Name :
TIREL Nicolas
X1085033

Team member :
VENANT-VALERY Thomas
X1085032

<u>FPGA Mapping Project</u>

1. Compilation and execution:

Our project is build with a makefile, so the only commands you need to execute to compile is:
make
in the src repository, and to execute:
./main ../testcases/fileName.aag K path/to/file.out

2. Final depth toggling rate and runtime:

All the following tests are computed with a 3-LUT:
alu4.aag: final depth: 14 - toggling rate: 2763.33 - runtime: 65 ms
bigkey.aag: final depth: 10 - toggling rate: 3138.45 - runtime: 76 ms
c1908.aag: final depth: 32 - toggling rate: 610.36 - runtime: 13 ms
c5315.aag: final depth: 38 - toggling rate: 2028.63 - runtime: 51 ms
sample.aag: final depth: 4 - toggling rate: 9.26 - runtime: 1 ms

3. Algorithm flow and explanation:

We tried to implement two personal algorithms before the final algorithm:
- For the first one, the algorithm flow is designed in two parts:
On the first part, we test if we can connect directly the fanins of the output with the first inputs, than means that there are less parents than the size of the LUT we want. And after that, with all of the nodes remaining, we select which one can be used as a LUT to complete the big LUT containing the final outputs. This solution works well with a sample, but only with 3 LUT, and only with the 'sample.aag' example, so we tried another solution.
-For the second one, the algorithm flow is again separated into two parts:
As the previous implementation, we try first to make a big LUT with the fanin of the output with the firs inputs. If we can't, then we try to merge as much nodes as we can into smaller LUT to make the big merge possible. This solution was really hard to implement in the C language because we needed a lot of lists and we needed to be very careful when we managed the storage of these lists. Finally, we figure out that the output wasn't a feasible solution when testing the 'verifier' executable file.

After those two tries which are in commentary in the 'mapping.c' code, we decided to implement a new solution that will work with any samples we want, and with any size of LUT, but we know that it will also not be the most optimized solution. We decided to consider every nodes as a 2-LUT, and we write it in the output file. The algorithm flow is described as the following: we have a file called 'mapping.h' which contains all of the structures, and all of the functions of our program. We have two big structures called DAG and LUT which contains the representation of a directed acyclic graph with all the internal nodes, input nodes and output nodes, and the LUT stores the final LUT to write in the output file. To store the internal nodes, input nodes and output nodes, we make 3 strucutres called PI, PO and Nodes which contain the ID of each node, and also the rate if it's a input or a internal node, and the fanin(s) if it's an output or an internal node. The file 'mapping.h' contains also a declaration of all the functions that are more described in the 'mapping.c' file. The 'mapping.c' contains all the code for our different functions. And finally the 'main.c' used the parameters and the implemented functions to read the file, compute a feasible solution and write the output in a different file.

4. List of tasks performed by each team member:

Thomas and I thought about the two personal algorithms, Thomas was in charge to find the best algorithm and to write it with pseudo code, and I was in charge to transform this using the C programming language.