Circuit Partition Algorithm

Yunhui Ma

***Abstract*—**The larger size of chips allows us to implement complicated logic and design. However, yield decreases significantly as the size of chip increases. The circuit partition becomes an important topic today. Partitioning divides a chip into portions which can be implemented on separate components. This project is to design a simple partitioning algorithm.

I. Introduction

In this project, I use OpenAccess to implement the partition algorithm. First, I convert all .lef and top_level.v files to oa. Then, I separate into two new chiplets and new top_level files. Finally, I convert all the oa files back to Verilog and run the Innovus to get the result.

II. Approach

For the partition algorithm, I first loop through all modules in the top level. In the OpenAccess, the modules are called instances. Then, I create two new oaDesigns called chiplet0 and chiplet1. I separate the instances into these two new oaDesigns randomly. The way that I decide how the instances are assigned into chiplets is the following. A counter is set to calculate how many instances that it loops through from the top level. By using the function counter mod 2, it will return 0 or 1 as the counter increases. If it returns 0, this instance will be assigned to chiplet0. If it returns 1, the instance will be assigned to chiplet1. For example, when iterating the first instance, the counter is 1. 1mod 2 returns 1. Therefore, the first instance will be assigned to chiplet1. The counter for the second instance is 2. 2 mod 2 returns 0. The second instance is assigned to chiplet0. After assigning all instances into two new chiplets. I begin to calculate their FM score.

To calculate the FM for each instance, I first iterate through all instances in chiplet0. From each instance, I iterate its instTerms which are the pins. Each instTerm would have a net which is the wire connected. From the net, I re-iterate through its instTerms so that I can get what instance the net connects to. oaInstTerm calls a function called getNumBits() which will return the number of bits of the net. For example, in Figure 1, the net has 32 bits length. It returns 32 integer number. The connected instance would be checked whether it is in the same chiplet. If the connected instance is in the same chiplet, the FM score will be subtracted. Otherwise, the FM score will be added. The case of Figure 1 is that the FM score of the instance in chiplet0 will be added 32 because another instance that it connects to is in another chiplet. Using this algorithm iterates through all instances and calculate their FM score. After getting the FM score for all instances, the instances with positive FM score will be moved to opposite chiplet. Therefore, the algorithm reduces the wire connectivity on the top level.

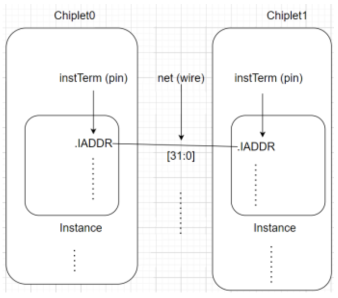


Figure 1, overall partition structure.

Another important algorithm is the area calculation for the chiplets. First, from chiplet0, I sort the instances with their height from high to low as showed in Figure 2. , I choose the first instances, record its height as y, and width as x. The next step is to add the second instance's width to x. It also means that the instance is placed on the right side of the first instance. I continue adding the instances to the right until the x value is greater than 2 times of y. I start a new row on the top and update the y and x. Each time a new row is started, I will record the max value of x. Finally, it returns x and y which are the width and height for the new chiplet. The height of top level chiplet is the max value of the height of two chiplets, and the width of top level is the sum of two chiplets' width.

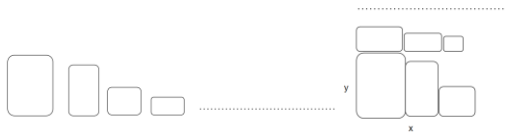
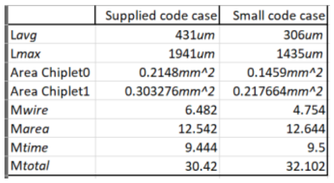


Figure2, Chiplet Floorplan

III. Results

The results are showed in Figure 3. The running time for both cases is about 20 minutes. Both results have improvement compared to the reference result. The Mtotal is larger than the reference value.



	Supplied code case	Small code case
Lavg	431*um*	306*um*
Lmax	1941*um*	1435*um*
Area Chiplet0	0.2148*mm^2*	0.1459*mm^2*
Area Chiplet1	0.303276*mm^2*	0.217664*mm^2*
Mwire	6.482	4.754
Marea	12.542	12.644
Mtime	9.444	9.5
Mtotal	30.42	32.102

Figure 3. metric results

IV. Conclusion

The partition algorithm is still needed to improve. It only generates two chiplets. The FM partition only computes one time, meaning that I only move the instances with positive FM score to opposite chiplet once. The algorithm does not consider the size of instances that are too large or small when they are assigned into new chiplets. Therefore, I still need to improve the partition algorithm. The algorithm should generate more chiplets and try to achieve lowest FM score. It also needs to consider the size of each instance to avoid the area of chiplets can be too large or too small.