Hammer: Top-Down Hierarchical Flow

*Spring 2023, Ver 2.0

Daniel Chang
University of California, Berkeley
Berkeley, CA, USA

Harrison Liew
University of California, Berkeley
Berkeley, CA, USA

Borivoje Nikolic
Univeristy of California, Berkeley
Berkeley, CA, USA

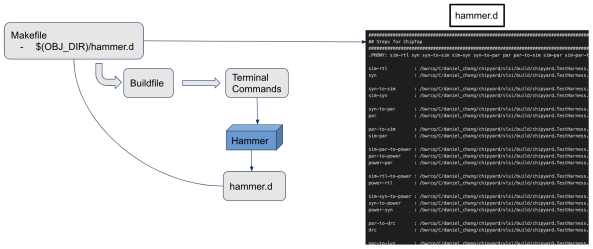
I. Introduction

The top-down hierarchical flow is an essential feature that will bring greater flexibility and freedom in partitioning the design of a chip by exploiting the division of labor, in addition to what the original Bottom-Up Flow Hammer already has. An inspection of the already-established Bottom-Up Flow started early in the semester and later transitioned into re-constructing the flow graph for the new Top-Down Flow while researching other potential functionalities that might come with the new feature. The second half of the semester was fully dedicated to transcribing the Top-Down Flow to corresponding Innovus commands. Currently, it is at the state where we are attempting to troubleshoot setting a constraint file for the command proto_design.

II. Hammer VLSI Flow Overview

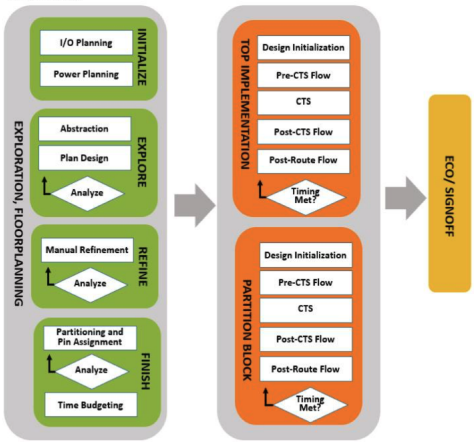
A. Hammer.d

The VLSI flow management and dependencies of the bottom-up flow are encoded in the auto-generated Makefile hammer.d, pointed by the -OBJ_DIR command line flag. Below is a diagram that summarizes the file structure and what the file format looks like. Based on the dependencies in the bottom-up hierarchy, I have recreated the dependencies that will be implemented in the top-down hierarchy in the later section, defining the overall flow steps before diving into the Innovus command script.



B. Hierarchical Design Closure

There are 4 general steps in the hierarchical design closure Innovus suggests in their user guide. First is Chip Planning, where we specify the Input/Output requirements and the overall constraints regarding power, resource, area, etc. Based on such information, partitions will be generated along with proper time budgeting. The second step would be the implementation of the partitions created, yet the methodology would vary depending on which partitioning method we choose to use in the design. The final step would be assembling the partitions both on the top implementation and on the partition scope along with the crucial validation processes. Below is a diagram provided by Innovus that will help you visualize the flow.



III. Partitioning

The essential question that must be posed at this point is – "Why is partitioning needed?" With growing design size with millions of instances, flat design flow becomes unavailable due to huge memory requirements and excessive run time. This is where hierarchical flow comes into play, dividing the design into several manageable partitions and potentially splitting up the work into several teams that are capable of developing the design in parallel. hierarchical flows can also be used for prototyping, in such cases when we are not too sure how certain partitions are supposed to look. By setting mere preset constraints in a partition, it is possible to finish up the top implementation first. Such structure decouples the implementation of the partitions and the top-level as the design of the partition can be iteratively completed without affecting the assembly. Partitioning also makes it easier to place macroblocks, or pre-designed/pre-verified functional units(such as microprocessors, memory controllers, etc).

Below is a brief summary of how the partitions can be set and be interconnected.

