Low-Power Contest 22/23: Dual Vth Optimization Algorithm

Members: Matteo **Fragassi** (s317636), Davide **Giuffrida** (s310265), Diego **Porto** (s313169).

The main idea behind the optimization process is the one to swap all the cells to HVT at the beginning, sort them through a metric, swap some of them at LVT (the upper half of the cell list, the one with the highest values for the metric) and check if constraints are met: if true, try to reduce the cells swapped to LVT, otherwise select additional cells to be swapped to HVT and repeat the check in both cases.

In our case the selection of the cells is performed by starting with half of the cells swapped to LVT, then moving with a step that is a geometric progression from there, depending on constraints being met or not: if they are met, we reduce the LVT cells to $\frac{1}{4}$ of the total, otherwise we increase them to $\frac{3}{4}$ and we go on in this fashion also in the following steps. Starting from this idea, we devised an algorithm to be implemented through the *dualVth* function that can be described in brief terms in the following way:

1. We swap all the cells to HVT and we do a first $constraints\_met$ check. In all the $constraints\_met$ checks we also compute the list of cells in the worst path (min slack) for each cell.
2. We compute for each cell a metric p, defined as follows:
   $p = weight_{deltaL}^{4} \cdot \frac{max_{deltaL}}{deltaL} + 1.2 \cdot \frac{max_{FEC}}{FEC}$ when $slack \geq 0$
   $p = -0.4 \cdot \frac{slack}{min_{slack}} \cdot \left(\frac{max_{deltaL}}{deltaL}\right)^{weigh\ deltaL} + 1.2 \cdot \frac{max_{FEC}}{FEC} + (1.1^{iter})$ when $slack < 0$.
   Where:
   - $deltaL$ is the delta leakage of the cell;
   - FEC is the Fanout Endpoint Cost for the cell computed in the previous execution of the $constraints\_met$ function;
   - $slack$ is the slack of the worst path through the cell, $iter$ is the current iteration count (starting from 0, increased after a $constraints\_met$ check);
   - $weight_{deltaL}$ is defined as: $\frac{\sum_{i=0}^{n_{cells\ still\ to\ HVT}} -deltaL_i \cdot slack_i}{\sum_{i=0}^{n_{cells\ still\ to\ HVT}} max_{deltaL} \cdot min_{slack}}$ considering only the cells that are still HVT (right side of the array) with a negative slack for the worst path (the sign of min_{slack} is changed to positive before being used for the formulas: that's why we don't have minus at the denominator).
3. For each cell, we check the list of cells in the worst path computed in the previous $constraints\_met$ check and we decrease the value of the metric p computed before according to the following formula:
   $p' = p \cdot \frac{7}{10}$ if the slack for the cell is positive;
   $p' = -p \cdot \frac{slack}{min_{slack}}$ if the slack is negative.
   This is done to avoid swapping to LVT too many cells from critical paths: indeed if we are trying to swap some cells, the value for the metric of the cells on the same path is going to decrease by a quantity that depends on the slack (if we are on a path with a very low slack, the p will decrease by a small quantity because it may be necessary to take some other cells from that path to make the slack reach 0).
4. We sort the cells depending on the values of p (updated after the previous step): the sort is descending, so we will swap to LVT only the cells with the highest value for the metric.
5. Depending on the previous outcome of the $constraints\_met$ call perform one of the following (to handle the binary search described in the summary at the beginning):
   - at the beginning of the execution: swap to LVT the higher half of the sorted array of cells
   - if previously constraints were <u>not</u> met: swap to LVT a number of cells corresponding to the current step (if after taking half of the cells the constraints are not met, we recompute p for the remaining half of the cells, we re-sort them and we swap to LVT another quarter of the total, thus arriving at $\frac{3}{4}$ of cells swapped to LVT)
   - if previously constraints were met: swap to HVT a number of cells corresponding to the current step (if we do as before and constraints are met, swap back to HVT the part from $\frac{1}{4}$ to $\frac{1}{2}$ of the array)
6. Do the $constraints\_met$ call over the new array with the new configuration of cells defined in the previous point.
7. If constraints are met, save the current configuration of cells in two arrays: one for the names of LVT cells, the other one for HVTs. So that it is ready to be loaded back at the end if we end up with an invalid configuration when the step reaches 0.
8. Reduce the step by half and go back to point 2, until the step becomes 0.
9. Explore some additional cells to compensate for rounding errors in the divisions by two: this is critical if we don't find any valid solution with the search algorithm implemented before, because we need to also explore the remaining cells to swap everything to LVT if required.