4 x 3 = 25 cells.
- Adder 2 = 6 x buffer + 5 x gray cell + 6 x black cell = 6 + 5 x 2 + 6 x 3 = 34 cells.
- Bitwise PG logic (for Adder 1) = 8 cells.
- Sum logic (for Adder 1) = 4 cells.
- Bitwise PG logic (for Adder 2) = 10 cells.
- Sum logic (for Adder 2) = 5 cells.

→ We estimated in total 119 cells/components from gsclib045 that we are going to use in the ALU.

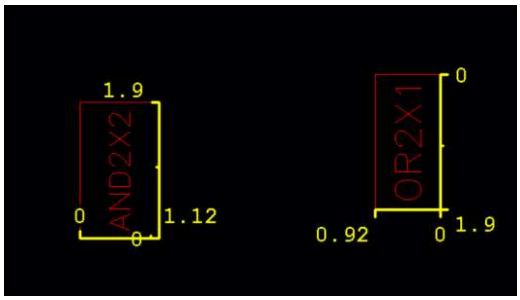
Component	Area $(um)^2$
AND2X2	2.128
OR2X1	1.748
XOR2X1	3.268
BUFX2	2.128
INVX1	0.988
DFFHQX1	6.308

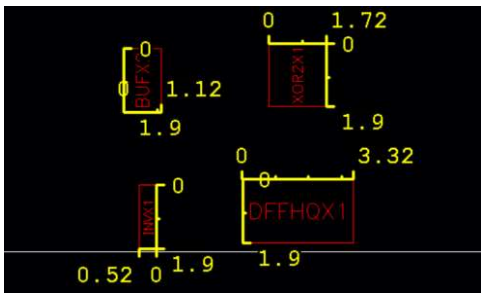
Sizes of used components:

- We measured the area using the detentions in layout XL, as shown in the images above.

Based on the number of cells and the sizes of each cell, we estimated the required area for a layout:

Size (23 x DFFHQX1 + 5 x INVX1 + 11 x BUFX2 +38 x AND2X2 + 19 x OR2X1 + 20 x XOR2X1) = 352.868[$(um)^2$]


1.9
AND2X2
0
1.12
0
0
OR2X1
0.92
0
1.9


0
BUFX
0
1.12
1.9
0
1.72
0
XOR2X1
1.9
0
3.32
0
DFFHQX1
1.9
0
INVX1
0.52
0
1.9