We will demo a novel high-level backannotation flow that reports routing congestion issues at the C++ source level by analyzing reports from FPGA physical design (Xilinx Vivado) and internal debugging files of the Vivado HLS tool. The flow annotates the C++ source code, identifying likely causes of congestion, e.g., on-chip memories or the DSP units. These shared resources often cause routing problems on FPGAs because they cannot be replicated by physical design to reduce pressure on routing resources. We demonstrate on realistic large designs how the information provided by our flow can be used to both identify congestion issues at the C++ source level and solve them using HLS directives. The main demo steps are:

- Extraction of the source-level debugging information from the Vivado HLS database
- Generation of a list of net names involved in congestion areas and of their relative significance from the Vivado post global-routing database
- Visualization of the C++ code lines that contribute most to congestion

We show that the backannotation of post-placement global routing congestion information can be used to effectively and easily improve performance (in particular clock period) by performing simple transformations using HLS directives. We can see from Figure 1 that congested areas (in red) are reduced. The recipes that we followed were relatively straightforward: if congestion was due to BRAM, we partitioned it, typically resulting also in increased performance. If DSPs were involved, we either allocated more units or created one level of function hierarchy to simplify the scheduler’s job and create a more regular netlist. These recommendations can be provided directly to the designer, based on the findings of our tool, and some day they could even be tried automatically. We believe that our approach can effectively help solving a serious problem that affects fast HLS-based design flows for modern large accelerators implemented on FPGAs.

Figure 1: physical implementation of Convolution on FPGA before (left) and after (right) optimization.