```
public class FullAdder extends Logic {
 /* Define the ports for a 1-bit full adder */
  public static CellInterface cell interface[] = {
    in(”a”, 1), in(”b”, 1), in(”cin”, 1),
    out(”s”, 1), out(”co”, 1)
  };

  public FullAdder(Wire a, Wire b, Wire ci, Wire s, Wire co) {
    /* Connect wires to my ports */
    connect(‘‘a’’, a); connect(‘‘b’’, b); connect(‘‘ci’’, ci);
    connect(‘‘s’’, s); connect(‘‘co’’, co);

    /* Instantiate the logic functions */
    or_o( and(a,b), and(a,ci), and(b,ci), co ); /* co is output */
    xor_o( a, b, ci, s ); /* s is output */

    /* Map the gates to LUTs, and place. */
    map( a, b, ci, s ); place( s, ‘‘R0C0.F’’ );
    map( a, b, ci, co ); place( co, ‘‘R0C0.G’’ );
  }
```

A CAD Suite for High-Performance FPGA Design

Brad Hutchings, Peter Bellows, Joseph Hawkins, Scott Hemmert, Brent Nelson, Mike Rytting

Year of publication: *1999*
Area: *Languages and Compute Models*

It's funny how languages and tools get established, spread and become standard. For decades now, digital hardware in ASICs and FPGAs has been designed in languages originally defined for documentation (VHDL) and simulation (Verilog). ASIC engineers put up with these languages and their synthesis foibles despite the fact neither was conceived or defined to be a design language.. We still use the ASIC model and languages for FPGAs in FCCMs, even though FPGAs don't even have logic gates, and a "silicon turn" takes minutes. Out of a number of efforts to escape from this and design FPGAs in FPGA terms, arguably the most successful was JHDL, a landmark in FPGA design.

JHDL started as a design tool for reconfigurable systems that change over time, supporting run-time and partial reconfiguration, as originally reported in an excellent FCCM 1998 paper (with "JHDL" in the title, that paper is often cited for this paper's work). JHDL's major impact came with further development into the unified, graphical FPGA tool suite reported in this paper.

You design in JHDL by writing a program in a well-known language, Java, using library objects for hardware components. JHDL is constructive and parametric. Higher-level design and instance specialization comes naturally from object-oriented programming. You construct your design as a hierarchy of objects instantiated with constructor parameters. What you write is what you get, avoiding pitfalls and uncertainty of synthesis, and providing fine control over all levels of the FPGA implementation, including placement. This is particularly important for datapaths and computing applications where automated tools often fail to find the mappings the designer knows the FPGA can realize. Executing your Java program generates its netlist or runs its simulation. Since JHDL design and simulation are unified in a simple graphical toolset, shifting between editing and testing is instantaneous. It happens so fast it's addictive and fun. JHDL also makes it easy to switch between software simulation and hardware execution in the same tool.

BYU released JHDL in open-source form, so it came into wide use by FCCM researchers and students. BYU faculty and students provided strong support through several FPGA generations. A substantial number of reconfigurable computing projects were developed in JHDL, and it had a strong influence on high-level FPGA and FCCM design tools to follow. This put it in the top tier of FCCM citations. JHDL is one of the most important and effective design methodologies to come out of the first twenty years of FCCM.

Mike Butts

DOI: http://dx.doi.org/10.1109/FPGA.1999.803663