Go with the flow

Design exploration: changing the fpga design flow.
By Ajay Jagtiani.

FPGAs have undergone significant architectural changes in the last few years. Beginning with hard blocks such as ram and dsp, fpgas now also include transceivers and hard IP blocks, such as Ethernet and PCIe Express. With these new functional blocks, fpga designers can now create complex designs. However, these designs can sometimes push the cost, power and performance specification requirements of the targeted fpga device.

In consumer and mobile applications, for example, low power and low cost are essential, so designers may have to:
- Explore different tool settings; for example, extra effort settings for routing or placement to try to improve performance or reduce area
- Make design changes, such as selecting block ram over distributed ram for performance
- Change the architecture of the design, for example, selecting a parallel, rather than serial, protocol.

These design approaches are typically explored sequentially. As a result, overall compile time is increased, which can create issues with the schedule. Almost always, designers – particularly those designing consumer products – are under huge pressures to get their product to market first. As a result, designers need to complete their fpga design in the shortest time while meeting challenging specifications.

Traditional design flow issues
Using the traditional design flow, fpga designers often struggle to meet targeted design specifications without affecting the schedule. This is because the traditional design flow was conceived to be fundamentally sequential: each time the user makes a change to the design, it needs to be recompiled assess its impact. This process is repeated until the specifications are met. This approach can extend the schedule and a solution is needed that will reduce overall compile time within the design flow.

FPGA design software introduced a tool called ‘settings explorer’, which allows the designer to select optimisation settings, such as retiming or fanout control, for the design and then archive the results of the entire space; all runs done by exploring the different settings or to save only the run with the best results. The designer can let the tool select the settings automatically based on some high level goals, such as ‘design for lower power’ or ‘reduce area of the design’ and explore the entire space [see fig 1]. While this feature improved the ability to meet targets, it did not alleviate the schedule pressure completely, since any change to the design could cause the user to launch the tool again and cause a lengthy compile time.

Another innovation allowed fpga design software to use any number of cores to reduce compile time. However, changes to the design still required another compile, which added to the total elapsed time, even if the compile time of one design iteration had been reduced.

Incremental design flow
FPGA design software has borrowed from asic design methodology and introduced an incremental design flow [fig 2]. In this flow, users can partition their designs based on logical hierarchies for runtime reduction and timing preservation. Using this approach, users can create a partition on the logical hierarchies where design changes could occur and would need to be recompiled. This approach can help to reduce overall compile time while preserving
While a major step in the right direction, this approach does not address the sequential nature of the design flow. Users can have only one implementation of the design active at a given time and need to wait for two compiles to be run sequentially in order to compare the results.

What is needed is a more radical change in the FPGA design flow, so users can compile multiple implementations in parallel and compare the results of two implementations after one compile, rather than after two sequential compiles. Users then could reject or accept the changes quickly and with limited impact on their schedule.

Design scenarios
Consider an example where the user has to change the design implementation from using block ram to distributed ram. Block ram is ideal for storing operations on coefficients in DSP-centric designs because it provides faster throughput. If the user wants to make such a change, they need to complete two runs sequentially before the impact of this major change can be assessed. However, with multiple implementations run in parallel, the impact can be seen more rapidly.

Another example where running multiple implementations in parallel is valuable is when the user changes the architecture of the design. A typical case is in high-speed mobile applications, where data traffic management is changed from a serial to a parallel implementation.

With such a fundamental change to the FPGA design flow, users can speed their schedule or, at least, reduce schedule pressures and improve productivity.

A contemporary design flow
One example of a contemporary design flow can be found within Lattice Semiconductor’s Lattice Diamond FPGA design software.

The software includes a feature called Run Manager: the user can have two RTL files for a single design captured as two implementations and run these in parallel. This reduces compile time compared to a traditional design flow and quickly compare the results of the two implementations. If satisfied with the results, the user can immediately program the device by selecting the better of the two implementations. If the user is not satisfied with any of the changes, they can create new implementations and run them again to assess their impact.

There is no limit to the number of implementations users can have in a run (fig 3).

Run time examples
Consider these two examples that were compiled with the Run Manager on two different devices.

Example 1: A DSP-centric design targeting the LatticeECP3-95EA 1156-8 FPGA. There were two implementations run for this example on a Windows-based machine with four cores and 4GB of RAM. The compile time for two serial implementations was 13 minutes, while the compile time using Run Manager for the two implementations in parallel was eight minutes: an overall reduction of close to 40% in compile time.

Example 2: A traffic manager design targeting a LatticeECP3-35EA 484-6 FPGA. Again, there were two implementations for this example run on a Windows-based machine with four cores and 4GB of RAM. The compile time for the two serial implementations was almost three hours, while the compile time with Run Manager for the two parallel implementations was 90 minutes: a savings of 50% in compile time.

Conclusion
In a world of fast-changing applications, meeting tight-to-market schedules is critical and designers are always under pressure to deliver their designs faster. In turn, designers are asking for faster compile times from their FPGA design software tools. Run Manager can provide them with that competitive advantage, leveraging the computer architecture and features in Lattice Diamond. Multiple implementations with an RTL file for the same design allows designers to compare their results quickly and improve productivity.

Author profile:
Ajay Jagtiani is senior product marketing manager with Lattice Semiconductor.