A. Pin Assignment

Pin Assignment is what sets constraints on the partition where it has to be optimized for specific needs. Given a set of user-specified pin assignments and constraints for area IO design, they are directly related to how the power supply and the necessary pins/pads will be routed at the top level. Additionally, there are many other factors to consider such as the placement of IO pads, routing the pads into the internal logic of the chip, physical constraints of the package, etc. Instead of manually assigning each pin, which is not quite necessary for simpler partitions and prototyping, we will be exploiting the automated pin-assignment features offered by Innovus. Once the top-level constraints for the pins are configured, the pins are assigned for each partition. Based on how the design has to be optimized, there can be pin blockages in specified areas in order to blockade certain areas from assigning pins on metal layers. Pin size, spacing, and layers can also be manipulated.

B. Partition Types

Among several partition types Innovus offers, starting off with a channel-based methodology seems to be the easiest way. While it is not the most optimal partition type performance-wise, the channel-based methodology has the least factors to put into account. While channel-based methodology implements the top-level logic in the channel spaces between the partitions, the other two partitioning types require feed-through buffers. One is a channel-less methodology where we get rid of the channels in between and pack the top-level with partitions. The other is narrow-channel methodology where the width of the channels that separate the partitions is minimized. Because there is not enough space for the top logic to be implemented in the channel space for both methodologies, the implementation of feed-through buffers or the overpass highway buffers that connect two distant partitions through other partition(s) in between is crucial. Because implementing feed-through buffers might be tricky in developing a prototype for a new hierarchical flow, it is a feature that can be added later after we stabilize the flow with the channel-based methodology mentioned above.

C. Physical Dividers

Innovus offers 4 types of physical dividers that generate a partition. Soft Guide, Guide, and Region are the dividers with relaxed constraints. The new hierarchical flow will be using Fence as a hard physical boundary for the partitions, which makes it easier to distinguish and instantiate each partition.

D. Time Budgeting

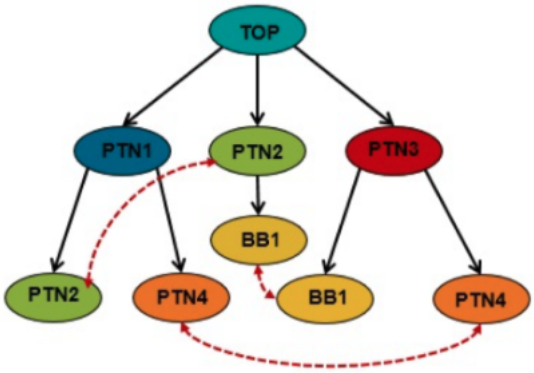
Time Budgeting is a crucial step that regulates the maximum allowed delays and clock frequencies for the logic paths and thus is correlated with the critical path of the design and the specific performance requirements defined for each purpose. Variables such as the maximum clock frequency, setup time, hold time, propagation delay, etc should be specified for each partition block. Time Budgeting is a step that is run on an early-stage non-optimized design as Innovus provides virtual design optimization that derives accurate timing budgets for each partition.

E. Assembling Partitions

As introduced in the hierarchical design closure subsection, Assembling Partitions is the final step that puts all partitions into the top-level, building the design from top to bottom. (top design to parent partitions to child nodes)

