Formal Verification of Tree-Structured Carry-Lookahead Adders

Sae Hwan Kim
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244
sakim@syr.edu

Shiu-Kai Chin
Department of Electrical Engineering and Computer Science
Syracuse University, Syracuse, NY 13244
skchin@syr.edu

Abstract

Quad trees – trees with four branches, are used to abstract the carry lookahead adder.
3.2. Hierarchical Hardware Descriptions

The abstract quad tree structural descriptions are converted into hierarchical hardware descriptions with input and output signals, 4-bit carry-lookahead adders cla4b_imp, and carry-lookahead generators CLG, by the predicate tree.cla.imp. The arguments of tree.cla.imp are a quad tree t, carry input cin, adder inputs a and b, output f, carry output co, and carry generate and propagate signals G and P.

In the base case, the quad tree argument t is LEAF. tree.cla.imp describes a 4-bit carry-lookahead adder, cla4b_imp:

\[
\text{tree.cla.imp (a b f co G P) = (cin a b cin b co co 0 P = (3a0 a1 a2 a3 b0 b1 b2 b3 f0 f1 f2 f3. (a = WORD a0; a1; a2; a3) \land (b = WORD b0; b1; b2; b3) \land (f = WORD f0; f1; f2; f3) \land cla4b_imp cin a b a2 a3 b0 b1 b2 b3 f0 f1 f2 f3))}
\]

For the recursive case where the quad tree argument t is of the form NODE t3 t2 t1 t0, tree.cla.imp connects four subtrees t3, t2, t1, and t0 together using carry-lookahead generator CLG. Subtree input and output words a3, a2, a1, a0, b3, b2, b1, b0 are concatenated together using WCAT to form the input and output words a and b of the whole adder. The output f is obtained by concatenating subtree outputs f3, f2, f1, and f0. The detailed structure of subtrees t3, t2, t1, and t0 is obtained by recursively applying tree.cla.imp to each subtree.

\[
\text{tree.cla.imp (NODE t3 t2 t1 t0) cin a b f co G P = (cin a b cin b co co 0 P = (3a0 a1 a2 a3 b0 b1 b2 b3 f0 f1 f2 f3 cin4 cin8 cin12 co4 co8 co12 G P0 G1 P1 G2 P2 G3 P3. (a = WCAT (a3,WCAT (a2,WCAT (a1,WCAT (a0,)))) \land (b = WCAT (b3,WCAT (b2,WCAT (b1,WCAT (b0,)))) \land (f = WCAT (f3,WCAT (f2,WCAT (f1,))) \land}
\]

4. Correctness

Informally, the behavioral specification is the sum of the values of the inputs cin, a, and b equals the value of the outputs f and co summed together. This is formally specified by add.spec where BNVAL, BV, EXP, and WORDLEN denote the binary word value, bit value, exponent, and word length functions, respectively.

\[
\text{tree.cla.imp t cin a b f co G P = \text{add.spec cin a b f co = BNVAL a + BNVAL b + BV cin = BNVAL f + BV co + EXP WORDLEN f}}
\]

The correctness of tree.cla.imp with respect to add.spec is proved in [2] using the HOL theorem prover [1]. The following correctness theorem states that the behavior of tree.cla.imp is completely contained by add.spec.

5. Conclusions

Many hardware devices are built using tree-shaped structures. The approach outlined here, using parameterization and tree data types for abstract structural descriptions, should be applicable beyond carry-lookahead adders. The advantage of using abstract structural descriptions is the support of hierarchical descriptions that ease the verification of implementations with respect to purely behavioral specifications.

References