Alessandro Tempia Calvino s251072 28/06/18
Fausto Chiatante s254271

Low-Power Contest for course SYSTHESIS AND OPTIMIZATION OF DIGITAL SYSTEMS Professor Calimera

The algorithm aims to lower the leakage power given a defined saving percentage and, in this process, to find an acceptable balance among slack losses and elaboration time.
To minimize the total slack worsening, we always switch to HVT the cells with the highest slack, as they are less critical in the design.

First of all, the script saves some information about the design, such as the initial leakage power and the final leakage power to be obtained.
Every time we extract an attribute related to power or time from the design, pt_shell re-elaborate the whole design, wasting computing time. To avoid the extractions of this kind of attributes we first save the leakage power of each LVT cell in the initial design and put them in a hash table with **`full_name`** as key. We then switch the whole design to HVT cells and execute the same strategy, ending up with the two hash tables **`initial_cell_leak`** and **`final_cell_leak`**. We finally revert the design back to LVT to start the computations.

With the external while loop, we check at each iteration that the current power of the design is not yet under the wanted threshold and also that we still have cells to switch to HVT. We choose 5 different levels in the optimization process, each one of them set a step, based on how large is the distance between the current design and the final wanted configuration (difference between wanted leakage and current leakage). The size of the step is also scaled on the size of the design, in term of the number of cells. The chosen step defines how many cells at a time are switched to HVT without recalculating the list of cells sorted by increasing slack. This last operation (line 86) is particularly critical, as it is the only operation in the script that implies a complete recalculation of the whole system's timing. Therefore, the biggest effort in our script is to reduce to minimum the execution of this operation. On the contrary, we are conscious of the trade-off that switching multiple cells at a time brings: not recalculating the slack of the whole design after each cell swap, we are more prone to switch many cells on the same path, making it a critical one.

To do a cell swap we just extract the leading element from a collection of output pins, sorted by decreasing slack. We then use the pin to obtain the cell to which it belongs and perform the switch to HVT elaborating its **`ref_name`**. Knowing the cell's **`full_name`**, we extract from the hash tables the leakage power of the HVT and LVT versions and update the current leakage power (a subtle approximation of the actual one) of the design.
Finally, we check that the current leakage power is still above the threshold, otherwise we skip the sorting of the collection of pins in the last iteration.

In conclusion, our script invests some fixed time at the beginning to track all the leakage power values of LVT and HVT cells, in order to gain performance and precision during the following iterations. A different path could've been taken not extracting all the leakage power values, saving a lot of time on shorter executions (ex. -savings 0.1, 0.15). This would've meant losing precision in slack containment during the execution of a step, and also it would've lead to way longer executions on higher savings (especially on large designs).
We picked step's sizes and leakage intervals in order to find an acceptable balance between execution time and slack control. Furthermore, execution times of under 2 seconds were not considered necessary if they meant a big loss in precision.