HLS tools: Portable generated HDL code is a must

I will make an argument here: Portable generated HDL from high-level synthesis must be readable. Manual tracing by the human expert is still invaluable and should always be there.

There are certain rules that should apply to generating portable, generic and readable HDL code:

- Maintain program symbols (variables) in the generated code. Temporaries should keep certain, easy to follow, naming conventions. I do this in HercuLeS HLS: http://www.nkavvadias.com/hercules/.
- Cross-tagging between source level, intermediate representation, graph-based representation and final HDL code. I'm currently investigating cross-tagging approaches since HercuLeS should support tracing among source (C w/wo GMP API for now), IR (N-Address Code: http://www.nkavvadias.com/hercules/nac-refman.html), low-level IR (Graphviz [http://www.graphviz.org] CDFGs) and RTL VHDL.
- Keep control steps (FSM/FSMD states) clear and visible. Consistent naming conventions can help (to associate states with the corresponding basic blocks or regions for instance). Again cross-tagging can make this more elegant.
- Exploit the casual FSMD feature: do embed datapath actions into the next state logic code of your FSMD code. Don't do it Vivado HLS style. The Vivado HLS approach is an ugly mess. Datapath actions are thrown out in concurrent code form, and nobody can follow anything. There is more to pay here that the minor gains in easier resource sharing. X and A people, I'm talking to you: your backend tools are better than that, not much is lost in optimization if you embed datapath actions.

That's for now.