2H−9

Synthesis Algorithm for Asynchronous Circuits from STG specifications

Mohit Sahni
Computer Science Department
Tokyo Institute of Technology

Takashi Nanya
Research Center for Advanced Science & Technology
University of Tokyo

Abstract

In this paper we propose methods for synthesis of asynchronous logic from graph theoretic specifications (STG). Our methods operate purely at the STG level and can handle non-persistent STGs with choice operation.

1 Introduction

The role of asynchronous design is gaining importance due to inherent limitations of synchronous circuits[1]. Signal Transition Graphs(STGs), which can be used to specify the behaviour of asynchronous control circuits, were first introduced by Chu[2]. STGs can efficiently capture the concurrent, sequential, conflict or choice relations between signal transitions and thus can consisely describe the behaviour of asynchronous circuits.

There has been considerable research to generate asynchronous logic automatically. Some use the State Graph(SG) of exponential complexity as an intermediate step and other methods work directly on the STG. Our algorithm also uses the STG as the main data structure and can generate circuits from a large domain of STGs.

2 Preliminaries

Speed-independent circuits are defined to be a class of asynchronous circuits that operate correctly in the presence of unbounded gate delays and zero wire delays. In this model the circuit may not be stable when new inputs are applied.

2.1 STG

An STG is an interpreted Petri net whose transitions are interpreted as the rising and falling of signals in a circuit. Each place in the Petri net is used for specifying choice operation and is eliminated in the corresponding STG if it has one predecessor transition and one successor transition. Fig. 1 shows an STG. The *input* signals are underlined to distinguish then from the *non-input (internal and output)* signals.

Each transition of a signal is represented by the signal name and a direction eg. t^+ (t^-) reprents the rising (falling) of the signal t. If there are two or more rising (falling) transitions of the same signal, numbers are assigned to distinguish the different transitions. If transition t^* follows directly after transition s^* then s^* is called the *trigger* transition of t^*. A transition is said to be enabled when all it's trigger transitions have fired.

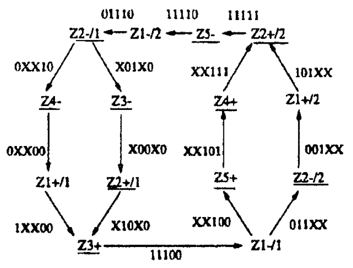

Figure 1: An STG

2.2 Properties of STGs

Definition 1.[2] A *simple-cycle* in an STG is a cycle which includes all the rising and falling transitions of a particular signal.

Definition 2[2] An STG is *live* if
1. every simple cycle has exactly one token
2. it is a strongly connected graph
3. the rising and falling transitions are alternated.

Definition 3[4] A live STG satisfies the *CSC(complete state coding) property*, iff
1. every state has a unique code or
2. if two states have the same code then all the signal transitions enabled in these states are the same.

Liveness and CSC are the only two necessary conditions for the given STG to be implementable however other methods may impose other severe restrictions. From now on we assume that the given STG satisfies these two properties.

3 Basic Circuit Structure

We use a generalized circuit structure (Fig. 2) for synthesizing the circuits. For each non-input signal we construct a partial circuit and the total circuit consists of all the parial circuits. The partial circuit consists of *set(reset) region networks* and the *set(reset) acknowledgement networks.*

Definition 4 *Set/Reset Region*: The maximum set of transitions firable after $t^{+/*}(t^{-/*})$ is enabled and before the next transition of t is enabled is called the set(reset) region of $t^{+/*}(t^{-/*})$.

A set(reset) region network is associated with each rising(falling) transition of a signal. Under speed independent conditions, the output of the set/reset region network for transition t^* will become 1 when the t^* is ready to fire and it remains 1 for sometime after t^* fires but will become 0 before the next transition of t is enabled.