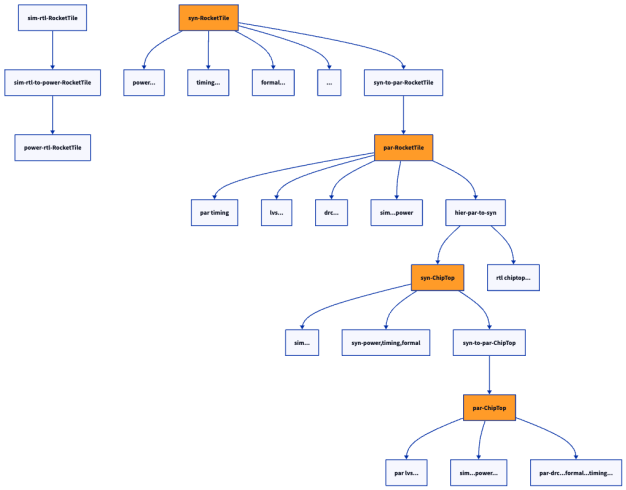
IV. Hierarchical Flow

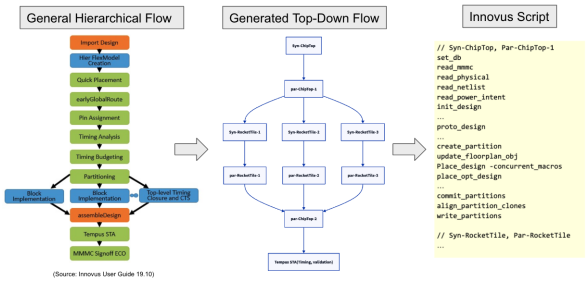
Hammer currently has a pre-implemented Bottom-Up hierarchical flow, which starts from the Block-level implementation and assembles the final design at the Top-level. The Top-level flow brings all the partitions together, leading to the Chip Assembly step. The top-down hierarchical approach is intuitively the exact opposite, starting from Chip Planning and setting the Top-level logic along with setting partitions that will later implement the Block-level logic. The diagram below illustrates the hierarchical relationship between the Top-level logic and the partitions. The 'Top' node, or the ChipTop contains multiple partitions, in this case, PTN1, PTN2, and PTN3. Nested partitions can be implemented when necessary, as in the diagram we have children nodes for each partition. Partitions can also be shared across different levels of the tree dependency diagram when appropriate types of physical constraints are applied. Depending when whether we start with the synthesis of the ChipTop(root node) or the Partitions(leaf nodes), the two different approaches to the hierarchical flow diverge.



A. Bottom-Up Methodology – D2 Diagram

In order to regenerate a better visualization of the bottom-up hierarchical flow, I have created a D2 Diagram of the dependencies specified in Hammer.d, where there is a single RocketTile in a ChipTop. D2 is a Declarative Diagramming modern language that turns text into diagrams, facilitating a quick conversion of relatively complex concepts into graphs. Below is a simplified version of the D2 diagram I have generated.





As of May 12th, I am working on running proto_model, an Innovus command that runs automatic floorplan synthesis.

REFERENCES

[1] Innovus Stylus Common UI Text Command Reference, Product Version 21.15
[2] Innovus User Guide, Product Version 21.15

B. Top-Down Methodology

A general hierarchical flow starts from importing the design and creating flex models as implied in the diagram below. A Flex Model is an abstraction model provided by Innovus that can be used for partition implementation flow while reducing the size of the netlist. Flex Model reduces the size of the netlist by adding modules called FlexFillers, which perform like black boxes. Without running the entire netlist and simply running the FlexModel consisting of FlexFillers, the runtime can be fastened up to 20 times faster. Because it significantly simplifies a partition, it is also used for prototyping with mere constraints set for each block. Flexmodel also facilitates faster runtime without degrading the accuracy of timing or area analysis. What happens after the creation of FlexModels is EarlyGlobalRoute followed by Quick Placement of the modules. EarlyGlobalRoute is a step in Innovus design flow that performs quick global routing for estimating routing-related congestion and resistance/capacitance values. In the early stage of the flow, EarlyGlobalRoute is great for optimizing the top-level logic and timing analysis. When used for prototyping, it automatically creates actual wires that implement early optimization of the flow. Routing must occur before partitions are converted into blocks with fences since Innovus will not allow routing through hard physical constraints once the fences are generated. Once the block implementation steps are completed, the design is assembled at a top-level and Tempus STA(Stating Timing Analysis and verification) and MMMC(Multi-Mode Multi-Corner) signoff ECO are run, which are validation/debug steps that will polish up the design. Once Again, using the D2 language, I have generated a simplified diagram of how it will look in the scope of the Top-Down flow. Below demonstrates how the generated Innovus script is derived from the general hierarchical flow diagram from the Innovus User Guide.