An Asynchronous Synthesis Toolset using Verilog

Frank Burns, Delong Shang, Albert Koelmans and Alex Yakovlev

School of Electrical, Electronic & Computer Engineering, University of Newcastle Upon Tyne
Newcastle Upon Tyne, NE7 1RU, UK, f.p.burns@ncl.ac.uk

Abstract

We present a new CAD tool set for generating asynchronous circuits from high-level Verilog level-sensitive specifications. Initially high-level Verilog descriptions are compiled and converted into a novel intermediate Petri-net format. The intermediate format is subsequently passed to optimization tools and mapping tools where it is directly mapped into asynchronous datapath and control circuits using David Cells (DCs). Finally logic optimization tools are applied to generate speed independent (SI) circuits. The speed independent circuits generated perform well compared to circuits generated by existing asynchronous tools.

1. Introduction

Compared to synchronous CAD tools which are very mature and accepted by industry, there is still a shortage of mature CAD tools to support asynchronous circuit designs. There are only a few tools available which generate complete datapath and control asynchronous solutions from high-level descriptions, e.g. Balsa [1] is completely asynchronous and uses its own specification languages based on handshaking. The Verilog language is used in [2] to specify micropipelined designs out of which controllers are generated. This technique, however, suffers from the state explosion problem when designs become larger because they refine to signal transition graphs and subsequently use logic sythesis.

Our aim is to avoid the state explosion problem by using a Petri-net synthesis approach like [3] but which combines the advantages of using a direct mapping method and using Verilog as a specification language. We wish to be able to generate guaranteed speed independent asynchronous circuits for designs which compete with those generated from mature asynchronous tools such as Balsa.

Fig. 1 gives a block diagram which shows the synthesis flow exhibited by our technique.

Our synthesis approach inputs a high-level Verilog specification, optimizes and schedules it, and translates it into an intermediate multi-Petri-net representation. This is subsequently synthesized into datapath circuits and control circuits using direct mapping techniques. After mapping optimized speed independent circuits are generated.

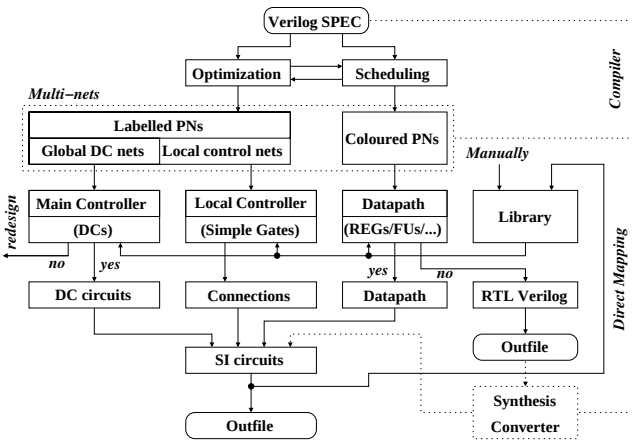

Figure 1. Diagram of synthesis flow

2. Synthesizing Petri nets from Verilog

We use the ICARUS compiler for compiling the Verilog specifications. The scheduler schedules sequential assignments in their natural order allowed by their dependencies and consecutive independent assignments in parallel. The scheduler handles basic Verilog constructs including: sequential assignments, conditional statements, i.e. if, case, and fork and join statements, cyclic behaviour, e.g. always, repetitive behaviour, i.e. while, repeat and for loops, and higher level constructs such as functions and tasks.

The Petri-net synthesizer uses the control output from the scheduler to generate the intermediate Petri-net description. The intermediate Petri-net format is a multi-net format comprised of datapath nets based on Coloured Petri Nets (CPNs) and control nets based on Labelled Petri nets (LPNs). The control nets are split into two types for mapping: (i) global control nets which are used for direct mapping to David Cells (DCs) [5] and (ii) local control nets for mapping to simple control gates (see Fig. 1).

The datapath net Coloured PN (CPN) is constructed us-

1530-1591/04 $20.00 (c) 2004 IEEE

ing transitions to represent functional units (FUs) and pairs of similarly named transitions to represent multiplexing.

The control nets are constructed in two parts: the global DC control nets which are based on the number of control steps required by the scheduler and the local control nets which translate the DC control net output signals into the firing signals required for the datapath net transitions.

