Electronic Design Automation (EDA) tool development: Performance enhancements to circuit extraction

Rithvik

[1]Poolesville High School

Purpose

The purpose of the research was to optimize the Magic VLSI Layout Tool to accommodate larger chip designs. Chip designs have labels which store information on their given cells along with their locations. Currently, Magic uses a linked list to hold the labels which has a time complexity of $\mathcal{O}(n)$, making it inefficient for larger designs. Magic requires that labels be accessed by location or content. Therefore, the goal of this research is to optimize the current label storage for both access by location and content.

Background

Magic is a VLSI (Very Large Scale Integration) Layout tool intended for chip design. This tool is easier to use than layout tools of the past due to its increased knowledge of design rules and connectivity. Previous tools such as Caesar and KIC2 had been used for various smaller layouts but provided little support in assiting with routing on chip designs, making the process error prone (Ousterhout et al., 1984).

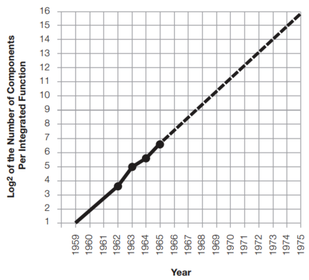

Figure: Original trajectory predicted by Moore, taken from Moore, 1998

Chips have been increasing in complexity and size as per Moore's Law (Moore, 1998). Therefore, the chip design tools need to adapt to accommodate these designs. Magic stores information on the design through labels, which describe the structure of the cell. Labels are used by the synthesis tools to communicate information about the given area. Labels are accessed inside of the synthesis tool both by region and by name. Thus, there must be two data structures used to access the labels associated with a file. The current implementation to store the labels is a linked list, creating a bottleneck when trying to load designs. The proposed solution is to use a hash table for storing labels by content and a binned plane for label storage by location.

Data Structures

Linked Lists

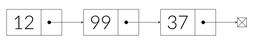

Figure: Sample linked list. Each node carries a value and a reference to the next node. Figure uses sample data.

Search and deletion have time complexity of $\mathcal{O}(n)$. This is not optimal for larger designs where there can be hundreds of thousands of labels.

Hash Tables

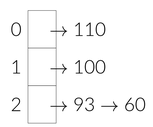

Figure: Sample hash table. Elements are indexed using a hash. Chaining is used to manage collisions. Figure uses sample data.

Theoretical time complexity of $\mathcal{O}(1)$ for all operations. So, the performance of the table will not be affected by increasing chip design size.

Binned Planes

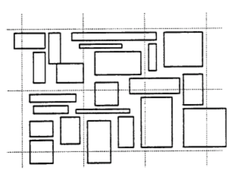

Figure: Sample binned collection. Each cell maintains a list of all rectangles intersecting it. Figure uses sample data.

Optimal solution because labels are almost completely equal in size and distribution.

Circuit Extraction

The first area of optimization was extraction, which is the generation of a netlist file showing the cells and their connections. The **extract all** command in Magic extracts all cells in the window and outputs them into an .ext file. With the striVe chip loaded into Magic, perf was run with a sampling rate of 99 samples per second during extraction. Upon generating a flame graph, **ExtFreeLabRegions**, **ExtLabelRegions**, and **extHierConnections** were found to consume the most time. The bplane enumeration was added to the extHierConnections function in place of the linked list iteration, leading to the following results:

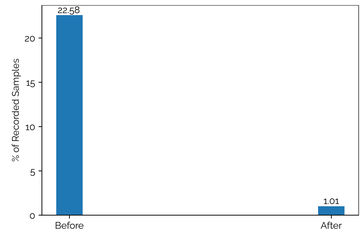

Figure: Figure taken from collected data

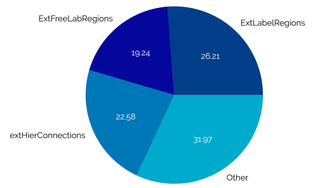

Figure: Figure taken from collected data

Net Selection

The second area of consideration for optimization was Net Selection. When a net is selected, Magic finds the connected geometry for the net and in the process. During net selection, the **DBTreeSrLabels** which recursively searched for all labels with a given mask, took the most time. This function checked two flags: **TF_LABEL_ATTACH** and **TF_LABEL_DISPLAY**. TF_LABEL_DISPLAY looked for labels that where either **lab_rect** or **lab_bbox** is in the search area while the other, TF_LABEL_ATTACH, looked for labels where lab_rect was in the search area. Since the bplane enumeration could only see if lab_rect was in the search area, both the bplane and the linked list had to be used depending on what flag was being used. This led to the following results:

