Politecnico di Torino

Report

Matteo Battilana s281389
Salvatore Gabriele La Greca s281589
Giovanni Pollo s290136

1 Explanation

The idea of the code is to start by inserting one functional unit per type and the chosen the slowest and the smallest one.

1.1 Heuristic

Starting from the constraints assigned in the previous stage, we apply an optimization loop based on a heuristic algorithm. The heuristic algorithm makes use of a "triple priority" logic to choose a better (the one that has a reduced delay but the minimum area difference compared to the actual one) functional unit for each node. The idea behind this choice of the new FU is done in order not to saturate the area immediately to leave space for other operations.

Nodes are assigned to the faster unit in order of priority and when a node gets a new FU assigned, all its children (the subgraph) that perform the same operation are updated. This is done because they can share the same unit (they are not concurrent).

The parameters taken into consideration for the priority are (in order of importance):

1. Distance actual node - sink node: higher distance means higher priority
2. Fanout: higher fanout means higher priority
3. Fanin: lower fanin means higher priority

All these operations are repeated in a loop until we don't find any difference in the bind OP $\leftrightarrow$ FU. Each time, we execute the MLAC algorithm in order to know the new latency. At the end, the best configuration is returned.

1.2 Filling of Area

Now we are in a situation where we have only one instance of FU for each operation. Now it's time to fill the area if there is space available in order by increasing the number of FUs.

So, we start a new optimization loop in order to explore some possibilities and find the best one. The used logic is based on these steps:

1. Fill the area as much as possible
2. Execute "Minimum Latency Area Constrained" algorithm, in order to know the new latency and to understand which FU can be used in parallel. In this way, we can highlight the unused functional units and save some area
3. Saving the latency and the functional units mapping, if it is a better result

This optimization loop stops when we iterate again and we don't find any difference in the assigned FUs wrt to the previous loop. In fact, when we reach that condition, it means that our algorithm has reached its best state.

This optimization loop is then repeated again, starting from a new FUs as starting point in order to exploit different combinations of assigned FUs.

The trick we used is to perform the sharing inside the MLAC algorithm, along with the schedule operation. In fact, every iteration, we check the maximum number of unit needed in order to obtain that scheduling. For example, if we launch the scheduling with 30 MUL but we can only run 15 MUL in parallel, we are wasting area. By doing this, at each iteration we are able to save some area that can be invested in adding other FUs for other operations (if needed).

An important decision was taken in order to satisfy the tradeoff precision-speed. When we execute the step of filling the area, we used a multiplier. When the free space is larger than ten times the biggest unit available, we set the multiplier to 2. By doing this, we increment the number of the unit by 2. When we are reaching the area limit, we start incrementing by 1. This allows to improve performance but not losing much in the granularity of the algorithm. The value of the multiplier can be changed based on the precision-speed ratio we would like to obtain. This allows the algorithm to be reusable and suitable for different use cases. Another optimization for running time is to use a LUT in order not to recalculate combination with the same constraints.

The minimum latency area constrained algorithm is taken from the laboratory exercises with little adaptation for managing the presence of different speed and area units.

The algorithm to compute the latency uses the last scheduled node time.