Claq: Classical-to-quantum circuit synthesizer

Tsuyoshi Ito
NEC Laboratories America, Inc.

Claq is a tool which converts a classical combinational circuit to a quantum circuit.

1 Usage

Claq is invoked by the following command line:

`claq` [⟨*option*⟩...] ⟨*input*⟩...

Each ⟨*input*⟩ is the name of a file containing the description of a classical combinational circuit in the format described in Section 2. For each ⟨*filename*⟩ on the command line, Claq produces its report to the standard output. All ⟨*option*⟩s apply to all ⟨*input*⟩s on the command line, and the relative order of ⟨*option*⟩s and ⟨*input*⟩s is irrelevant.

Valid ⟨*option*⟩s are as follows.

- `--report=`⟨*report-type*⟩: This option specifies the type of the report that Claq produces. ⟨*report-type*⟩ is one of the following.
  - `circuit` (default): The .QC circuit which consists of NOT, CNOT, and Toffoli gates. CNOT and Toffoli gates may have positive and negative controls.
  - `circuitstd`: The .QC circuit which consists of gates in the standard gate set (Clifford and T gates).
  - `raw`: The circuit in the internal format (primarily for debugging).
- `-s`: A shorthand for `--report=circuitstd`.
- `-n`, `--noncoherent`: By default, Claq produces a quantum circuit which implements the specified classical function f in a coherent manner: $|x\rangle|y\rangle|0\rangle \mapsto |x\rangle|y \oplus f(x)\rangle|0\rangle$. When `-n` is specified, Claq produces a quantum circuit which implements f in a noncoherent manner: $|x\rangle|y\rangle|0\rangle \mapsto |x\rangle|y \oplus f(x)\rangle|\varphi_x\rangle$, where $|\varphi_x\rangle$ is some quantum state depending on x. Specify `-n` when you know that the noncoherent version is sufficient and want a smaller circuit.

2 Input specification

Claq reads classical circuits from files specified on the command line. Each file contains one classical circuit, whose grammar will be specified in this section.

An example of a simple classical circuit is:

```
-- 2-bit adder with carry
.inputs a0, a1, b0, b1, c0;
.outputs s0, s1, s2;
s0 = a0 ^ b0 ^ c0;
c1 = a0 & (b0 | c0) | b0 & c0;
s1 = a1 ^ b1 ^ c1;
s2 = a1 & (b1 | c1) | b1 & c1;
```

2.1 Lexical structure

The lexical structure of an input file is similar to that of Haskell 2010 with the notable exception that identifiers can start or contain a period (“`.`”).

A comment either begins with two consecutive dashes (“`--`”) and extends to the next newline, or begins with the sequence of an open brace and a dash (“`{-`”) and ends with the sequence of a dash and a close brace (“`-}`”). Comments of the latter kind can be nested.

2.2 Syntax

The following is the syntax for the input file after lexical analysis in a variation of the Backus–Naur form. A vertical bar (“ | ”) means a choice, brackets (“[. . .]”) mean an optional part, and braces (“{. . . }”) mean zero or more repetitions.

```
        ⟨input-file⟩ ::= {⟨statement⟩}

         ⟨statement⟩ ::= ⟨inputs-statement⟩
                       | ⟨outputs-statement⟩
                       | ⟨equation-statement⟩
                       | ⟨empty-statement⟩
  ⟨inputs-statement⟩ ::= .inputs [⟨wire-name⟩ {, ⟨wire-name⟩}] ;
 ⟨outputs-statement⟩ ::= .outputs [⟨expr⟩ {, ⟨expr⟩} ;
⟨equation-statement⟩ ::= ⟨wire-name⟩ = ⟨expr⟩ ;
   ⟨empty-statement⟩ ::= ;

              ⟨expr⟩ ::= 0
                       | 1
                       | ⟨wire-name⟩
                       | ( ⟨expr⟩ )
                       | ~ ⟨expr⟩
                       | ⟨expr⟩ & ⟨expr⟩
                       | ⟨expr⟩ ^ ⟨expr⟩
                       | ⟨expr⟩ | ⟨expr⟩

         ⟨wire-name⟩ ::= ⟨identifier which is not a keyword⟩
           ⟨keyword⟩ ::= .inputs | .outputs
```

Keywords and identifiers are case-sensitive.

In the production rules for ⟨*expr*⟩, the unary operator ~ has the highest precedence, the binary operator & has the second highest, the binary operator ^ has the third highest, and the binary operator | has the lowest precedence. The associativity of the binary operators &, ^, and | does not matter because of their semantics.

A valid input file must contain exactly one inputs statement and exactly one outputs statement. An input file may contain any number of equation statements and empty statements. Empty statements do not have any meaning and completely ignored. The order of equation statements does not have any meaning, nor does the relative order of the inputs statement, the outputs statement, and equation statements.

2.3 Wire definitions

All the wires that appear in expressions (in the outputs statement or the right-hand side of an equation statement) must be defined.

A wire is considered to be defined if it appears in the inputs statement or it appears on the left-hand side of an equation statement.

No wires may be defined more than once.

3 To-dos

- Detect unused wires in input circuits and remove them.
- Detect wires which are guaranteed to contain the same value (or the opposite values) in input circuits and merge them into one.
- Implement any nontrivial optimizations.

About license of Claq

Acknowledgments

The author thanks Martin Roetteler for helpful discussions. The development of Claq was supported in part by the Quantum Computer Science Program of the Intelligence Advanced Research Projects Activity.