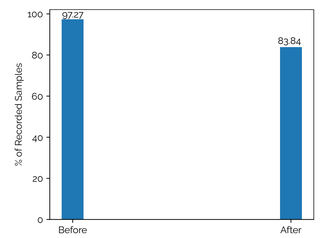

Figure: Figure taken from collected data

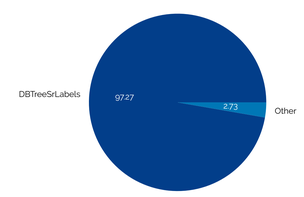

Figure: Figure taken from collected data

Label Filtering by Content

Locating labels by name was done through the **DBCheckLabelsByContent** function. The hash table implementation was used for filtering labels by their content. It was added alongside the binned plane in the cell definition and was populated alongside the binned plane. The hash table was then used in place of the linked list iterator in the function. The binned plane would still need to be used during label filtering by content to resolve collisions because labels are unique by location. The **select short** command was used in testing because it located labels by name to detect a short.

Results

With all errors resolved in extraction, time spent in a recording period on the extHierConnections function went from 22.58% to 1.01%. With the added bplane implementation, there was no noticeable difference, but this was because only one sample was detected for the extHierConnections function.

For net selection, after implementing the bplane data structure, there was a 15% improvement in the performance. Originally accounting for 97.27% of CPU cycles, the DBTreeSrLabels function decreased in run time to 83.84%.

For label filtering by content, the effects of the changes could not be measured due to the perf tool not finding a sample in the recording with the **DBCheckLabelsByContent** function in it with a sampling rate of 30,000. Theoretically, there would be a performance enhancement because the hash table has a search complexity of $\mathcal{O}(1)$ compared to the $\mathcal{O}(n)$ of the linked list.

Conclusions

Although testing of the optimizations of the program were limited to the striVe and AES chip, initial results have been favorable. There was a general decrease in the amount of time spent on extraction, net selection, and theoretically filtering by content using the binned plane and hash table implementations. For the extHierConnections function, after correcting the optimizations, there was a decrease of 21.47% in the time spent on the function. However, taking the entire execution into account, there was added overhead involved. The use of another class for the bplane enumeration meant that more time was spent on instantiation, however this would be a worthwile tradeoff for larger designs.

For net selection, the smaller difference in time spent could have been attributed to the fact that not all of the labels can be handled by the bplane implementation. For the striVe chip, there might still have been many more instances where the linked list could have been traversed, and therefore lead to a not as significant decrease in the time spent on the function as before.

With the optimizations implemented, Magic will be able to handle larger chip designs and allow developers to interact with them at an increased speed.

Future Work

For the future, a wider variety of chips could be used in testing the effectiveness of the optimizations discussed in the paper in order to ensure that they do not only apply to the AES or striVe chip. A limiting factor in assessing the performance of the commands run was having to stop the performance recording. Because the recording had to be manually stopped, in some instances, idle time overshadowed the recording of Magic, making the flame graphs less detailed because of the relatively less time spent on the functions of interest.

References

Moore, G. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, *86*(1), 82–85. https://doi.org/10.1109/jproc.1998.658762

Ousterhout, J. (1984). Corner stitching: A data-structuring technique for vlsi layout tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *3*(1), 87–100. https://doi.org/10.1109/tcad.1984.1270061

Ousterhout, J., Hamachi, G., Mayo, R., Scott, W., & Taylor, G. (1984). Magic: A vlsi layout system. *21st Design Automation Conference Proceedings*. https://doi.org/10.1109/dac.1984.1585789

Taylor, G., & Ousterhout, J. (1984). Magics incremental design-rule checker. *21st Design Automation Conference Proceedings*. https://doi.org/10.1109/dac.1984.1585790