Optimizations are applied to minimize the number of DC cells in the DC-net and to reduce nodes in the local net prior to merging the nets together and logic mapping.

3. Generation of SI circuits

Once the datapath nets and control nets have been constructed a syntax-based approach is used to map them directly into circuits. The data Petri-net fragments are translated into dual-rail datapath net-lists. The DC control nets are translated into DC net-lists. The local control nets are translated into local control flow circuits which are merged with the datapath net-lists and DC net-lists to give the local control circuit connections. Subsequently an optimized SI circuit is generated. An example output circuit from a Verilog sequential example is shown in Fig. 2.

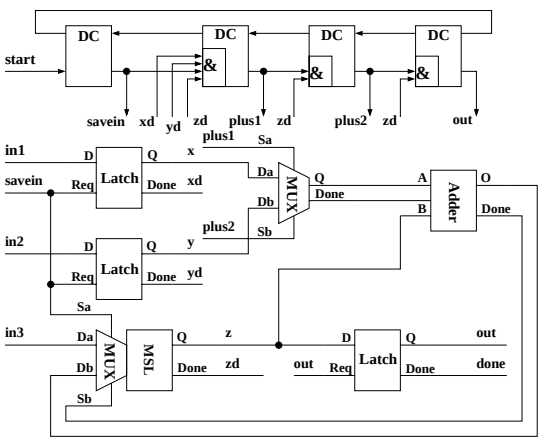

Figure 2. Example of generated logic circuit.

Mapping is achieved using a library of specialized dual-rail components (AMS 0.35um cell library) which include datapath elements, C-elements and various kinds of DCs, for example, the DCs at the top of Fig. 2 'AND' different numbers of input signals together from the datapath below.

In our method, DCs are used instead of a global clock signal to activate the control steps [4]. This divides all the operations into several stages based on the sequence of operations. One cell is equivalent to one global DC place and transition at the top of Fig. 2.

DCs can be joined to make more elaborate Verilog control constructs such as fork, join, conditions and loops.

4. Results

Our synthesis toolset has been successfully applied to a range of Verilog examples. The example set covers most of the basic Verilog level-sensitive constructs so far that are accepted by commercial synthesis tools. In addition we have tried to test our toolset against examples entered into the asynchronous Balsa tool. As a comparison we have chosen the GCD (Greatest Common Divisor) benchmark.

Table 1. GCD Comparison

start to data out	Verilog	Balsa
x = y	14.3ns	20.8ns
x = 12, y = 16	108.9ns	187.6ns

Table 1 shows an improvement in time for our synthesized results over those generated by Balsa using the same technology (AMS 0.35um cell library). The speed up is due mainly to our more efficient Petri-net based implementation methodology which we have used to synthesize our circuits. This methodology exploits a semantical, execution-flow, link from Verilog to circuits. Balsa uses syntax-directed techniques, based on handshakes, and uses excessive memory elements.

5. Conclusions

A new synthesis approach has been described for synthesizing asynchronous datapaths and controllers from Verilog using a direct mapping approach in order to generate speed independent implementations. A combination of Petri nets has been found to be beneficial in direct mapping of the behaviour into asynchronous datapath and control circuits.

We can generate asynchronous circuits quickly and, in addition, the 1-hot approach guarantees better safety than other asynchronous approaches. Comparisons with other asynchronous tool sets (Balsa) show our synthesized circuits to be competetive from a performance view.

References

[1] A. Bardsley and D. Edwards, Compiling the language Balsa to delay-insensitive hardware, Hardware Description Languages and their Applications (CHDL), pages 89-91, 1997.

[2] I. Blunno and L. Lavagno. Automated synthesis of micro-pipelines from behavioral Verilog HDL, Proc. of IEEE Symp. on Adv. Res. in Async. Cir. and Syst. (ASYNC'2000), pp. 84-92.

[3] P. Eles, K. Kuchcinski and Z. Peng, System Synthesis with VHDL, Kluwer Academic Publishers, P.O. Box 17,3300 AA Dordrecht, The Netherlands, 1998.

[4] D. Shang, F. Xia and A. Yakovlev. "Asynchronous Circuit Synthesis via Direct Translation", ISCAS 2002, IEEE International Symposium on Circuits and Systems, May 2002.