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 abstractly describe tree-structured carry-lookahead adders using 4-bit components. The specification and implementation descriptions are parameterized and describe tree-structured adders having arbitrarily large inputs and outputs. The descriptions are formally verified using the HOL theorem prover.

1. Introduction

Carry-lookahead adders are widely used due to their superior performance over ripple adders. The expressive power of higher-order logic makes it possible to define a wide variety of data types and to prove theorems that concisely and abstractly state the properties of these types.

Implementations and specifications are described parametrically using higher-order logic and types. These parametric descriptions are proved correct once and subsequently are specialized to specific values, e.g., to specific word lengths.

Carry-lookahead adders are explained in Section 2. The quad tree type qtree is defined in Section 3. Using qtree, a parameterized implementation description of tree-structured carry-lookahead adders is defined. Its correctness is stated in Section 4. The conclusions are in Section 5.

2. Carry-Lookahead Adders

Carry-lookahead adder (CLA) circuits are constructed using carry-lookahead adders and carry-lookahead generators (CLG). Figure 1 shows a 16-bit adder as an example. Each 4-bit CLA produces a 4-bit slice of the sum. The CLG generates the carry inputs for the three most significant carry-lookahead adders.

![Figure 1. A 16-Bit Carry-Lookahead Adder](image)

The structure of the adder in Figure 1 is a quad tree. The carry-lookahead generator is the node of the quad tree. Each 4-bit carry-lookahead adder is a leaf of the quad tree.

3. Tree-Structured Adders

3.1. Describing Implementations Using Quad Trees

Quad trees belong to the recursive type qtree described informally by:

\[ qtree ::= \text{LEAF} \mid \text{NODE } qtree \ qtree \ qtree \ qtree \]

Quad trees are either a LEAF or a NODE connected to four other qtrees.

Tree-structured adders using 4-bit carry-lookahead circuits are described abstractly using quad trees. Each NODE corresponds to a carry-lookahead generator. Each LEAF corresponds to a 4-bit carry-lookahead adder. For example, the 40-bit adder in Figure 2 is described by:

\[ \text{NODE} (\text{NODE} \ \text{LEAF} \ \text{LEAF} \ \text{LEAF} \ \text{LEAF}) (\text{NODE} \ \text{LEAF} \ \text{LEAF} \ \text{LEAF}) \ \text{LEAF} \ \text{LEAF} \ \text{LEAF} \]

232
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 basic case, the quad tree argument t is LEAF. tree.cla_imp describes a 4-bit carry-lookahead adder, cla4b_imp:

\[ \text{tree.cla_imp t0 cin a0 b0 f0 co4 G0 P0} \wedge \text{tree.cla_imp t1 cin a1 b1 f1 co5 G1 P1} \wedge \text{tree.cla_imp t2 cin a2 b2 f2 co12 G2 P2} \wedge \text{tree.cla_imp t3 cin a3 b3 f3 co G3 P3} \wedge \text{CLG cin G0 P0 G1 P1 G2 P2 G3 P3 cin4 cin8 cin12 G P} \]

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.

\[ \forall cin a b f c o. \]
\[ \text{add.spec cin a b f c o} = \]
\[ \text{BNVAL a + BNVAL b + BV cin} = \]
\[ \text{BNVAL f + BV co * 2} \exp \text{WORDLEN f} \]

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.

\[ \forall t \text{ cin a b f c o G P}. \]
\[ \text{tree.cla_imp t cin a b f c o G P} \wedge \]
\[ \text{add.spec cin a b f c o} \]

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
