Low power contest – Description of the script

Group 20: 231676 FORNO, 231562 JIANG, 230073 MONTISCI.

First of all, we verify that the arrival time of the all-LVT circuit is below or equal the user-defined constraint: if it is not, the script cannot proceed, because swapping cells to HVT can only increase it. Then we verify that the arrival time is above the clock cycle: if it is, the slack window can be redefined so that the slack can be negative; it is up to the user to define a clock period coherent with the arrival time that he/she passes as an input to the script. Then we verify that the number of critical paths inside the specified slack window is below or equal to the number of critical paths passed as a parameter: if it is not, no cells can be moved inside the window, and the script terminates.

For each LVT cell belonging to any path outside the slack window, we retrieve its delay and leakage power consumption, saving them into 2 arrays. Then we swap the same cells into HVT and retrieve again delay and leakage power consumption, saving them into other 2 arrays. Each cell is swapped back to LVT after this. Now we define the priority index k of each of these cells:

$$k_{cell} = \frac{\Delta leakage_{cell}}{\Delta delay_{cell}} = \frac{leakage_{HVT_{cell}} - leakage_{LVT\,cell}}{delay_{HVT_{cell}} - delay_{LVT\,cell}}$$

We define an array with key = K and value = cellname, where K is always negative. Higher absolute values of K correspond to cells with higher priority, which will be the first to be swapped to HVT if feasible. K values are also put into a list sorted increasingly (decreasingly if the absolute values are considered), which will be used for the core of the algorithm.

Commentato [DM1]: Da decidere se va bene o meno la mia modifica al codice.

The first phase of the algorithm operates a coarse swap. ½ of the cells of the array, those with the highest K value, are swapped into HVT, then the constraints on slack, number of critical paths and arrival time are checked. If they are satisfied we swap ½ of the remaining cells of the array (¼ of the total) with the same methodology, otherwise we go to the second pass. We iterate using a geometric series of ½ (which means 1/2, 1/4, 1/8, 1/16, etc…), so the last cell will be swapped (if feasible) in the second phase. This allow us to avoid calling the "get_timing_paths" function (which is time consuming) every time that we swap a cell, so this phase has a logarithmic complexity instead of a linear one.

The second phase of the algorithm refines the results of the first one. If the first phase was interrupted because the previously mentioned constraints were not satisfied, one cell at a time is swapped back to LVT until they are met, otherwise the last cell is swapped to HVT (always under the constraints). This time the complexity is linear, but few cells should be involved. In the worst case, no cell could be swapped without violating the constraints, so to revert the changes made by the first phase, half of the cells of the array have to be swapped back one by one.

Commentato [DM2]: Nessuna cella o una? Cosa dite?

Execution time includes from the initial parameter checking phase to the end of the second phase. The main script uses some auxiliary custom procedures: "cell_swapper" can swap a cell from/to LVT/HVT given its name, and relies on "size_cell" command; "leak_power" and "total_leak_power" parse the output of the "power_report" command to get the desired numerical value; "get_gate_percentage" parses the output of the "report_threshold_voltage_group" command and returns the percentages of LVT and HVT cells as a list.

Commentato [DM3]: "swap_all_LVT" non viene mai usata?

Finally, we return

1. power-savings: % of leakage reduction w.r.t. the initial configuration;
2. execution-time: difference between starting-time and end-time [seconds]
3. lvt: % of LVT gates
4. hvt: % of HVT gates

In case of errors, an empty list is returned.