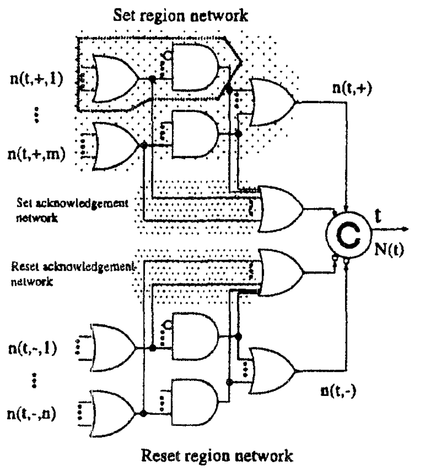

Figure 2: Generalized Signal Network

4 Speed Independent Implementation

In this section we first define some lock relations. Note that $\mathcal{A}$ denotes the sum or product of some literals.

Definition 5 *Partial Lock*: A signal t is partial-locked with $\mathcal{A}$ if between two consequtive transitions of signal t the value of $\mathcal{A}$ either remains constant or changes only once.

Definition 6 *Region Lock*: A function f has a region-lock with $t^{*/i}$ if the value of the function becomes 1 when $t^{*/i}$ is enabled. And in the region after the firing of $t^{*/i}$ and before the firing of the next transition of t, the value of the function becomes 0 and remains so until $t^{*/i}$ is enabled again.

Our synthesis procedure is basically a means of finding a region network such that all the AND gates in set/reset region network have a region lock with the corresponding transition, and all the OR gates in the set/reset region network have a partial-lock with the corresponding signal. If these two conditions can be satisfied then it can be guaranteed that no change in the output of a gate goes unrecognized and hence we get a hazard-free speed-independent circuit.

Extended Graph(EG): All the computation is carried on the STG by transforming the original petri net into an *Extended Graph*[5] as shown in the Fig. 1. Codes with dont cares are assigned to each place in the petrinet.

The EG by definition is equivalent to the SG so there is no need to generate the SG as all computation can be carried on the EG.

Region Network Synthesis: The literals needed for the AND gate to establish a region-lock can be selected by simple traversals on the EG. For finding the the set $\mathcal{A}$ for partial-locks we use simple heurestics as the problem tends to be NP-complete. The steps in the synthesis method can be briefly stated as below. Let $t^{*/i}$ be the non-input transition under consideration

1. Starting from the trigger transition signals try to add literals until the cube is region-locked with t
2. If no such cube exists then we also need to consider signals which are parallel to $t^{*/i}$.
   - (a) Find a set of literals $\mathcal{A}$ from the signals which are parallel to $t^{*/i}$ such that $\mathcal{A}$ is partial-locked with t through $t^{*/i}$ and $\mathcal{A}$ has not been tried before. If no such literals exist return *LockError*.
   - (b) Consider $\mathcal{A}$ as a literal and goto Step 2.
   - (c) Add acknowledgement forks appropriately.
3. Optimization

The circuit for the STG in Fig. 1 is shown in Fig. 3.

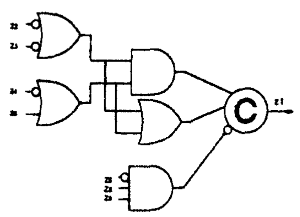

Figure 3: Circuit of STG in Fig.1

5 Conclusion

We propose a new polynomial time complexity method to generate asychronous logic from STGs which are live and have the CSC property. We do not assume the STG to satisfy any other sufficent conditions like persistency and neither do we assume the availability of complex gates which may have internal hazards. Hence we can generate circuits from a wide domain of STGs. As far as we know, existing methods can't handle the STG in Fig. 1.

References

[1] T.Nanya, "A new perspective on asynchronous VLSI system design (invited paper)", *Proc. of 3rd Asia Pacific Conf. on Hardware Description Languages*, pp.120-127 (Jan, 1996)

[2] T.Chu, "Synthesis of self-timed VLSI circuits from graph-theoretic specifications", *PhD Thesis*, Massachusetts Institute of Technology, June 1987.

[3] S.Park and T.Nanya, "Synthesis of Asynchronous Circuits from Signal Transition Graph Specifications", *To appear in IEICE Trans. Information and Systems*, March 1997

[4] C.W.Moon, "Synthesis and verification of asynchronous circuits from graphical specifications", *PhD thesis*, Univ of California at Berkeley, 1992

[5] E.Pastor, J.Cortadella, A.Kondratyev and O.Roig, "Structural methods for the synthesis of speed-independent circuits", *Proc. of European Design and Test Conference*, 1996