Circuit verification and conformal equivalence

Vikas Kumar Rao
University of Utah
U1072596

Outline

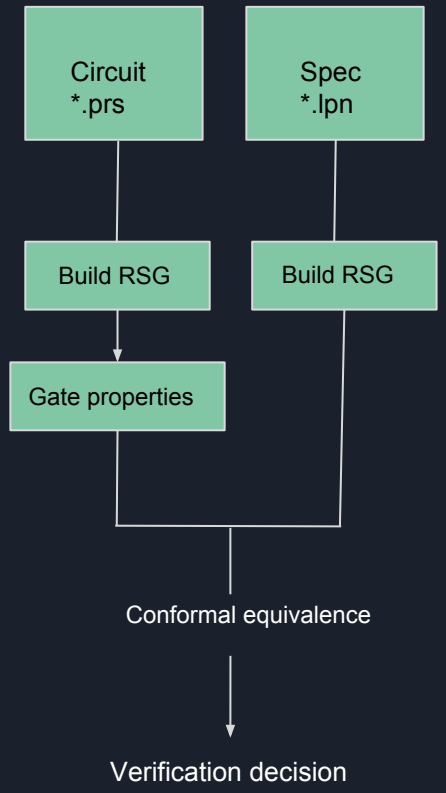
- Introduction
- Problem Statement
- Objective
- Implementation
- Current state
- Questions

Introduction

- Verifying whether a protocol, circuit has all the desired properties and is hazard free.
  - Protocol never deadlocks.
  - Acknowledge follows a request in bounded amount of time.
- Exhaustive simulation is not practical and does not guarantee correctness of the design.
- Hazard manifests only under a very particular set of delays especially in asynchronous designs.
- Verification needs to make sure all permissible delay behaviors are modelled and that the circuit operates correctly under all the delay combinations.
- System level verification challenge is to verify that all circuit behaviors are allowed by a given specification.

Problem statement - Inputs

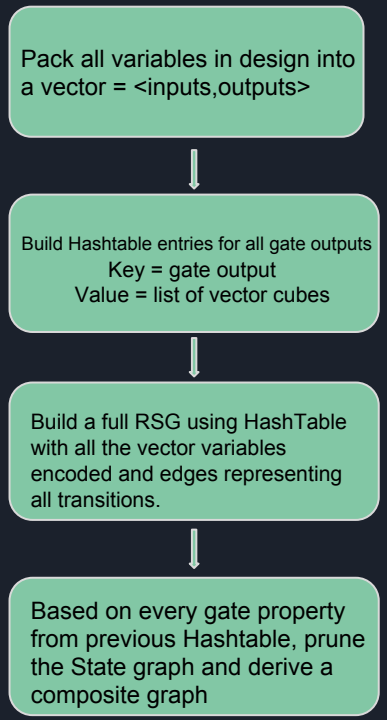
- Circuit -
  - Every gate in the circuit is modeled as either
    - Generalized C element (gC)
      - Where each signal '*u*' is given as a combination of
        - *set(u)* cube(s) -> [+x:(~y & ~z)]
        - *reset(u)* cube(s) -> [-x:(z)]
    - Combinational gate
      - Where each signal '*u*' will be represented as amalgam of combinational gate(s) -> [y:(z)]
- Specification (equivalence)-
  - Represented as a labeled petri net(lpn) with initial state and markings -> x+ z+ y+

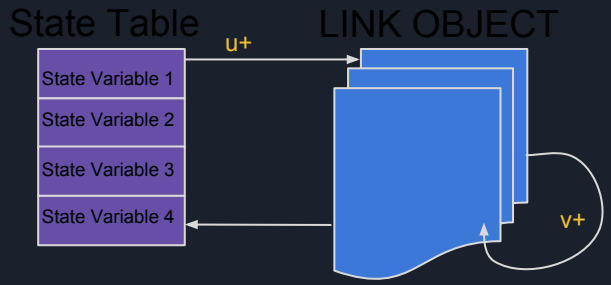


Objective

- Represent Circuit and Specification as Receptive State Graph(RSG).
  - state of the circuit cannot prevent an input from happening.
- Circuit -
  - Derive a composite RSG for the whole circuit -
    - Derive RSG for all individual gates, check for state inclusion, prune disallowed transitions, redirect illegal transitions at every node of every graph and come up with a composite RSG.
      - No initial state defined for gC gate descriptions.
      - Complicated to traverse individual gate graphs and modify all graphs.
    - Build a complete state graph for all the inputs mentioned in the design to start with and prune the state space and transitions based on the behavior described by the individual gates in the circuit.
      - Large state space (2^n : n-inputs) and transitions initially, but easier to traverse on one graph.
- Specification (If time permits)-
  - Derive RSG and verify that at all nodes, all transitions where the specification is failure-free, the circuit is also failure-free.

Implementation - Flowchart





Implementation contd..

- Every node in state graph which is implicitly covered by gate cubes should follow the behavior exhibited by individual gates.
- A transition is success in composite graph state node, if it is a success in every individual gate property.
- A transition is a failure in composite graph, if it is a failure in any of the individual gate property.
- A fail state(F) occurs when an input changes at the wrong time
    - A rising output transition cannot change to zero in consecutive state (should be redirected to fail sate).
    - A falling output transition cannot change to one in consecutive state (should be redirected to fail sate).
- For a set cube, the falling output transition from every implicit state node satisfying the state inclusion should be discarded.

Current state (pun intended)

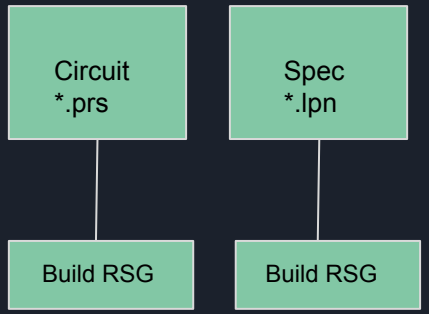
- Implemented a parser to read the *.prs file line by line and store the gates as single cube string (implemented using HashTable(key,value(list)) pair.
  - Every gC gate has set cube and reset cube entries.
  - Every combinational logic for distinct output has one entry with multiple cubes as values.
- The cube string is represented as -
  - literal = '1'
  - complemented literal = '0'
  - missing literal = '-'
- The combinational gate is represented as a list of set function only and the reset function cube is left blank.
- Implemented a Hash table data structure with (key, value(list of linked lists)) data structure to store/manipulate the state graph.

Next state...

- Implement the generic state graph for all given variables in the design with all inputs as valid events/transitions in every node.
- Implement a private data structure class for Hashtable to store, modify the state graph.
  - The Hashtable key to be stored will be the state vector(<inputs,outputs>)
  - The value will be the list of link objects representing the next state transitions which will be checked against every state graph value for state inclusion.
- Prune the transitions of the state graph and build a new state table based on the valid transitions and properties derived from each gate cube from circuit.

THANK YOU

QUESTIONS?


Circuit
*.prs
Spec
*.lpn
Build RSG
Build RSG