Decomposing Specifications to Resolve State Coding Conflicts in Asynchronous Logic Synthesis

Hemangee K. Kapoor  Mark B. Josephs
Centre for Concurrent Systems and VLSI
London South Bank University
{kapoorhk, josephmb}@lsbu.ac.uk

Abstract

Petri nets, interpreted as Signal Transition Graphs (STGs), are widely used to specify asynchronous control circuits. The tool Petrify inputs such a description and converts it into a state graph (SG) prior to synthesis. Construction of Petri nets manually is cumbersome and error prone. More conveniently, the front-end tool di2pn takes a program in the language of Delay-Insensitive Sequential Processes (DISP) and automatically generates a Petri net.

In order to synthesise a circuit, Petrify requires an SG to have a complete state coding (CSC). When an SG does not have CSC, the specification needs to be modified. Petrify employs heuristics to insert internal signals (extra state variables) in order that different markings might correspond to different states.

Two common causes leading to CSC conflicts identified by us are when the specification includes

- concurrent outputs and these outputs are absorbed by different components in the environment. In such a situation the environment may respond to these outputs with corresponding input events in an arbitrary order.

- a self-contained block, i.e., a block in which every signal is transitioned an even number of times, and the start state needs to be distinguished from the finish state.

We present the designer with heuristics that can be applied to decompose such specifications into a form in which Petrify can solve CSC. The language of DISP in which this decomposition is carried out is high level compared to SGs and Petri nets. The heuristics used in the case of concurrent outputs introduce Fork elements to reduce concurrency, while Wires are introduced to act as state variables in the self-contained blocks, thus reducing potential CSC conflicts. (Forming a Gray code with these variables is particularly effective.)

To evaluate the heuristics, they were applied to a total number of 19 benchmark examples. In the cases of pscsi-trcv, scsi-fast-initiator-send and loadable counter, Petrify could not solve CSC on the original specification, but could successfully synthesise the decomposed specification. In the other cases the decomposed specifications

- for concurrent outputs synthesised on average 2.7-times faster, saving 25% in area, 39% in set-reset pins required and 52% in state variables inserted by Petrify.

- for self-contained blocks synthesised on average 3.4-times faster, saving 11% in area, 23% in set-reset pins required and 100% in state variables inserted by Petrify (i.e. solving CSC in all cases).