Modeling Electrical Circuits

CS 6115 Small Project Report

Brittany Nkounkou and Yao Wang

Overview

In this project, we model electrical circuits and prove certain properties about them in Coq. Circuits are modeled by basic logic gate components (e.g. AND gate, OR gate) and two constructors that generate compositional and parallel circuits. Properties of these circuits include area, delay and behavior, implemented as functions on circuits. With these definitions, we are able to prove various properties about circuits and the named functions on them. For example, we can explore the tradeoff between area and delay of behaviorally equivalent circuits. The bulk of the proofs we provide exhibit basic structural properties that can be applied to any arbitrary circuits, and are followed by a few proofs specific to certain circuits.

Circuit Definition

Circuits are defined dependently, where the number of (ordered) inputs and outputs are included in a Circuit's type, i.e. `Circuit : nat -> nat -> Type`. Further, Circuits are defined inductively by the following two sets of constructors:

- Various nullary constructors, each modeling basic indivisible logic gate components, including:
  - `none : Circuit 0 0`
  - `high : Circuit 0 1`
  - `low : Circuit 0 1`
  - `wire : Circuit 1 1`
  - `inv : Circuit 1 1`
  - `split : Circuit 1 2`
  - `and : Circuit 2 1`
  - `nand : Circuit 2 1`
  - `and3 : Circuit 3 1`
  - `...`
- Two binary constructors modeling compositional circuits (circuits in series) and parallel circuits:
  - `comp : Circuit m n -> Circuit n o -> Circuit m o`
  - `par : Circuit m n -> Circuit o p -> Circuit (m+o) (n+p)`

Circuit Functions

We model basic properties of Circuits by defining three functions on the Circuit type:

- `area : Circuit m n -> nat`. This function computes the area of a Circuit. Areas for the compositional and parallel constructors are defined as follows:
  - `area (comp A B) = area(A) + area(B)`
  - `area (par A B) = area(A) + area(B)`
- `delay : Circuit m n -> nat`. This function computes an upper bound on the delay of a Circuit. Delays for the compositional and parallel constructors are defined as follows:
  - `delay (comp A B) = (delay A) + (delay B)`
  - `delay (par A B) = max (delay A) (delay B)`

It is important to recognize that this function computes an upper bound on the delay and not the delay itself. This is exemplified by the fact that while `par (comp A B) (comp C D)` and `comp (par A C) (par B D)` model the same real-life circuit, the delay function does not necessarily compute the same result for each. We later prove that the two results are ordered.

- `behavior : Circuit m n -> BoolVect m -> BoolVect n`. This function computes the behavior of a Circuit, modeled by a function from boolean vectors to boolean vectors, with lengths respective to the number of inputs and outputs of the Circuit. For example:
  - `behavior wire = fun x => x`
  - `behavior nand = fun x => [negb (fold_right andb x true)]`
  - `behavior (comp A B) = fun x => behavior B (behavior A x)`

Given the definitions above, we conveniently compare the areas and delays of different circuits via Coq's standard propositional (in)equality relations e.g. =, <=, <, etc. However, the = relation cannot be directly used to compare behaviors because of the dependent nature of the way they are modeled (functions dependent on the number of inputs and outputs of the Circuit). Overall, the = relation requires that its arguments' types are definitionally equal, disallowing those that are only propositionally equal. This for example prevents us from directly expressing the proposition `behavior A = behavior (par A none)` because `behavior A` has type `BoolVect m -> BoolVect n` and `behavior (par A none)` has type `BoolVect (m+0) -> BoolVect (n+0)`. To address this, we define our own notion of equality between behaviors (and vectors), making use of Coq's dependent propositional equality relation, `eq_dep`, as follows:

- `Notation "x =v y" := eq_dep nat BoolVect n x m y`
  where `x` is of type `BoolVect n` and `y` is of type `BoolVect m`.
- `Notation "a =b b" := forall (x : BoolVect n) (y : BoolVect m), x =v y -> eq_dep nat BoolVect o (a x) p (b y)`
  where `a` is of type `BoolVect n -> BoolVect o` and `b` is of type `BoolVect m -> BoolVect p`. Note that this definition further avoids the need for the functional extensionality axiom.

General Properties

We prove that the area, delay and behavior functions produce ring-like algebraic structures where the addition and multiplication operations correspond to the `par` and `comp` Circuit constructors, respectively. Below is an enumeration of the proved structural axioms.

Identities

We define `par_wire : forall n : nat, Circuit n n` as the identity for `comp`, where `par_wire n` represents n wires in parallel. We then prove:

- `area (comp (par_wire n) A) = area A`
- `area (comp A (par_wire m)) = area A`
- `delay (comp (par_wire n) A) = delay A`
- `delay (comp A (par_wire m)) = delay A`
- `behavior (comp (par_wire n) A) =b behavior A`
- `behavior (comp A (par_wire m)) =b behavior A`

where `A` has type `Circuit n m`.

We include `none : Circuit 0 0` in the definition of Circuit because it serves as the identity for `par`, proving:

- `area (par none A) = area A`
- `area (par A none) = area A`
- `delay (par none A) = delay A`
- `delay (par A none) = delay A`
- `behavior (par none A) =b behavior A`
- `behavior (par A none) =b behavior A`

Associativity

We prove `comp` is associative:

- `area (comp A (comp B C)) = area (comp (comp A B) C)`
- `delay (comp A (comp B C)) = delay (comp (comp A B) C)`
- `behavior (comp A (comp B C)) =b behavior (comp (comp A B) C)`

We prove `par` is associative:

- `area (par A (par B C)) = area (par (par A B) C)`
- `delay (par A (par B C)) = delay (par (par A B) C)`
- `behavior (par A (par B C)) =b behavior (par (par A B) C)`

Commutativity

We prove `comp` is commutative for `area` and `delay`:

- `area (comp A B) = area (comp B A)`
- `delay (comp A B) = delay (comp B A)`

We prove `par` is associative for `area` and `delay`:

- `area (par A B) = area (par B A)`
- `delay (par A B) = delay (par B A)`

Distributivity

We prove that `comp` distributes over `par` for `delay`:

- `delay (comp A (par B C)) =`
`delay (par (comp A (par B (par_wire p))) (comp A (par (par_wire n) C)))`
- `delay (comp (par B C) A) =`
`delay (par (comp (par B (par_wire p)) A) (comp (par (par_wire n) C) A))`

where `A` has type `Circuit m (n + p)`, `B` has type `Circuit n o` and `C` has type `Circuit p q`.

Idempotence

We prove that `par` is idempotent for `delay`:

- `delay (par A A) = delay A`

Par Comp

As mentioned above, `par (comp A B) (comp C D)` and `comp (par A C) (par B D)` model the same real-life circuit. We prove the following invariances between them:

- `area (par (comp A B) (comp C D)) = area (comp (par A C) (par B D))`

- `delay (par (comp A B) (comp C D)) <= delay (comp (par A C) (par B D))`
- `behavior (par (comp A B) (comp C D)) =b`
  `behavior (comp (par A C) (par B D))`

Circuit-Specific Facts

We prove the following relationships between `nand` and `comp and inv`.

- `behavior nand =b behavior (comp and inv)`
- `area nand < area (comp and inv)`
- `delay nand < delay (comp and inv)`

We prove the following relationships between `and3` and `comp (par and wire) and`.

- `behavior and3 =b behavior (comp (par and wire) and)`
- `area and3 < area (comp (par and wire) and)`
- `delay and3 < delay (comp (par and wire) and)`